Java aktuell

iJUG Verbund
www.ijug.eu

Topaktuell
Alle Neuheiten von Java 10

Aus der Praxis
Bastlerfreundliche Heim-Automatisierung

Der neue Trend
Serverless Java mit Fn Project

Java ist cool

Ausgabe 04/2018

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 04

adesso business. people. technology.

SOFTWARE DEVELOPMENT@adesso

Von Anfang an Teil des Java-Teams!

Entwickelt bereits kluge IT-Lösungen bei adesso:
Ihr neuer Kollege Kristof Zalecki | Software Engineer

Sie wollen dort einsteigen, wo Zukunft programmiert wird? Dann sind Sie mit einem Start in unserem Java-Team bei adesso genau richtig. Gemeinsam setzen wir herausfordernde Projekte für unsere Kunden um. Dafür brauchen wir Menschen, die Lust haben, ihr Wissen, ihre Talente und ihre Fähigkeiten einzubringen.

Planen und realisieren Sie in interdisziplinären Projektteams anspruchsvolle Anwendungen und Unternehmensportale auf Basis von Java/JavaScript-basierten Technologien als

→ (Senior) Software Engineer (m/w) Java
→ Software Architekt (m/w) Java
→ (Technischer) Projektleiter Softwareentwicklung (m/w) Java

CHANCENGEBER – WAS ADESSO AUSMACHT

Kontinuierlicher Austausch, Teamgeist und ein respektvoller, anerkennender Umgang sorgen für ein Arbeitsklima, das verbindet. So belegen wir beim Wettbewerb „Deutschlands Beste Arbeitgeber in der ITK 2018“ den 1. Platz!

Mehr als 650 Software Engineers Java bei adesso, über 120 Schulungen und Weiterbildungen – zum Beispiel in Angular2 oder Spring Boot – sowie ein Laptop und ein Smartphone ab dem ersten Tag warten auf Sie!

GREAT PLACE TO WORK® 2018 Beste Arbeitgeber™ ITK

IHRE BENEFITS – WIR HABEN EINE MENGE ZU BIETEN:

- Welcome Days
- Choose your own Device
- Weiterbildung
- Events: fachlich und mit Spaß
- Sport-förderung
- Mitarbeiter-prämien
- Auszeit-programm

Es wird Ihnen bei uns gefallen! Mehr Informationen auf **www.karriere.adesso.de**. Olivia Slotta aus dem Recruiting-Team freut sich auf Ihre Kontaktaufnahme: **adesso AG // Olivia Slotta // T +49 231 7000-7100 // jobs@adesso.de**

Die beste JavaLand aller Zeiten

Zum Jubiläum der fünften JavaLand kamen vom 13. bis 15. März 2018 mehr als 1.900 Entwickler aus zwanzig Nationen in das Phantasialand in Brühl. Nach dem Auftakt im Jahr 2014 mit 800 Teilnehmern ist die Veranstaltung ständig stark gewachsen. Sie bietet den Java-Enthusiasten mit Vorträgen nationaler und internationaler Speaker, zahlreichen Aktivitäten der Community und einem Schulungstag den idealen Treffpunkt zum Wissensaustausch und wird in Kooperation mit Heise Medien und dem Interessenverbund der Java User Groups e.V. (iJUG) organisiert.

Den Auftakt zur diesjährigen Konferenz gab Keynote-Speakerin Dr. Holly Cummins. In ihrem Vortrag „Cloudy with a Chance of Meatballs: Cloud Surprises for the Java Developer" teilte sie im bis auf den letzten Platz besetzten Vortragssaal neben einer historischen Perspektive auf Cloud Computing auch einige Gedanken darüber, wohin die Entwicklung in Zukunft gehen könnte. Dabei zeigte sie den Teilnehmern insbesondere einige Fallstricke auf, über die Entwickler beim Wechsel in die Cloud stolpern können.

Neben den rund 110 Vorträgen in acht Streams zu Themen wie „Architektur und Sicherheit", „Container und Cloud", „Enterprise und Microservices" sowie „Innovationen" sorgten auch die aktuellen Entwicklungen im Java-Kosmos rund um die Zukunft von Java EE/Jakarta EE für viel Gesprächsstoff. Auch die vielen Community-Aktivitäten, die zusammen mit zahlreichen Java User Groups gestaltet wurden, boten den Teilnehmern viele Möglichkeiten zum Ausprobieren, Entdecken und Spielen.

Bereits am Vortag fand die JavaLand4Kids, die kleine Schwester der JavaLand, statt. Vierzig Kinder und Jugendliche im Alter von acht bis achtzehn Jahren nutzten in vier spannenden Workshops die Gelegenheit, die Welt des Programmierens zu erkunden.

Nicht zu vergessen, die super Stimmung auf der Abendveranstaltung. Zwischen internationalen Restaurants, Fahrgeschäften des Phantasialands und der Band blieb viel Raum, sein Netzwerk zu pflegen und neue Kontakte zu knüpfen.

Die meisten Artikel in dieser Ausgabe sind von den Referenten der JavaLand verfasst, sodass Teilnehmer die Veranstaltung nochmals Revue passieren lassen können und alle anderen sehen, was sie auf der JavaLand erwartet.

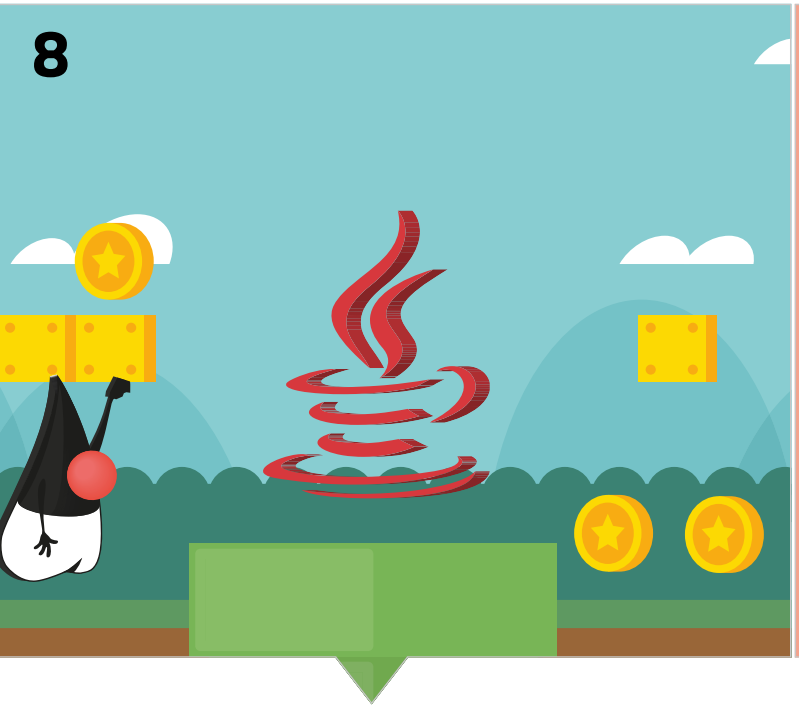
Das Datum für die sechste Ausgabe der JavaLand steht bereits fest: Sie findet vom 19. bis 21. März 2019 an gewohnter Stätte im Phantasialand statt. Interessierte können sich bereits jetzt für den Newsletter anmelden, der über Call for Papers und Registrierung informiert.

Ich bin gespannt, ob die JavaLand 2019 wieder einen neuen Besucherrekord bringen wird.

Ihr

Wolfgang Taschner
Chefredakteur Java aktuell

Inhalt



Java 10 bietet einige interessante Features

Soft Skills sind für jeden Software-Entwickler interessant

Die Steuerung von Geräten im Haus automatisieren

Die funktionale Programmierung steht vor der Tür

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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

8. März 2018

Kein JavaFX mehr im OpenJDK

Oracle will JavaFX komplett aus dem OpenJDK heraustrennen und das Projekt OpenJFX eigenständig weiterführen. Ab Java 11 soll es als getrennter Download angeboten werden – und „mit seiner eigenen Geschwindigkeit" weiterentwickelt werden. Oracle verspricht sich dadurch auch eine größere Beteiligung Dritter an dem Projekt. Java 11 ist das nächste „Long Term Support"-Release. Mit diesem Schritt ist aber langfristiger (kommerzieller) Support für JavaFX im OpenJDK nur noch bis Release 8 gewährleistet – also bis 2022. Kostenlose Updates gibt es für Java 8 dann noch bis mindestens Ende 2020 für private Nutzung beziehungsweise Januar 2019 für Unternehmen. Was das jetzt alles für die Zukunft von JavaFX bedeutet, hängt von der Community ab. Es lebe Swing!

https://blogs.oracle.com/java-platform-group/the-future-of-javafx-and-other-java-client-roadmap-updates

12. März 2018

JavaLand: Java Lobby Day und JUG Leaders Day, JavaLand4Kids

Etwas Neues für den Vortag der JavaLand-Konferenz: Neben der etablierten JavaLand4Kids gibt es heute jede Menge Gelegenheit zum Networking: Zum „Java Lobby Day" treffen sich Entwicklerinnen und Entwickler aus den zentralen Programmen des Java-Ökosystems – insbesondere Jakarta EE, MicroProfile und OpenJDK. Zum „JUG Leaders Day" sind Vertreter von JUGs aus aller Welt eingeladen. Die beiden Veranstaltungen werden aber – ganz agil – schnell zusammengelegt, weil die Interessen sehr ähnlich sind. Der Tag ist als Unconference organisiert – die Sessions und ihre Anordnung werden in der großen Runde festgelegt. Es geht um technische Themen, aber auch viel um den Status der genannten Projekte und einen Ausblick in die Zukunft von Java. „Long Term Support" ist eines davon – der Tenor: Oracle schafft sich hier zum ersten Mal ein tragfähiges Geschäftsmodell für die Java-Plattform; aber dieses können auch andere Hersteller mit eigenen JDK-Releases nutzen. Insofern könnte das Ergebnis im Sinne der Vielfalt und eines robusten Ökosystems sehr positiv sein. Die Resonanz zur Veranstaltung ist sehr gut. Solche Treffen sind extrem wertvoll für alle – persönlich und um die Projekte voranzutreiben. Viel Gelegenheit bekommen sie dazu sonst nicht (siehe Seite 64).

13. März 2018

JavaLand: Tag 1

Die Konferenz beginnt mit einer Keynote von Holly Cummins von IBM zur Cloud und ihren Fallstricken. „Serverless bedeutet nur: der Server eines anderen" ist eine zentrale Aussage. Der erste Tag bietet mehrere Panel-Diskussionen, etwa eine zum Thema „Java SE Release Train und Long-Term Support" – was gut ist, denn insbesondere die Änderungen zum Thema Support haben sich offensichtlich noch nicht weit herumgesprochen. Auch abends wird neben Karussells, verschiedenen Restaurants oder einfach einem Bier beim Warten auf die Band noch inhaltlich einiges geboten. Bei mehr als 1.900 Besuchern soll ja für alle etwas dabei sein. Und nächstes Jahr wird die 2.000er-Marke geknackt. Dann allerdings mit logistischen Verbesserungen insbesondere bei den Keynotes (wie Videoübertragung in einen anderen Saal), weil sonst die Kapazitäten nicht reichen.

https://www.javaland.eu/de/javaland-2018

14. März 2018

JavaLand: Tag 2

Auch am zweiten Tag der JavaLand gibt es eine Panel-Diskussion, diesmal über die Zukunft der Frontends. JavaFX ist natürlich ein großes Thema. Es gibt keine einheitliche Meinung zur Zukunft, aber offensichtlich Potenzial: „Wir haben jede Menge Code, den wir zu OpenJFX beitragen können, und mit dem neuen Modell ist das besser möglich", sagt Markus Karg, der auch in der Vergangenheit mit TeamFX schon versucht hat, die Entwicklungsgeschwindigkeit des Frameworks zu erhöhen. Die Community Hall mit Early Adopters Area, Innovation Lab, Code-Golf etc. ist gut gefüllt. Es sind mehr Aussteller da (welcome back, Oracle); aber dadurch, dass ein angrenzender Nebenraum zur Halle hin geöffnet wurde, ist es nicht zu eng. Letztes Jahr haben wir selbst Interviews mit Specification Leads und anderen Java-Größen geführt, die sich in der Community Hall aufhielten. Aber das hatte noch Potenzial, deshalb machen wir es diesmal anders: Wir bitten geeignete Paare, sich gegenseitig zu interviewen. Amelia Eiras von TomiTribe bietet organisatorische Unterstützung an und nach kurzer Zeit stehen die Interview-Kandidatinnen und -Kandidaten Schlange. Die gut gelungenen Videos sind demnächst auf javaland.eu zu sehen.

https://www.javaland.eu/de/javaland-2018

21. März 2018

Java SE 10

Java 10 ist erschienen. Durch den neuen Release-Train mit einem Release alle sechs Monate kommt es nicht mit einer Fülle von Features, aber dafür ist der Zug pünktlich. Automatische Typinferenz für lokale Variablen mit „var" und der neue, in Java geschriebene JIT-Compiler Graal als experimentelle Option sind zwei der neuen Features. Darüber hinaus wurde der JDK Forest in ein einziges Repository konsolidiert – mit Ausnahme von JavaFX natürlich. Das ist eine deutliche Erleichterung, etwa bei komplexeren Bug-Fixes, die bislang Änderungen an mehreren Repositories bedeutet hätten, macht es allerdings auch für alle leichter, die sich einfach mal so mit dem OpenJDK auseinandersetzen möchten.

http://openjdk.java.net/jeps/0

27. März 2018

Oracle „Java Client Roadmap“

Oracle hat ein neues Roadmap-Dokument veröffentlicht, in dem die Pläne zu Java auf dem Client zusammengefasst sind. Neben der bekannten Ausgliederung von JavaFX steht dort auch, dass Java Web Start ebenfalls ab Version 11 nicht mehr Bestandteil des Java-SE-Release sein wird.

https://bit.ly/2FOkZyJ

28. März 2018

MicroProfile 1.4 und 2.0 verzögern sich

Das MicroProfile-Projekt war im vergangenen Jahr überaus produktiv – trotz beziehungsweise nach überstandenem Umzug zur Eclipse Foundation. Die neuen Releases 1.4 (Fokus: „Developer Experience“ – also umfangreichere Samples, verbesserte Dokumentation etc.) und 2.0 (Fokus: von Java EE übernommene Spezifikationen auf EE-8-Stand bringen) verzögern sich allerdings. Beide sollten Ende März erscheinen. In der MicroProfile Google Group wird diskutiert, jetzt doch Einzel-Spezifikationen mit 1.4 auf eine neue Version zu heben, weil „Developer Enablement“ kein Release-Event sei. Es geht halt agil zu. Aktualisierte Einzel-Spezifikationen gibt es wohl genügend für ein Release (Config 1.3, REST Client 1.1, Fault Tolerance 1.1 etc.). Wäre schön, wenn sie bald verfügbar sind – nicht nur als Microprofile 1.4, sondern auch als fertige Implementierung der teilnehmenden Application-Server. Eine Referenz-Implementierung gibt es hier ja nicht.

https://projects.eclipse.org/projects/technology.microprofile

19. April 2018

Code One

Die Gerüchte der letzten Wochen um das Ende der JavaOne-Konferenz stimmen. Stephen Chin bloggt, dass die JavaOne vergrößert wird, um diesen Herbst in einem neuen Event namens „Code One“ aufzugehen, an gleicher Stelle in San Francisco. Die JavaOne ist tot – es lebe die Code One! „Go, Rust, Python, JavaScript, SQL, R zusammen mit mehr an großartigem Java Content“, so Stephen. Bislang habe ich das allerdings auf der JavaOne nicht vermisst. Der eine oder andere SQL-Schnipsel und alles, was auf der JVM kreucht und fleucht, war auch dort immer zu finden, wenn es sinnvoll war. Aber gut, Java ist nicht allein auf der Welt und Oracle möchte den kompletten „Zoo“ bedienen, der im Konzern entstanden ist.

https://blogs.oracle.com/author/stephen-chin

20. April 2018

JCP-Nachwahlen

NXP Semiconductors hat seinen Sitz im Executive Committee des JCP aufgegeben. Anfang Mai soll daher ein neuer Kandidat abgenickt werden (es ist ein „ratified seat“, also von Oracle vorgeschlagen). Der chinesische eCommerce-Konzern Alibaba könnte damit nach mehreren vergeblichen Kandidaturen für frei gewählte Sitze in Zukunft den riesigen weißen Fleck auf der Landkarte der EC-Mitglieder tilgen.

https://www.jcp.org

24. April 2018

Microsoft

Microsoft hat sich schon seit einiger Zeit wieder stärker im Java-Umfeld engagiert und tritt folgerichtig jetzt der Jakarta EE Working Group bei. Das könnte ein großer Schritt für „Java Enterprise“ und diejenigen sein, die dafür entwickeln.

https://dev.eclipse.org/mhonarc/lists/jakarta.ee-wg/msg00054.html

26. April 2018

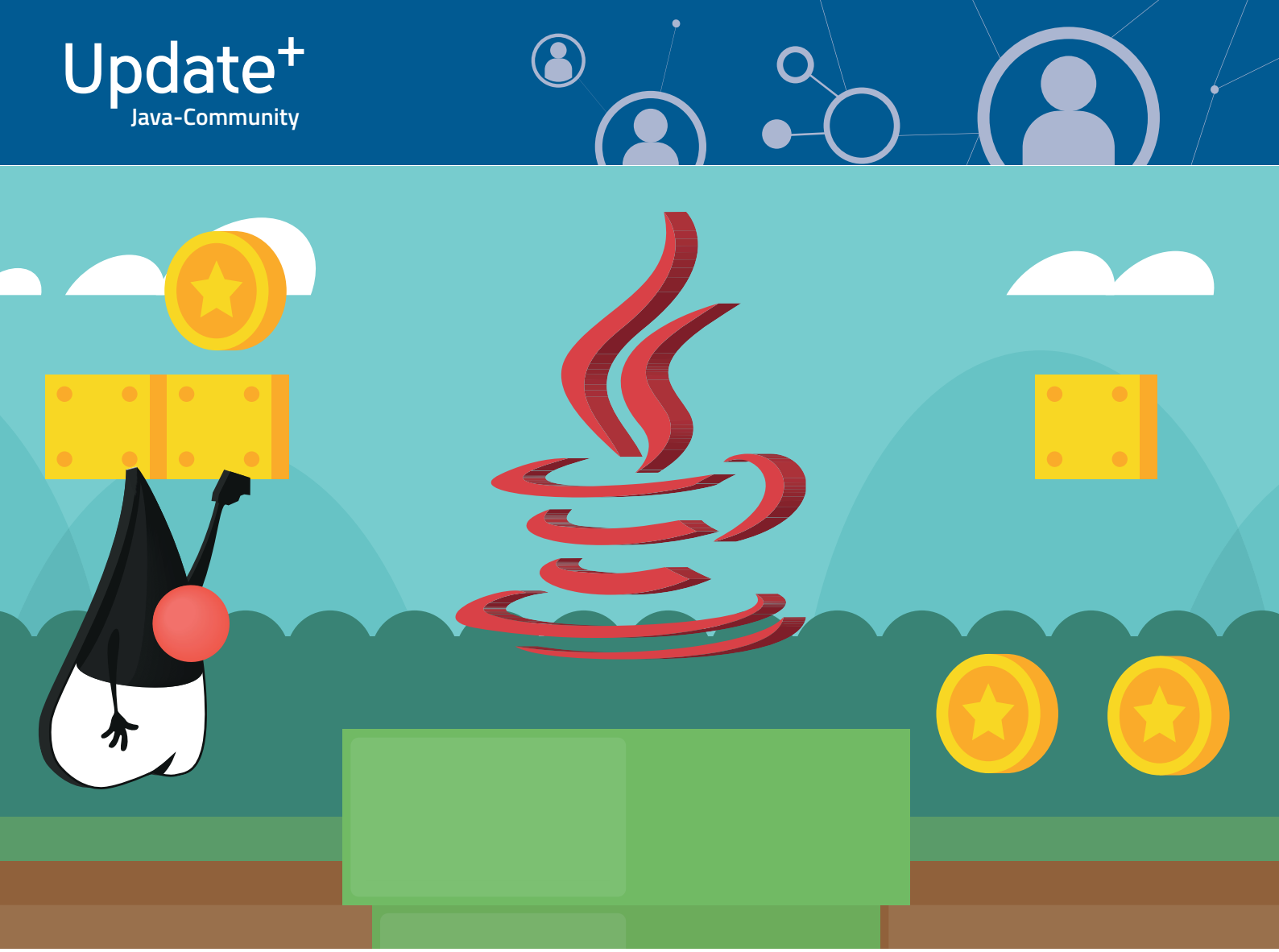
GraalVM

Das passt gut zur Code One: Oracle hat das Release 1.0 der GraalVM freigegeben. Mit dem neuen Graal-Compiler der JVM im Kern soll sie hohe Performance und Interoperabilität für eine ganze Reihe von Programmiersprachen bieten: Python 3, Ruby, R, JVM-Sprachen wie Java, Scala und Kotlin, JavaScript inklusive Node.js sowie LLVM-basierte Sprachen wie C und C++. Alles zusammen ist in einer universellen VM – standalone oder im Paket mit dem OpenJDK, Node.js, der Oracle DB oder mySQL. Die Tools wie Debugger, Profiler etc. sollen alle sprachunabhängig sein. Das Projekt gibt es als (Open Source) Community Edition – allerdings ohne jede Garantie und Support. Ansonsten ist die Enterprise Edition erforderlich, neben fertigen Binaries für Linux/x86 64-Bit sind auch MacOS-Binaries dabei.

http://www.graalvm.org

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

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



Java geht in die nächste Runde

Falk Sippach, Orientation in Objects GmbH

Nur die wenigsten haben die Neuerungen von Java 9 bereits verdaut, von daher kam das Release von Java 10 für viele überraschend. Das liegt einmal an der ungewohnt überpünktlichen Auslieferung, hauptsächlich aber an dem neuen, halbjährlichen Release-Zyklus. Natürlich darf man sich aufgrund der kurzen Zeitspanne keine großen Änderungen erwarten; doch es gibt einige interessante Features.

Erst vor einem halben Jahr hatte Java im September 2017 den letzten großen Versionssprung erlebt. Die damals eingeführten Modularisierungsmöglichkeiten rund um Jigsaw haben die meisten Entwickler bisher kaum verarbeitet und es gibt nur wenige kommerzielle Projekte, die bereits auf Java 9 migriert sind. Selbst viele Hersteller von Bibliotheken und Werkzeugen sind noch mit den Anpassungen auf das neue Modulsystem beschäftigt. So hat das JUnit-Team erst kürzlich die Version 5.1 veröffentlicht, die Tests in Jigsaw-Modulen unterstützt.

Dass viele Entwicklungsabteilungen die Migration nach Java 9 bisher gescheut haben, liegt nicht nur an der hohen Komplexität des Modulsystems; vielmehr erhalten durch das neue, halbjährliche Release-Modell nicht mehr alle Java-Versionen den gleichen langfristigen Support. Nach Java 8 wird Oracle voraussichtlich erst wieder für Java 11, das für September 2018 geplant ist, einen Long-Term-Support anbieten.

Für Entwickler haben die kürzeren Release-Zyklen den Vorteil, dass sie viel früher kleinere Änderungen ausprobieren können, auf die sie sonst wieder mehrere Jahre hätten warten müssen, weil sie im Schatten von großen Features (wie Generics, Lambdas, Streams,

```
// Type inference bei Parametern von Lambda Ausdrücken (Java 8)
Function<String, String> helloFunction = s -> "Hello " + s;

// Inference von Generics (Diamond Operator, seit Java 7)
List<String> strings = new ArrayList<>();
strings.add(helloFunction.apply("World"));

// Inference von Generics (Diamond Operator) bei anonymen inneren Klassen (Java 9 -> JEP 213)
Consumer<String> printer = new Consumer<>() {
  @Override
  public void accept(String string) {
     System.out.println(string);
  }
};
strings.forEach(printer::accept);
```

Listing 1

Jigsaw etc.) standen und sich dadurch die Auslieferung ungewollt immer wieder verzögerte.

Die neuen Funktionen

Kommen wir zurück zu Java 10 – dank des kurzen Zeitrahmens muss man sich nicht durch allzu lange Release Notes quälen. Die Liste auf der OpenJDK-Projektseite *[1]* ist überschaubar:

- 286: Local-Variable Type Inference
- 296: Consolidate the JDK Forest into a Single Repository
- 304: Garbage-Collector Interface
- 307: Parallel Full GC for G1
- 310: Application Class-Data Sharing
- 312: Thread-Local Handshakes
- 313: Remove the Native-Header Generation Tool (javah)
- 314: Additional Unicode Language-Tag Extensions
- 316: Heap Allocation on Alternative Memory Devices
- 317: Experimental Java-Based JIT Compiler
- 319: Root Certificates
- 322: Time-Based Release Versioning

Die Nummern vor den neuen Funktionen sind die Java Enhancement Proposals (JEP); aus Entwicklersicht ist insbesondere der erste am interessantesten, die Local-Variable Type Inference. Typ-Inferenz ist das Schlussfolgern auf Datentypen aus den restlichen Angaben des Codes und den Typisierungsregeln heraus. Dadurch kann man im Code gewisse Typ-Angaben weglassen. Die Lesbarkeit erhöht sich, weil der Quellcode nicht unnötig aufgebläht wird. Man kennt das in Java bereits bei Lambda-Parametern und beim Diamond-Operator für generische Daten-Container beziehungsweise seit Java 9 auch für anonyme innere Klassen *(siehe Listing 1)*.

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

Listing 2

Mit dem Schlüsselwort „var" lassen sich jetzt sehr prägnant lokale Variablen definieren, deren Datentyp sich direkt aus der Zuweisung des Werts ergibt. Während beim „Diamond"-Operator die Typ-Information aus der linken Seite der Zuweisung geschlussfolgert wird, ist es bei Local-Variable Type Inference genau andersherum *(siehe Listing 2)*.

Einmal deklarierte „var"-Variablen sind auf den zugewiesenen Datentyp festgelegt. Möchte man nachträglich Werte anderer Typen zuweisen, erhält man Compiler-Fehler aufgrund der fehlschlagenden Typ-Konvertierung *(siehe Listing 3)*.

Bei der Deklaration muss also immer ein Wert zugewiesen werden, sonst kommt es ebenfalls zu Compiler-Fehlern, auch wenn wie im folgenden Beispiel kurz darauf garantiert ein Wert zugewiesen werden würde *(siehe Listing 4)*.

Mit „var" deklarierte Variablen sind, wie der Name vermutet lässt, nicht automatisch unveränderbar. Man kann sie jedoch mit dem Schlüsselwort „final" kombinieren. Außerdem sind sie „effectively final" (wenn es nur eine Zuweisung gibt) und können somit auch aus inneren Klassen und Lambda-Ausdrücken verwendet werden, ohne sie explizit als final zu deklarieren *(siehe Listing 5)*.

```
// Type wird bei Deklaration und Erstzuweisung vom Compiler festgelegt
var zahl = 5;
zahl = 7L; // incompatible types: possible lossy conversion from long to int

var objekt = BigDecimal.ONE;
objekt = BigInteger.TEN; // incompatible types: BigInteger cannot be converted to BigDecimal
```

Listing 3

Die Typ-Inferenz funktioniert auch mit generischen Typen und innerhalb der „foreach"-Schleife. Die Kombination mit dem „Diamond"-Operator führt allerdings aufgrund der fehlenden Typ-Information zu einem Container von Objekt-Referenzen und somit zu weniger Typsicherheit *(siehe Listing 6)*.

Die Typ-Inferenz nutzt im Übrigen den konkreten Typ, bei anonymen inneren Klassen dürfen also zwei vom selben Interface abgeleitete Instanzen nicht derselben „var"-Variablen zugewiesen werden *(siehe Listing 7)*. Das bedeutet allerdings auch, dass bei lokalen Typen (anonymen inneren Klassen-Implementierungen) neu implementierte Methoden ohne Compiler-Fehler aufgerufen werden können („reverseMe()", *siehe Listing 8)*.

Für Java 11 wird noch an einer Erweiterung der Local-Variable Type Inference gearbeitet, die in Lambda-Ausdrücken funktioniert. Das ist für die Kombination von Typ-Inferenz mit Typ-Annotationen notwendig *(siehe Listing 9)*.

Die restlichen Punkte der Release Notes *[2]* betreffen eher den infrastrukturellen Bereich und den Betrieb von Java-Anwendungen. So kann der G1, der seit Java 9 der Standard GC ist, die Full Garbage Collection parallelisieren und dadurch die Stop-the-world-Zyklen verkürzen. Der Memory Footprint wird durch das Teilen von geladenen Klassen zwischen mehreren Java-Anwendungen verringert. Darüber hinaus gibt es einen neuen, noch experimentellen JIT- (Just-in-time) Compiler (Graal) und Verbesserungen an der JVM für die Zusammenarbeit mit Docker-Containern. Zudem wird der Trust Store des OpenJDK jetzt mit einer gewissen Menge an Root-Zertifikaten ausgeliefert, was sonst nur den Oracle-Java-SE-Versionen vorbehalten war.

An der Klassen-Bibliothek (JDK) gibt es auch kleine Änderungen. Das umfasst beispielsweise eine überladene Version von „orElseThrow()" in der Klasse „Optional" und diverse Fabrik-Methoden zur Erzeugung von „unmodifiable" Collections und Stream-Kollektoren. Weitere Änderungen kann man den Release Notes *[2]* entnehmen oder über das Tool „JDK-API-Diff" *[3]* herausfinden.

Fazit

Schlussendlich lässt sich sagen, dass Java 10 ein unaufgeregtes Release ist. Beeindruckend ist jedoch, dass es wie angekündigt just in time geliefert wurde. Nun muss sich in den nächsten Jahren

```
// Deklaration von "var" nur bei direkter Initialisierung der Variable
var flag = true;
var number; // Compiler-Fehler
if (flag) {
  number = 5;
} else {
  number = 7;
}
```

Listing 4

```
// Inference bei Neuzuweisung eines Wertes (var impliziert nicht "final")
var zahl = 5;
zahl = 7;

// Inference auch mit "final" nutzbar, ansonsten gilt seit Java 8 ohnehin die "effectively final" Semantik
final var zahl = 5;
```

Listing 5

```
// Inference generischer Typen (List<String>)
var strings = Arrays.asList("World", "Java 10");

// Inference in Schleifen
for (var string : strings) {
  System.out.println("Hello " + string);
}

// Kombination mit Diamond Operator führt zu Inferenz von List<Object>
var strings = new ArrayList<>();
strings.add("Hello World");
for (var string : strings) {
  System.out.println(string.replace("World", "Java 10")); // cannot find symbol 'replace'
}

var strings2 = new ArrayList<String>();
strings2.add("Hello World");
for (var string : strings2) {
  System.out.println(string.replace("World", "Java 10"));
}
```

Listing 6

```
// Inference nutzt konkrete Typisierung
var runnable = new Runnable() {
  @Override
  public void run() {
  }
};
// incompatible types: <anonymous Runnable> cannot be converted to <anonymous Runnable>
runnable = new Runnable() {
  @Override
  public void run() {
  }
};
```

Listing 7

```
// Inference von lokalen Typen
var myReversibleStringList = new ArrayList<String>() {
  List<String> reverseMe() {
     var reversed = new ArrayList<String>(this);
     Collections.reverse(reversed);
     return reversed;
  }
};
myReversibleStringList.add("World");
myReversibleStringList.add("Hello");

System.out.println(myReversibleStringList.reverseMe());
```

Listing 8

```
// Ausblick Java 11 (JEP 323)
// Inference von Lambda Parametern
Consumer<String> printer = (var s) -> System.out.println(s); // statt s -> System.out.println(s);

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

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

Listing 9

zeigen, wie die Java-Welt dieses neue Auslieferungsmodell annimmt. Als Entwickler dürfen wir jetzt häufiger mit kleinen, nützlichen Sprach-Features spielen. Das sind auf jeden Fall gute Aussichten. Dann viel Spaß beim Runterladen *[4]* und Ausprobieren. Nicht vergessen, im September 2018 steht mit Java 11 bereits das nächste Release auf dem Plan.

Referenzen

[1] OpenJDK-Projektseite: *http://openjdk.java.net/projects/jdk/10/*

[2] Release Notes: *http://www.oracle.com/technetwork/java/javase/10-relnote-issues-4108729.html*

[3] JDK-API-Diff Tool: *https://gunnarmorling.github.io/jdk-api-diff/jdk9-jdk10-api-diff.html*

[4] Download: *http://www.oracle.com/technetwork/java/javase/downloads/index.html*

Falk Sippach

falk.sippach@oio.de

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

Impressionen von der JavaLand 2018

Die Java-Konferenz im Phansialand Brühl verzeichnet mit mehr als 1.900 Teilnehmern ein neues Rekordhoch. Auch die Stimmung war an beiden Tagen bestens.

Play it again, Arun

Uwe Sauerbrei, Kids4IT

Man könnte durchaus „Play it again, Arun" sagen, auch wenn die vierte Auflage der JavaLand4Kids weder in Casablanca noch mit musikalischer Untermalung stattgefunden hat. Obwohl – so ganz richtig ist es auch wieder nicht.

Zum ersten Mal war nicht nur eine Grundschulklasse der Hans-Christian-Andersen Schule aus Sankt Augustin zu Gast, sondern auch Schüler des Max-Ernst-Gymnasiums aus Brühl. Mit insgesamt vierzig Teilnehmern, zwanzig Mentoren und einigen Lehrern mussten die Workshops in zwei Hotels geplant werden.

Wie in den vergangenen Jahren gab es im Hotel Matamba die Kurse für die Grundschüler im Alter von neun bis zehn Jahren. Thematisch drehte es sich um die Programmierung mit Scratch und musikalische Variationen mit Sonic Pi. Letzteres ist ein Open-Source-Musikprogramm, das für den Raspberry Pi entwickelt wurde und die musikalischen Instinkte der Kinder weckt. Innerhalb von neunzig Minuten durfte jede Gruppe basteln, programmieren und mit Kopfhörern ihren Werken lauschen.

Die Kinder arbeiteten sehr konzentriert und kreativ. Diese Beobachtung kommt sehr häufig, wenn es um die Workshops geht. Für den Autor ein Beweis dafür, dass man jedes Thema so interessant aufbereiten kann, dass es die Kinder in seinen Bann zieht. Natürlich haben die erfahrenen Mentoren einen nicht unerheblichen Anteil an diesem Erfolg.

Zur gleichen Zeit im Hotel Ling Bao: Die Informatik-Klasse des Gymnasiums wollte gerne in die große weite Welt der Programmierung hineinschnuppern. Hier kamen die bewährten Kurse für die Kinder nicht infrage. Mit Iryna Feuerstein und Oliver Milke standen jedoch zwei Mentoren bereit, die über reichlich Erfahrungen mit jugendlichen Hackern verfügten. So stand das Thema „Neo4J“ im Raum. Eine Graphen-Datenbank kennt man nicht aus dem Schulalltag und so waren alle gespannt, ob und wie es funktionieren würde. Natürlich bedurfte es einer etwas umfangreicheren Einführung, aber als es dann an das Hands-on ging, waren alle sofort dabei und tauchten ab in die spannende neue Welt der Visualisierung von Daten.

Last but not least freute sich der andere Teil der Klasse auf die gehobene Kunst des Minecraft Modding mit Arun Gupta. Er organisiert seit vielen Jahren die sehr aktive Devoxx4Kids-Gruppe in der Bay-Area. Mit mehr als hundert bereits absolvierten Veranstaltungen verfügt er sowohl über die Fähigkeit zur Präsentation der verschiedensten Technologien als auch über die Gelassenheit im Umgang mit unterschiedlichen Altersgruppen.

Im Vorfeld der Planungen gab es ein paar zögerliche Stimmen, die einen rein englisch geführten Workshop als schwierig erachteten. Für die Schüler war es im ersten Moment auch ungewohnt, jedoch schon nach kurzer Zeit begannen Diskussionen, wurden Fragen gestellt und beantwortet und die Kreativität nahm ihren Lauf. Vier Mentoren standen auch hier den Schülern zur Seite, sie wurden allerdings immer weniger technisch und noch weniger sprachlich benötigt. Aber es war natürlich auch eine gute Gelegenheit, hier selbst etwas zu lernen.

Noch am selben Tag kam sowohl von den Lehrern als auch von den Schülern die Rückmeldung, dass es sehr viel Spaß gemacht habe und sie gerne wiederkommen würden – ganz klar die Aufforderung, auch für das Jahr 2019 ein spannendes Programm auf die Beine zu stellen. An Ideen und Mentoren wird es keinen Mangel geben, das ist ganz sicher.

Uwe Sauerbrei
info@kids4it.de

Uwe Sauerbrei arbeitet als selbständiger IT-Berater in Hamburg. Er organisiert die dortige Java User Group und hat mit Kids4IT eine gemeinnützige User Group gegründet. Kids4IT veranstaltet monatliche Workshops in der Hansestadt, zeichnet für die Devoxx4Kids-HH verantwortlich und unterstützt die JavaLand4Kids.

Automatisiertes Access Management mit Keycloak

Jannik Hüls, codecentric AG

Viele Anwendungen nutzen Drittanwendungen für die Authentifizierung und Autorisierung der Endnutzer. Die Integration dieser sicherheitskritischen Dienste ist häufig nicht automatisiert, damit unpraktisch für die Entwickler und zudem fehleranfällig. Dieser Artikel stellt den Standard der „OpenID Connect Dynamic Client Registration" vor und auf dessen Basis ein automatisiertes Access Management, beispielhaft implementiert mit dem Open-Source-Tool Keycloak und dem CI-/CD-Server Jenkins.

Die Automatisierung manueller Schritte hat in der Anwendungsentwicklung mittlerweile breite Akzeptanz gefunden. CI-/CD-Server unterstützen dabei enorm. Entwickler veröffentlichen Code-Änderungen in Versionskontroll-Systemen, CI-/CD-Server übernehmen danach viele aufwändige Tasks. Vom Bauen und Testen bis hin zur Integration und Veröffentlichung der Anwendung sind im optimalen Fall keine manuellen Interaktionen mehr notwendig. Dabei werden die Tasks aus unterschiedlichen Gründen automatisiert. Zunächst

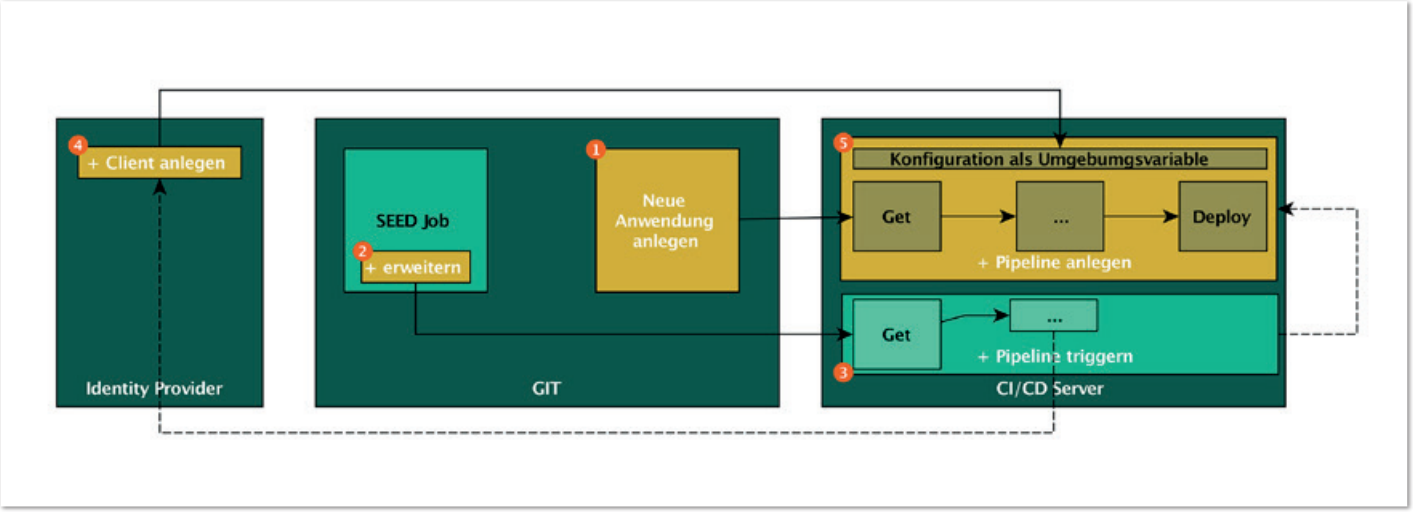


Abbildung 1: Automatische Integration einer neuen Anwendung in CI-/CD-Server

einmal soll dadurch die Geschwindigkeit erhöht werden. Es wird versucht, die Zeit zu minimieren, in der der Entwickler auf Rückmeldung wartet, ob seine Änderung erfolgreich integriert und somit veröffentlicht werden kann. Zudem sind Tasks reproduzierbar gespeichert. Automatisierungs-Artefakte enthalten möglichst alles an Expertenwissen; Tasks können somit wiederholt und ohne Abhängigkeiten zu bestimmten Verantwortlichen ausgeführt werden. Zudem sind manuelle Tasks fehleranfällig und ein hoher Automatisierungsgrad sorgt für sehr robuste Systeme.

Im CI-/CD-Server werden die einzelnen Schritte in einer sogenannten „Pipeline" ausgeführt. Diese enthält meist Zusatz-Informationen zur Ziel-Plattform der Veröffentlichung oder zu Abhängigkeiten von Drittanwendungen. Solche Abhängigkeiten treten beispielsweise auf, wenn die Authentifizierung der zu entwickelnden Anwendung gegen vorhandene Dienste erfolgen soll. Beim Einrichten der Pipeline müssen so vorab sicherheitskritische, authentifizierungsrelevante Informationen bekannt sein. Beim Anlegen eines neuen Projekts werden dabei zwar häufig die einzelnen Schritte der Pipeline automatisiert, die Erstellung der Pipeline ist allerdings meist noch ein manueller Task. Vor allem die Anbindung an Dienste wie die Authentifizierung erschwert die automatisierte Erstellung der Pipeline. Das ist ein Problem, da genau diese extrem sicherheitskritischen Tasks aus den oben genannten Gründen nicht immer wieder von den Entwicklern manuell ausgeführt werden sollten. Nachfolgend wird betrachtet, wie die Nutzung von Authentifizierungsdiensten automatisiert und in die Pipeline integriert werden kann. *Abbildung 1* zeigt dabei das gewünschte Ziel-Szenario.

Eine neue Anwendung wird im Versionskontroll-System angelegt (1). Die Erstellung der CI-/CD-Pipeline erfolgt dabei nicht manuell, sondern durch einen sogenannten „Seed-Job". Dies ist eine Pipeline, die ausschließlich für die Erstellung anderer Pipelines zuständig ist. Dieser Seed-Job wird erweitert (2), die entsprechende Pipeline des Jobs gestartet (3) und auf diese Weise vollautomatisch eine neue Pipeline erzeugt (5). Abhängigkeiten zu sicherheitskritischen Diensten wie dem Identity-Provider werden im Seed-Job programmatisch gelöst (4) und alle notwendigen Informationen stehen automatisch in der neuen Pipeline zur Verfügung. Essenziell bei diesem Vorgehen ist der Standard „OpenID Connect Dynamic Client Registration".

OpenID Connect Dynamic Client Registration

Aus Gründen der Sicherheit, der einfachen Anbindung sowie der gewünschten Unabhängigkeit soll in der neu zu entwickelnden Web-Anwendung für die notwendige Authentifizierung das Standardprotokoll „OpenID Connect" eingesetzt werden. Es handelt sich um eine Identitätsschicht, aufbauend auf dem Protokoll „OAuth 2.0". Mit OpenID Connect lässt sich sowohl die Frage der Authentifizierung („Wer ist der Benutzer?") als auch die Frage der Autorisierung („Was ist dem Benutzer erlaubt zu tun?") klären.

Die Web-Anwendung ist eine „Relying Party" im „OpenID Connect"-Terminus. Jede Authentifizierung erfolgt im Sinne des Endbenutzers, des „Resource Owner". Dies bedeutet, die Relying Party verweist zur Authentifizierung im Standardfall an den Identity-Provider. Der Resource Owner meldet sich dort an und wird nach erfolgreicher Anmeldung erneut zur „Relying Party" umgeleitet.

Benutzerspezifische Informationen werden im Anschluss als JSON Web Token (JWT) enkodiert an die Relying Party, und somit die Web-Anwendung, übermittelt. Diese verifiziert damit die Identität des Resource Owner, kann mit den erhaltenen Token Zugriffsrechte prüfen und auf Basis dieser Informationen Inhalte anzeigen.

Jede Relying Party kommuniziert also zur Authentifizierung mit dem Identity-Provider. Diese Kommunikations-Schnittstelle wird im Iden-

```
POST /connect/register HTTP/1.1
Content-Type: application/json
Accept: application/json
Host: server.example.com

{
  "redirect_uris":
    [
      "https://client.example.org/callback",
      "https://client.example.org/test"
    ],
  "client_name": "My Example",
  "application_type": "web"

}
```

Abbildung 2: HTTP-Post-Nachricht, um einen neuen Client zu registrieren

tity-Provider „Client" genannt. Für jede Relying Party muss, wenn sie einen Identity-Provider verwenden möchte, demnach in genau diesem ein entsprechender Client registriert und konfiguriert werden.

Die Registrierung des Clients kann manuell erfolgen, beispielsweise über eine webbasierte Administrationsoberfläche des Identity-Providers. Dazu ist zur Anlage des Clients der Zugriff auf den Identity-Provider notwendig. Die Konfiguration der Relying Party erfolgt dann mit dem Identity-Provider-URL, der „Client ID" sowie einem „Client Secret".

Diese manuellen Schritte widersprechen der gewünschten Automatisierung. Beim Anlegen eines neuen Entwicklungsprojekts sind diese Konfigurationen wiederkehrend durchzuführen, was fehleranfällig und zeitraubend ist. Zudem ist Expertenwissen und -unterstützung notwendig. Die Authentifizierung kann erst dann implementiert werden, wenn ein Client im Identity-Provider angelegt wurde oder wenn der Entwickler Zugriff auf den Identity-Provider bekommt, um dies im Self-Service zu erledigen. Beides sind häufig keine gewünschten Szenarien, da solche Abhängigkeiten Zeit rauben und umfangreiche Zugriffe auf extrem sicherheitsrelevante Systeme meist unerwünscht sind.

„OpenID Dynamic Client Registration" ist ein Standard, der es einer Relying Party erlaubt, sich dynamisch beim Identity-Provider zu registrieren. Die Relying Party nutzt dabei ein RESTful-HTTP-API, um die notwendigen Informationen eines Clients zu übermitteln und alle Konfigurationsdaten für die lokale Anwendung zu erhalten – ideal also für die angestrebte Automatisierung und die Integration in die Pipeline. Es wird zwischen einem Registrierungs- und einem Konfigurations-Endpunkt unterschieden. Neue Clients können am Registrierungs-Endpunkt dynamisch angelegt und die Einstellungen existierender Clients am Konfigurations-Endpunkt im Self-Service geändert werden. Um einen neuen Client zu registrieren, wird eine HTTP-Post-Nachricht an den Identity-Provider gesendet. *Abbildung 2* zeigt beispielhaft eine solche Nachricht.

Die mitgesendeten Client-Metadaten konfigurieren den Client im Identity-Provider. Lediglich der Parameter „redirect_uris", also die erlaubten Umleitungen nach erfolgreicher Authentifizierung, ist dabei Pflicht. Der Identity-Provider antwortet bei erfolgreicher Registrierung dann mit einer Client-ID, einem Client Secret sowie allen erstellten Client-Metadaten *(siehe Abbildung 3)*.

```
HTTP/1.1 201 Created
Content-Type: application/json
Cache-Control: no-store
Pragma: no-cache
Host: server.example.com

{
  "client_id": "s6BhdRkqt3",
  "client_secret": "ZJYCqe3GGRvdrudKyZS0XhGv_Z45DuKhCUk0gk",
  "registration_access_tkoen": "tokenstring",

  ...

  "redirect_uris":
    [
      "https://client.example.org/callback",
      "https://client.example.org/test"
    ],
  "client_name":"My Example"
  "application_type":"web",

  ...

}
```

Abbildung 3: HTTP-Response nach erfolgreicher Registrierung eines Clients

Client-ID und Secret werden in der Applikation verwendet, um die Authentifizierung mit dem Identity-Provider durchführen zu können. Das „Registration Access Token" ist zudem von Interesse, da es für die Authentifizierung bei Anfragen an den Konfigurationsendpunkt verwendet werden muss. Auffällig ist, dass keine Authentifizierung bei der Erstellung eines Clients notwendig ist. Die Spezifikation sieht explizit vor, dass auch Anfragen ohne Authentifizierung möglich sein sollen. Dem Identity-Provider obliegt es dann, diese durch gesonderte Mechanismen wie Beschränkungen in der maximalen Anzahl von Anfragen und vertraute Domänen abzusichern.

Mit der Dynamic Client Registration bietet OpenID Connect einen Standard, der für die Nutzung in der Automatisierungspipeline wie gemacht ist. Am Beispiel des Identity-Providers Keycloak und des CI-/CD-Servers Jenkins wird nun gezeigt, wie eine neue Pipeline erstellt und die entsprechende Anwendung automatisch in Keycloak registriert wird.

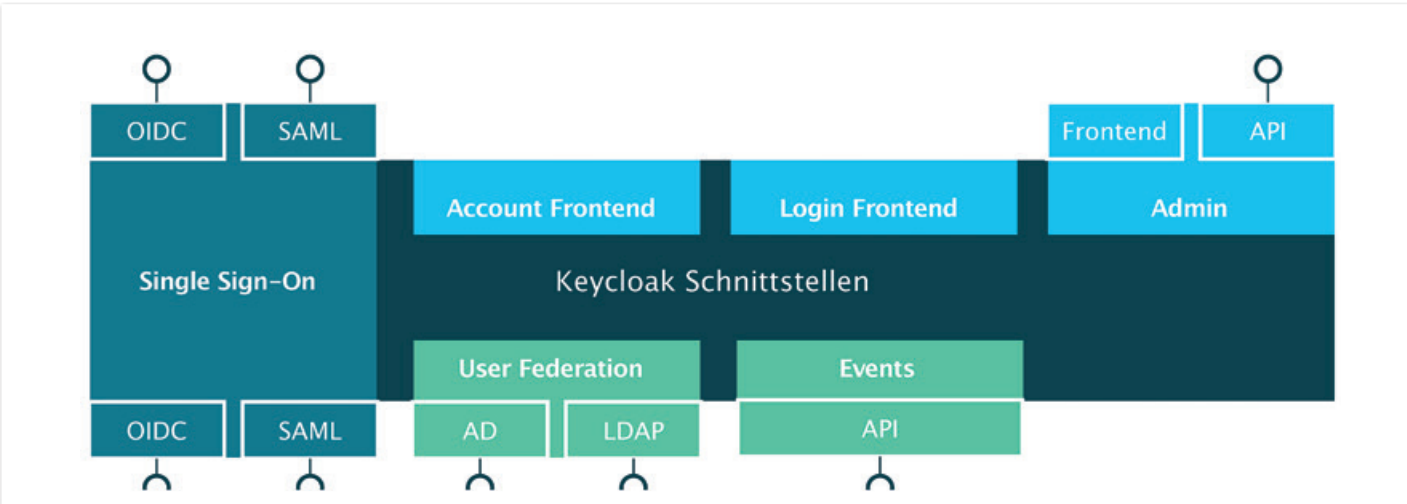


Abbildung 4: Keycloak-Schnittstellen für Anwendung und Administration

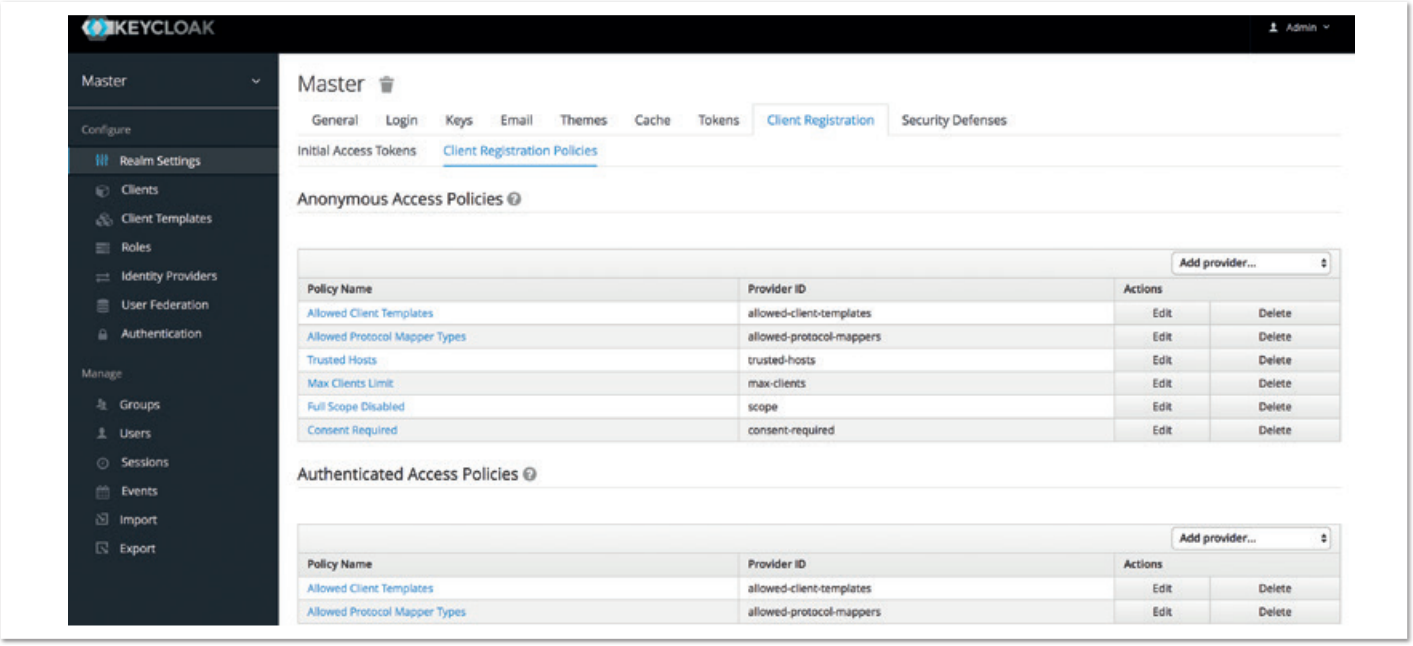


Abbildung 5: Keycloak-Client-Registrierungs-Policies

Dynamische Client-Registrierung in der Praxis

Keycloak *(siehe „http://www.keycloak.org")* ist ein Open-Source-Identity-Provider aus dem Hause Red Hat. Ehrlicherweise nur einer von vielen Identity-Providern, mit dem der Autor bereits in vielen Projekten gute Erfahrungen machen konnte. Keycloak bedient sich sehr ausgiebig am Red-Hat-Stack. Standardmäßig wird WildFly als Application-Server eingesetzt und dessen Clustering- und Hochverfügbarkeits-Funktionen genutzt. Viele Adapter, unter anderem für Tomcat, Spring Boot oder Spring Security, machen die Integration für Entwickler extrem komfortabel. Keycloak setzt auf Standards für

```
1 : def kcRealm = <realmstring>
2 : def kcUrl = <identity provider url>
3 : def kcDcrUrl = kcUrl + "/realms/" + kcRealm + "/clients-registrations/openid-connect"
4 : def registerClient(String urlString, String queryString) {
5 :  def url = new URL(urlString)
6 :  def connection = url.openConnection()
7 :  connection.setRequestMethod("POST")
8 :  connection.doOutput = true
9 :  connection.setRequestProperty("Content-Type", "application/json")
10 :  def writer = new OutputStreamWriter(connection.outputStream)
11 :  writer.write(queryString)
12 :  writer.flush()
13 :  writer.close()
14 :  connection.connect()
15 :  new groovy.json.JsonSlurper().parseText(connection.content.text)
16 : }
17 : job('Webanwendung mit Authentifizierung') {
18 :  def metadata = '{"redirect_uris":["https://client.example.org/callback","https://client.example.org/
callback2"],"client_name": "My Example"}'
19 :  def response = registerClient(kcDcrUrl, metadata)
20 :  environmentVariables {
21 :    env('CLIENT_ID', response.client_id)
22 :    env('REALM', kcRealm)
23 :    env('IDENTITY_PROVIDER', kcUrl)
24 :    env('CLIENT_SECRET', response.client_secret)
25 :    env('REGISTRATION_ACCESS_TOKEN', response.registration_access_token)
26 :   }
27 :  scm {
28 :    git ('https://github.com/jannikhuels/kc-jenkins-dcr')
29 :  }
30 :  triggers {
31 :   scm('*/15 * * * *')
32 :  }
33 :  steps {
34 :    maven("build")
35 :   …
36 :    maven("package")
37 : }
38 : }
```

Listing 1

die Anbindung von Identity-Providern sowie User Federation und ist dabei sehr breit aufgestellt. Neben OpenID Connect und SAML werden standardmäßig auch LDAP und Active Directory unterstützt. Zudem ist Keycloak auf Erweiterbarkeit ausgelegt. Sogenannte „Service Provider Interfaces" (SPIs) sind für alle wesentlichen Funktionalitäten von Keycloak implementiert und schaffen der Erweiterbarkeit extrem großzügige und komfortable Möglichkeiten.

Abbildung 4 zeigt die Schnittstellen von Keycloak. Nicht nur die Automatisierung der Client-Registrierung lässt sich über die standardisierten OIDC-Schnittstellen durchführen. Das RESTful-Admin-API ermöglicht noch viele weitere Möglichkeiten. Von einer automatischen Konfiguration neuer Keycloak-Instanzen bis hin zu einer Self-Service-User-Registrierung werden damit sehr viele Möglichkeiten geboten. Die automatische Client-Registrierung ist für die Anwendungsentwicklung allerdings der Startpunkt; im Folgenden wird erläutert, wie diese in Keycloak verwendet werden kann.

Keycloak bietet vier verschiedene Provider für die Client-Registrierung: (1) default, (2) install, (3) openid-connect und (4) saml2-entity-descriptor. Die ersten beiden implementieren eine Keycloak-spezifische Client-Beschreibung in JSON. Provider drei und vier implementieren die jeweilige standardkonforme Client-Registrierung. An allen vier Schnittstellen können sowohl ein Bearer- als auch ein Initial-Access-Token für die Authentifizierung verwendet werden. Zudem ist eine Registrierung ohne vorherige Authentifizierung möglich. Sicherheit wird dabei auf Basis von konfigurierbaren Richtlinien hergestellt. Diese sogenannten „Policies" sind initial schon sehr umfangreich und zudem über das entsprechende Serial Peripheral Interface (SPI) ierweiterbar. *Abbildung 5* zeigt das User-Interface für die Konfiguration der dynamischen Client-Registrierung in Keycloak.

Keycloak implementiert die OIDC Dynamic Client Registration konform mit dem Standard. Um die Unabhängigkeit zu bewahren, soll ein Standardprotokoll eingesetzt und daher in der Pipeline diese Schnittstelle verwendet werden. Dabei wird die Policy-basierte Authentifizierung genutzt.

Für unsere Web-Anwendung soll nun eine Pipeline entstehen. Dies muss für Anwendungsentwickler möglichst komfortabel sein und daher weitestgehend automatisiert ablaufen. Als CI-/CD-Server wird Jenkins eingesetzt. Die Erstellung der Pipeline erfolgt in Jenkins über einen Seed-Job. *Listing 1* zeigt das Groovy-Script des entsprechenden Jobs.

In den Zeilen 1 bis 3 werden die Identity-Provider-spezifischen Variablen gesetzt. Dies muss nicht notwendigerweise Keycloak sein, ist es jedoch in diesem Beispiel. Bei abweichenden Identity-Providern würde sich die URL der Zeile 3 ändern. Die Funktion „registerClient" (Zeilen 4 bis 16) setzt einen HTTP-Post-Request zur Registrierung eines neuen Clients ab. Dieser entspricht dem in *Abbildung 2* dargestellten Request. Dieser Seed-Job erstellt genau eine Pipeline. Die anwendungsspezifischen Metadaten (Zeile 18) beschreiben die Konfiguration des Clients. Der Client wird einmalig im Job angelegt (Zeile 19), die Daten der Antwort (analog zu *Abbildung 3)* werden als Umgebungsvariablen der Pipeline gespeichert (Zeilen 20 bis 26). Diese Umgebungsvariablen können dann in den einzelnen Tasks der Pipeline verwendet werden.

Die Tasks der Pipeline sind hier nicht von Relevanz, wurden aber der Vollständigkeit halber hinzugefügt. In der Pipeline wird ein Git-Repository alle 15 Minuten gepollt und auf Änderungen überprüft. Die Web-Anwendung ist ein Java-Projekt, folglich entsprechen die Steps der Pipeline „maven"-Tasks (Zeilen 33 bis 37). Bei Ausführung dieses Jobs wird demnach im Jenkins eine Pipeline erzeugt, in der alle Konfigurationsdaten des Identity-Providers hinterlegt sind. Diese können in der Anwendung dann für die Konfiguration der Authentifizierung verwendet werden.

Fazit

Das Tool Keycloak bietet durch den modularen Aufbau und die vielen Schnittstellen umfangreiche Automatisierungsmöglichkeiten. In diesem Post wurde beispielhaft die initiale Registrierung neuer Anwendungen auf Basis der „OpenID Connect Client Registration" betrachtet. Dies ist der erste Schritt für eine sehr komfortable Erstellung neuer Anwendungs-Pipelines, aber nur einer von wenigen zu einer vollständigen Self-Service-Implementierung. Viele Dinge lassen sich in der Implementierung deutlich optimieren. Beispielsweise sollte eine sichere Verwaltung der Token implementiert werden. Für die Idee dieses Artikels ist dies nicht wichtig, für eine produktionsreife Implementierung allerdings umso mehr. Dieser Beitrag soll dazu anregen, vor allem komplexe und sicherheitsrelevante Integrationen zu automatisieren und die notwendigen manuellen Schritte auf ein Minimum zu reduzieren.

Jannik Hüls
jannik.huels@codecentric.de

Jannik Hüls unterstützt seit dem Jahr 2016 das codecentric Team in Münster. Er beschäftigt sich mit der technischen Umsetzung von IT-Architekturen und der Integration von Anwendungskomponenten, aktuell vor allem mit den Möglichkeiten der Serverless-Architektur. Zudem berät er Kunden im Identity- und Access-Management.

Soft Skills – nur etwas für Dampfplauderer?

Christoph Menzel, Method Park Engineering GmbH

Sind Soft Skills wirklich nur etwas für Dampfplauderer oder sind sie heutzutage für jeden von uns wichtig, auch für den normalen, einfachen Software-Entwickler? Der vorliegende Artikel geht dieser Frage nach und zeigt im Anschluss zu einigen ausgewählten Soft Skills, wie man an ihnen arbeiten kann.

Was sind Soft Skills eigentlich? Wie sehen ihre Gegenstücke, die Hard Skills, aus? Für Hard Skills (Fachkompetenzen) findet sich eine der treffendsten Definitionen bei Wikipedia: „... die Fähigkeit von Personen, berufstypische Aufgaben und Sachverhalte den theoretischen Anforderungen gemäß selbstständig und eigenverantwortlich zu bewältigen ..." *(siehe „https://de.wikipedia.org/wiki/Fachkompetenz").*

Fachkompetenzen sind also Fähigkeiten, die sich von Beruf zu Beruf unterscheiden: Ein Schreinermeister benötigt andere Fachkompetenzen als ein Bauingenieur oder ein Software-Entwickler. Diese theoretischen Fähigkeiten lassen sich zum Beispiel durch das Lesen von Büchern, das Absolvieren einer Ausbildung oder in der Zusammenarbeit mit Kollegen erlernen.

Das selbstständige und eigenverantwortliche Bewältigen von Aufgaben mithilfe der Hard Skills ist ein Bestandteil unserer Ausbildung. In der Software-Entwicklung wird diese Kompetenz beispielsweise bei agilen Vorgehensweisen sehr gestärkt. Welche Fachkompetenzen muss ein Software-Entwickler konkret beherrschen? Dazu einige Beispiele *(siehe Abbildung 1):*

- Programmiersprachen
- Versionsverwaltung
- Frameworks
- Algorithmen und Datenstrukturen
- Testen
- Debugging
- Patterns

Soft Skills

Was aber sind Soft Skills, die oft auch „weiche Fähigkeiten" genannt werden? Aus der Vielzahl der Definitionen erscheint wieder die bei Wikipedia passend: „... combination of interpersonal people skills, social skills, communication skills, character traits, attitudes, ... that *enable* people to ... work well *with others* ... achieve their goals with *complementing* hard skills." *(siehe „https://en.wikipedia.org/wiki/Soft_skills").* Die kursiv gesetzten Keywords geben Aufschluss über die Bedeutung von Soft Skills:

- **Combination**
  Kombination bedeutet, dass Soft Skills selten alleine stehen und nur schwer voneinander abzugrenzen sind. Ganz im Gegenteil, sie gehen ineinander über und die Kombination vieler Fähigkeiten ist besonders wichtig.
- **Enable**
  Soft Skills befähigen uns, gut mit anderen zusammenzuarbeiten. Soft Skills sind also vor allem dann wichtig, wenn es um die Zusammenarbeit mehrerer Menschen geht.
- **Complementing**
  Soft Skills ergänzen und vervollständigen die Hard Skills. Jede Fähigkeit für sich alleine genommen ist nutzlos. Erst ihre Kombination macht Hard und Soft Skills extrem wert- und machtvoll. Nachfolgend einige Beispiele, die Soft Skills noch greifbarer machen:

Warum braucht man in der Software-Entwicklung Soft Skills?

Ein Software-Entwickler muss hauptsächlich eine Programmiersprache beherrschen und performante, sichere Software schreiben können. Wozu braucht er Soft Skills? Die wichtigsten Gründe lassen sich in vier Schlagwörtern zusammenfassen:

- Komplexität
- Zunehmend rascherer Technologiewandel
- Qualitätsansprüche
- Teamwork

Komplexität

Produkte und die dazugehörige Software werden immer komplexer. Ein Beispiel ist die smarte Zahnbürste: Sie stellt fest, wie gut oder schlecht und wie lange man Zähne putzt. Die gesammelten Daten kann man sich in einer iOS oder Android App ansehen, vermutlich auch in der Cloud speichern und vielleicht sogar per Facebook oder Twitter teilen.

Bei so etwas Simplem wie einer Zahnbürste kommen schon sehr viele verschiedene Technologien und Software-Produkte zum Einsatz. Bei einem Auto oder einem Medizinprodukt, wie etwa einem Computertomografen, steigt die Komplexität um ein Vielfaches. Um mit dieser Komplexität besser zurechtzukommen, helfen Soft Skills wie zum Beispiel Disziplinierheit, Gründlichkeit, Lernbereitschaft oder Ordnungssinn.

Technologiewandel

Die Software-Entwicklung ist wohl eine der Branchen, die sich am schnellsten verändert. Dies zeigt beispielsweise die Zahl neuer Java Script Frameworks, die jeden Monat entstehen. Auch neue Technologien, wie etwa 3D-Druck, Virtual und Augmented Reality oder Drohnen, stellen immer wieder andere, neue Herausforderungen an einen Software-Entwickler. Um mit diesem schnellen Wandel mithalten zu können, braucht es Soft Skills wie Lernbereitschaft, Lern-, Anpassungs- oder Begeisterungsfähigkeit.

Qualitätsanspruch

Vor allem in sicherheitskritischen Branchen wie der Automobilindustrie, der Medizintechnik und dem Energie-Sektor werden besonders hohe und einmalige Qualitätsansprüche an Software gestellt. Oftmals geht es dort um Menschenleben oder um sehr viel Geld. Aber auch in anderen, weniger sicherheitssensiblen Branchen sollte man hohe Qualitätsansprüche verfolgen, denn Qualität ist ein „Unique Selling Point". Soft Skills vermitteln das dafür notwendige Bewusstsein; sie unterstützen Qualität durch Gründlichkeit, Ordnungssinn und Organisationsfähigkeit.

Teamwork

Kaum jemand arbeitet heutzutage nicht im Team oder nicht mit anderen Menschen zusammen. In der Software-Entwicklung schreiben vor allem neue Entwicklungspraktiken wie Scrum oder SAFe Teamwork und Kommunikation sehr groß. Unterstützende Soft Skills sind in diesem Fall Selbstorganisation, Eigenverantwortlichkeit und Kommunikationsfähigkeit.

Alleine mit Hard Skills lassen sich die gerade genannten Gründe nicht befriedigen. Genau deshalb werden Soft Skills immer wichtiger und ein entscheidender Faktor für den Erfolg von Teams oder sogar Unternehmen.

Hard Skills werden, wie erwähnt, vor allem durch Bücher und Ausbildung vermittelt. Wie erlernt man Soft Skills? Sie sind weder expliziter Teil einer Berufsausbildung noch eines Studiums. Soft Skills werden oftmals erst im Lauf einer beruflichen Entwicklung aktiv gefördert, etwa wenn man eine höhere Position erreicht. Dann ist es jedoch eigentlich zu spät, denn bereits ein Junior-Software-Entwickler benötigt viele der vorher genannten Soft Skills in seinem Alltag. So sind heutzutage zum Beispiel Kommunikationsfähigkeit, Lernbereitschaft, Gründlichkeit und Organisationsfähigkeit ein täglicher Begleiter jedes guten Software-Entwicklers.

Soft Skills werden jedoch weniger erlernt als vielmehr adaptiert: Man schaut sich Soft Skills von Eltern, Lehrern, Freunden, Kollegen oder Mitmenschen ab. Allerdings können Soft Skills auch trainiert und weiterentwickelt werden. Hierzu einige Tipps und Tricks.

Social Skills

Unter Social Skills versteht man eine Vielzahl an Soft Skills wie zum Beispiel Kommunikation, Wortwahl, aktives Zuhören, Körpersprache, Augenkontakt, Empathie, Gefühlskontrolle oder Selbstbeherrschung. Sie helfen dabei, selbstbewusst aufzutreten, neue Leute kennenzulernen – dadurch vielleicht sogar den Traumjob zu bekommen –, Beziehungen aufzubauen und zu pflegen oder das Arbeitsklima zu verbessern. Um Social Skills zu fördern und sich häufiger aus der eigenen Komfortzone zu bewegen, bieten sich zwei Übungen an: das 60-Seconds-Game und das Compliments Game.

Im 60-Seconds-Game geht es darum, sich jemandem persönlich vorzustellen, wenn man einen neuen Ort betritt: eine Veranstaltung, ein Café oder einen Bus. Es ist egal wem, es ist egal wie – man muss es nur innerhalb von 60 Sekunden tun, denn nach dieser ersten Minute wird die Angst sich vorzustellen zu groß.

Diese Angst ist jedoch unbegründet, denn man hat nichts zu verlieren. Im Gegenteil, man kann nur gewinnen und sich durch diese Übungen für die wirklich wichtigen Momente vorbereiten. Man stärkt das eigene Selbstbewusstsein und lernt, sich in solchen doch eher ungewöhnlichen Situationen wohlzufühlen und souverän aufzutreten.

Das Compliments-Game ist ähnlich: Dreimal täglich sollte man ein aufrichtiges Kompliment vergeben. Es ist vollkommen egal wem, die

Abbildung 1: Beispiele für Soft Skills

Hauptsache ist, das Kompliment ist ehrlich gemeint. Auch hierbei lernt man, in ungewohnten Situationen selbstbewusst aufzutreten. Beide Übungen helfen, aus der Komfortzone herauszukommen. Nur wenn man gelernt hat, diese zu verlassen, dann kann man sich wirklich weiterentwickeln und etwas Neues lernen.

Learn and Teach

Gerade in Branchen, die sich wie die Software-Entwicklung schnell verändern, ist lebenslanges Lernen unabdingbar. Mindestens genauso wichtig ist die Weitergabe von Wissen. Dahinter steht eine wichtige Sammlung von Soft Skills, denn das Hamstern von Wissen ist für niemanden von Vorteil. Es gibt viele verschiedene Möglichkeiten, sowohl sein Wissen weiterzugeben als auch neues Wissen aufzubauen.

Das Etablieren sogenannter „Tech Talks" wäre ein Anfang und eine große Chance für Arbeitnehmer wie Arbeitgeber. Pro Woche stehen ein bis zwei Stunden zur Verfügung, um gegenseitigen Wissen auszutauschen. Dies kann unterschiedlich ausgestaltet werden, von Erfahrungsberichten über Diskussionen bis hin zum Live Coding. In diesem geschützten und bekannten Umfeld kann jeder seine Präsentationsfähigkeiten üben und verbessern, bevor der eventuelle nächste Schritt ansteht, etwa ein Vortrag auf einer Konferenz.

Zwei mittlerweile sehr verbreitete Formen der Wissensweitergabe sind Code Review und Pair Programming. Diese zwei Techniken sind sehr mächtig und wirkungsvoll. Sie sollten in keinem Software-Entwicklungsprojekt fehlen. Beide Techniken beinhalten immer ein Geben und Nehmen. Selbst Senior-Software-Entwickler können dabei häufig etwas von Junior-Entwicklern lernen.

Eine dritte Möglichkeit ist das Etablieren von Mentoren. Jeder Software-Entwickler sollte einen Mentor zur Seite haben, der ihn fordert und fördert. Eric Elliott twitterte dazu: „Want to really learn your craft? Mentor a junior developer. If you can't explain it simply, you don't know it well enough."

Kommunikation

Es gibt viele Formen der Kommunikation, so ist zum Beispiel dieser Artikel eine Form der schriftlichen Kommunikation von Autor zu Leser. Körpersprache, mündliche oder indirekte Kommunikation über Dritte sind weitere Möglichkeiten. Jedoch kann man nicht nicht kommunizieren (Paul Wazlawick). Selbst das Fernbleiben von einem Meeting oder einem Event ist eine Form der Kommunikation. Jeder kommuniziert somit immer und ständig, meistens auch unbewusst.

Schau mir in die Augen

Gerade bei der direkten, zwischenmenschlichen Kommunikation ist wichtig: der Augenkontakt. Er vermittelt zum einen Interesse und Wertschätzung und zum anderen ermutigt er das Gegenüber, das Gespräch weiterzuführen.

Heutzutage schaut man viel zu oft aufs Handy oder in den PC und redet nur nebenbei miteinander, geschweige denn, dass man richtig zuhört. Dies kann sehr leicht als ein Zeichen von Desinteresse aufgefasst werden.

Vielen fällt es zudem schwer, jemandem in die Augen zusehen. Hier kann jedoch Abhilfe geschaffen werden: Man stellt sich ein virtuelles „T" vor, das von den Augenbrauen und der Nase geformt wird. Läuft man dies mit den Augen ab, so schaut man seinem Gegenüber an,

ohne ihm in die Augen sehen zu müssen. Oder man schaut auf eine andere Partie des Gesichts, zum Beispiel die Nase oder den Mund.

Zuhören

Neben dem Augenkontakt ist auch das richtige Zuhören sehr wichtig für eine gute und zielführende Kommunikation. Oftmals nimmt man das Gesagte nicht wirklich auf. Ein aktives Zuhören hilft, mehr aufzufassen und dem Gegenüber ein Feedback zu senden. So kann es hilfreich sein, das Gehörte mental zu rekapitulieren, Fragen zu stellen, zusammenzufassen, zu wiederholen oder zuzustimmen.

Aber auch schon kleine Gesten wie ein Nicken oder ein Lächeln geben Feedback und zeugen von Aufmerksamkeit. Wichtig dabei ist es den Gesprächspartner niemals zu unterbrechen, seine Sätze nicht für ihn zu beenden und nicht die Kontrolle über das Gespräch an sich zu reißen. Das gibt dem Redenden nur ein Gefühl von Unselbstständigkeit. Der Zuhörende muss seinem Gegenüber die Zeit geben, die dieser braucht, um eine These oder ein Argument zu entwickeln.

Ein in der Software-Entwicklung sehr bekanntes Prinzip ist „Keep it simple, stupid", kurz „KISS". Dieses Prinzip lässt sich auch sehr gut bei alltäglichen Kommunikationen verwenden. Vor allem bei schriftlicher Kommunikation ist es extrem hilfreich.

Oftmals zeugt das absichtliche Verwenden komplizierter Wörter und Sätze nicht von Intelligenz, sondern vielmehr von Hochnäsigkeit und Arroganz. Deshalb sollte man immer versuchen, die Kommunikation einfach zu halten, eine unkomplizierte Sprache zu verwenden, nur maximal drei bis fünf verschiedene Punkte anzusprechen und auf direktem Weg mit dem richtigen Empfänger zu kommunizieren. Ist Kommunikation einfach und simpel, dann ist sie auch effizient.

Fokussieren

In der heutigen Zeit gibt es so viele Störfaktoren wie noch nie. Um nur ein paar zu nennen: Facebook, Twitter, Handy, Mail, Chat, Kollegen oder Smart Watches. All dies und noch mehr hindert daran, uns zu fokussieren und konzentriert an einer Sache zu arbeiten. Der Versuch des Multitaskings verschärft das Problem noch. Es kann aufschlussreich sein, Störfaktoren aufzuschreiben und zu sammeln. Um seinen Fokus zu finden und ihn beizubehalten, gibt es einige Tricks.

Eine Methode ist „Pomodoro", relativ einfach mit einem hohen Wirkungsgrad. Zu Beginn formuliert man seine Aufgabe schriftlich, um sie festzuhalten und sich klar zu machen, woran man arbeiten möchte. Danach stellt man sich einen Wecker auf 25 Minuten; es geht auch kürzer, jedoch keinesfalls länger. In diesen 25 Minuten bearbeitet man die vorher formulierte Aufgabe und lässt keinerlei Störungen zu. Man liest keine Mails, öffnet Twitter nicht, führt keine Telefonate und lässt auch keine Ablenkung von Kollegen zu, außer eines davon wäre Teil der Aufgabe.

Oftmals muss man seine Kollegen zu Beginn darüber informieren und ihnen zum Beispiel durch eine Lampe oder ein Schild signalisieren, dass gerade nicht gestört werden darf. Natürlich kann es Ausnahmen geben, jedoch sollte es dann dafür einen sehr triftigen Grund geben.

Sind die 25 Minuten vorbei, wird eine Fünf-Minuten-Pause eingelegt, um neue Kraft für die nächste Runde zu sammeln. Je nach Aufgabengröße kann an der gleichen Aufgabe weitergearbeitet oder eine neue formuliert werden. Nach vier Runden sollte man eine längere Pause machen, etwa 20 bis 30 Minuten.

Manche Störfaktoren lassen sich leichter, andere schwieriger blockieren. Das Aufsetzen von Kopfhörern kann ein Signal für Kollegen sein, dass man sich gerade in einer Pomodoro-Runde befindet, und schafft gleichzeitig einen Raum ohne Störgeräusche. Auch die Benachrichtigungen von Handy, Facebook, Twitter oder Mail-Client sollte man ausschalten. Es genügt normalerweise völlig, diese Medien zwei-, drei- oder viermal am Tag zu lesen und zu beantworten. Hierfür gibt es verschiedene Apps und Plug-ins für alle gängigen Browser und Handy-Plattformen.

Pausen werden leider bei der Fokussierung auf eine Arbeit oder Aufgabe meist sehr unterschätzt. Deswegen sind bei der Pomodoro-Technik die Pausen ein fester Bestandteil, da sie extrem hilfreich und förderlich für die Konzentration sind. Als Faustregel sollte man nach 45 Minuten Arbeit für fünf Minuten Pause, nach 90 Minuten zehn Minuten Pause und nach vier Stunden 30 Minuten Pause machen.

Man kann die Pausen auch gut nutzen, um mit Kollegen einen Kaffee oder Tee trinken zu gehen oder zusammen einen kleinen Spaziergang zu unternehmen. Im Gespräch mit Kollegen kommt man entweder gemeinsam auf eine Lösung oder ein Kollege hatte bereits das gleiche Problem und eine gute Lösung dafür. Pausen sind nicht unproduktiv, ganz im Gegenteil, sie erhöhen die Produktivität und den Fokus enorm.

Fazit

Soft Skills sind keinesfalls soft. Sie sind vielmehr ein wichtiger Bestandteil unseres alltäglichen Lebens, sowohl beruflich als auch privat, und können den Unterschied zwischen Scheitern und Erfolg ausmachen. Sie sind nicht nur etwas für Dampfplauderer, sondern werden auch in der Software-Entwicklung immer wichtiger. Jeder muss an seinen Soft Skills arbeiten, Fortbildungen dazu gibt es mindestens genauso viele wie für die sogenannten „Hard Skills".

Christoph Menzel
Christoph.Menzel@methodpark.de

Christoph Menzel ist bei Method Park als Senior Software Engineer tätig. Seine Themen-Schwerpunkte sind agile Entwicklungsmethoden und die Testautomatisierung. Als agiler Coach und zertifizierter Scrum-Master unterstützt er seine Kunden bei der Einführung von Scrum oder bei der Optimierung von Entwicklungsprozessen. Außerdem hält er Hands-on-Workshops zu Themen wie „Unit Tests", „Clean Code" oder „Continuous Integration & Delivery". Darüber hinaus begleitet Christoph Menzel seine Kunden aktiv bei der Entwicklung von Software und engagiert sich beim Software Engineering Camp Erlangen.

Die Suche nach dem Heiligen Gral der agilen Transformation

Andreas Meurer, PENTASYS AG

Der digitale Fortschritt bringt fundamentale Veränderungen und zahlreiche Komplexitäten mit sich – in Bezug auf Märkte, Technologien sowie den Wandel von Organisationen selbst. Die Notwendigkeit, bei Digitalisierungsstrategien auch den Aspekt der Agilität berücksichtigen zu müssen, ergibt sich aus den immer schneller verlaufenden Lebenszyklen der technologischen Entwicklung. Der Artikel zeigt, wie Unternehmen zu agilen Prozessen und dadurch auch zu besseren Software-Lösungen finden können.

Wenn Unternehmen ihre IT-Abteilung vollständig auf agile Methoden umstellen möchten, steht am Beginn der Reise eine große Unbekannte. Vor allem, wenn es um hundert Prozent Agilität geht – von Anfang bis Ende der Wertschöpfungskette. Bisher sind Insel-Lösungen für einzelne Teams oder Abteilungen der Standard. Die Integration agiler Prozesse in benachbarte Abteilungen des Unternehmens, die nach wie vor auf klassische Vorgehensmodelle setzen, steigert die Komplexität des agilen Wandels.

Die agile Transformation ist also in vollem Gange und beinahe jeder auf der Suche nach ihrem Heiligen Gral. Dies gleicht den zahlreichen Geschichten von einer Reise ins Ungewisse – egal, ob im Mittelalter bei Parzival und den Rittern der Tafelrunde oder in der Neuzeit bei Indiana Jones. Wie der Protagonist muss das Unternehmen in diesem Abenteuer vier Stufen durchschreiten: den „Aufbruch ins Ungewisse", den „Märchenwald", den „dunklen Turm" und schließlich das „Finden und Lösen des Rätsels um den Heiligen Gral" selbst. Wer am Ende erfolgreich sein will, kann von anderen lernen, wie die Reise am einfachsten ist.

Der Ausbruch aus den Routinen

In aller Regel beginnen Abenteuer im Alltäglichen. Ob bei Parzival oder Indiana Jones, zunächst befindet sich der Held in seinem gewohnten Umfeld bei gewohnten Aufgaben. Dann kommt es jedoch vor, dass von weither die Nachricht eines bevorstehenden Umbruchs kommt und dringend Handlung geboten ist. Es gilt, eine Lösung zu finden. Auch im Beispiel eines heute auf dem Markt agierenden Unternehmens stellt sich die Situation ähnlich dar; Schlagworte wie „Agile", „Lean Management" oder „Digitalisierung" dringen von weither. Klar ist: Es muss ein Heilmittel gefunden werden. Was in den Legenden der Heilige Gral, ist für uns in der heutigen Zeit die Frage nach der vollumfassenden agilen Transformation.

Der erste Schritt dorthin ist, zu prüfen, was Agilität überhaupt bedeutet und welche schwerwiegenden Missverständnisse sich mit diesem Begriff mitunter verknüpfen. Wer den Wandel anpacken will, muss sich daher zunächst in den Märchenwald begeben, wo es gilt, Dichtung und Wahrheit auseinanderzuhalten.

Die häufigsten Mythen über Agilität und worin ihr wahrer Kern liegt

Die Verschlagwortung des Begriffs „Agilität" hat es mit sich gebracht, dass auch in Kontexten außerhalb des Projekt-Managements von ihm gesprochen wird. Dadurch sind einige Missverständnisse zur Funktionsweise und zu den zu erwartenden Resultaten agiler Methoden entstanden. Diese müssen – obwohl sie oftmals einen wahren Kern besitzen – ausgeräumt werden, bevor es weitergehen kann. Auch im Märchen muss der Held zunächst die Mythen prüfen, die über den Heiligen Gral kursieren, und schließlich Wahrheit und Dichtung voneinander trennen – erst dann wird die Zielrichtung der weiteren Reise wirklich klar. Nachfolgend die beliebtesten Mythen über Agilität – durch eine realistische Erwartungshaltung ersetzt.

„Agile Programmierung geht schneller": In dieser Pauschalität geäußert, ist diese Aussage falsch, denn der Software-Entwickler in einem agilen Projekt schreibt seine hundert Zeilen Code weder schneller noch langsamer als sein Kollege, der nach dem Wasserfall-Modell arbeitet. Nimmt man jedoch eine übergeordnete Perspektive ein, bringt Agilität natürlich den Vorteil einer niedrigen Zeitspanne für die Marktreifung eines Produkts („time to market"). Da es inkrementell entwickelt wird, stehen Teil-Segmente und -Funktionen der Software früher zur Verfügung. Zeit lässt sich durchaus bei den Korrektur-Schleifen einsparen. Schließlich erfolgt beim agilen Vorgehen das Debugging immer kleinschrittig, was Folgefehler vermeiden kann. Zudem sinkt durch schnelles Feedback das Risiko, dass ein Software-Projekt komplett am Bedarf vorbei entwickelt wird. Ein solcher „Worst Case" kostet sehr viel Zeit – und Nerven.

„Agile Projekte kosten weniger": Analog zum Zeitfaktor gestaltet sich das Argument bei den Kosten. Der Software-Entwickler kostet im agilen Projekt nicht weniger als im klassischen. Ganz im Gegenteil: Personal, das auf Agilität zertifiziert ist, hat oftmals sogar einen höheren Stundensatz. Dies ist wiederum auch deshalb der Fall, weil es sich um eine Investition für spätere Effizienzgewinne handelt: Das engere Zusammenspiel zwischen Software-Design und -Implementierung schafft einen Hebel für tatsächliche Einsparungen.

„Bei Scrum & Co. wird viel geredet, aber wenig gearbeitet": Wer sich neu mit agiler Software-Entwicklung beschäftigt, mag von der Meeting-Kultur überrascht sein. Tatsächlich werden im agilen Modell quantitativ scheinbar mehr Meetings einberufen, als in klassischen Projekten üblich. Über Länge und Produktivität der Meetings sagt dies jedoch noch rein gar nichts aus. Das Daily sollte – zumindest nach Scrum-Lehrbuch – nie länger als fünfzehn Minuten dauern. Ob die Teilnehmer vorbereitet ins Meeting gehen und dort ergebnisorientiert miteinander kommunizieren, hängt in erster Linie von hilfegebender Führung und entsprechender Selbstdisziplin ab. Von beidem braucht es auch in der agilen Welt eine Menge.

„Planung ist für agile Projekte ein Fremdwort": Oftmals entsteht gerade beim fachfremden Publikum der Eindruck, Agilität sei eine Art „geplante Anarchie". In diesem Fall wurde aber einfach nur der Grundsatz „Reagieren auf Veränderung mehr als das Befolgen eines Plans" aus dem Manifest für agile Software-Entwicklung falsch interpretiert. „Mehr als" bedeutet nämlich nicht „statt". Planung ist nach wie vor unabdingbar; besonders, wenn es darum geht, an eine Abteilung oder auch einen Kunden auszuliefern, der mit klassischen Methoden der Projekt-Entwicklung arbeitet und dementsprechend auf die jeweilig angemessene Dokumentation angewiesen ist. Da im klassischen Projekt-Kontext allerdings stark an einmal gemachten Plänen festgehalten wird, man diese jedoch auf Grund von Erkenntnissen, die man zur Projekt-Laufzeit gewinnt, ändern können sollte, sei dies im agilen Kontext eben nur nochmals besonders betont.

„Agilität benötigt keine Dokumentation": Dieser Irrtum beruht ebenfalls auf einer Fehl-Interpretation des Manifests für agile Software-Entwicklung. Dort heißt es nämlich: „Funktionierende Software mehr als umfassende Dokumentation". Sicher ist es sinnvoll, sich mehr auf das Funktionieren einer Software zu fokussieren als auf eine umfängliche Dokumentation ihres Nichtfunktionierens, allerdings entbindet dies nicht aus der Pflicht, trotzdem sauber zu dokumentieren. Besonders die „Definition of Done" ist ein geeignetes Mittel, um die Schnittstelle zwischen Entwicklung und Auslieferung sauber abzubilden und die Qualität hochzuhalten. Bei Übergaben, Team-Änderungen, aber auch unter Wartungs-Aspekten, ist die Dokumentation eine sehr wichtige Ressource, um das Projekt weiterzuführen.

Fazit: Wer diese Missverständnisse beiseitegeräumt hat, ist zumindest schon einmal im Bilde, was die Zielrichtung angeht. Nun ist er in der Lage, die nächste, noch gefährlichere Etappe zu beschreiten: den „dunklen Turm".

Wo es bei der Umsetzung schiefgehen kann

In jeder Sage muss der Held sich einer Umgebung stellen, die voller Gefahren ist: zum Beispiel ein Labyrinth, ein Verlies oder eben auch der dunkle Turm eines Magiers, der mit Fallen gespickt ist. Diesen muss man sich bewusst stellen und gezielt damit umgehen, sonst ist es mit der Heldenreise schnell vorbei, denn im Rahmen der agilen Transformation lauern einige Schwierigkeiten.

Angst vor Veränderung: Dies stellt sich insbesondere dann bei der Belegschaft ein, wenn die Umstellung zu schnell ist, als zu einfach wahrgenommen beziehungsweise dargestellt wird und damit oft im Ergebnis zu wenig Begleitung erfolgt. Natürlich fragen sich besonders die langjährigen Mitarbeiter zu Recht, ob sie ihren Platz auch in der agilen Welt noch behalten können. Zerfällt die Belegschaft allerdings in die zwei Lager – der Euphoriker und der Skeptiker – steigt die Unsicherheit. Die Wahrheit liegt tatsächlich irgendwo dazwischen. Hier kann der kulturelle Faktor im Unternehmen nicht genug betont werden: Es bedarf eines klaren Commitments der Führungspersönlichkeiten, für die Erwartungen und auch Sorgen der Mitarbeiter offen zu sein, um auf Augenhöhe Transparenz und Orientierung anzubieten.

Kein „Ende zu Ende"-Ansatz: Gefährlich wird es auch, wenn versucht wird, die agile Transformation auf den Bereich der Entwicklung zu beschränken – ohne sie im gesamten Kontext der Wertschöpfungskette zu verstehen. Dies kann dazu führen, dass die Schnittstellen zu den angrenzenden Bereichen oder auch Zulieferern nicht passen. Dadurch verspielen Unternehmen ihre durch Agilität gewonnenen Effizienzgewinne wieder. Eine „Ende zu Ende"-Transformation ist dementsprechend Pflicht.

Projekt vs. Linie: Da die meisten Unternehmen anhand von siloartigen Abteilungen organisiert sind, werden Projekte in der Regel

zwischen diesen und einer eigens gegründeten Projektgruppe koordiniert (im Englischen als „Line" bezeichnet). Widersprüchliche Weisungsbefugnisse und unpassende Ressourcen-Zuordnung können die Koordination an dieser Schnittstelle erheblich erschweren. Hier ist für eine saubere Einordnung zu sorgen.

Tagesgeschäft: Auch wenn die Transformation zweifelsohne wichtig für die Unternehmensentwicklung ist – die tägliche Arbeit fällt nach wie vor an. Dies gilt es zu akzeptieren und diszipliniert anzupacken. Dazu müssen die Mitarbeiter Wertschätzung und Unterstützung bei der Bewältigung dieser Doppelbelastung erfahren, denn letztendlich erwartet der Kunde außerhalb der Werktore gleichbleibende Qualität und Sorgfalt, egal was sich sonst so momentan im Inneren des Unternehmens tut.

Fazit: Hat der Held unserer Geschichte nun diese Gefahren gemeistert, kann er aus dem dunklen Turm entkommen und ist so gut wie am Ziel: dem Heiligen Gral.

Wie die agile Transformation gelingt

Natürlich ist der Heilige Gral nicht das einfache und schnelle Heilmittel für komplexe Herausforderungen, die der Held zu Beginn noch erwartet hat. Wie so oft in Erzählungen (und auch im wahren Leben) besteht der eigentliche Schatz in den Erkenntnissen, die er auf seinen Reisen gesammelt hat. Wer die Irrtümer über Agilität durch realistische Erwartungshaltungen ersetzt, die schlimmsten Fallstricke vermieden hat und die Erkenntnis akzeptiert, dass agile Methoden die üblichen Projekt-Probleme nicht automatisch lösen, hat gute Voraussetzungen, die agile Transformation erfolgreich zu Ende zu bringen. Die Umstellung auf hundert Prozent Agilität verläuft entlang der folgenden sechs Bereiche.

Agile Methoden: Methoden sind zwar sehr wichtig, doch noch lange kein Selbstläufer. Zunächst bedarf es Disziplin bei der kontinuierlichen Umsetzung, was einen kulturellen Wandel erforderlich macht. Diesen kann die Tatsache erschweren, dass agile Methoden Fehler schneller sichtbar machen. Schließlich sollte die Methode auch angemessen Anwendung finden; das bedeutet, nicht blind am Lehrbuch zu kleben, sondern Regeln je nach Praktikabilität zu betonen oder auch zu modifizieren.

Führung und Skalierung: Die agile Transformation ändert das Rollenbild von Führungskräften erheblich, nämlich vom Weisungsbefugten zum Enabler. Dies beschreibt das allgegenwärtige Schlagwort „Management 3.0"; die Führungskraft soll das Richtige für das Team tun, jeden in die Weiterentwicklung der Organisation einbeziehen und Mitarbeiter zu einer proaktiven Rolle motivieren. Was die Skalierung der agilen Arbeitsweise auf viele Teams und viele Produkte betrifft, stehen mittlerweile zahlreiche Methoden bereit, die jeweils an einen bestimmten Organisationstyp angepasst sind (wie Scrum of Scrums, NEXUS, SAFe, LeSS oder Spotify).

Continuous Delivery und DevOps: Eine wichtige Erkenntnis lautet, dass „schnell entwickelt" nicht automatisch „schnell live" bedeutet. Der Schritt zwischen Release und Produktion erfordert daher besondere Aufmerksamkeit. Mit einem dedizierten DevOps-Ansatz lassen sich die Freigabe und das Deployment beschleunigen, sodass die finale Version der Software in einer kürzeren Time-to-Market vorliegt.

Skill-Set: Die neue, agile IT-Welt stellt nicht zuletzt für die Mitarbeiter eine große Herausforderung dar; die kontinuierliche Erweiterung des Skill-Sets wird von der Kür zur Pflicht. Der Einsatz von „State of the art"-Technologie kann auf der anderen Seite auch einen nicht zu unterschätzenden Motivationsfaktor darstellen. Die Technologie-Auswahl muss zum Team passen, um weder eine Über- noch eine Unterforderung hervorzurufen.

Agile Architektur: In der Wasserfall-Methodik überwacht der Software-Architekt die Architektur. Was nun, wenn diese Rolle durch agile Methoden wegfällt? Logischerweise muss auch die Architektur agilen Maßstäben gehorchen. „Microservices" stellen einen vielversprechenden Ansatz dar, um Abhängigkeiten bewusst zu managen. Damit gelingt es nicht nur, Software agil zu entwickeln, sondern auch, agile Software zu entwickeln.

Managed Evolution: Der Umbau der Organisation darf im Zuge der agilen Transformation nicht planlos erfolgen, sondern sollte einem zielgerichteten Verfahren unterliegen. Für große Systeme ist der Ansatz der Evolution besser als die Neuentwicklung, weshalb diese in einem Prozess der „Managed Evolution" in agile Strukturen überführt werden. Insbesondere wichtig ist, agilitätshemmende Strukturen abzubauen. Die Priorität agiler Architekturen sollte daher der Priorität der Business-Ziele gleichgestellt sein.

Fazit: Alle sechs Bausteine sollten parallel entwickelt werden, um die notwendigen Ressourcen und Strukturen aufzubauen. Es kann eine große Hilfe bei der agilen Transformation sein, sich für bestimmte Spezialgebiete externes Know-how zuzukaufen, um auf wertvolle Erfahrungswerte, die natürlich zum Teil auch aus gemachten Fehlern entstanden sind, zurückgreifen zu können.

Andreas Meurer
andreas.meurer@pentasys.de

Andreas Meurer ist seit sechs Jahren in unterschiedlichen klassischen und agilen Rollen für die PENTASYS AG tätig: als Agile Coach, IT-Projekt-Manager sowie als Line Manager mit Personalverantwortung. Insgesamt bringt er knapp fünfzehn Jahre Erfahrung in der IT-Beratung mit. Er ist unter anderem als Projektmanager (IPMA Level D) sowie als Business Process Manager (OCEB-F) zertifiziert. Dabei fließen ständig neue Erfahrungswerte aus den beiden Welten „Personalführung" und „Projektarbeit" in seine Tätigkeiten ein, in denen er oft im Spannungsfeld klassischer Konzern-Strukturen und agiler Start-up-Mentalität steht.

Erweiterbare und bastlerfreundliche Heim-Automatisierung mit openHAB

Philipp Hertweck, Fraunhofer IOSB

Die Vernetzung unterschiedlicher Dinge nimmt immer mehr an Fahrt auf. Auch die eigene Wohnung bleibt davon nicht unberührt, sodass das „Smart Home" an Bedeutung gewinnt. Dafür gibt es bereits Lösungen verschiedener Hersteller, passende App und Cloud-Anbindung inklusive. Die Plattform openHAB liefert die Antwort, damit Geräte unterschiedlicher Anbieter interagieren und Privatsphäre-gerecht, ohne Bekanntgabe sensibler Daten, integriert werden können.

Auf openHAB können unterschiedliche Komponenten (und die damit verbundenen Protokolle) unter einer einheitlichen Benutzeroberfläche dargestellt und gesteuert werden. Neben einer Einführung in openHAB zeigt dieser Artikel Möglichkeiten, wie eigene Sensoren und Aktoren entwickelt und eingebunden werden können, sodass sich mithilfe von Micro-Controllern wie Arduino & Co. (fast) jeder Wunsch im Smart Home erfüllen lässt.

Die Heim-Automatisierung

Wäre es nicht schön, wenn sich die Steuerung von Geräten im Haus automatisieren ließe? Das Licht in der Küche nicht mehr vergessen werden kann, da es beim Verlassen des Raumes automatisch ausgeht? Beim Weggehen eine Warnung erscheint, dass das Fenster im Wohnzimmer noch offen steht? Eine Benachrichtigung kommt, wenn die Waschmaschine im Keller fertig ist? Dank vernetzter Heimgeräte lässt sich dies bereits heute umsetzen.

Unterschiedliche Hersteller bieten Systeme mit verschiedensten Sensoren und Aktoren an. Häufig ist eine App verfügbar, um die einzelnen Komponenten zu verwalten und zu steuern. Wenn die eigenen Wünsche mithilfe dieses Systems vollständig abgedeckt sind, ist dies eine einfache und komfortable Lösung. Anders sieht es aus, wenn Komponenten unterschiedlicher Hersteller kombiniert werden sollen. Bei geschlossenen Systemen ist dies kaum möglich. Abhilfe versuchen (Funk-)Standards wie ZWave, ZigBee, Bluetooth smart oder (kabelgebunden) KNX zu schaffen. Jedoch ist selbst innerhalb dieser Standards eine Interoperabilität verschiedener Hersteller nicht immer sichergestellt, obwohl diese, im Vergleich zu geschlossenen Systemen, häufig teurer sind.

Im Rahmen dieser Diskussion soll auch nicht außer Acht gelassen werden, dass einige proprietäre Systeme auf eine Cloud-Anbindung angewiesen sind; sie funktionieren nur, wenn die Dienste des Herstellers verfügbar und erreichbar sind. Die Vergangenheit hat gezeigt, dass dies nicht immer der Fall ist *(siehe „https://www.heise.de/newsticker/meldung/Serverausfall-bei-Homematic-IP-3903589.html“).* Außerdem stellt sich in diesem Zusammenhang die Frage nach dem Datenschutz: Was passiert mit den Daten aus meinem Smart Home? Wer erhält Zugriff darauf? Wie werden diese ausgewertet *(siehe „https://www.heise.de/newsticker/meldung/Datenschutzpanne-Testgeraete-von-Google-Home-Mini-hoerten-staendig-zu-3856399.html“)?* Ebenfalls sollte man darauf achten, wie lange ein Hersteller seine Cloud-Infrastruktur betreibt. Was passiert, wenn dieser den Betrieb einstellt? Sind dann alle Smart-Home-Komponenten nutzlos *(siehe „https://www.golem.de/news/revolv-google-macht-heimautomatisierung-kaputt-1604-120128.html“)?*

Allen diesen Problemen kann man mit openHAB *(siehe „http://www.openhab.org“)* begegnen, einem in Java geschriebenen Open-Source-Heimautomatisierungs-Framework, das auf dem Eclipse-Smart-Home-Projekt *(siehe „https://www.eclipse.org/smarthome“)* basiert. Es bietet eine hersteller- und protokollunabhängige Unterstützung sehr vieler proprietärer Smart-Home-Komponenten sowie gängiger Standards. Eine Abstraktions- und Integrationsebene erlaubt eine einheitliche Steuerung und das Zusammenspiel über Hersteller- und Protokollgrenzen hinweg. Ein mächtiges Regelsystem bietet die Automatisierung vieler Vorgänge. Die Hardware-Anforderungen von openHAB sind gering, sodass es problemlos auf einem Raspberry Pi ausgeführt werden kann. Dadurch lässt sich die Steuerung des Smart Homes in eigener Verantwortung betreiben und private Daten werden Dritten nicht bekanntgegeben.

Grundlegende Konzepte in openHAB

Nachfolgend werden einige Begriffe und Konzepte vorgestellt, die in openHAB eine zentrale Rolle spielen. Ein „Thing“ repräsentiert ein physikalisches oder virtuelles Objekt, beispielsweise einen Schalter für eine Lampe, einen Sensor, aber auch einen Webservice. Jede managebare Informationsquelle oder Funktion wird dadurch repräsentiert. „Channels“ bilden die unterschiedlichen Fähigkeiten eines Thing ab. Beispielsweise kann ein Sensor Temperatur und Luftfeuchtigkeit messen; für eine Lampe lassen sich Helligkeit und Lichtfarbe separat regeln. Jede dieser Eigenschaften wird durch einen Channel verfügbar gemacht. Ein „Binding“ ist ein Software-Adapter, der Things in openHAB integriert. Innerhalb eines solchen Add-ons findet die Kommunikation mit Things über unterschiedlichste Protokolle und Technologien statt. Es bietet somit eine Abstraktionsschicht der Smart-Home-Komponenten. Die Fähigkeiten, die in openHAB (durch die GUI oder in Regeln) verwendet werden können, sind durch „Items“ definiert. Jedes Item hat einen Zustand (die Temperatur, die ein Sensor gemessen hat, oder die Angabe, ob ein Schalter an oder aus ist) und kann gegebenenfalls Kommandos empfangen (Ein-/Aus-Schalten, Dimmen etc.). Ein „Link“ verbindet Items und Channels.

Auf dem Weg zum eigenen Smart Home

Nachdem openHAB installiert ist, sind einige Schritte erforderlich, um mit openHAB ein eigenes Smart Home aufzubauen. Anleitungen für verschiedene Systeme finden sich in der sehr guten openHAB-Dokumentation *(siehe „https://docs.openhab.org“).*

Die ersten Schritte können mithilfe des Paper UI ausgeführt werden, einer intuitiven Web-Oberfläche, die nach der Installation zur Verfügung steht. Zu Beginn ist das notwendige Binding für die vorhandenen Smart-Home-Komponenten zu installieren. Davon sind bereits mehr als 150 verfügbar, die die gängigsten auf dem Markt erhältlichen Systeme abdecken. Darüber hinaus sind Bindings verfügbar, mit denen Komponenten generisch integriert werden können. Das Exec-Binding erlaubt die Ausführung beliebiger Programme; mit dem HTTP-Binding lassen sich Komponenten integrieren, die per HTTP erreichbar sind. Das MQTT-Binding ermöglicht die Einbindung von Geräten, die über das gleichnamige Nachrichtenprotokoll angesprochen werden, das nachfolgend zum Einsatz kommt, um selbst gebaute Komponenten zu integrieren.

Sobald das passende Binding installiert ist, sind die Things zu konfigurieren. Werden proprietäre Systeme eingebunden, ist eine Bridge (IP-Gateway für Heimautomatisierungssysteme) häufig die Startkomponente. Sie ermöglicht die Integration weiterer Things, die per Funk mit dem Gateway kommunizieren. Am einfachsten funktioniert dies, wenn die Things per Autodiscovery automatisch gefunden und über Paper UI direkt angelegt werden können *(siehe Abbildung 1)*.

Vor der weiteren Einrichtung zunächst ein Blick auf die openHAB-Konfiguration. Eine komfortable Möglichkeit ist das bereits beschriebene Paper UI. Die Installation von Add-ons, das Anlegen erkannter Things mittels Autodiscovery sowie das Zuordnen von Channels zu Items ist dort möglich. openHAB lässt sich auch über Konfigurationsdateien einrichten. Diese Möglichkeit bietet sich vor allem an für Items, Sitemaps (konfigurierbare Benutzeroberfläche) und Rules (Regeln für die Automatisierung). In der Praxis hat sich eine Kombination beider Möglichkeiten bewährt. Vorlagen für Item- und Sitemap-Konfigurationen lassen sich mithilfe eines Generators (Home Builder, *siehe „https://docs.openhab.org/configuration/homebuilder.html“)* erstellen.

Anschließend sollen auf dem Weg zum eigenen Smart Home Items angelegt werden. Diese beschreiben die funktionale Sicht, also sämtliche durch UIs und Regeln steuerbaren Objekte. Um eine konsistente Sicht auf unterschiedliche Arten von Geräten zu erhalten, existieren verschiedene Item-Typen, beispielsweise „Contact“ (Zustand eines Tür-/Fensterkontakts Open/Close), „Dimmer“ (Prozentwert für dimmbare Items On/Off, Increase/Decrease, Percent), „Number“ (Zahlenwert) oder „Switch“ (Schalter On/Off). Die vollständige Liste steht in der openHAB-Dokumentation. *Listing 1* zeigt das Format einer Item-Definition.

Eine weitere praktische Möglichkeit zur Strukturierung ist die Gruppenbildung von Items. Sie ermöglicht die Anzeige (basierend auf einer Aggregats-Funktion) und Steuerung aller Gruppen-Items. Zusätzlich können Gruppen wiederum Gruppen zugeordnet sein, sodass eine hierarchische Gliederung möglich ist. *Listing 2* zeigt eine beispielhafte Items-Definition. Eine Sitemap definiert die in *Abbildung 2* gezeigte Benutzeroberfläche, auf der Items und Gruppen angeordnet werden.

„Rules“ sind ein mächtiges Werkzeug, um Abläufe zu automatisieren. Sie bestehen immer aus einem Namen und einem Skript-Block. *Listing 3* zeigt ein Beispiel, wie täglich um 6:30 Uhr das Item „Wake-Up_Light“ einschaltet wird.

```
itemtype itemname "labeltext [stateformat]" <iconname> (group1, group2, ...) ["tag1", "tag2", ...] {bindingconfig}
```

Listing 1

```
Group    Home                   "Daheim"            <house>
Group    Office                 "Büro"              <office>          (Home)
Group    LivingDining           "Wohn-Esszimmer"    <sofa>            (Home)

Switch   Office_Light           "Licht"             <light>           (Office, gLight)
Number   Office_Humidity        "Luftfeuchtigkeit"  <humidity>        (Office, gHumidity)
Number   Office_Temperature     "Temperatur"        <temperature>     (Office, gTemperature)
Switch   LivingDining_Light     "Licht"             <light>           (LivingDining, gLight)
Number   LivingDining_Humidity  "Luftfeuchtigkeit"  <humidity>        (LivingDining, gHumidity)
Number   LivingDining_Temperature "Temperatur"      <temperature>     (LivingDining, gTemperature)

Group:Switch:OR(ON, OFF) gLight         "Licht"              <light>          (Home)
Group:Number:AVG         gHumidity      "Luftfeuchtigkeit"   <humidity>       (Home)
Group:Number:AVG         gTemperature   "Temperatur"         <temperature>    (Home)
```

Listing 2

Als „Trigger" können ein Item-Event (Zustand eines Items ändert sich), die Uhrzeit, System- oder Thing-Events eingesetzt werden. Eine Java-ähnliche DSL, um den Zustand von Items auszulesen und um Commands an Items zu verschicken, ermöglicht somit eine mächtige Möglichkeit, das Smart Home zu automatisieren.

Selbstbau-Komponenten

Wie bereits angedeutet, soll dieser Artikel auch aufzeigen, wie man eigene Komponenten bauen und in das Gesamtsystem integrieren kann. Dafür kommt mit dem ESP8266 ein Low-Power-32-Bit-Microcontroller von espressif zum Einsatz, der für wenige Euro erhältlich ist. Neben einem integrierten WLAN-Chip ist dieser Mikrocontroller kompatibel zu Arduino. Dadurch lassen sich sämtlichen Tools wie beispielsweise die IDE, aber auch Bibliotheken aus dem Arduino-Umfeld, nutzen. Für Einsteiger ist es auch sehr hilfreich, dass viel Arduino-Hardware (Achtung, der ESP8266 arbeitet im Gegensatz zum Arduino mit 3,3 V statt 5 V) und Anleitungen mit wenig Aufwand angepasst und umgesetzt werden können.

Nachdem die Hard- und Softwareseite geklärt sind, muss im nächsten Schritt eine Integration in openHAB stattfinden. Auf den ersten Blick gibt es mehrere Möglichkeiten: Nutzung der openHAB-REST-Schnittstelle, eigenes Binding mit eigenem Protokoll implementieren oder Nutzen eines MQTT-Message-Bus. In diesem Artikel ist Letzteres umgesetzt, weil dadurch eine lose Kopplung von Komponenten und openHAB möglich ist. Durch die freie Wahl der Topics auf dem Message-Bus kann man schnell den Überblick verlieren. Um dies zu verhindern, hat Marvin Roger mit Homie *(siehe „https://github.com/marvinroger/homie")* eine leichtgewichtige MQTT-Konvention für das Internet der Dinge geschaffen. Dabei werden die Topics „homie/[deviceID]/$[deviceInfo]" und „homie/[deviceID]/[node]/[nodeProperty]" eingesetzt.

Stellt man die Begrifflichkeiten in Zusammenhang mit den Konzepten von openHAB, so entspricht ein Device in Homie einem Thing und jeder Node eines Device entspricht einem Channel. Für den ESP8266-Microcontroller existiert bereits eine Implementierung dieser Konvention *(siehe „https://github.com/marvinroger/homie-esp8266").* Diese übernimmt die vollständige WLAN- und MQTT-

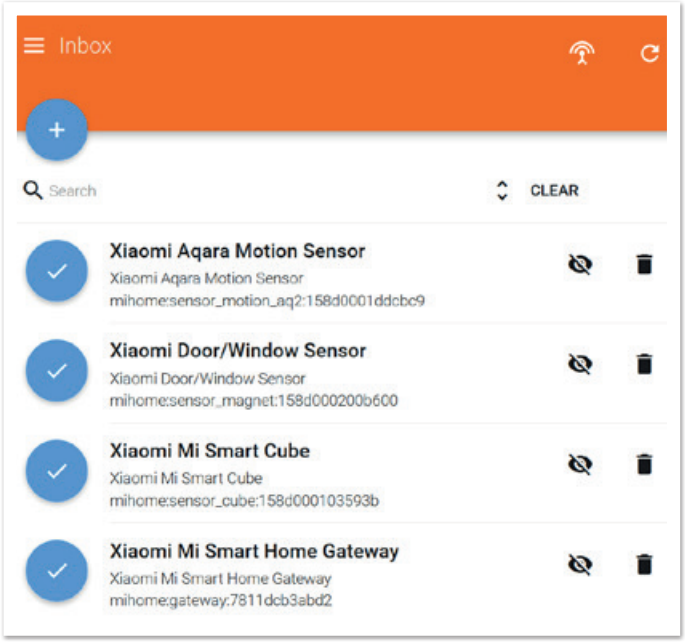


Abbildung 1: Per Autodiscovery erkannte Things

```
rule "WakeUp Light"
when
  Time cron "0 30 6 ? * *"
then
  sendCommand(WakeUp_Light, ON)
end
```

Listing 3

Verbindungsverwaltung (etwa erneutes Verbinden nach Abbruch) sowie das Bereitstellen von Meta-Informationen per MQTT. Ebenfalls müssen die WLAN- und MQTT-Konfiguration nicht im Quelltext hart codiert, sondern können per App über einen automatisch aufgespannten Access-Point eingestellt werden, wie man es auch von kommerziellen Produkten her kennt. Somit kann man sich bei der Programmierung auf die eigentliche Anbindung der Sensoren/Aktoren konzentrieren. Zuletzt muss in openHAB noch ein Item kon-

```
Number Living_Temperature "Temperatur"<temperature> {mqtt="<[local:homie/a020a616d1a3/temperature/
temperature:state:default]"}
```

Listing 4

figuriert werden, das die Temperatur vom Message-Bus liest und bereitstellt *(siehe Listing 4)*.

Sicherheit im Smart Home

Aufgrund des tiefen Eingriffs eines Smart Homes in den persönlichen Lebensbereich darf das Thema „Sicherheit" nicht außer Acht gelassen werden. Zu Beginn dieser Diskussion ist zu erwähnen, dass openHAB selbst keine Authentifizierung und keinen Zugriffsschutz bietet. Dadurch darf die openHAB-Instanz keinesfalls aus dem Internet erreichbar sein.

Wünscht man dennoch eine Steuerung/Überwachung aus der Ferne (beispielsweise mit der openHAB-App), gibt es mehrere Möglichkeiten der Absicherung. Zum einen kann mittels VPN eine gesicherte Verbindung ins Heimnetz aufgebaut werden, sodass ein Zugriff von außerhalb nicht notwendig wird. Alternativ existiert mit dem myopenHAB-Cloud-Service *(siehe „https://myopenhab.org")* ein Online-Dienst, der bestimmte Items nach Eingabe von Benutzername und Passwort erreichbar macht.

Es ist ebenfalls möglich, einen Reverse-Proxy vor openHAB zu konfigurieren, der die Authentifizierung und Autorisierung regelt. Dieser kann zusätzlich durch den Einsatz von SSL sicherstellen, dass der Zugriff vertraulich stattfindet.

An dieser Stelle ist jedoch jede einzelne Komponente des Smart Homes zu betrachten: Ist diese hinsichtlich Authentizität und Vertraulichkeit sicher integriert? Diese Frage ist nicht immer einfach zu beantworten, da oft proprietäre Komponenten und Protokolle zum Einsatz kommen. Wie sieht es aber mit den selbst gebauten Komponenten aus? Wurde die Standard-Einstellung des MQTT-Message-Brokers angepasst und dieser abgesichert?

Auch die Verfügbarkeit eines Smart Homes ist wichtig. Es wäre sehr ärgerlich, wenn das Licht nicht mehr eingeschaltet werden kann, weil das Netzwerk streikt. openHAB selbst läuft sehr stabil, auch auf einem Raspberry Pi. Dennoch kann ein Ausfall nicht ausgeschlossen werden. Daher ist beim Kauf der Komponenten darauf zu achten, dass die Basis-Funktionalität ohne openHAB realisiert werden kann und lediglich Komfort-Funktionen darauf angewiesen sind. Dadurch ist sichergestellt, dass man bei Problemen nicht zwingend im Dunklen oder Kalten sitzt. Ebenfalls sollten kritische Komponenten, wie beispielsweise eine Alarmanlage, nicht auf openHAB angewiesen sein. Hier sollte man auf zertifizierte Hardware-Systeme setzen, die durch optionale Funktionen (wie Benachrichtigung) ergänzt werden.

Weitere Funktionen

Bis hierhin wurden lediglich die grundlegenden Funktionen von openHAB vorgestellt. Durch dessen modularen Aufbau besteht eine Vielzahl an Erweiterungen. Beispielsweise existiert mit HABPanel ein leichtgewichtiges und konfigurierbares UI, das problemlos für Wand-Displays eingesetzt werden kann. Ebenfalls sind Benachrichtigungen, etwa per E-Mail oder Smartphone-Push-Meldung, beim Eintritt eines bestimmten Ereignisses möglich.

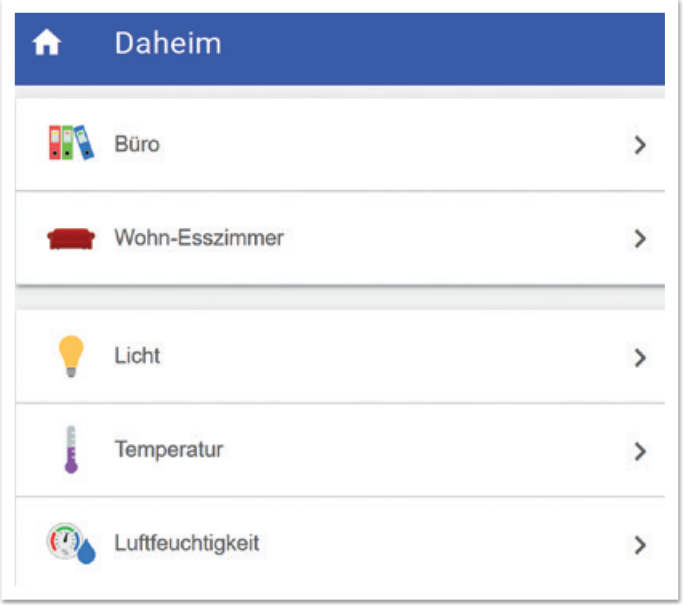
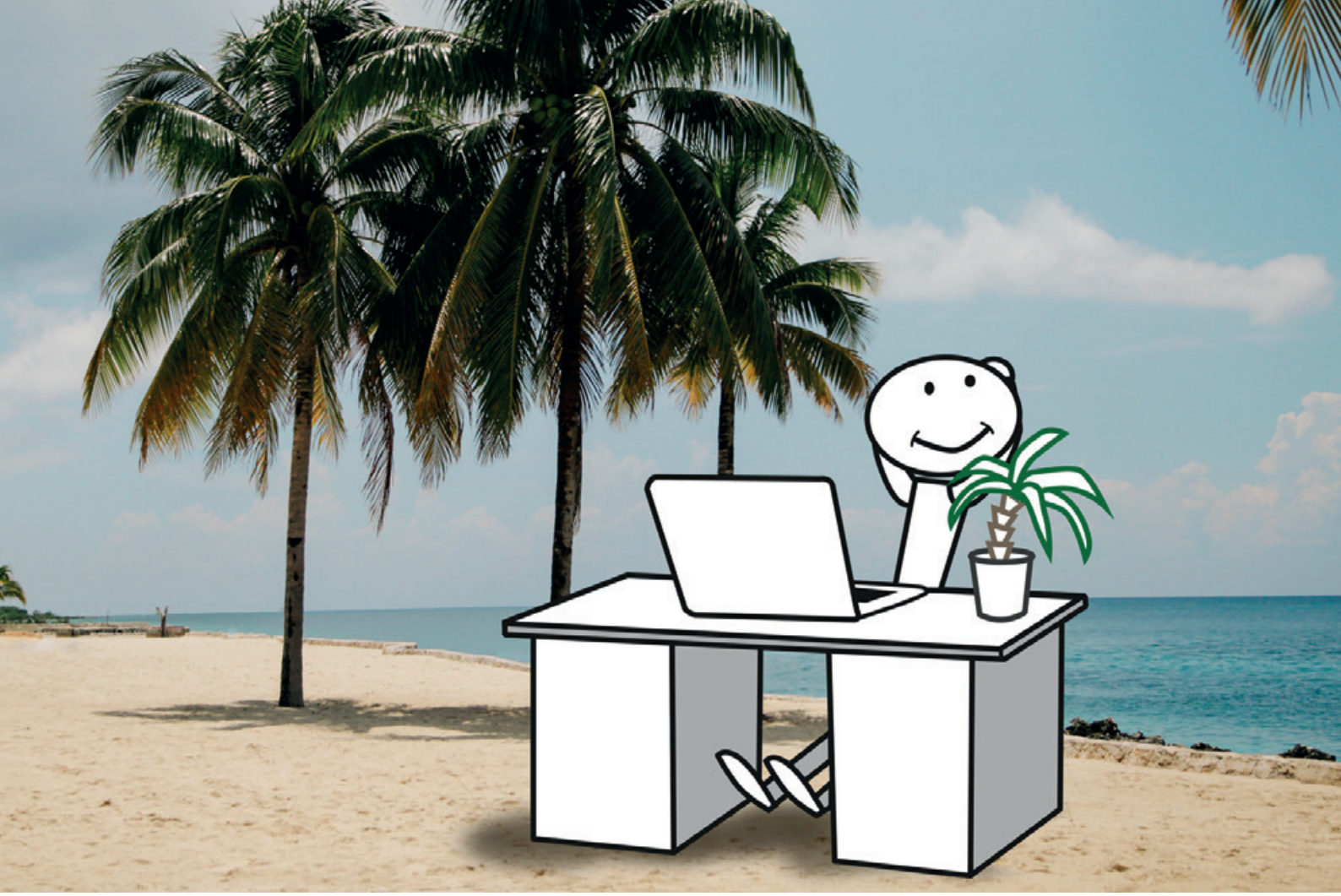


Abbildung 2: Konfigurierbare Benutzeroberfläche

Schnittstellen für Sprach-Ein- und -Ausgabe, wie sie beispielsweise durch Alexa und Google Home bekannt sind, sind ebenfalls vorhanden. Sollten selbst diese Funktionen die eigenen Wünsche nicht vollständig abdecken, gibt es für openHAB selbst ein Alexa Skill sowie eine Google-Home-Anbindung. Eine Integration mit IFTTT ist ebenfalls problemlos möglich. Dank einer großen Community mit einem aktiven Forum existieren sehr viele Projekte, Ideen und fertige Add-ons.

Philipp Hertweck
philipp.hertweck@iosb.fraunhofer.de

Nach dem Abschluss seines Informatik-Studiums am Karlsruher Institut für Technologie arbeitet Philipp Hertweck als wissenschaftlicher Mitarbeiter am Fraunhofer-Institut für Optronik, Systemtechnik und Bildauswertung IOSB in Karlsruhe. Dort beschäftigt er sich unter anderem mit Software-Architekturen sowie der semantischen Modellierung, insbesondere im Zusammenhang mit Entscheidungsunterstützungssystemen.



„Niemand schaut mir über die Schulter, und trotzdem machen alle das Richtige“ – Führung in verteilten Teams

Steven Schwenke, msg DAVID

Nachdem in den letzten beiden Ausgaben grundsätzliche Prinzipien des Remote-Arbeitens erläutert und Erfolgskonzepte für das Moderieren verteilter Meetings gezeigt wurden, beleuchtet dieser vorerst letzte Artikel der Serie die Führung in verteilten Teams.

Die wichtige Nachricht gleich vorneweg: „Führung“ ist im Rahmen dieses Artikels als die Summe aller Maßnahmen zu verstehen, ein verteiltes Entwicklungsteam zu einem gemeinsamen Erfolg zu führen. Dabei wird Wert darauf gelegt, sämtliche Aufgaben als eine organisatorische Einheit zu lösen, anstatt Aufgabenpakete „über den Zaun zu Nearshore“ zu werfen.

Hintergrund dieses alle Maßnahmen bestimmenden Gedankens ist der Entstehungsprozess des Projekts, in dem der Autor als Tech Lead für die technische Qualität und den Entwicklungsprozess verantwortlich ist. Als das Projekt auf der grünen Wiese gestartet wurde, war der Autor einer der beiden Entwickler, die von einem erfahrenen Architekten geführt wurden. Aufgrund der Größe und insbesondere der Firmenkultur wurde sehr viel Wert auf das gemeinsame Lernen und das nachhaltige Miteinander-Arbeiten ge-

legt. Sämtliche Fachlichkeit wurde von Anfang an allen Teammitgliedern im Detail erklärt und auch wichtige Entscheidungen, wie zum Beispiel das Schneiden von Aufgaben, wurden offen im Team diskutiert und entschieden.

Nach einiger Zeit wuchs das Team auf insgesamt sechs Kollegen an und der Autor unterstützte die Entwicklung ein halbes Jahr als Scrum-Master, um dann in die Rolle des Tech Lead zu wechseln. Aufgrund des bereits langen gemeinsamen Arbeitens auf Augenhöhe wurde die Tradition der größtmöglichen Transparenz und Offenheit im Team weitergeführt. Dies führte zu einer natürlichen und von allen respektierten Rollenverteilung, die zusammen mit ständiger Transparenz in sämtlichen Entscheidungsprozessen zum Projekterfolg beitrug.

Leading by Example

Wer kennt sie nicht, die sonore und scheinbar über allen potenziellen Problemen stehende Pilotenstimme aus dem Cockpit. Wenige kennen den Ursprung dieser Eigenart vieler Berufspiloten. Chuck Yeager war der erste Pilot, der am 14. Oktober 1947 mit der „X-1" die Schallmauer durchbrach *(siehe Abbildung 1)*. Aufgrund seiner Herkunft aus einer ländlichen Gegend in West Virginia und des damals unter Testpiloten verbreiteten Überlegenheits-Denkens *[1]* entwickelte er eine besondere Sprechweise, die ihn gegen alle Arten von Problemen komplett immun erschienen ließ. Spätestens mit seinem erfolgreichen Überschallflug galt er als Ikone der Fliegerei und selbst viele Berufspiloten ahmten diese distanzierte Sprechweise nach.

Auch wenn die Herkunft dieses besonderen Dialekts nichts mit Problemen moderner Software-Entwicklung zu tun hat, so hilft eine ruhige Stimmlage in kritischen Situationen aller Art sehr. Eine gute Führungspersönlichkeit nimmt sich selbst dann die Zeit für ein paar ruhige Atemzüge, wenn der Produktiv-Server das Log mit Fehlermeldungen flutet. Sie spricht ruhig, unaufgeregt und faktiziert und stellt mehr Fragen, als Anweisungen zu geben. Doch nicht nur in Ausnahmesituationen ist Führung wichtig. Stets zu Terminen pünktlich zu sein und Team-Mitglieder zu fragen, ob sie gerade Zeit haben, statt einfach Aufmerksamkeit zu nehmen, zeugt von gegenseitigem Respekt und erhöht mittelfristig die Produktivität aller Beteiligten.

All dies sind Beispiele für „Leading by Example". Sehr anschaulich ist dies in der bekannten Fernsehserie „Star Trek: The Next Generation" dargestellt. In kritischen Situationen versammeln sich alle Brückenoffiziere in einem Besprechungsraum, um dem Captain Handlungsoptionen aufzuzeigen. Dieser hört (meist) ruhig zu und stellt Verständnisfragen. Erst ganz zum Schluss, wenn alle Informationen allen bekannt sind, trifft er eine Entscheidung und verteilt eindeutige Aufgaben.

Trust Equation

Neben dem gegenseitigen Respekt ist Vertrauen eine der wichtigsten Komponenten eines gut funktionierenden Teams. Dieses kann im Wesentlichen durch drei Faktoren vergrößert werden:

1. Glaubwürdigkeit
2. Verlässlichkeit
3. Vertrautheit

Abbildung 1: Chuck Yeagers X-1

Glaubwürdigkeit ist in der Theorie einfach zu erreichen, nämlich indem man genau das vorlebt, was man sagt. Verlässlichkeit in Teams ist das Verhältnis von Versprochenem zu Eingehaltenem. Dieser Aspekt lässt sich insbesondere bei der Übertragung von Aufgaben und Verantwortung sicherstellen. In dieser Situation ist die explizite und unmissverständliche Kommunikation von Erwartungshaltungen und Rahmenbedingungen wichtig. Nach den Erläuterungen hilft die einfache Frage „Fühlst du dich mit dieser neuen Aufgabe wohl?", da mit ihr nicht nur die Erfolgswahrscheinlichkeit erhöht werden kann, sondern auch die Zuversicht auf ein gutes Ergebnis.

Vertrautheit-Schaffen ist eine sehr individuelle Angelegenheit. Das Einbringen von Persönlichem zusätzlich zur Arbeitsebene ist wohl der einfachste Weg. Über den eigenen Urlaub, die eigenen Hobbies oder sogar die Familie zu reden, lockert in den richtigen Situationen nicht nur auf, sondern schafft auch Verbundenheit auf einer ganz menschlichen Ebene. Dies ist im Umfeld des verteilten Arbeitens wichtig, da der sogenannte „Watercooler-Effect" (im Deutschen am besten „Flurfunk") fehlt und man sich neben den geschäftlichen Meetings oft wenig Zeit für ein einfaches Gespräch nimmt.

Glaubwürdigkeit, Verlässlichkeit und Vertrautheit steigern, wie beschrieben, das Vertrauen in einem Team. Laut der Trust Equation gibt es nur einen Aspekt, der diesem positiven Effekt entgegenwirkt: Selbstorientierung oder Egoismus. Stellt man das Team voran und achtet darauf, dass sich jeder im Team abgeholt, eingebunden und nicht hintergangen fühlt, stellt sich eine vertraute Atmosphäre wie von selbst ein. Diese Betrachtung ist bis hierhin natürlich reichlich theoretisch und trocken, deshalb folgen ein paar Beispiele.

Der Urlaubsantrag eines rumänischen Teammitglieds für nur einen Vormittag hat uns in Deutschland verwundert; normalerweise nimmt man sich ja immer einen ganzen Tag frei. Auf Nachfrage stellte sich heraus, dass eine Motorrad-Führerscheinprüfung abgenommen wurde. Aus diesem kleinen Detail ergaben sich mehrere Gespräche über Motorräder, schöne Strecken in Rumänien und Unterschiede im allgemeinen Straßenverkehr. Außerdem wissen nun

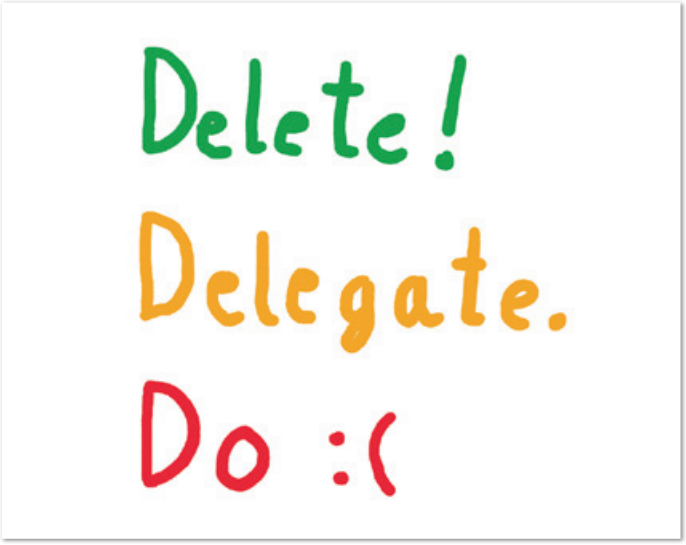


Abbildung 2: „3Ds" aus „Getting Things done"

alle im Team, dass die rumänische Entwicklerin und der Projektleiter ein gemeinsames Hobby haben.

Wie in vergangenen Artikeln erwähnt, arbeitet der Autor viel von Zuhause aus. Dadurch ergeben sich fast schon zwangsmäßig Einblicke in das Privatleben. Angefangen beim Bücherregal, das bei jeder Skye-Session im Hintergrund zu sehen ist und von einigen Kollegen bei Gelegenheit auf Veränderungen geprüft wird, bis hin zur über den Schreibtisch laufenden Katze. Letztere ist bestens geeignet, um angestrengte Fachgespräche aufzulockern.

Als letztes Beispiel soll die offene und direkte Kommunikation im Team des Autors dienen. Es hat sich als extrem vertrauensbildend herausgestellt, möglichst viel Drumherum in alle Richtungen des Teams zu kommunizieren. Organisatorische Details, Berichte darüber, was man so für andere Baustellen außerhalb des Teams getan hat, das Aufklären von Gerüchten oder einfach nur interessante Begebenheiten aus anderen Bereichen tragen nicht direkt zum Projekt-Fortschritt bei, schaffen aber Nähe. Mit einer solchen stabilen und über lange Zeit erarbeiteten Vertrauensgrundlage werden potenziell negative Ereignisse viel entspannter aufgenommen: „Wenn sich wirklich etwas Schlechtes für uns ergeben sollte, werden wir wie immer zeitnah und direkt informiert. Wir brauchen uns jetzt noch keine Gedanken zu machen."

Delegation

Gekonntes Delegieren erleichtert das (Projekt-)Leben ungemein. Schon im Selbstmanagement-Klassiker „Getting Things done" (GTD) *[2]* wird empfohlen, eintreffende Aufgaben am besten gleich wieder zu löschen, wenn es denn geht. Falls das nicht der Fall ist, sollte man sie (richtig) delegieren *(siehe Abbildung 2)*. Nur im Notfall ist eigenes Handeln angesagt. An dieser Stelle ist es wichtig zu unterstreichen, dass dieses Prinzip nicht zur Handlungsunfähigkeit und zum „Nur Weiterleiten" aufruft. Es geht hier um den bestmöglichen Einsatz der einem zur Verfügung stehenden Zeit. Ziel ist es, dass jeder das tut, was er am besten kann.

Wenn man dieses Prinzip verfolgt, stellt sich natürlich die Frage: Wie delegiert man nun am besten? In verteilten Teams ist dieses Thema noch viel wichtiger, als bei an einem Ort arbeitenden Kollegen. Bei falscher Delegation besteht nicht nur die Gefahr eines falschen oder ungewünschten Ergebnisses. Schlimmer: Dieses wenig brauchbare Ergebnis wird erst nach einigen Tagen bekannt, wenn man sich nicht regelmäßig über den Fortschritt austauscht. So kann sehr effektiv und massiv Zeit verbraucht werden. Einem verteilten Team das richtige Delegieren beizubringen, ist also eine wichtige Investition in die Arbeitsfähigkeit des Teams. Außerdem wird so die in diesem Setting oft drohende „Micro-Management-Trap" vermieden, in der jeder kleinste Schritt erklärt und überprüft wird, statt Verantwortlichkeiten zu übertragen und nur das Endergebnis zu prüfen.

Ein wichtiger Aspekt beim Delegieren ist die Deadline. Nach dem Parkinson'schen Gesetz *[3]* nimmt jede Aufgabe so viel Zeit in Anspruch, wie man ihr einräumt. Wird also keine Deadline gesetzt, werden Aufgaben potenziell nie fertig. Deshalb ist bei jeder delegierten Aufgabe immer eine Deadline zu setzen. Diese hat nicht die Aufgabe, unnötig sportlich anzutreiben. Wenn man als Delegierender weiß, dass die Aufgabe vermutlich fünf Tage dauert, hat es keinen Sinn, nur drei Tage Zeit zu gewähren.

Das Ziel einer Deadline ist nicht die Optimierung, sondern die Ermöglichung der Selbstorganisation bei dem ausführenden Teammitglied. Die Angabe des Zeitpunkts der Deadline muss so exakt und genau wie möglich erfolgen. Die Auskünfte „in den nächsten Tagen" oder „im Winter 2018" sind nicht nur ungenau, sondern auch fehleranfällig: Ist der Winter der Zeitpunkt, an dem es das erste Mal schneit oder ganz exakt der 21. Dezember? Besser ist folgender Satz: „Ich benötige den Text am Freitag, den 9. Juni um 15 Uhr deutscher Zeit. Dann kann ich ihn über das Wochenende lesen, um ihn am Montag bei meinem Termin mit dem Kunden in den Vertrag aufzunehmen." Schon an der Länge erkennt man, dass hier viel mehr Informationen übermittelt werden.

Sicher ist es nicht unbedingt notwendig zu wissen, was mit dem textuellen Ergebnis der übertragenen Aufgabe passiert. Ein vertragsrelevanter Text, der eins zu eins so dem Kunden übergeben wird, erzeugt jedoch sicherlich ein anderes Gefühl beim Schreiber als eine unwichtigere Aufgabe. Die Information, dass der Delegierende sich wirklich auf das Eintreffen des Textes bis zur angegebenen Zeit verlassen können muss, da ansonsten der ganze Zeitplan durcheinanderkommt, bietet Einblick in den viel größeren Kontext und ermöglicht damit mehr Verständnis. Der Zeitpunkt der Deadline ist jedenfalls absolut unmissverständlich – selbst über Zeitzonen hinweg.

Das hervorragende Buch „Management 3.0" von Jurgen Appelo *[4]* beinhaltet eine ganze Checkliste von Aspekten guter Delegation. Angefangen beim Risiko-Faktor, nämlich der Problematik, ob das Team-Mitglied eine Aufgabe aufgrund seiner Fähigkeiten und Kenntnisse überhaupt sinnvoll bearbeiten kann, bis hin zur Frage „Weiß der Kollege, wie Fortschritt an der Aufgabe aussieht?" werden viele unterschiedliche Aspekte abgedeckt.

Als letzter, aber absolut nicht unwichtiger Punkt zum Thema „Delegation" sei das Verhalten im Fehlerfall genannt. Wenn Delegation schiefläuft, und das wird früher oder später garantiert der Fall sein, stellt sich manchmal eine ernüchterte Stimmung ein. „Warum haben die Kollegen die Aufgabe nicht verstanden?" „Was war daran so schwer?" – diese Fragen kommen schnell, helfen jedoch nicht. Die

einzige Frage, die wirklich hilft, ist: „Was kann ich als Delegierender beim nächsten Mal besser machen, damit es funktioniert?"

Der Zuschauer-Effekt

Einer der ungünstigeren Zustände, in die ein verteiltes Team fallen kann, hat mit dem Zuschauer-Effekt *[5]* zu tun. Dieser beschreibt das Verhalten größerer Menschenansammlungen bei Notfällen oder in besonderen Situationen. Je größer die Menge, desto geringer die Chance, dass helfend eingegriffen wird. „Es sind so viele Menschen hier, irgendeiner wird schon helfen", wird oft gedacht. Blöd nur, wenn das alle denken.

Insbesondere bei Retrospektiven, wenn über Probleme und Hindernisse geredet wird, kann dieser Effekt auch ein verteiltes Team treffen. Das beispielhaft genannte Problem schlechter User-Stories ist von den Entwicklern in der Retro genannt und pflichtbewusst vom Protokollanten niedergeschrieben wurden. Dann schweigen alle, auch die Entwickler und sogar der im Team vorhandene Anforderungsmanager, der die Stories geschrieben hat. Warum sagt keiner etwas?

Erst nach einiger Moderationsarbeit kommt heraus, dass sich tatsächlich jeder zwar als Zuschauer und irgendwie Leidtragender, aber eben nicht als Verantwortlicher sieht: Die Entwickler haben den Punkt ja aufgebracht – der Anforderungsmanager muss besser schreiben, klarer Fall. Dieser ist mit seinen Stories allerdings zufrieden und muss sein Verhalten nicht ändern. Jetzt, wo die Entwickler grundsätzlich willkommene Vorschläge für die Stories haben, werden sie sich ja schon mit Fragen melden.

Der Tech-Lead denkt sich, dass er damit ja sowieso nicht viel zu tun hat. So blicken alle Anwesenden auf das eben notierte Problem und jeder denkt vom anderen, er würde handeln. Doch es passiert nichts. Deshalb muss der Moderator (oder Scrum-Master, je nach Rollenverteilung) dafür sorgen, dass konkrete und korrekt delegierte Aufgaben mit dem Team entwickelt und dann ebenfalls ins Protokoll aufgenommen werden. Das Niederschreiben schafft Verbindlichkeit. Das Nennen genau eines Verantwortlichen schafft Ansprechbarkeit. Das Verwenden direkter Sprache, wie „Steven, wir wollen, dass du das eben beschriebene Problem bis Mittwoch analysierst und uns bis spätestens 15 Uhr eine Mail mit den Ergebnissen schickst.", schafft Klarheit über die nächsten Schritte.

Genau diese nächsten Schritte werden oft vergessen: Eine sehr einfache Technik zur Vermeidung des Zuschauereffekts ist die Frage „Was ist der nächste Schritt?", die man immer dann stellen sollte, wenn eine Aufgabe scheinbar in der Luft hängt und droht, vom nächsten Tagesordnungspunkt überholt zu werden.

Regelmäßige One-on-Ones

Als „One-on-Ones" werden regelmäßig stattfindende Treffen zwischen zwei Team-Mitgliedern bezeichnet. Der Autor führt solche

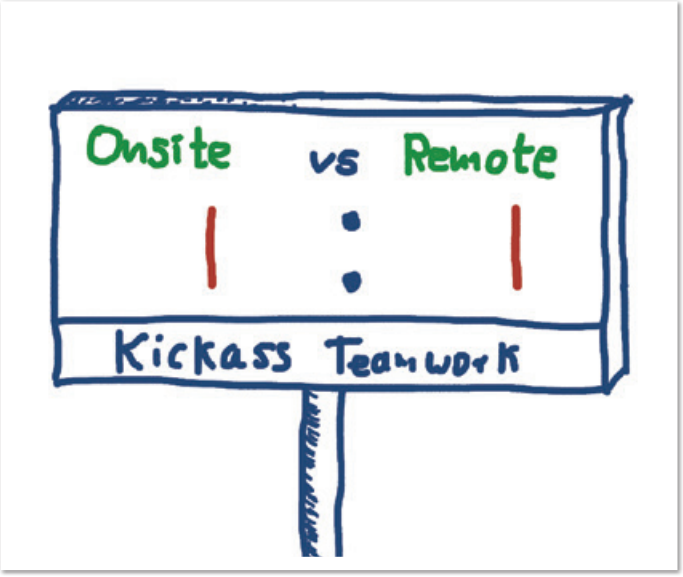


Abbildung 3: Regelmäßige One-on-Ones sind nicht nur ein Gewinn für das Team-Mitglied, sondern für das ganze Team.

Treffen mit jedem Mitglied seines Teams mindestens alle sechs Monate durch. Ziel ist, in einer entspannten und vertrauten Atmosphäre darüber zu reden, was dem Teammitglied für seine tägliche Arbeit wichtig ist, und daraus konkrete Maßnahmen abzuleiten *(siehe Abbildung 3)*.

Einführen kann man solche Gespräche immer sehr gut mit einer Wiederholung, wie die Welt beim letzten Gespräch aussah. Was gab es für Herausforderungen, was lief damals gut und was nicht? Beim ersten One-on-One kann man auf dem technischen Background und der bisherigen Werdegeschichte aufbauen.

Anschließend sollten Parallelen und Unterschiede zur aktuellen Situation identifiziert werden. Welche Probleme gibt es noch immer? Was hat sich wesentlich verbessert? Was wurde seit dem letzten Treffen gelernt und wie kann das neue Wissen nun angewendet werden?

Der dritte Teil betrifft die Zukunft bis zum nächsten Treffen und sollte zur Formulierung klar zugeordneter Aufgaben führen. Dazu ein Beispiel aus dem Team des Autors: Das zweite One-on-One mit einem im Team relativ neuen Entwickler wurde in der ersten Phase genutzt, um die Einarbeitung ins Projekt zu besprechen. Der Entwickler hat sich im Team gut integriert und eine Nische gefunden, in der er sich gut einbringen konnte. In der zweiten Phase wurde festgestellt, dass mit den ersten Erfolgen und einem sicheren Umgang mit dem Projekt eine Weiterentwicklung beginnen sollte.

Im dritten Teil wurde die Möglichkeit besprochen, in eine Architekten-nahe Richtung zu gehen und das Projekt zum Beispiel durch die Untersuchung geeigneter neuer Bibliotheken oder Tools zu ergänzen. Als ganz konkrete Aufgabe hat der Autor sich verpflichtet, dem Kollegen Einblick in seine Architektur-Aufgaben-Liste zu geben und erste, kleinere Aufgaben daraus zu übertragen. Diese sollten als Sprungbrett in kompliziertere Aufgabenbereiche dienen. Zum Thema One-on-Ones ist der Artikel „Stop wasting my Time with your shitty One-on-One Meetings" von Mike Acton empfohlen *[6]*.

Fazit

Wie ganz am Anfang erwähnt, ist das Wort „Führung" nicht unbedingt passend für die in diesem Artikel beschriebenen Maßnahmen. Ein erfolgreiches Team ist selbstorganisierend und findet eigene Wege, Tools und Prozesse, um die bei verteilten Teams vorhandene geographische Entfernung abzubauen und so erfolgreich zu sein. Einer der wichtigsten Aspekte dabei ist, wie so oft, die gelungene Kommunikation. Angefangen bei einer auch in Stresssituationen ruhigen Stimme, die faktiziert spricht und nach einem gemeinsamen Findungsprozess Aufgaben korrekt delegiert, über langfristig angelegten Vertrauensaufbau bis hin zu regelmäßigen One-on-One-Meetings ist der Weg zur erfolgreichen Führung lang und teilweise mühselig. Doch es lohnt sich. Ein derart empowertes Team wird Schwierigkeiten wie von selbst souverän meistern und braucht langfristig wesentlich weniger Management-Aufmerksamkeit.

Der Autor jedenfalls ist immens stolz auf die Kollegen, mit denen er die in dieser Artikelserie beschriebenen Erkenntnisse gemeinsam erarbeitet hat und mit denen er sowohl durch gute als auch durch etwas mühseligere Projekt-Situationen ging und dabei stets etwas gelernt hat. Vielleicht tragen die hier beschriebenen Konzepte zum Erfolg anderer Teams bei. Damit diese auch über Aktualisierungen des Themas „Remote Working" informiert bleiben, sei noch einmal auf das Repository der msg DAVID verwiesen, in dem all das hier Geschriebene und einiges mehr frei erhältlich ist *[7]*.

[1] Siehe „The right Stuff", Tom Wolfe
[2] *https://gettingthingsdone.com*
[3] *https://de.wikipedia.org/wiki/Parkinsonsche_Gesetze*
[4] *https://management30.com*
[5] *https://de.wikipedia.org/wiki/Zuschauereffekt*
[6] *https://medium.com/@mike_acton/stop-wasting-my-time-with-your-shitty-one-on-one-meetings-258b5c5c10cc*
[7] *https://github.com/msg-DAVID-GmbH/RemoteWorking*

Steven Schwenke
steven.schwenke@msg-david.de

Steven Schwenke ist Software Craftsman und liebt, was er tut. Als Technical Teamlead leitet er ein verteiltes Team und führt regelmäßig Inhouse-Workshops durch. Zusätzlich organisiert er Veranstaltungen wie den HackTalk und hält Vorträge bei Konferenzen. Er teilt seine Erfahrung gern mit anderen und freut sich auf spannende Gespräche.

Serverless Java mit Fn Project

Marcel Amende, Oracle Deutschland B.V. & Co. KG

Das Serverless-Computing-Programmiermodell ist der neue Trend beim Entwickeln in der und für die Cloud. Man braucht sich dabei keine Gedanken mehr über Server, Storage, Netzwerk, virtuelle Maschine und selbst zu implementierende, infrastrukturelle Hilfsroutinen zu machen und kann sich ganz auf die Hauptsache konzentrieren: den eigenen Code, der die Geschäfts-Anforderungen modular in Form von kleinen Funktionspaketen (Function-as-a-Service, FaaS) umsetzt. Server gibt es beim Serverless Computing natürlich dennoch. Sie bleiben allerdings für den Entwickler unsichtbar und brauchen selbst bei sich ändernden Last-Anforderungen keinerlei Administration.

Mit AWS Lambda *[1]*, Cloud Functions *[2]* und Azure Functions *[3]* gibt es bereits einige kommerzielle, weitgehend geschlossene Serverless-Computing-Angebote am Markt. Fn Project ist eine Open-Source-Serverless-Plattform als Alternative, die zudem erstklassige Unterstützung für die Programmiersprache Java bietet. Auch Fn Project hat bereits eine längere Historie. Das Team dahinter gehört zu den Pionieren des Serverless Computing und rekrutiert sich aus den Entwicklern der IronFunctions-Serverless-Plattform *[4]*.

Im Unterschied zu den erstgenannten Plattformen bekommt man bei Fn Project nicht nur ein API oder ein Benutzer-Interface, um den Code hochzuladen; Fn Project ist komplett Container-basiert. Die Funktion ist hier der Container und der Container ist die Funktion. Dadurch ist sie unverändert überall lauffähig: als Docker-Container auf dem Laptop des Entwicklers, auf einer Container-Plattform im eigenen Rechenzentrum oder als Kubernetes-Cluster in der Cloud jedes beliebigen Anbieters.

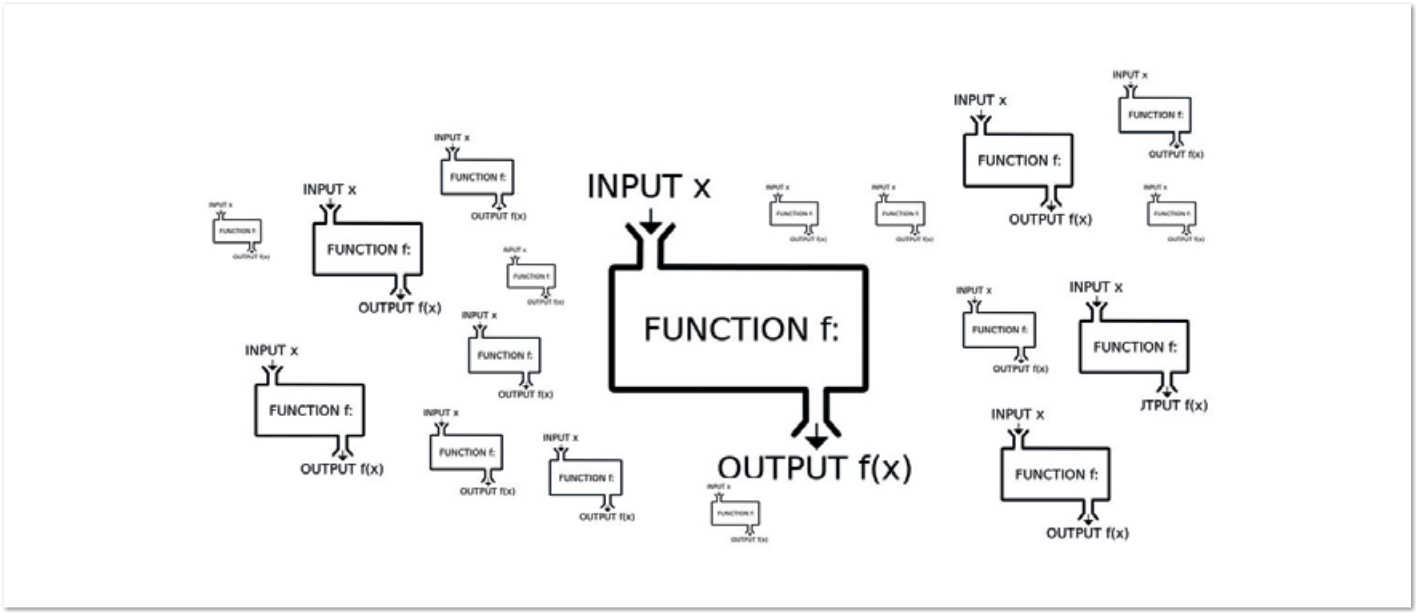


Abbildung 1: Funktionen im Serverless-Computing-Programmiermodell

Einführung in Fn Project

Der Fokus von Fn Project liegt auf der einfachen Nutzbarkeit. Es ist in Go implementiert, kann aber mit jeder beliebigen Programmiersprache genutzt und einfach um zusätzliche erweitert werden.

Funktionen sind das Herzstück der Serverless-Computing-Plattform. Sie stellen kleine, isolierte Codefragmente dar, die eine bestimmte Anforderung möglichst optimal erledigen. Sie lesen von der (oft Standard-)Eingabe, tätigen ihre Berechnungen und erzeugen eine Ausgabe *(siehe Abbildung 1)*.

Die Funktion stellt auch die Ausbringungseinheit dar. Um alles weitere, die Provisionierung der benötigten Infrastruktur, die Ausbringung, Skalierung, Verfügbarkeit, Elastizität, Sicherheit und Abrechnung, kümmert sich die Plattform. Code lässt sich in dieser Form sehr stark verdichtet betreiben, was letztendlich Kosten spart.

Bei Fn Project entwickelt man Funktionen von der Kommandozeile aus durch Nutzung des Kommandos „fn". Einzige Abhängigkeit ist Docker. Dieses muss vorinstalliert sein, da jede Funktion in Form eines Docker-Containers erstellt und ausgebracht wird. Für die einfache Erstellung von Funktionen gibt es bereits eine Reihe von Entwicklungsbaukästen (FDK, Function Development Kit) für Java, Go, JavaScript, Ruby, Rust oder Python.

Den größten Funktionsumfang und die größte Robustheit hat derzeit das Java FDK, das in enger Zusammenarbeit mit dem Java-Entwicklerteam von Oracle entsteht. Ein FDK ist ein vorgefertigter Do-

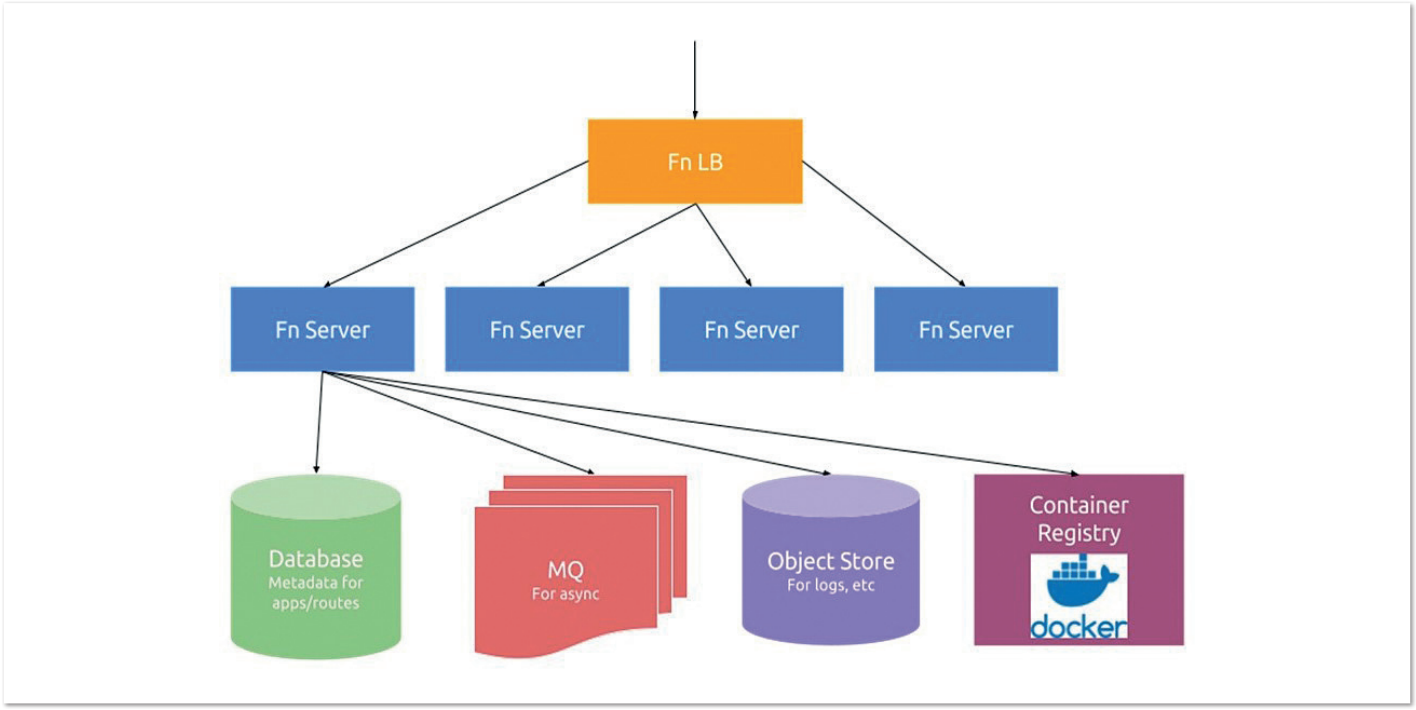


Abbildung 2: Architektur von Fn Project

cker-Container mit den benötigten Laufzeit-Komponenten, in den der Funktionscode eingebracht wird. Aber auch jeder selbst erstellte Docker-Container kann als Funktion betrieben werden. Dadurch ist es möglich, Lambda-Funktionen lokal auszuführen oder auf Cloud-Plattformen anderer Anbieter zu migrieren.

```
$ fn start
        ______
       / ____/___
      / /_  / __ \
     / __/ / / / /
    /_/   /_/ /_/
        v0.3.313

time="2018-03-03" msg="Fn serving on `:8080`" type=full
```

Listing 1

```
$ fn init --runtime=java javaaktuell
Creating function at: /javaaktuell
        ______
       / ____/___
      / /_  / __ \
     / __/ / / / /
    /_/   /_/ /_/

Runtime: java
Function boilerplate generated.
func.yaml created.
$
```

Listing 2

```
$ tree
.
├── func.yaml
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── example
    │               └── fn
    │                   └── HelloFunction.java
    └── test
        └── java
            └── com
                └── example
                    └── fn
                        └── HelloFunctionTest.java
```

Listing 3

```
$ fn run
Building image javaaktuell:0.0.1
Sending build context to Docker daemon  13.82kB
Step 1/11 : FROM fnproject/fn-java-fdk-build:jdk9-
latest as build-stage
...
Step 11/11 : CMD ["com.example.fn.HelloFunction::handle
Request"]
 ---> Using cache
 ---> 79717de5757b
Successfully built 79717de5757b
Successfully tagged javaaktuell:0.0.1
Hello, world!
```

Listing 4

Zusätzlich gibt es noch eine Reihe weiterer architektonischer Grundkomponenten, die für den Betrieb von Fn Project notwendig sind *(siehe Abbildung 2)*. Zentral ist der Fn Server für die Ausführung der Funktionen und eine lokale oder zentrale Container Registry für deren Ausbringung. Für die verteilte Nutzung und Protokoll-Adaption nutzt man vor den Server-Komponenten einen Load Balancer. Bei asynchronen Funktionen wird eine Message Queue (etwa auf Basis von Redis oder Bolt) benötigt. Hinzu kommt eine Datenbank (wie MySQL, Progress oder sqlite3) für persistente Operationen, die aus Latenzgründen aber möglichst wenig und schreibend nur außerhalb der Anfragezyklen einer Funktion genutzt wird.

Erste Schritte mit Fn Project

Als Open-Source-Projekt unter Apache 2.0 Lizenz findet man den Fn-Project-Quellcode und die direkt ausführbare Variante auf GitHub *[5]*. Auch die Schnellstart-Anleitung und Anwendungsbeispiele sind hier veröffentlicht. Weitere, ausführliche Tutorien finden sich auf der Projektseite „fnproject.io“ *[7]*. Auf MacOS lässt sich Fn Project mit dem Paketmanager „Homebrew“ per „$ brew install fn“ installieren. Alternativ können die Installationsroutinen für die Kommandozeilen-Ausführung unter MacOS, Linux und Windows von GitHub bezogen werden *[6]*. Nach der Installation lässt sich der Fn Server starten, um später Funktionen lokal ausführen und testen zu können *(siehe Listing 1)*.

Eine erste Java-Funktion

Die Erstellung einer einfachen Funktion aus einer Vorlage dauert mit dem Kommando „fn init“ nur wenige Sekunden. Die gewünschte Laufzeit-Umgebung übergibt man als Parameter. In dem Beispielprojekt dieses Artikels wird eine auf Java 9 basierende Funktion entwickelt, daher heißt der Parameter „Java“. Mit derzeit unter anderem Go, Node, Python, Ruby oder Rust ist gleichsam eine Reihe weiterer Laufzeit-Umgebungen unterstützt *(siehe Listing 2)*. Die Initialisierungsroutine erzeugt eine Projektverzeichnisstruktur mit vier Dateien *(siehe Listing 3)*.

Eine Datei „func.yaml“ wird unabhängig von der gewählten Laufzeit-Umgebung immer erzeugt. Im Falle einer Java-Funktion enthält diese Parameter für Laufzeit und Version sowie die vollqualifizierten Namen der Java-Funktionsklasse und der aufzurufenden Methode. Zusätzlich findet man den Quellcode einer Beispielfunktion „HelloFunction.java“ mit einem JUnit Testcase „HelloFunctionTest.java“ und eine Maven-„pom.xml“-Datei zum Kompilieren und Testen der Funktion.

Build und Run

Im nächsten Schritt wird der Build-Prozess für die Funktion gestartet, um diese dann lokal zu testen. Dafür kommt zunächst die lokale Docker-Registry zum Einsatz, deren Benutzername mit „$ export FN_REGISTRY=<local_docker_user>“ als Umgebungsvariable gesetzt wird.

Der erste Build- und Deployment-Prozess dauert länger als nachfolgende, da die benötigten Docker-Basis-Images aus dem Docker-Hub geladen werden. Gleiches gilt für die Verwendung von Maven, bei dem zusätzliche Java-Packages geladen werden. Mit dem Befehl „fn run“ lassen sich der Build-Prozess der Funktion starten und diese sofort einmalig ausführen *(siehe Listing 4)*.

```
$ echo "Java aktuell" | fn run
Building image javaaktuell:0.0.1
...
Successfully built 79717de5757b
Successfully tagged javaaktuell:0.0.1
Hello, Java aktuell!
```

Listing 5

```
package com.example.fn;

public class HelloFunction {

    public String handleRequest(String input) {
        String name = (input == null || input.isEmpty())
                ? "world"  : input;
        return "Hello, " + name + "!";
    }
}
```

Listing 6

```
    // My POJO
    public static class MyCustomer(){
        String name;
    };
...
    // Function Method
    public String handleRequest(MyCustomer cust){
        ....
    }
```

Listing 7

```
$ fn build
Building image javaaktuell:0.0.1
...
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.example.fn.HelloFunctionTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time
elapsed: 0.941 sec

Results :

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...
Function javaaktuell:0.0.1 built successfully.
$
```

Listing 8

Im Hintergrund wurde die Funktion mit Maven kompiliert, in einen Docker-Container eingepackt, an die lokale Docker-Registry übergeben und ausgeführt. Die Funktion gibt den Text „Hello, world!" auf der Standard-Ausgabe aus. Man kann der Aufruf-Funktion per Pipe-Befehl auch Parameter über die Standardeingabe übergeben *(siehe Listing 5)*.

Die Funktion hat offensichtlich den Text „Java aktuell" von der Standard-Eingabe gelesen und daraufhin die Ausgabe „Hallo, Java aktuell!" generiert. Ein Blick in den Quellcode der generierten Klasse „com.example.fn.HelloFunction" zeigt, dass es sich bei der Funktion um eine Methode „handleRequest" in einem einfachen Java-POJO handelt *(siehe Listing 6)*.

Die Methode akzeptiert einen String und gibt ebenfalls einen String zurück. Das Java-FDK unterstützt allerdings auch das Binden anderer Ein- und Ausgabe-Typen wie Streams, primitive Daten-Typen, Byte-Arrays und in POJOs gewandelte JSON-Objekte. Will man JSON verwenden, ersetzt man die Aufruf- beziehungsweise Rückgabe-Argumente der Funktionsmethode einfach durch POJOs und setzt in der Datei „func.yaml" den Parameter „Format" zu „format: json" *(siehe Listing 7)*.

Das Java-FDK bindet basierend auf diesen Argumenten die Eingabedaten automatisch. Diese JSON-Unterstützung ist bereits in das FDK eingebaut. Wandler für andere Datenformate, wie XML oder Protobuf, lassen sich einfach ergänzen.

Zum generierten Beispiel gehört mit der Klasse „com.example.fn.HelloFunctionTest" auch ein JUnit-Test. Dieser wird von Maven ausgeführt, wenn man den Build per „fn build" explizit startet *(siehe Listing 8)*.

Mit Fn Server Funktionen ausbringen

Nachdem der Funktions-Container erstellt ist, kann dieser über das Kommando „fn deploy" auf einem Fn Server ausgebracht werden. Generell können dafür lokale und entfernte beziehungsweise öffentliche Docker-Registraturen verwendet werden. Wir nutzen die lokale Registratur in Kombination mit dem bereits gestarteten, lokalen Fn Server. Das spart dem Entwickler viel Zeit und unterscheidet Fn Project signifikant von anderen Serverless-Computing-Frame-

```
$ fn deploy --local -app myapp
Deploying javaaktuell to app: javaapp at path: /javaaktuell
Bumped to version 0.0.2
Building image javaaktuell:0.0.2
...
Successfully built 031e7a11ede4
Successfully tagged javaaktuell:0.0.2
Updating route /javaaktuell using image javaaktuell:0.0.2...
$
```

Listing 9

```
$ curl --data "Test" http://localhost:8080/r/myapp/javaaktuell
Hello, Test!
$
```

Listing 10

works und Implementierungen. Diese sind meist nur als Services in der Cloud eines bestimmten Anbieters verfügbar und können lokal höchstens die Ausführung einer Funktion simulieren. Bei Fn Project bestehen keine Unterschiede zwischen lokalem Server und Service in der Cloud; man nutzt immer die exakt selbe Codebasis und Implementierung. Für das lokale Ausbringen wird die Option „--local" gesetzt *(siehe Listing 9)*.

Der Parameter „--app" dient dazu, mehrere Funktionen zu einem Dienst zusammenzuführen. In der letzten Zeile der Ausgabe sieht man, dass mit dem Ausbringen einer Funktion auf dem Fn Server automatisch eine Route mit der Funktion assoziiert wird, in diesem Fall mit „/javaaktuell" der Name des Funktions-Verzeichnisses. Dies bedeutet, dass die Funktion auch als Webservice über das Hilfskommando „curl" oder aus einem Browser aufrufbar ist *(siehe Listing 10)*.

Es fällt auf, dass die Ausführung dieser einfachen Funktion mit mehreren Hundert Millisekunden recht lange dauert. Dies liegt daran, dass die vorliegende Funktion eine sogenannte „Cold Function" ist, bei der die Funktion inklusive ihres Containers und ihrer Laufzeit-Umgebung für jeden Aufruf neu gestartet wird. Für einen Webservice, der oftmals ausgeführt wird, ist dies nicht praktikabel. Durch Setzen des Parameters „format: http" in der Datei „func.yaml" ändert man die Funktion in eine „Hot Function", die für das Bedienen mehrerer Anfragen bis zu einer voreingestellten Auszeit von dreißig Sekunden am Leben gehalten wird.

Asynchrone Funktionen

Asynchrone Funktionen eignen sich besonders für rechenintensive Aufgaben und Massen-Operationen. Die Funktions-Aufrufe werden in eine Warteschlange eingestellt und zu einem späteren Zeitpunkt ausgeführt. Der Aufrufer wartet nicht auf eine sofortige Rückantwort wie bei einer synchronen Funktion. Er erhält stattdessen ein JSON-Objekt mit einer Aufruf-ID. Das Ergebnis der eigentlichen Funktionsausführung wird in ein Log eingestellt. Die Aufruf-ID lässt sich später nutzen, um das Ergebnis aus dem Log abzurufen. Für die Erstellung einer asynchronen Funktion muss die „func.yaml" um das Attribut „type: async" ergänzt werden. Nach erneuter Ausbringung kann die asynchrone Funktion aufgerufen werden, in diesem Beispiel durch das Fn-Kommandozeilen-Interface *(siehe Listing 11)*.

Die Aufruf-ID wird in Form eines JSON-Objektes zurückgegeben. Ebenfalls über die Fn-Kommandozeile lässt sich per Status-Endpunkt-API der Status der Funktionsausführung abfragen *(siehe Listing 12)*.

Der Status „success" bedeutet, dass der Aufruf erfolgreich beendet wurde. Zusätzlich wird die Einstellungs-, Start- und Endzeit ausgegeben. Um zu überprüfen, ob die Funktion die richtige Ausgabe generiert hat, kann der Log-Endpunkt abgefragt werden, um das Ausgabeergebnis zu erhalten *(siehe Listing 13)*.

Komplexe Funktionen mit Fn Flow

Fn Flow stellt dem Entwickler ein flexibles Modell zur Verfügung, um einzelne Funktionen zu komplexeren Arbeitsabläufen zusammenzusetzen. Es behandelt Aspekte wie die Ausführungs-Reihenfolge, Parallelisierung, Wiederanläufe und Fehlerbehandlung. Es ist dabei kein weiteres schwergewichtiges, grafisches Workflow-Werkzeug, wie ein Service-Bus, ein BPEL- oder BPMN-Implementierungen sie

```
$ fn call javaapp /javaaktuell
{"call_id":"01C96TVVR247WHT00000000000"}
$
```

Listing 11

```
$ fn calls get javaapp 01C96TVVR247WHT00000000
ID: 01C96TVVR247WHT00000000000
App: javaapp
Route: /javaaktuell
Created At: 2018-03-22T12:27:47.330Z
Started At: 2018-03-22T12:27:48.829Z
Completed At: 2018-03-22T12:27:49.331Z
Status: success
$
```

Listing 12

```
$ fn logs get javaapp 01C96TVVR247WHT00000000000
HTTP/1.1 200 INVOKED
Content-Length: 13
Content-Type: text/plain

Hello, world!
$
```

Listing 13

```
Flow f = Flows.currentFlow();
FlowFuture flight = f.invokeFunction("/flight/book",
...);
FlowFuture hotel = f.invokeFunction("/hotel/book", ...);
flight.thenCompose(
    (flightRes) -> hotel.whenComplete(
        (hotelRes) -> Email.send(flightRes, hotelRes)
    ).exceptionallyCompose((e) -> cancel(/hotel/cancel))
).exceptionallyCompose((e) -> cancel(/flight/cancel));
```

Listing 14

darstellen, sondern wird im Code definiert und ganz natürlich als Funktion ausgeführt. Das zugehörige API ermöglicht Einsicht in den Status des Ablaufs, in Logs und Stack Traces. Auch wenn die erste Implementierung von Fn Flow für Java verfügbar ist, ist es konzeptionell sprachagnostisch.

Implementierungen in Go, JavaScript und Python werden in Kürze folgen. Ein typisches Beispiel ist die Buchung einer Reise, bei der folgender Ablauf denkbar wäre: Stelle parallel Buchungsanfragen für einen Flug und ein Hotel; verschicke im Erfolgsfall eine Bestätigungsmail und storniere das Hotel, wenn kein Flug verfügbar ist. Als Java-Code sieht dies in leicht vereinfachter Form wie in *Listing 14* aus.

Das hier verwendete asynchrone und verteilte Programmiermodell nutzt sogenannte „Futures" und „Promises". Ein Objekt wird mit einer bestimmten Ausführung starten, zu einem zukünftigen Zeitpunkt eine Ausgabe ermitteln oder einen Grund liefern, warum dies nicht erfolgen konnte. Registrierte „Callbacks" verarbeiten die Rückgaben. Die Klasse „Flow" übergibt jeden Schritt des Arbeitsablaufs an den Flow-Server, der die Schritte als individuelle Aufrufe an den FN-Server orchestriert und auch die Rückantworten auswertet. Den

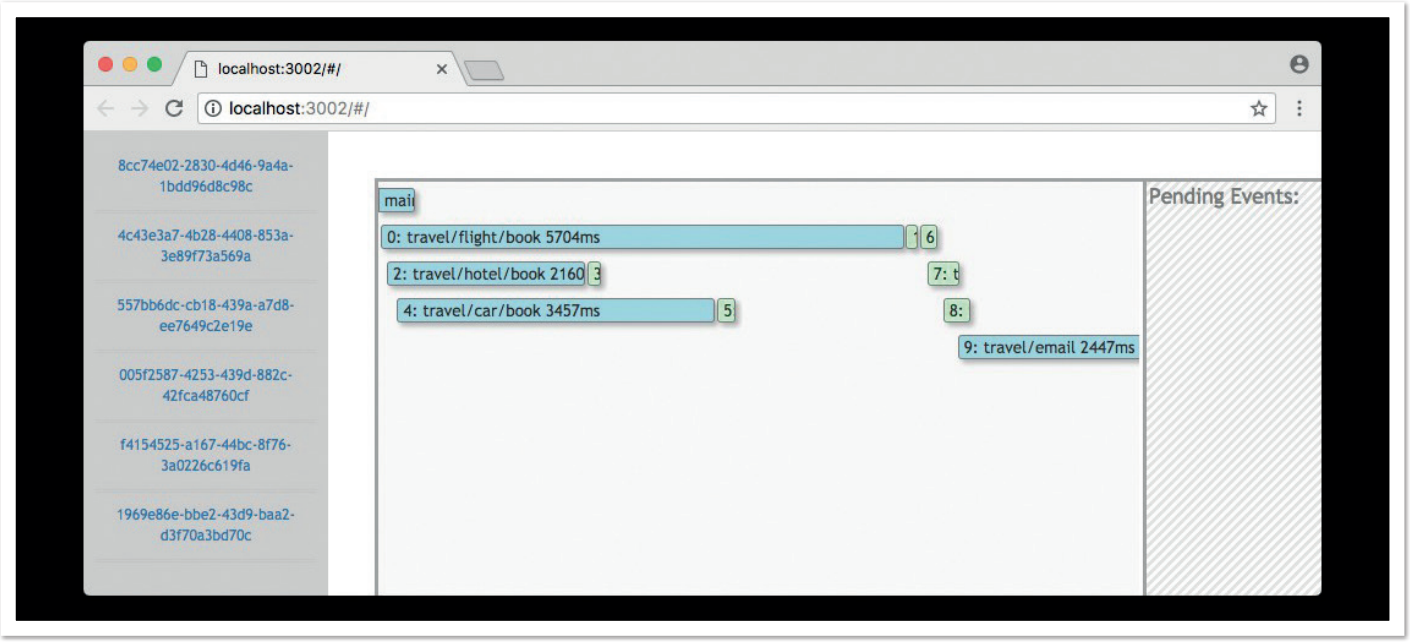


Abbildung 3: Flow-Benutzerinterface - Reisebuchung

tatsächlichen, detaillierten Ablauf inklusive der Ausführungszeiten kann man sich im Flow-Benutzerinterface ansehen. Dieses ist derzeit noch in einem experimentellen Stadium, eignet sich aber hervorragend zur Problem- und Fehler-Analyse. Man findet in der Visualisierung beispielsweise leicht sich blockierende Schritte und Aufrufe. *Abbildung 3* zeigt eine Ausführungs-Instanz der Reisebuchung.

Fn Project in der Produktion

Eine mit Fn erstellte Funktion ist ein Docker-Container. Daher ist eine mit Kubernetes verwaltete Docker-Umgebung eine gute Wahl, um Fn-Funktionen ablaufen zu lassen. Für die einfache Installation der benötigten Fn-Ressourcen wie Fn-Service, Fn-UI, Flow-Service, Flow-UI und Persistenz-Datenbank kann ein vorgefertigtes Helm-Chart genutzt werden. Dabei handelt es sich um einen kommandozeilenbasierten Paketmanager für Kubernetes. Das Helm-Repository mit den für Fn benötigten Konfigurationsdaten kann direkt mit „$ git clone git@github.com:fnproject/fn-helm.git" aus GitHub bezogen werden *[9]*. Danach müssen noch mit „helm dep build fn" die Abhängigkeiten installiert werden. Schließlich installiert man das Chart und setzt mit diesem sowie „helm install --name my-release fn" in einem Arbeitsschritt die Fn-Plattform als verwaltete Docker-Umgebung auf.

Zusammenfassung und Ausblick

Das bestehende Angebot kommerzieller Serverless-Computing- und Functions-as-a-Service-Angebote ist breit. Im Vergleich dazu ist Fn Project spät am Markt. Die kommerziellen Angebote sind allerdings oft proprietär und auf die Nutzung mit den zugehörigen Diensten der Cloud des jeweiligen Anbieters begrenzt. Fn Project geht – auf der langjährigen Erfahrung mit Iron.io *[4]* aufbauend – einen anderen Weg. Es ist vollständig Open Source und konsequent Container-basiert. Dadurch läuft es auf Basis der exakt selben Codebasis *[5]* auf den Laptops der Entwickler, unter Nutzung von Docker-Verwaltungswerkzeugen wie Kubernetes in den eigenen Rechenzentren der Anwender oder in jeder beliebigen öffentlichen Cloud. Zudem ist es einfach erweiterbar und zählt auf die Erfahrung und die Mitwirkung der Entwicklergemeinde.

Während Java – die Programmiersprache mit der auch in Zeiten der Cloud immer noch weitesten Verbreitung *[10]* – von anderen Serverless-Computing-Anbietern stiefmütterlich behandelt wird, startet Fn Project dank des Erfahrungsschatzes der Oracle-Entwickler mit exzellenter Unterstützung. Was zur Vollständigkeit noch fehlt, ist der passende und automatisch skalierende Serverless-Computing-Plattformdienst in der Oracle-Cloud *[8]*.

Weitere Informationen

[1] *aws.amazon.com/lambda*
[2] *cloud.google.com/functions*
[3] *azure.microsoft.com/serverless/functions*
[4] *open.iron.io*
[5] *github.com/fnproject/fn*
[6] *github.com/fnproject/cli/releases*
[7] *fnproject.io/tutorials*
[8] *cloud.oracle.com/tryit*
[9] *github.com/fnproject/fn-helm*
[10] *www.tiobe.com/tiobe-index*

Marcel Amende
marcel.amende@oracle.com

Geboren als Ingenieur, aufgewachsen bei Oracle, zu Hause in der Cloud: Marcel Amende repräsentiert die „Cloud Native"-Entwicklergemeinde bei Oracle, um aus den Diensten der Oracle-Cloud kreative Kundenlösungen zu gestalten. Sein besonderes Interesse gilt dabei den großen Innovationsthemen dieser Tage wie IoT, Blockchain, Mobile & Bots sowie Serverless Computing & Co.

Jenkins

Coding Continuous Delivery – statische Code-Analyse mit SonarQube und Deployment auf Kubernetes et al. mit dem Jenkins-Pipeline-Plug-in

Johannes Schnatterer, Cloudogu GmbH

Die ersten drei Teile dieser Artikelserie in den letzten drei Ausgaben der Java aktuell zum Thema „Jenkins Pipelines" beschreiben Grundlagen, Performance und Werkzeuge wie Shared Libraries und Docker. Dieser letzte Teil widmet sich der Integration statischer Code-Analyse mittels SonarQube. Er zeigt, wie man kontinuierlich auf Kubernetes ausliefern kann, und gibt Anregungen für Continuous Delivery (CD) auf anderen Plattformen.

Die Pipeline-Beispiele aus den Vorgänger-Beiträgen werden in diesem Artikel sukzessive erweitert. Das Beispiel-Projekt ist weiter der „kitchensink"-Quickstart von WildFly. Alle Pipeline-Beispiele sind sowohl in „declarative"- als auch in „scripted"-Syntax realisiert. Den aktuellen Stand jeder Erweiterung kann man bei GitHub *[1]* nachverfolgen und ausprobieren. Hier gibt es für jeden Abschnitt jeweils für „declarative" und „scripted" einen Branch, der das vollständige Beispiel umfasst. Das Ergebnis der Builds jedes Branch lässt sich außerdem direkt auf der Jenkins-Instanz *[2]* einsehen. Die Nummerierung der Branches setzt sich damit aus den vorhergehenden Teilen fort. Docker war Nummer neun, also ist die statische Code-Analyse das zehnte Beispiel.

Statische Code-Analyse mit SonarQube

Die statische Code-Analyse erfasst gängige Fehlermuster, Code Style und Metriken (wie Code Coverage) anhand des Source- oder Byte-Codes. Auf Basis der Metriken können dann Quality Goals festgelegt werden, etwa „Code Coverage muss bei neuem Code größer 80 Prozent sein".

Diese Schritte sind gut automatisierbar und deshalb ein weiteres Mittel zur Qualitätssicherung in einer CD-Pipeline, neben den bereits gezeigten Unit- und Integrationstests. Weit verbreitet ist dabei das Werkzeug SonarQube (SQ), das in einer eigenen Web-Anwendung gekapselt die Möglichkeit zur statischen Code-Analyse bietet *[3]*. Dabei können über Plug-ins viele Programmiersprachen und die Regeln von Tools wie FindBugs, PMD und Checkstyle integriert werden.

Die Integration in die Jenkins Pipeline erfolgt per Jenkins-Plug-in *[4]*. Nach der Installation wird in Jenkins die SQ-Instanz konfiguriert, typischerweise mithilfe von URL und Authentication Token. In SQ wird zudem ein Webhook eingerichtet, der Jenkins über die Ergebnisse der Quality Gate Checks asynchron informiert. Der Webhook wird vom SQ-Plug-in auf Jenkins bereitgestellt: „/sonarqube-webhook". Die Analyse kann man dann unter anderem vom SQ-Plug-in für Maven auf Jenkins durchführen lassen. *Listing 1* zeigt, wie das mit „scripted"-Syntax in der Jenkins Pipeline abgebildet werden kann.

Das SQ-Plugin stellt auf der Pipeline verschiedene Steps bereit:

- „withSonarQubeEnv()" injiziert die in der Konfiguration für eine SQ-Instanz (hier mit der ID „sonarcloud.io") angegebenen Werte wie die URL als Environment-Variablen in den entsprechenden Block
- „waitForQualityGate()" wartet auf den Aufruf des Webhook, der über den Zustand des Quality Gate informiert

In *Listing 1* wird bei negativem Ergebnis der Build auf „unstable" (gelb) gesetzt. Hier ist alternativ auch ein Aufruf des „error()"-Steps möglich, der den Build auf „failed" (rot) setzt. Um zu verhindern, dass der Build hier unbegrenzt wartet, wird ein Timeout gesetzt. Ist beispielsweise kein Webhook konfiguriert, bricht der Build nach zwei Minuten ab. Auch hier empfiehlt es sich, die Logik in einen eigenen Step „analyzeWithSonarQubeAndWaitForQualityGoal()" auszulagern, um den Pipeline-Code lesbarer zu gestalten. In „declarative"-Syntax muss mindestens die Prüfung des Quality Gate in einen eigenen Step oder „script"-Block geschrieben werden.

Die offizielle Pipeline-Dokumentation *[5]* empfiehlt zwar, den Step „waitForQualityGate()" außerhalb eines Node durchzuführen, um diesen so wenig wie möglich zu blockieren. Dies führt allerdings zu schwerer wartbarem Code (siehe dazu die Beispiele bei GitHub im Branch 10a, jeweils für „scripted" und „declarative" *[1]*). In „declarative"-Syntax führt dies unter anderem dazu, dass jede Stage in einem eigenen Build Executor ausgeführt wird, womit sich zudem die Gesamtlaufzeit der Pipeline deutlich verlängert *[2]*. Da die Antwortzeiten von SQ typischerweise bei wenigen Millisekunden liegen, zeigt *Listing 1* den pragmatischen Weg innerhalb des Node.

Es gibt noch einige fortgeschrittene Themen im Umgang mit SQ, etwa die Ergebnisse der Analyse direkt als Kommentar in Pull Requests schreiben zu lassen oder die Verwendung des Branch-Features (nicht in der Community Edition). Diese Features sind zum Beispiel von der Shared Library „ces-build-lib" *[6]* komfortabel bereitgestellt.

Deployment

Wenn alle qualitätssichernden Maßnahmen erfolgreich waren, kann zum Abschluss der CD-Pipeline das Deployment erfolgen. Je nach Grad der Automatisierung steht hier am Ende das Deployment in Produktion. Um dabei die Risiken zu verringern, ist es empfehlenswert, mindestens eine Staging-Umgebung zu betreiben. Eine einfach umsetzbare Logik, um sowohl Staging als auch Produktion automatisiert einzurichten, ist die Verwendung von Branches im Source Code Management (SCM).

```
node {
    stage('Statical Code Analysis') {
        analyzeWithSonarQubeAndWaitForQualityGoal()
    }
}
// ...
void analyzeWithSonarQubeAndWaitForQualityGoal() {
    withSonarQubeEnv('sonarcloud.io') {
        mvn "${SONAR_MAVEN_GOAL} -Dsonar.host.
url=${SONAR_HOST_URL} -Dsonar.login=${SONAR_AUTH_TOKEN}
${SONAR_EXTRA_PROPS} "
    }
    timeout(time: 2, unit: 'MINUTES') {
        def qg = waitForQualityGate()
        if (qg.status != 'OK') {
            currentBuild.result = 'UNSTABLE'
        }
    }
}
```

Listing 1

Viele Teams arbeiten mit Feature-Branches oder Git Flow, in denen der integrierte Entwicklungsstand auf dem Develop-Branch zusammenfließt und der Master-Branch die produktiven Versionen enthält. Darauf kann man einfach seine CD-Strategie aufbauen: Jeder Push auf Develop führt zu einem Deployment auf die Staging-Umgebung, jeder Push auf Master geht in Produktion. So hat man stets die letzte integrierte Version auf Staging und kann dort funktionale oder manuelle Tests durchführen, bevor man durch einen Merge auf Master das Deployment in Produktion anstößt. Zudem ist ein Deployment pro Feature-Branch denkbar.

Eine solche Deployment-Logik lässt sich mit Jenkins Pipelines einfach realisieren, da man den Branch-Namen in Multibranch Builds aus dem Environment abfragen kann (siehe *Listing 2*, Stage „deploy"). Wohin und wie man Software einrichtet, hängt vom Projekt ab. In den letzten Jahren haben sich Container Orchestration Platforms als flexibles Mittel für DevOps-Teams erwiesen. Hier hat sich Kubernetes (K8s) als De-facto-Standard herauskristallisiert, weshalb dieser Artikel exemplarisch das Deployment auf K8s beschreibt. Um eine Anwendung auf K8s einzurichten, sind vier Schritte notwendig:

1. Versionsnamen festlegen
2. Docker-Image bereitstellen (Image bauen, mit Version als Tag und in Registry hochladen)
3. Image-Version in Deployment-Beschreibung (typischerweise in YAML) aktualisieren
4. YAML-Datei auf K8s-Cluster anwenden.

Listing 2 zeigt die Umsetzung dieser Schritte in „scripted"-Syntax.

Der Versionsname lässt sich mit Groovy erzeugen, etwa als Zeitstempel in *Listing 2*. Hier könnte man durch Anhängen des Git Commit Hash mehr Eindeutigkeit schaffen. Denkbar wäre es außerdem, aus der Pipeline einen Git Tag zu setzen. Um diese Version auch in Maven zu verwenden, bietet sich die Nutzung der ab Maven 3.5.0 verfügbaren „CI Friendly Versions" *[8]* an. *Listing 3* zeigt, wie dies in der „pom.xml" umgesetzt wird.

Im Build wird die Version dann mithilfe des Arguments „-Drevision" *(siehe Listing 2)* an Maven übergeben. Die weiteren Schritte in *Listing 2* sind in einem eigenen Step „deployToKubernetes()" realisiert. Dieser wird jedoch erst nach der Prüfung daraufhin, ob der Build noch

```
node {
    String versionName = createVersion()
    stage('Build') {
        mvn "clean install -DskipTests -Drevision=${versionName}"
    }
    // ...
    stage('Deploy') {
        if (currentBuild.currentResult == 'SUCCESS') {
            if (env.BRANCH_NAME == "master") {
                deployToKubernetes(versionName, 'kubeconfig-prod', 'hostname.com')
            } else if (env.BRANCH_NAME == 'develop') {
                deployToKubernetes(versionName, 'kubeconfig-staging', 'staging-hostname.com')
            }
        }
    }
}
String createVersion() {
    String versionName = "${new Date().format('yyyyMMddHHmm')}"

    if (env.BRANCH_NAME != "master") {
        versionName += '-SNAPSHOT'
    }
    currentBuild.description = versionName
    return versionName
}
void deployToKubernetes(String versionName, String credentialsId, String hostname) {

    String dockerRegistry = 'your.docker.registry.com'
    String imageName = "${dockerRegistry}/kitchensink:${versionName}"
    docker.withRegistry("https://${dockerRegistry}", 'docker-reg-credentials') {
        docker.build(imageName, '.').push()
    }

    withCredentials([file(credentialsId: credentialsId, variable: 'kubeconfig')]) {
        withEnv(["IMAGE_NAME=${imageName}"]) {
            kubernetesDeploy(
                    credentialsType: 'KubeConfig',
                    kubeConfig: [path: kubeconfig],
                    configs: 'k8s/deployment.yaml',
                    enableConfigSubstitution: true
            )
        }
    }

    timeout(time: 2, unit: 'MINUTES') {
        waitUntil {
            sleep(time: 10, unit: 'SECONDS')
            isVersionDeployed(versionName, "http://${hostname}/rest/version")
        }
    }
}
boolean isVersionDeployed(String expectedVersion, String versionEndpoint) {
    def deployedVersion = sh(returnStdout: true, script: "curl -s ${versionEndpoint}").trim()
    return expectedVersion == deployedVersion
}
```

Listing 2

stabil ist, aufgerufen. Wenn das Quality Gate fehlschlug, soll natürlich nicht eingerichtet werden.

Das Bauen und Hochladen des Image lässt sich dank der in der letzten Ausgabe beschriebenen Docker-Integration leicht mit Jenkins-Bordmitteln realisieren. Dabei muss man sich bei der Docker-Registry authentifizieren. Dazu hinterlegt man in Jenkins Username und Password-Credentials an („docker-reg-credentials" in *Listing 2*). Deren Herkunft hängen vom Anbieter der Registry ab, beispielsweise ist das Passwort bei der Google-Container-Registry eine JSON-Datei *[7]*, die man in einfachen Anführungszeichen ohne Zeilenumbrüche in Jenkins einfügt.

Um den Versions-Namen in die YAML-Datei zu schreiben, kann man einen eigenen Step zum Ersetzen im Jenkinsfile schreiben oder man verwendet ein Plug-in. Eine komfortable Möglichkeit ist das „kubernetes-cd-plugin" *[9]*. Es stellt den Step „kubernetesDeploy()" zur Verfügung, der YAML-Dateien filtert und direkt auf den Cluster anwendet. Dabei werden alle Einträge mit der „$VARIABLE"-Syntax in den YAML-Dateien durch entsprechende Environment-Variablen aus der Jenkins Pipeline ersetzt (in *Listing 2*, zum Beispiel „IMAGE_NAME").

Um die YAML-Datei auf den Cluster anwenden zu können, muss sich das Plug-in beim K8s-Master authentifizieren. Dafür erzeugt man einen K8s-Service-Account für Jenkins und legt dessen Rechte mit Role-Based Access Control fest. *Listing 4* zeigt exemplarisch, wie

```
<project>
    <version>${version}</version>
    <properties>
        <version>-SNAPSHOT</version>
    </properties>
</project>
```

Listing 3

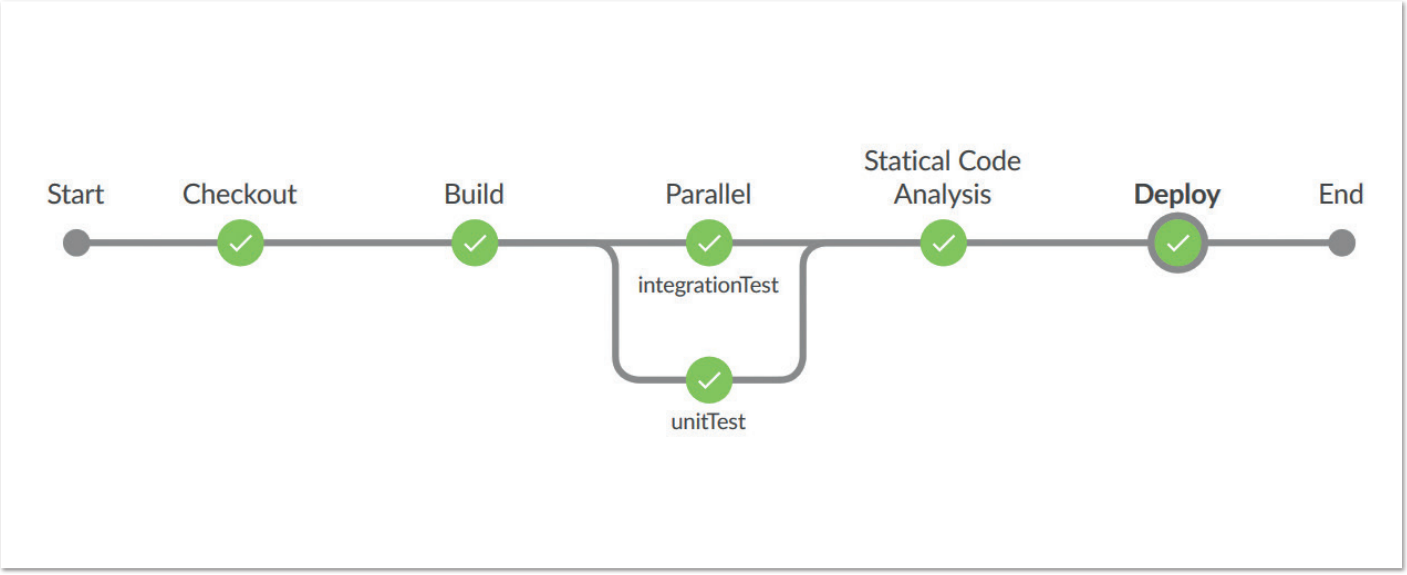


Abbildung 1: Die finale Pipeline im „Blue Ocean"-Theme

man imperativ einen Service-Account auf einen K8s-Namespace einschränkt. Für die deklarative Variante (in YAML) siehe *[1]*.

Mit diesem Service-Account kann man über das K8s-HTTP-API, „kubectl" oder „kubernetes-cd-plugin" auf den Cluster zugreifen. An das Plug-in übergibt man den Service-Account als „kubeconfig"-Datei. Diese lässt sich mit einem Script von GitHub *[10]* erstellen, wie die letzte Zeile in *Listing 4* zeigt. Die durch das Script erstellte Datei „kubeconfig" lädt man in Jenkins in ein Secret File Credential hoch, etwa mit der ID „kubeconfig-prod" *(siehe Listing 2)*. Wenn man Staging-Umgebungen in einem anderen Namespace hat, würde man an dieser Stelle weitere „kubeconfig"-Dateien für diese erstellen (wie „kubeconfig-staging" in *Listing 2)*.

Da das Anwenden der Datei auf den Cluster bei K8s nur das serverseitige Deployment anstößt, ist danach noch nicht klar, ob dieses erfolgreich war. Deshalb wird in *Listing 2* abschließend geprüft, ob die neue Version erreichbar ist. Dazu muss die Anwendung den Versions-Namen bereitstellen. Wie man dies mit Maven und REST erledigt, zeigt zum Beispiel dieser Blog-Post *[11]*. In *Listing 2* ist zu sehen, wie man den Versions-Namen abfragt und vergleicht, ob die Version mit der gewünschten übereinstimmt. Taucht sie nach einer bestimmten Zeit nicht auf, schlägt der Build fehl und die Entwickler werden von Jenkins informiert. In einem solchen Fall zahlt sich die Verwendung von K8s aus: Durch dessen Rolling-Update-Strategie bleibt die Anwendung weiterhin eingeschränkt verfügbar. In *Listing 2* ist der Hostname hart codiert. Alternativ kann man die externe IP-Adresse des Service mittels „kubectl" abfragen, siehe *[1]*. Die Umsetzung des Deployments in „declarative"-Syntax erfolgt wie in *Listing 2*, bis auf folgende Ausnahmen:

- „createVersion()" muss in einem „steps"-Block (etwa innerhalb der „build"-Stage) aufgerufen werden und schreibt sein Ergebnis nach „env.versionName", da keine Variablen im „steps"-Block möglich sind
- Die Prüfung, ob der Build noch stabil ist, kann in einer „when"-Directive erfolgen (siehe Artikel in der vorletzten Ausgabe)
- Die Prüfung des Branch in der „Deploy"-Stage muss innerhalb eines „steps"- und „script"-Blocks oder in einem eigenen Step stattfinden

Das komplette Beispiel findet man bei GitHub *[1]*. Selbstverständlich kann man Jenkins Pipelines nicht nur auf K8s einrichten; beispielsweise ist das Deployment auf die Container Orchestration Platform Docker Swarm dank des eingebauter Docker-Supports einfach *[12]*. Docker selbst kann man außerdem nutzen, um einfache Staging-Umgebungen aufzubauen, indem man einen Rechner mit Docker-Host als Jenkins-Worker einbindet und darauf aus der Pipeline die Container des Staging-Systems startet. Auch Deployments auf PaaS-Plattformen sind möglich, so gibt es für CloudFoundry ein Plug-in mit Pipeline-Unterstützung *[13]*. Außer Web-Anwendungen kann man auch andere Arten von Anwendungen kontinuierlich ausliefern. Zum Abschluss einige Anregungen aus der Praxis:

- Java-Libraries lassen sich mit wenigen Zeilen Pipeline-Code nach Maven Central einrichten. Ein Beispiel ist der „test-data-loader" [14], der die Shared Library „ces-build-lib" [6] verwendet.
- **Docs-as-Code**
  Eine in einer Markup-Sprache verfasste und im SCM gespeicherte Dokumentation kann automatisiert in ein fertiges Dokument überführt werden. Dieses Beispiel [15] zeigt, wie man aus Markdown mit Pandoc verschiedene Dokumenten-Formate

```
kubectl create namespace jenkins-ns
kubectl create serviceaccount jenkins-sa --namespace=jenkins-ns
kubectl create rolebinding jenkins-ns-admin --clusterrole=admin --namespace=jenkins-ns --serviceaccount=jenkins-
ns:jenkins-sa
./create-kubeconfig jenkins-sa --namespace=jenkins-ns > kubeconfig
```

Listing 4

wie PDF oder ODT erzeugt. Über den Pandocs-Template-Mechanismus können die Dokumente im Corporate Design gerendert werden. In diesem Beispiel wird das Build-Tool Gulp verwendet, damit man den Build auch lokal durchführen kann. Im Jenkinsfile wird die für Gulp notwendige Umgebung (node.js, yarn) in einem „yarn"-Container bereitgestellt. Darin nutzt Gulp den „cloudogu/pandoc"-Container [16] zur Dokumenten-Erstellung. Daher ergibt sich hier die im letzten Teil beschriebene „Docker in Docker"-Herausforderung. Um weitere Container aus dem „yarn"-Container zu starten, werden der Docker Socket durchgereicht und der Docker-Client installiert. Auch dies kann in wenigen Zeilen durch „ces-build-lib" [6] gelöst werden. Wenn man das Markup in einem Git-basierten Wiki wie Gollum [17] oder Smeagol [18] editiert, wird direkt bei Speicherung im Wiki durch die CD-Pipeline ein PDF ausgeliefert. Dadurch ist Docs-as-Code für Nicht-Entwickler besser zugänglich. Das funktioniert auch für Präsentationen. Dieses Beispiel [19] zeigt, wie man in Markdown Präsentationen mit „reveal.js" erstellen und in einer Maven Site (Nexus Repository) oder per Kubernetes (NGINX-Container) im Web bereitstellen kann.

- **Infrastructure as Code**
  Auch ganze virtuelle Maschinen können in der Cloud provisioniert werden, etwa mithilfe des Tools „Terraform". Damit wird beispielsweise die öffentliche Demo-Instanz des Cloudogu EcoSystem [20] mittels Blue-Green Deployment [21] täglich aktualisiert.

Fazit und Ausblick

Diese Artikelserie zeigt einige der Möglichkeiten, die das Jenkins-Pipeline-Plug-in bietet. Es kombiniert die bereits vorhandene große Auswahl an Jenkins-Plug-ins mit einer DSL zur Beschreibung von Build Jobs. So kann man diese als Code formulieren. Sie sind schneller verständlich, können im SCM verwaltet, einfacher wiederverwendet (beispielsweise durch Shared Libraries) und automatisch getestet werden. Durch Parallelisierung kann man außerdem vorhandene Ressourcen nutzen, um mit geringem Aufwand die Laufzeiten der Pipelines zu verkürzen. Die Docker-Integration ermöglicht ohne weitere Konfiguration die Benutzung von weiteren Werkzeugen und bietet die Möglichkeit, Images während des Deployments in eine Registry bereitzustellen.

Ob man Pipelines in „scripted"- oder „declarative"-Syntax beschreibt, bleibt Geschmackssache. Beim Schreiben der Beispiele für den Artikel fiel auf, dass gerade bei komplexeren Aufgaben die „declarative"-Lösung oft umständlicher, aber generell machbar war. Der Vorteil von „declarative"-Pipelines ist, dass sie besser in das „Blue Ocean"-Theme integriert sind und dort visuell editiert werden können.

Die finale Pipeline *(siehe Abbildung 1)* umfasst in beiden Varianten ungefähr 150 Zeilen. Dabei entspricht sie von der Komplexität her durchaus der eines echten Projekts.

An wenigen Stellen (wie Nightly Builds, Möglichkeiten für Unit- und Integrationstests der Pipeline) zeigt sich zwar, dass noch nicht alles perfekt ist. Dennoch ist das Pipeline-Plug-in die wichtigste Neuerung der letzten Jahre für Jenkins und sorgt dafür, dass wir den altgedienten Butler auch weiterhin für moderne Software-Entwicklung einsetzen können.

Weitere Informationen

[1] *https://github.com/triologygmbh/jenkinsfile*
[2] *https://opensource.triology.de/jenkins/job/triologygmbh-github/job/jenkinsfile/*
[3] J. v. Gizycki, Statische Code-Analyse mit SonarQube, Java aktuell 04/2017
[4] *https://docs.sonarqube.org/display/SCAN/Analyzing+with+SonarQube+Scanner+for+Jenkins*
[5] *https://jenkins.io/doc/pipeline/steps/sonar*
[6] *https://github.com/cloudogu/ces-build-lib*
[7] *https://cloud.google.com/container-registry/docs/advanced-authentication*
[8] *https://maven.apache.org/maven-ci-friendly.html*
[9] *https://plugins.jenkins.io/kubernetes-cd*
[10] *https://github.com/zlabjp/kubernetes-scripts/blob/master/create-kubeconfig*
[11] *https://www.triology.de/blog/versionsnamen-mit-maven-auslesen-des-versionsnamens*
[12] *https://jenkins.io/doc/book/pipeline/docker/#using-a-remote-docker-server*
[13] *https://plugins.jenkins.io/cloudfoundry*
[14] *https://github.com/triologygmbh/test-data-loader*
[15] *https://github.com/cloudogu/continuous-delivery-docs-example*
[16] *https://github.com/cloudogu/docker-pandoc*
[17] *https://github.com/gollum/gollum*
[18] *https://github.com/cloudogu/smeagol*
[19] *https://github.com/cloudogu/continuous-delivery-slides-example*
[20] *https://demo.cloudogu.net*
[21] *https://www.martinfowler.com/bliki/BlueGreenDeployment.html*

Johannes Schnatterer
johannes.schnatterer@triology.de

Johannes ist Software-Entwickler und Solution Architect bei der Cloudogu GmbH und dort Teil des DevOps-Teams, um das Backend des Cloudogu EcoSystem zu betreiben. Er ist Continuous-Delivery-Fan, fokussiert auf Software-Qualität, hat einen ausgeprägten Open-Source-Enthusiasmus und ist überzeugt, dass prägnante Dokumentation entscheidend sein kann.

Functional Load Testing mit Gatling

Gerald Mücke, DevCon5 GmbH (Schweiz)

Die Hauptaufgabe im Testen besteht darin, Informationen über ein Software-System zu gewinnen; dazu gehören auch Last- und Performance-Tests. Mit funktionaler Programmierung können wir Ereignisse und Szenarien modellieren, die mit klassischen Ansätzen gar nicht oder nur schwer abbildbar sind. Dieser Artikel beschreibt diese Szenarien und zeigt die Möglichkeiten von Gatling auf, sie funktional zu testen.

Für viele Entwickler besteht Testen in erster Linie aus dem Schreiben automatisierter Unit-Tests. Dies sind jedoch keine Tests im eigentlichen Sinne, denn sie prüfen nur bekannte Bedingungen und Ereignisse ab und werden daher auch „Checks" genannt, die verhindern sollen, dass bekannte oder vorhersehbare Bugs (wieder) auftreten. Testen im eigentlichen Sinne hat daher auch nicht zum Ziel, Bugs zu finden, sondern vielmehr, Informationen über das zu testende System zu gewinnen. Nicht jedes Verhalten ist spezifiziert, nicht jedes unspezifizierte Verhalten ist ein Bug. Ein Bug ist etwas, das jemanden stört, der genug Einfluss hat – Einfluss, eine Entscheidung zu treffen. Informationen sind wiederum essenziell für eine fundierte Entscheidung.

Ein Teilgebiet des Testens ist der Last- und Performance-Test, dessen Zweck es ist, Informationen über die Leistungsfähigkeit eines Software-Systems zu gewinnen. Dabei unterscheidet man zwischen verschiedenen Arten mit jeweils unterschiedlicher Zielsetzung:

- **Soak Testing**
  Ein lang laufender Test mit moderater Last, der vor allem dazu dient, Ressourcen-Lecks wie Memory Leaks zu finden.
- **Stress Testing**
  Ein Test mit Hoch- oder Überlast, um das Verhalten eines Systems im Grenzbereich seiner Kapazität zu ermitteln.
- **Benchmarking**
  Ein Test, um die Unterschiede zwischen Versionen zu erkennen. Dabei geht es vor allem um die Leistungsmessung.

Diese etablierten Praktiken helfen dabei, Informationen über Robustheit, Stabilität oder Leistungsfähigkeit zu gewinnen. Sie sind aber auch ein Werkzeug für eine gezielte Leistungsanalyse mit dem Ziel, Engpässe zu finden und die Leistung des Systems zu verbessern.

Anatomie eines Last-Tests

Betrachten wir zunächst einmal, wie man einen typischen Last- und Performance-Test entwickelt. Der Last-Test hat drei Hauptbestandteile:

- Seiten-Skripte mit einzelnen Requests pro Seite definieren. Diese lassen sich mit einem Recorder aufnehmen und nach Bedarf nachbearbeiten oder manuell entwickeln.
- User-Szenarien, bestehend aus einer Abfolge von Seiten, mit Verzweigungswahrscheinlichkeiten der Benutzerströme inklusive Eintritts- und Austrittspunkte. Diese Navigationspfade sind oft nicht klar definiert und müssen in Zusammenarbeit mit verschiedenen Stakeholdern ermittelt werden. Das Ziel ist, die Be-

nutzerströme so realistisch wie möglich zu definieren, um möglichst verschiedene Teile des Systems zu belasten.
- Lastprofile definieren die Verteilung der Benutzer über die Zeit, die nachfolgend genauer beschrieben sind.

Für das Lastprofil sind die Qualitätsanforderungen an das System erforderlich. Dabei helfen Anforderungsdokumente oder aber auch historische Daten und Prognosen. Manchmal ist es nötig, verschiedene Stakeholder zu interviewen, um genügend detaillierte Anforderungen zu bekommen.

In einem der Projekte des Autors lautete die spezifizierte Anforderung: „Das System muss in der Lage sein, tausend „concurrent user" mit einer durchschnittlichen Antwortzeit von eineinhalb Sekunden zu bedienen". Das Wesentliche an dieser Anforderung sind nicht die Zahlen, sondern der Kontext, in dem diese Zahlen verwendet werden: „concurrent user" und „durchschnittlich". Wie sich auf Nachfrage herausstellte, waren damit „tausend User innerhalb einer Stunde" gemeint und der Durchschnitt bezog sich auf den „wahrgenommenen Durchschnitt", der in der Literatur häufig der Neunzig-Prozent-Perzentilen entspricht *[1]*, also haben neunzig Prozent der Benutzer eine Antwortzeit von unter eineinhalb Sekunden. Diese Angaben deckten sich wiederum mit historischen Daten.

Anhand dieser Daten würde man nun klassisch ein Szenario entwerfen, bei dem eine Anzahl virtueller User sich durch die Applikation klickt, um das System hinreichend zu durchdringen. Die virtuellen User haben eine bestimmte „Think Time", also eine künstliche Wartezeit zwischen den Requests. Anhand dieser lassen sich virtuelle in reale User umrechnen.

Beim genannten Projekt lag beispielsweise die durchschnittliche Sessionlänge bei 210 Sekunden, mit vier Page Requests pro Session. Damit lag die reale Think Time bei 52 Sekunden. Mit einer Think Time von fünf Sekunden bilden hundert virtuelle User tausend reale User ab. Ein Thread-basierter Lastgenerator wie JMeter benötigte also hundert Threads.

Diese hundert virtuellen User würden so auf eine Stunde verteilt, dass sie das System kontinuierlich beschäftigen. Der Test würde durch eine Ramp-up-Phase eingeleitet, der sowohl die User gleichmäßig verteilt als auch das System aufwärmt. Zweck des Aufwärmens ist es, den Einfluss von nicht-linearem Systemverhalten auf die Messung zu reduzieren, also Verhalten, das sich nicht proportional zur anliegenden Last verändert. Dazu zählen folgende Punkte:

- Just-in-time-Kompilierung und Optimierung
- Befüllen von Caches
- Ressourcen-Initialisierung wie das Laden von Ressourcen oder der Aufbau von Datenbank-Verbindungen
- Füllen von Queues und Puffern auf ein stabiles Niveau

Hat sich das System auf einen stabilen Zustand eingependelt, kann gemessen werden. *Abbildung 1* zeigt die Messung eines elastischen Systems mit konstanter Last. Deutlich zu sehen ist zum einen, wie das System während des Ramp-up schlechtere Minimal-Werte hat, sich also erstmal aufwärmen muss. Auch zu sehen ist, dass das System erst ab etwa vierzig Prozent der Gesamtlaufzeit in einem stabilen Zustand ist, da das System auf die initial steigende Last mit mehreren trägen Skalierungsvorgängen reagiert. Aus Sicht des Endanwenders – und hier kommt der Tester als dessen Advokat ins Spiel – besitzen die Mess-Ergebnisse jedoch nur begrenzte Relevanz.

Im Zeitalter der Cloud sind viele Applikationen als elastische Systeme konzipiert, sie werden also unter Last durch Zufügen weiterer Ressourcen hoch- beziehungsweise bei sinkender Last herunterskaliert, um Kosten und User Experience in einem ausgewogenen Verhältnis zu halten. Ansätze wie Serverless-Computing besitzen sogar keine dedizierten Server mehr und müssen unter Umständen viel häufiger mit einem Kaltstart, also einem nicht-optimierten System zurechtkommen.

Ein weiterer Faktor ist das Nutzerverhalten selbst. Durch die hohe Durchdringung des Internets im öffentlichen Leben ist die potenzielle Zielgruppe sehr groß und vielschichtig und damit schwerer vorauszusagen. Einzelne Ereignisse wie der Black-Friday-Sale oder Online-Werbekampagnen werden durch soziale Medien verstärkt und sorgen für schwer planbare Belastungsspitzen. Aus Perspektive eines Testers lässt sich mit dem klassischen Vorgehen nur schwer die Frage beantworten, wie sich das System in solchen extremen Grenzfällen verhält und wie sich das auf die User Experience auswirkt. Anders formuliert: Wie testet man das nichtlineare Verhalten?

Auftritt der Statistiker

Zur Abbildung von Verteilungen unabhängiger und unbekannter Ereignisse liefert die Statistik eine Reihe von Verteilungsfunktionen. Die bekannteste ist die Gauß'sche Normalverteilung *[3]*, die Wahrscheinlichkeitsverteilung unabhängiger Ereignisse. Mit deren Hilfe, die mit der in *Abbildung 2* gezeigten Formel berechnet wird, lässt sich eine realitätsnahe Last über einen längeren Zeitraum abbilden,

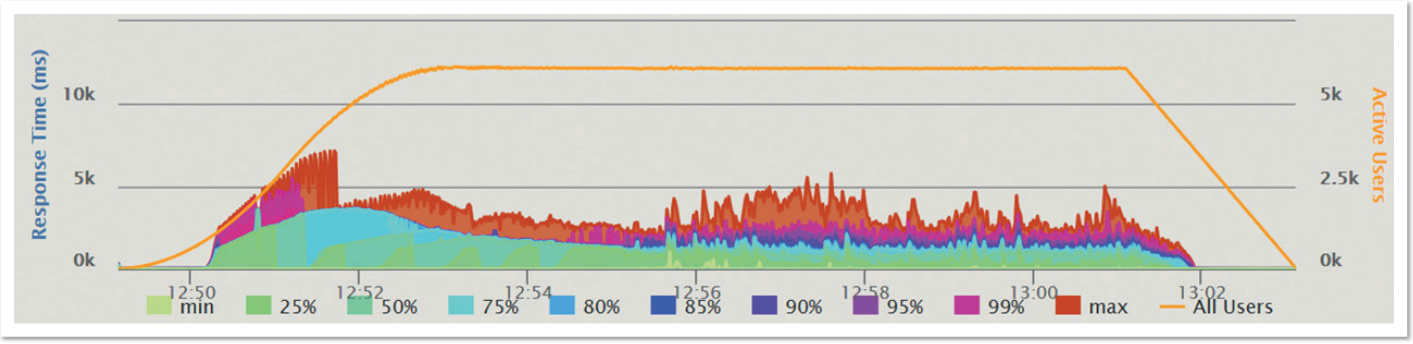


Abbildung 1: Beispiel Messung mit Ramp-up und konstanter Last

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Abbildung 2: Die Gauß'sche Normalverteilungsfunktion

$$B(k|p,n) = \binom{n}{k} p^k (1-p)^{n-k}$$

Abbildung 3: Die Binomial-Verteilung

$$P_\lambda(k) = \frac{\lambda^k}{k!} e^{-\lambda}$$

Abbildung 4: Die Poisson-Verteilung

```
val page1 = exec(
 http("request_0")
  .get("/")
  .headers(acceptHtml)
  .resources(...))
```

Listing 1: Seitenskript

```
val scn = scenario("Simple").exec(page1)
```

Listing 2: Szenario

```
scn.inject(
  rampUsersPerSecond(10) to 500 during (5 minutes)
  constantUsersPerSec(500) during (10 minutes)
  )
```

Listing 3: Lastprofil

dessen tatsächliche Verteilung unbekannt ist. Als Beispiel sei hier auf die eingangs erwähnte Anforderung von tausend Benutzern pro Stunde verwiesen.

Neben der Normalverteilung sind die Binomial-Verteilung *[4] (siehe Abbildung 3)* sowie die Poisson-Verteilung *[5] (siehe Abbildung 4)*, die einen Grenzfall der Binomial-Verteilung darstellt, als diskrete Funktionen zur Abbildung unabhängiger Zufallsereignisse geeignet. Die Poisson-Verteilung wird dabei häufig zur Modellierung von Ankunftsraten in Warteschlangen-Systemen verwendet, zu denen auch Computersysteme gehören.

Diese statistischen Verteilungen setzen allerdings voraus, dass jedes Ereignis, also die anzutreffenden Benutzer, unabhängig voneinander eintritt. Mit Thread-basierten Lastgeneratoren wie JMeter lassen sich derartige Szenarien nur schwer abbilden. Zum einen sind die Konfigurationsmöglichkeiten sich kontinuierlich verändernder Lasten zu begrenzt, zum anderen leiden Thread-basierte Lastgeneratoren an der sogenannten „coordinated omission" (koordinierte Auslassung); Lastgenerator und das zu testende System bilden also über eine Rückkopplung einen geschlossenen Kreislauf. So sind die Requests, die ein Lastgenerator-Thread für einen virtuellen User absetzen kann, von den Antworten des vorhergehenden virtuellen Users abhängig. Für die Abbildung statistisch unabhängiger Ereignisse ist diese Art der Lastgenerierung also nicht geeignet.

Gatling

Ganz anders ist die Situation mit Gatling. Es basiert auf Event-getriebener, asynchroner Lastgenerierung. Jeder Request wird unabhängig von vorherigen Ereignissen von einem Timer ausgelöst und es besteht keine Rückkopplung des zu testenden Systems auf den Lastgenerator. Damit lassen sich unabhängige Ereignisse gemäß den beschriebenen statistischen Verteilungsfunktionen, aber auch anderen Funktionen, abbilden.

Ein weiterer wesentlicher Vorteil von Gatling ist, dass die Szenarien in einer Programmiersprache – im Fall von Gatling in Scala – geschrieben sind. Die drei eingangs beschriebenen Komponenten eines Last-Tests sind in Gatling:

- Seiten-Skripte, die eine Abfolge von Requests beschreiben *(siehe Listing 1)*
- Szenarien, bestehend aus seiner Abfolge von Seiten *(siehe Listing 2)*
- Lastprofile, die als Abfolge von „InjectionSteps" definiert werden *(siehe Listing 3)*

```
def continuousUserRate(
duration: FiniteDuration,
     totalUsers: Int,
     distrFun: (Int) => (Double => Double),
     stepFun: (Duration) => (Int) = (d) => d.toMinutes.toInt):
List[InjectionStep] = {
  val steps = stepFun(duration)
  val stepDuration = duration / steps
  def fun(x: Double) : Double =
       totalUsers * distrFun(steps)(x) / stepDuration.toSeconds
  List.range(0, steps).map(
       step => rampUsersPerSec(fun(step)) to fun(step + 1) during stepDuration)
}
```

Listing 4: Generierung der „InjectionSteps"

„InjectionSteps“ im Lastprofil beschreiben, wie viele User in welchem Zeitrahmen aktiv sind und dem Szenario folgen. Gatling bietet eine Reihe von Basis-Steps an. Für klassische Lastszenarien sind das unter anderen:

- **rampUsersPerSecond**
  Steigert die Ankunftsrate linear während eines bestimmten Zeitraums
- **constantUsersPerSecond**
  erzeugt in regelmäßigen Abständen neue Benutzer, die am System eintreffen

„InjectionSteps“ werden als Liste an das Szenario übergeben. Diese kann aber auch das Ergebnis eines Funktionsaufrufs wie „def f()= _ => List[InjectionStep]“ sein. Auf diesem Wege lassen sich aus den beschriebenen mathematischen Funktionen Lastprofile für Gatling generieren.

Da Gatling von Hause aus keine funktionsbasierte Lastgenerierung anbietet, sind die mathematischen Funktionen durch ein Mapping auf die Basisschritte anzunähern. Naheliegend ist hier eine Abfolge von „RampUserPerSecond“-Steps, was hinreichend genau für einen Last-Test ist. Die Funktion in *Listing 4* übernimmt das Erzeugen dieser Liste für einen definierten Zeitraum und eine bestimmte Anzahl Gesamtbenutzer. Der dritte Parameter ist die Verteilungsfunktion für eine spezifische Anzahl von Schritten. Der optionale vierte Parameter bestimmt die Länge der linearen Schritte, wobei der Default-Wert bei einminütigen Schritten liegt.

Mit der Funktion aus *Listing 4* lässt sich aus einer beliebigen mathematischen Funktion, wie der Gauß-Verteilungsfunktion aus *Listing 5*, eine Liste von „InjectionSteps“ erzeugen und auf ein Gatling-Szenario anwenden, wie *Listing 6* zeigt.

Beispiel-Messungen

Abbildung 5 zeigt das gleiche System, das in *Abbildung 1* mit konstanter Last getestet wurde, mit einer normalverteilten Last. Bei einer effektiv geringeren Zahl von Usern sind die Antwortzeiten höher; insbesondere im Bereich der höheren Last zur Mitte des Tests steigen die minimalen Antwortzeiten mit der Last weiter an. Dementsprechend sind auch die Perzentilen der Antwortzeiten, die für die Definition sinnvoller Anforderungen oder SLAs wichtig sind, höher als bei dem Szenario mit konstanter Last.

Wiederum deutlich verschärfter sieht die Situation bei der Binomial-Verteilung aus *(siehe Abbildung 6)*. Die Gesamtzahl der Nutzer ist gleich wie bei *Abbildung 1*, jedoch sind die Auswirkungen einer schnell steigenden Ankunftsrate gravierender. Die einzelnen Zacken bei den hohen Perzentilen sind das Ergebnis von Garbage Collections, die sich zur Lastspitze zu Timeouts für den User aufschaukeln. Deutlich sind auch die Auswirkungen des Scale-Downs auf die Antwortzeiten ab der zweiten Hälfte zu erkennen.

Fazit

Mit dem eventbasierten Lastgenerator Gatling lassen sich mit mathematischen Funktionen Lastprofile definieren, die näher an realen Situationen liegen. Die dadurch gewonnenen Erkenntnisse

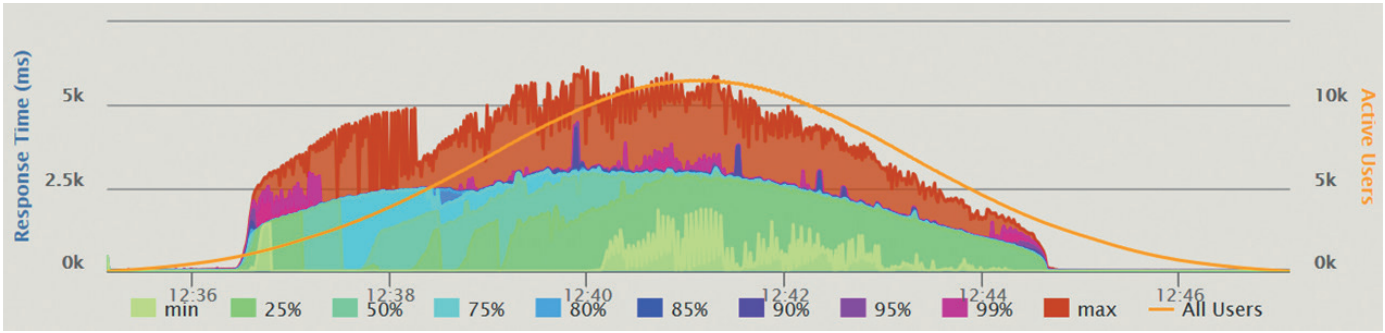


Abbildung 5: Antwortzeiten bei Gaußverteilung

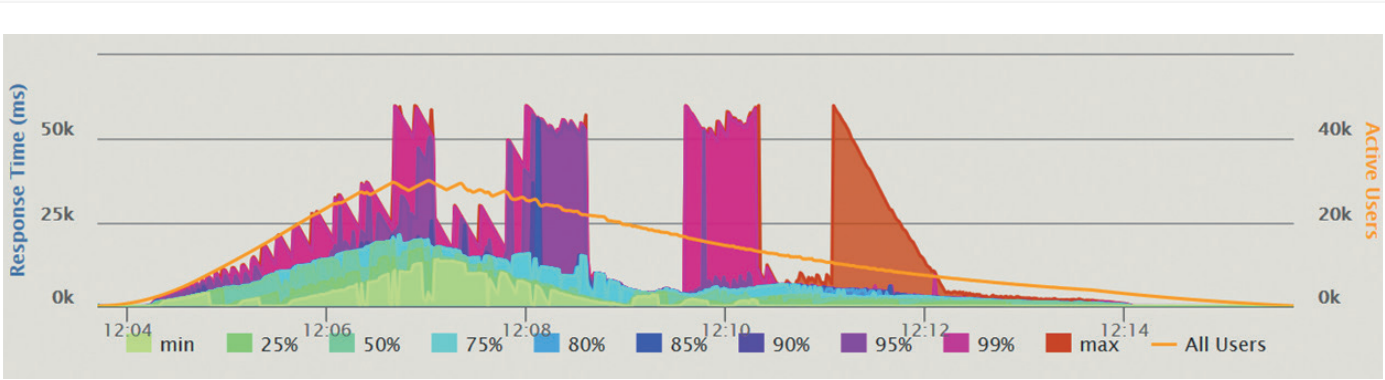


Abbildung 6: Antwortzeiten bei Binomial-Verteilung

bieten bessere Rückschlüsse auf die zu erwartende User Experience unter Last als mit herkömmlichen Methoden zur Modellierung konstanter Lasten. Außerdem können mit statistischen Verteilungsfunktionen Szenarien zum Testen elastischer Systeme entwickelt werden.

Wie alle Testpraktiken hat auch das funktionale Load Testing seine Grenzen und ersetzt daher nicht das klassische Vorgehen, sondern ergänzt es vielmehr.

```
def gauss(sigma: Double, mu: Double)(x: Double): Double
= {
  (1 / (sigma * sqrt(2 * Pi)))
  * exp(-0.5 * pow((x - mu) / sigma, 2))
}
```

Listing 5: Die Gauß-Verteilungsfunktion in Scala

```
def gaussDistr(sigma: Double = 2, muPercent: Double = 0.5)
              (duration: FiniteDuration, totalUsers: Int) =
  continuousUserRate(duration, totalUsers,
                     steps => gauss(sigma, steps * muPercent))

setUp(
  ExampleScenario.helloWorld
                 .inject(
    gaussDistr()(10 minutes, 30000)
  )).protocols(httpServer)
```

Listing 6: Erzeugung der InjectionSteps mithilfe der Gauß-Funktion

Weitere Informationen

[1] Java Performance, The Definitive Guide, Scott Oaks, 2014
[2] User Experience, Not Metrics, Scott Barber, 2006:
http://www.perftestplus.com/resources/UENM4.pdf
[3] *https://de.wikipedia.org/wiki/Normalverteilung*
[4] *https://de.wikipedia.org/wiki/Binomialverteilung*
[5] *https://de.wikipedia.org/wiki/Poisson-Verteilung*
[6] *https://gatling.io/docs/2.3/general/simulation_setup*

Gerald Mücke
gerald.muecke@devcon5.ch

Gerald Mücke ist ein Java-Begeisterter mit mehr als zwölf Jahren Branchenerfahrung. Er begann seine Karriere als Entwickler von Testautomatisierungs-Werkzeugen für einen der weltweit führenden Hersteller von Enterprise-Software. Nachdem er in verschiedenen Funktionen, Branchen und Projekten gearbeitet hatte, bietet er heute als unabhängiger Berater mit seiner eigenen Firma DevCon5 (CH) Beratungs-Dienstleistungen in den Bereichen Software-Entwicklung und -Test an. Er interessiert sich insbesondere für Test-Automation, Performance-Testing und DevOps.

WebAssembly – ein Jahr danach

Mirko Sertic, Thalia Bücher

WebAssembly wurde Anfang 2017 in Form eines Minimum Viable Product (MVP) in der Version 1.0 veröffentlicht. Dieser Artikel gibt einen kurzen Rückblick auf die Entstehungsgeschichte, zeigt den aktuellen Stand der Implementierung und bringt einen Ausblick auf mögliche Anwendungsfälle sowie zukünftige Erweiterungen.

Wir schreiben das Jahr 2018. Das Internet ist allgegenwärtig. Als Anwender und Entwickler steht uns eine Vielzahl unterschiedlichster Technologien für die Entwicklung moderner Web-Anwendungen zur Verfügung. Es gibt den HTML5-Standard. Wir haben CSS. Wir nutzen IndexedDB oder auch den LocalStore im Browser. Mit diesen Technologien lassen sich schnell und effizient Online- oder auch Offline-Anwendungen bauen und verteilen. Seit Anfang 2017 gibt es in diesem Universum neu WebAssembly. Um WebAssembly und dessen Bedeutung genauer zu verstehen, müssen wir jedoch eine kleine Zeitreise in die Vergangenheit unternehmen.

Wie alles begann

Im Jahr 1995 wurde ein wichtiger Meilenstein für das gesamte Internet gelegt. Als Ergebnis des Projekts „Mocha" stand zusammen mit dem Netscape Navigator Version 2.02 eine Skript-Sprache für dynamische HTML-Seiten der breiten Öffentlichkeit zur Verfügung. Diese Skript-Sprache hatte den Namen „Netscape LiveWire" und wurde später in „JavaScript" umbenannt. Damit war ein Fundament geschaffen, auf dem noch heute unsere Anwendungen aufbauen.

Die Adaption von JavaScript verlief anfangs eher zögerlich. Hintergrund war der laufende Prozess der Sprach-Normierung und -Weiterentwicklung gepaart mit den Browser-Kriegen der damaligen Zeit. Es gab noch nicht viele Standards beziehungsweise die Standards wurden sehr hart erkämpft.

Im Jahr 2005 erwähnte James Garret in seinem Aufsatz das erste Mal das Wort „AJAX". Nun war es möglich, die dynamischen Inhalte im Browser auch mit Backend-Logik zu koppeln, um noch interaktivere Anwendungen zu bauen. JavaScript wurde immer beliebter.

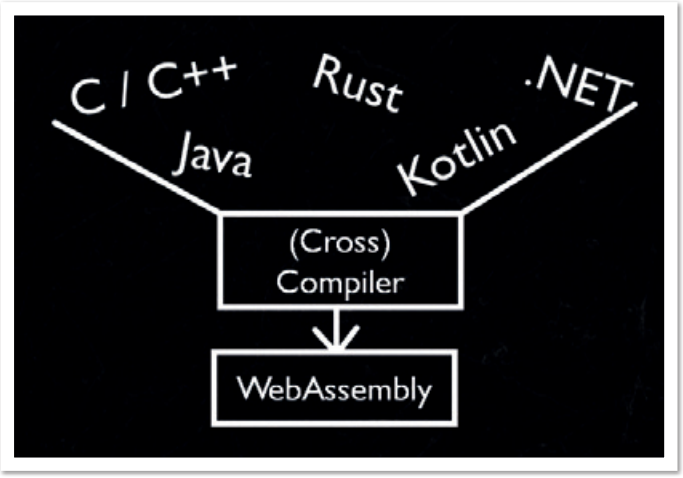


Abbildung 1: WebAssembly ist ein Compile-Target

Neue Horizonte und Möglichkeiten

Wir spulen die Zeit nun etwas vor bis in das Jahr 2010. Das Internet ist überall; auf dem Desktop; auf unseren Smartphones; auf embedded Devices; in der Gebäude-Automatisierung. JavaScript ist überall. Diese standardisierte Plattform ist eine ideale Grundlage für weitere Anwendungsfälle. In diesem Jahr betritt ein sehr interessantes Projekt die Bühne: Emscripten, im Wesentlichen ein C- beziehungsweise C++-Compiler. Dieser erzeugt jedoch keinen Binärcode in Form von X86-Instruktionen, sondern er übersetzt das Programm nach JavaScript. Somit sind völlig neue Deployment-Szenarien für bestehende C++-Anwendungen möglich.

Der Early-Adopter von Emscripten war die Entertainment-Branche, darunter große Spiele-Studios wie Epic. Durch den Einsatz von Emscripten war es möglich, die Unreal Game Engine sowie bestehende Desktop-Spiele mit minimalem Aufwand auch im Browser laufen zu lassen – aus wirtschaftlicher Sicht ein gigantischer Sprung. Die Unreal Engine komplett in JavaScript neu zu schreiben, hätte sicherlich ein großes Team über mehrere Monate bis Jahre ausgelastet. Emscripten als Compiler hat das deutlich optimiert. Ergebnis war die Unreal-Cathedral-Demo, die im Browser lief. Als Laufzeitgeschwindigkeit wurde die halbe Framerate im Vergleich zu der Desktop-Variante erreicht. Das Spiel war spielbar, jedoch gab es noch deutlich Luft nach oben.

Es geht noch schneller

Nach intensiver Zusammenarbeit zwischen Mozilla, Epic und dem Emscripten-Team wurde im Jahr 2013 „asm.js" präsentiert, eine Untermenge von JavaScript, die auf Laufzeit-Effizienz ausgelegt ist. Durch die Annotation von Ausdrücken ist es möglich, der JavaScript-Runtime Hinweise zu geben, um den Code effizienter auszuführen. Diese Typisierung von JavaScript ermöglicht es, Integer- und Fließkomma-Datentypen genauer zu unterscheiden und somit gerade die Integer-Berechnung deutlich zu verbessern. Diese Unterscheidung ist in JavaScript nicht möglich, da es hier nur Fließkomma-Zahlen gibt. Emscripten wurde mit „asm.js" als Compile-Target erweitert. Ergebnis war die Unreal Engine mit der Cathedral-Demo und annähernd nativer Geschwindigkeit im Browser.

Zu diesem Zeitpunkt gab es Browser mit sehr guten JavaScript-Just-in-time-Compilern. Durch „asm.js" wurden noch zusätzliche Hinweise für die Optimierung des Laufzeit-Verhaltens gegeben. Insgesamt war das Ergebnis im Sinne der Laufzeit sehr ansehnlich. Jedoch hatte das Ganze auch seine Schattenseite. Die Code-Basen wurden immer größer, der Quellcode immer komplexer.

Zusammen mit der Größe und der Komplexität stieg natürlich auch die Download-Zeit für Anwendungen deutlich an; auch die Parsing-Zeit für JavaScript im Browser. Als Ergebnis war die Time-to-Interactive bei sehr großen Anwendungen für spontane Internet-User nur noch bedingt akzeptabel. Die Suche nach einem Werkzeug begann, das sowohl das Laufzeit-Verhalten als auch die Download- und Parsing-Zeiten optimiert.

Die Entstehung von WebAssembly

Im Jahr 2015 wurde die WebAssembly Working Group als Reaktion auf die genannte Fragestellung gegründet. Das Ergebnis ihrer Arbeit war ein Format, das unterschiedlichste Anforderungen unter einen Hut bringt. WebAssembly ist Bytecode für das Web. Es handelt sich hier um ein binäres Format für transportable Programme, das auf Download-Größe sowie Ausführungsgeschwindigkeit optimiert ist.

Interessant an diesem Format ist, dass es in Host-Umgebungen integrierbar ist. Als primäre Host-Umgebung steht natürlich der Browser im Vordergrund. Es ist jedoch auch denkbar, WebAssembly im Backend zu nutzen. Dazu ist nur eine Host-Umgebung erforderlich, die den WebAssembly-Standard implementiert. Die Browser-Host-Umgebung setzt auf der bestehenden JavaScript-Runtime im Browser auf. Durch diese Integration wird die bestehende Just-in-time-Compiler-Mechanik im Browser wiederverwendet. Für die Browser-Hersteller ist es so wesentlich einfacher, WebAssembly zu integrieren.

WebAssembly 1.0 MVP

Anfang 2017 war es dann soweit. WebAssembly wurde in Form eines Minimum Viable Product in Version 1.0 veröffentlicht. Bemerkenswert war, dass alle Browser-Hersteller im Abstand von nur wenigen Tagen den WebAssembly-Support ausrollten. Was ist nun Teil des MVP? Schon im initialen Design wurden bewusst Stolpersteine ausgeklammert, um die erste Version des Produktes schlank zu halten. Allerdings wurde gleichzeitig dafür gesorgt, dass diese Abgrenzung keine größeren Nachteile verursacht.

Beispielhaft beinhaltet die WebAssembly-Host-Umgebung eine Sandbox, in der das Programm abgeschottet läuft. Sie ermöglicht den Zugriff auf einen linearen Speicherbereich. Dieser ist jedoch „nicht managed". Der WebAssembly-Autor muss also das Speichermanagement und gegebenenfalls auch einen Garbage-Collector selbst implementieren. Dies ist jedoch kein größerer Nachteil, da WebAssembly ein Modul-System für dynamisches Linken beinhaltet. Damit lässt sich ein bestehender Memory-Manager einfach in das WebAssembly linken, um darauf aufzubauen.

Ebenfalls wurden im MVP bewusst kein DOM-Zugriff oder die „opaque"-Data-Types implementiert. Es ist aus der WebAssembly-Sandbox nicht direkt möglich, auf das DOM oder Browser-APIs zuzugreifen. Auch dies ist kein größerer Nachteil, da über das Modul-System der Zugriff durch Wrapper-Types emuliert werden kann. Diese verursachen zwar gewisse Laufzeit-Nachteile, was jedoch im Vergleich mit der reduzierten Komplexität zu verkraften ist.

Im MVP wurde bewusst auf Unterstützung für Threading und Thread-Synchronisierung verzichtet. Dies ist einleuchtend, da die WebAssembly-Host-Umgebung auf der JavaScript-Integration im Browser aufbaut und es in JavaScript keine Threads und keine Thread-Synchronisierung gibt.

Dieser reduzierte Funktionsumfang ist jedoch kein Widerspruch. Durch die bewusste Ausklammerung dieser Funktionalitäten und gleichzeitige Unterstützung eines Modul-Systems ist eine konsistente, nutzbare und vor allem erweiterbare WebAssembly-Version 1.0 entstanden.

Kernkonzepte und Bootstrap

Genug der Theorie. Wie funktioniert nun WebAssembly? Es kommt in zwei verschiedenen Ausprägungen vor, der textuellen Repräsentation und der binären Repräsentation. Die Textform (WAT) ist für uns Menschen gedacht und soll vor allem beim Debuggen unterstützen. Es ist jedoch nicht besonders effizient für einen Computer lesbar. Deshalb gibt es das Binärformat (WASM), das nur für den Computer gedacht und auf besonders effiziente Verarbeitung optimiert ist.

Beide Formate sind sehr Hardware-nah. Als Entwickler könnten wir solche Programme schreiben, was allerdings sehr ineffizient wäre. Der eigentliche Zweck von WebAssembly ist ein anderer: Entwickler schreiben Programme in einer Hochsprache wie C++, Rust, .NET, Java oder Kotlin. Ein Compiler übersetzt diese Sprache dann nach WebAssembly. WebAssembly ist also primär ein Compile-Target für eine Hochsprache unserer Wahl. Wir haben somit das Beste aus zwei Welten: hohe Produktivität in einer bekannten Hochsprache und maximale Laufzeit-Optimierung durch Einsatz eines optimierten Binärformats *(siehe Abbildung 1)*.

WebAssembly kennt sogenannte „Module" und „Instanzen". Ein Modul ist eine Art Schablone, aus der mehrere Instanzen erzeugt werden. Jede Instanz bekommt ihre eigene Sandbox und ist darin ausführbar. Das WebAssembly-Modul hat eine Liste von Imports und Exports. Ein Export ist eine Funktion, die von außerhalb der WebAssembly-Instanz aufgerufen werden kann, also aus der WebAssembly-Host-Umgebung. Der naheliegendste Anwendungsfall ist hier der Aufruf der Main Function des Programms.

Interessant sind die Imports. Ein Import ist eine Funktion, die von innerhalb der WebAssembly-Instanz aufgerufen werden kann, jedoch nicht in der Instanz definiert ist. Importierte Funktionen sind beispielsweise für die Interaktion mit der WebAssembly-Host-Umgebung erforderlich. Jede Form von I/O wie eine Konsolen-Ausgabe oder auch ein DOM-Zugriff wird üblicherweise in Form von Imports in WebAssembly gelinkt *(siehe Abbildung 2)*.

Um eine lauffähige WebAssembly-Instanz zu erhalten, ist eine Bootstrap-Sequenz auf dem WebAssembly-Host zu durchlaufen. Im Browser gibt es dafür ein JavaScript-API. Im ersten Schritt muss die binäre WebAssembly-Repräsentation geladen werden, was zum Beispiel über einen XML-HTTP-Request erfolgen kann. Wichtig ist, dass wir kein JSON- oder XML-Dokument als Antwort erwarten, sondern einen Array-Buffer mit den geladenen Binärdaten *(siehe Listing 1)*.

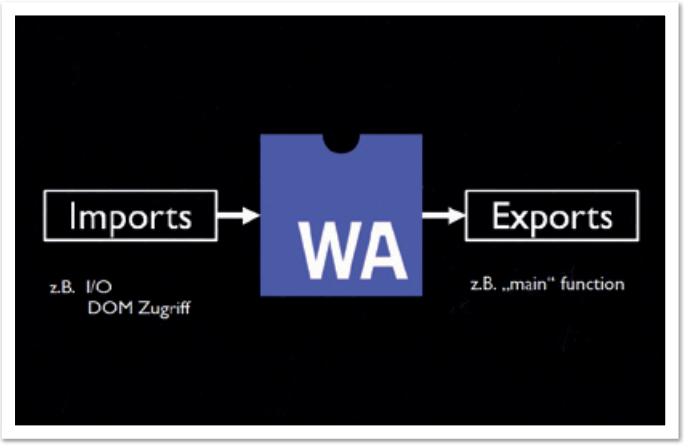


Abbildung 2: Das WebAssembly-Modulsystem

Anschließend wird via WebAssembly-JavaScript-API die Instanziierung gestartet. An diesem Punkt sind die Imports für das WebAssembly-Modulsystem anzugeben. Ein fehlender Import führt zum Abbruch der Instanziierung *(siehe Listing 2)*.

„WebAssembly.instantiate“ liefert ein Promise zurück. Bei der Instanziierung wird die WebAssembly-binäre Repräsentation validiert und kompiliert. Um den Main-Thread im Browser nicht zu blockieren, findet dies asynchron statt. Sobald dieser Vorgang abgeschlossen ist, wird das Promise mit den Referenzen auf das WebAssembly-Modul und die WebAssembly-Instanz erfüllt (fulfilled, siehe *Listing 3)*. Jetzt lässt sich der Kontrollfluss an die WebAssembly-Instanz übergeben, um etwa eine exportierte Main Function aufzurufen.

Lasst uns spielen

So weit, so gut. Welche Möglichkeiten bestehen, um erste Erfahrungen mit WebAssembly zu machen? Zum Glück gibt es da eine sehr interessante Webseite: WebAssembly Studio *(siehe „https://github.com/wasdk/WebAssemblyStudio“).* WebAssembly Studio ist „JSFiddle“ für WebAssembly. Über diese IDE kann sehr einfach ein C- oder Rust-Projekt angelegt und dieses dann nach WebAssembly kompiliert werden. Der JavaScript-Code für die Bootstrap-Sequenz wird automatisch generiert und lässt sich auch nachträglich editieren. Es ist auch möglich, die WebAssembly-Text- oder -Binärform genauer unter die Lupe zu nehmen und so tiefer in die Funktionsweise abzutauchen *(siehe Abbildung 3)*.

Wer nicht direkt Quellcode schreiben möchte, sondern eher eine lauffähige Anwendung benötigt, kann das Unity-Tanks-Demo spielen. Dabei handelt es sich um ein kleines Spiel, in dem Panzer in einer Landschaft bewegt werden können. Die Grafik ist sehr schön über WebGL umgesetzt. Hier zeigt sich sehr gut, wie WebAssembly bereits heute in Lösungen wie die Unity-Game-Engine integriert ist *(siehe Abbildung 4)*.

Andere Anwendungsfälle

Es gibt natürlich neben dem Gaming-Bereich auch weitere Anwendungsfälle. Einer davon ist „PocketSpinx.js“, ein System zur Spracherkennung. Es wurde via Emscripten nach WebAssembly übersetzt. Damit ist es möglich, Spracherkennung in Applikationen zu nutzen, ohne Remote-Services wie Alexa in eine Anwendung zu integrieren.

```
var request = new XMLHttpRequest();
request.open('GET', 'bytecoder.wasm');
request.responseType = 'arraybuffer';
request.send();
```

Listing 1

```
request.onload = function() {
  var bytes = request.response;

  WebAssembly.instantiate(bytes, {
     // Imports
     mymodule: {
add: function(a, b) {
return a + b;
}
     }
  }
    );
};
```

Listing 2

```
WebAssembly.instantiate(… ).then(function(result) {

  // Zugriff auf Modul und Instanz
  var wasmModule = result.module;
  var runningInstance = result.instance;

  // exportierte Funktion aufrufen
  runningInstance.exports.main();
});
```

Listing 3

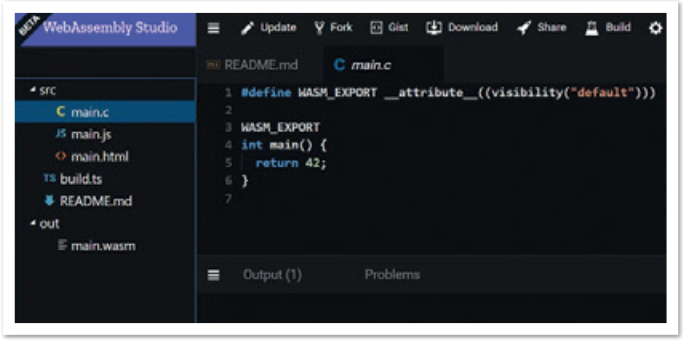


Abbildung 3: WebAssembly Studio

Abbildung 4: Unity-Tanks-Demo

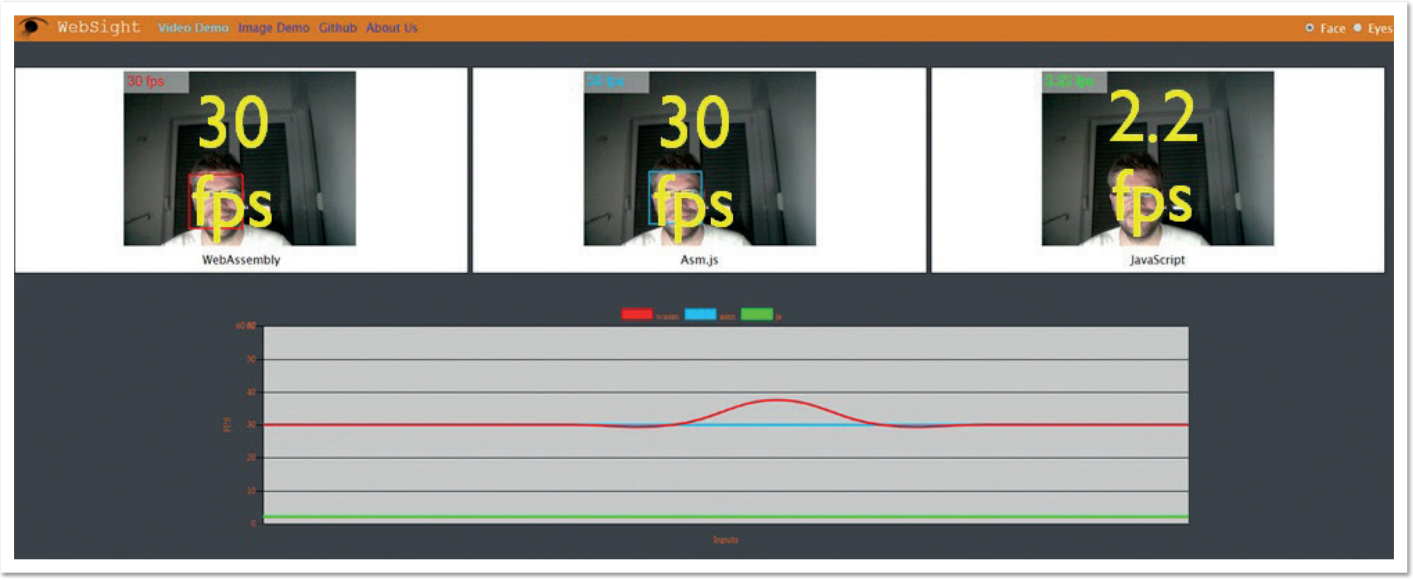


Abbildung 5: WebSight-Demo

„WebSight" ist ein anderer Anwendungsfall. Hier wurde OpenCV via Emscripten nach WebAssembly übersetzt. Diese Demo greift auf die Webcam zu und sucht in dem Live Video Stream nach Gesichtern. Interessant ist, dass gleich drei Varianten der Bildverarbeitung in Form von WebWorker-Instanzen parallel laufen. Eine WebAssembly-, eine „asm.js"- und eine JavaScript-Variante. Hier offenbart sich der mögliche Performance-Gewinn deutlich: Die WebAssembly-Variante ist um den Faktor 15 schneller als die JavaScript-Variante *(siehe Abbildung 5)*.

Missing Parts

Wo Licht ist, ist auch Schatten. Das WebAssembly-MVP funktioniert, jedoch zeigen sich in der aktuellen Version besonders die Stellen, an denen bewusst abgespeckt wurde. Aus Entwickler-Sicht fällt besonders das noch sehr einfache Tooling rund um die WebAssembly-Integration im Browser auf. Es gibt im Moment nur sehr einfache Debugging-Möglichkeiten.

Die ersten Browser beinhalten Sourcemap-Unterstützung für WebAssembly, diese könnte jedoch noch deutlich optimiert werden. Durch das Fehlen eines Memory-Managers und einer Garbage-Collection-Integration ist diese Funktionalität via Modul-System zu importieren. In der Konsequenz verursacht dieser Runtime-Code ein unnötig großes WebAssembly-Binary.

Das Fehlen eines direkten DOM-API oder des Zero-cost-Exception-Handling im MVP verursacht ebenfalls Glue Code zur Emulation von Hochsprachen mit Exception-Handling, was eigentlich nicht notwendig wäre. Unterstützung für Threading wäre wünschenswert, jedoch kann im MVP auch ohne native Thread-Unterstützung gearbeitet werden. Die WebSight-Demo zeigt es: Threads lassen sich zusammen mit dem WebWorker-API emulieren. Dies ist für die meisten Anwendungsfälle eine gute Grundlage.

Ausblick

Auch wenn das WebAssembly-MVP ein paar Ecken und Kanten hat, ist es doch schon sehr gut einsetzbar. Die Unity-Game-Engine nutzt bereits heute WebAssembly für die Spiele-Entwicklung. Unity kann jedoch mehr. Denkbar sind hier alle Formen von besonders interaktiven Produkt-Demonstrationen, die von WebAssembly profitieren können. Besonders in Verbindung mit Virtual Reality oder Augmented Reality ergeben sich völlig neue Anwendungsfälle.

Eine weitere interessante Möglichkeit ist die Migration von Legacy Code ins Web. Durch moderne Compiler können wir funktionierende und getestete Software in neuen Umgebungen laufen lassen, wie am Beispiel von WebSight-OpenCV auf einer Webseite. Die Möglichkeiten sind grenzenlos. Untermauert wird diese Perspektive durch die Übergabe der WebAssembly-Core-Spezifikation an das W3C im Februar 2018. WebAssembly ist somit ein offizieller Standard mit unglaublich viel Potenzial für die Zukunft. Das Web war schon immer gut für Überraschungen und wird es mit dieser Technologie auch weiterhin bleiben.

Mirko Sertic
mirko@mirkosertic.de

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

Neuer reaktiver Ansatz für die GUI-Programmierung mit Sodium

Sven Reinck, FLUXparticle

In jüngster Zeit ist der Begriff „reaktive Programmierung" in aller Munde. Es besteht allerdings noch viel Uneinigkeit darüber, was reaktive Programmierung überhaupt ist. Der Artikel bringt etwas Licht ins Dunkel, denn der Autor ist der Meinung, dass dieses Paradigma nach der funktionalen Programmierung das nächste sein wird, das in der breiten Masse Einzug hält.

Neben der reaktiven Programmierung gibt es auch die funktionale reaktive Programmierung (FRP). Auch wenn die Lambdas, die beim reaktiven Framework RxJava zum Einsatz kommen, frei von Nebeneffekten sind und somit als funktional bezeichnet werden könnten, lohnt es sich dennoch, ein funktional reaktives Framework wie Sodium anzuschauen, um nochmal ein gänzlich anderes Konzept kennenzulernen, das auch seine Anwendungsfelder hat.

RxJava ist zwar für die Verarbeitung von Events in einer GUI interessant, allerdings ist hier das Konzept von Sodium besser. Deshalb stellt der Autor seine Lösung für die ersten fünf GUIs des 7GUIs-Benchmarks von Eugen Kiss *(siehe „https://eugenkiss.github.io/7guis/tasks")* vor, die er mit seinem Groovy-Wrapper für Sodium *(siehe „https://github.com/SodiumFRP/sodium")* namens „Fenja" *(siehe „https://github.com/FLUXparticle/fenja")* erstellt hat. Im 7GUIs-Benchmark geht es darum, sieben typische Aufgaben der GUI-Programmierung in einem bestimmten Framework zu lösen, um anschließend verschiedene Frameworks hinsichtlich ihrer Funktionalität und Effektivität vergleichen zu können.

Zunächst ein Blick darauf, was „Functional Reactive Programming" mit Sodium von reaktiver Programmierung mit RxJava unterscheidet. In RxJava gibt es sogenannte „Observables", die einen oder mehrere Werte ausgeben können; ähnlich zu den Streams in Java 8. Diese können dann gefiltert, gemappt oder sonst wie verarbeitet werden. Zum Beispiel lassen sich mithilfe dieser Observables die Anfragen an einen REST-Service sehr bequem und mit nur wenigen Zeilen parallelisieren, und zwar besser als es mit Streams möglich wäre.

In einer GUI ist die Situation allerdings etwas anders gelagert. Hier gibt es zwei verschiedene Arten von Funktionalitäten, mit denen man umgehen muss. Auf der einen Seite sind da beispielsweise Textfelder, die zu jeder Zeit einen bestimmten Wert beinhalten. Diese sind in Fenja als „Value" abgebildet *(siehe Abbildung 1)*. Auf der anderen Seite gibt es auch Buttons, die zu einzelnen Zeitpunkten Ereignisse auslösen. Sie kommen in Fenja als „EventStream" vor *(siehe Abbildung 2)*.

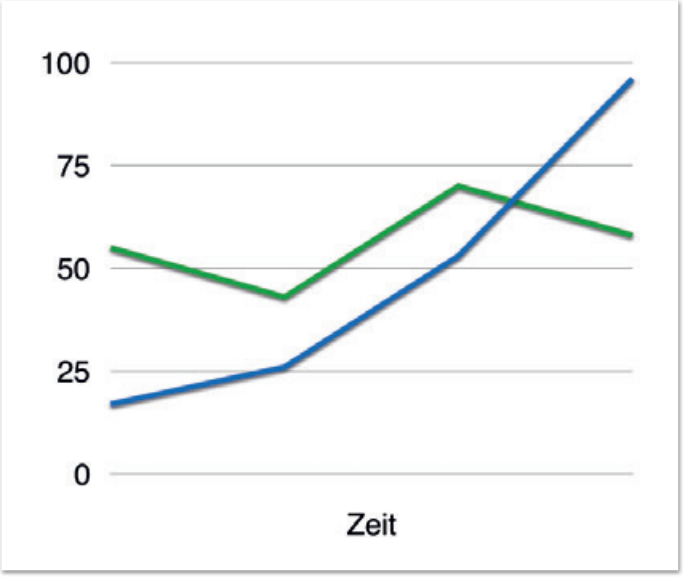


Abbildung 1: Darstellung von Werten, die sich über die Zeit verändern

Grundoperationen

Sowohl „Value" als auch „EventStream" können durch „map()" transformiert werden. Dagegen unterstützt nur „EventStream" die Operation „filter()". Würde diese Operation auch auf „Value" funktionieren, hätte der resultierende „Value" in bestimmten Fällen keinen Wert, und genau das würde der Definition eines „Value" widersprechen. Es gibt noch einige weitere Grundoperationen, die jedoch hier nicht wichtig beziehungsweise in den nachfolgenden Beispielen erklärt sind. Durch die Verknüpfung dieser Typen und Operationen entsteht ein Graph, der jeweils bei den Beispielen abgebildet ist.

Erste GUI: Ein Counter

Der Counter ist die einfachste Aufgabe aus dem Benchmark. Er besteht lediglich aus einem Textfeld und einem Button *(siehe Abbildung 3)*. Immer wenn man auf den Button klickt, wird die Zahl im Textfeld um „1" erhöht.

Der Graph ist auch entsprechend einfach und ermöglicht es damit, erstmal ein paar Konventionen für die Grafiken einzuführen. Die „EventStreams" sind als dicke, blaue Pfeile dargestellt und ihre Bezeichnung beginnt mit einem kleinen „s". Der Button erzeugt einen solchen „EventStream", der bei jedem Klick ein Event auslöst. Aus

Abbildung 2: Darstellung von Ereignissen zu unterschiedlichen Zeitpunkten

diesem „EventStream" und einem „Value" entsteht eine Schleife, die bei jedem Event den Wert des „Value" nimmt, um „1" erhöht und wieder in das „Value" zurückschreibt.

So eine Schleife würde bei einem herkömmlichen reaktiven System nicht funktionieren, da das Erhöhen des Zählers ja sofort wieder die Schleife triggern und somit der Zähler unendlich oft erhöht werden würde. Sodium verwendet dafür Transaktionen; jede Veränderung, die durch ein Ereignis ausgelöst wurde, führt erst in der nächsten Transaktion zu einer tatsächlichen Veränderung. Damit sind solche Schleifen ohne Weiteres möglich. Zu guter Letzt wird die Zahl im „vCounter" noch mit „map()" in einen „String" umgewandelt, der in JavaFX ganz einfach an die „textProperty()" des Textfelds gebunden werden kann. *Listing 1* zeigt, wie dieser Graph mithilfe von Fenja in Groovy dargestellt wird.

Alle hier vorgestellten Codes sind nach dem EVA-Prinzip unterteilt, um mehr Übersicht zu erreichen. Der erste Teil ist die Eingabe, bei der mithilfe der Convenience-Methode „streamOf()" ein „EventStream" aus dem „ActionEvent" des Buttons erzeugt wird. Im dritten Teil, der Ausgabe, wird mithilfe des überladenen Operators „<<" (die Idee stammt aus C++) der Counter an „textProperty()" des Textfelds gebunden.

Das Interessanteste ist der mittlere Teil; dort findet die eigentliche Verarbeitung statt – hier ist auch die Schleife zu erkennen –, denn die Variable „vCount" kommt dort vor, bevor sie überhaupt erzeugt wurde. Das würde normalerweise gar nicht gehen – hier kommt jetzt der Grund zum Tragen, aus dem der Autor überhaupt Groovy verwendet hat, um einen Wrapper für Sodium zu bauen.

Da Schleifen in FRP-Graphen etwas ganz Normales sind, hat Sodium dafür extra Datentypen geschaffen, nämlich eine „EventStreamLoop" und eine „ValueLoop". Diese dienen als Platzhalter für die echten Datentypen und werden erst später mit einem Wert belegt. Hier bietet Groovy die Möglichkeit, definieren zu können, was passieren soll, wenn eine Variable benutzt wird, die es noch gar nicht gibt. In dem Fall kommt einfach ein Loop-Objekt statt eines richtigen Ob-

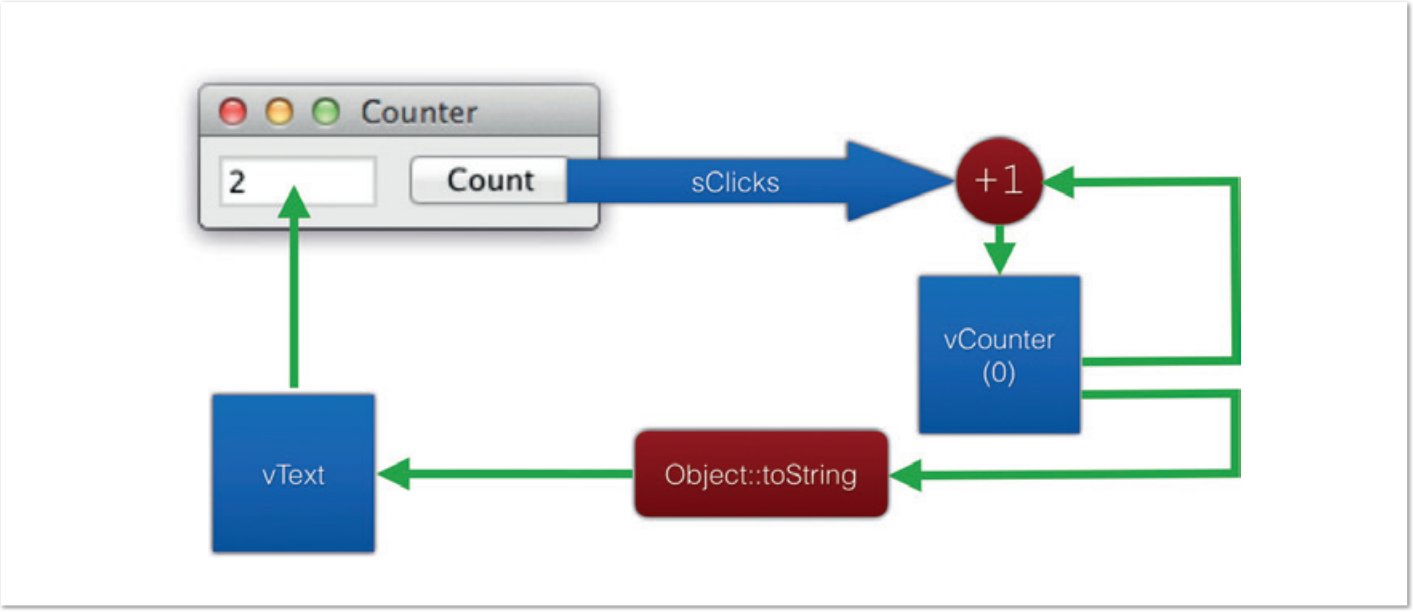


Abbildung 3: Darstellung des Graphen für den Counter

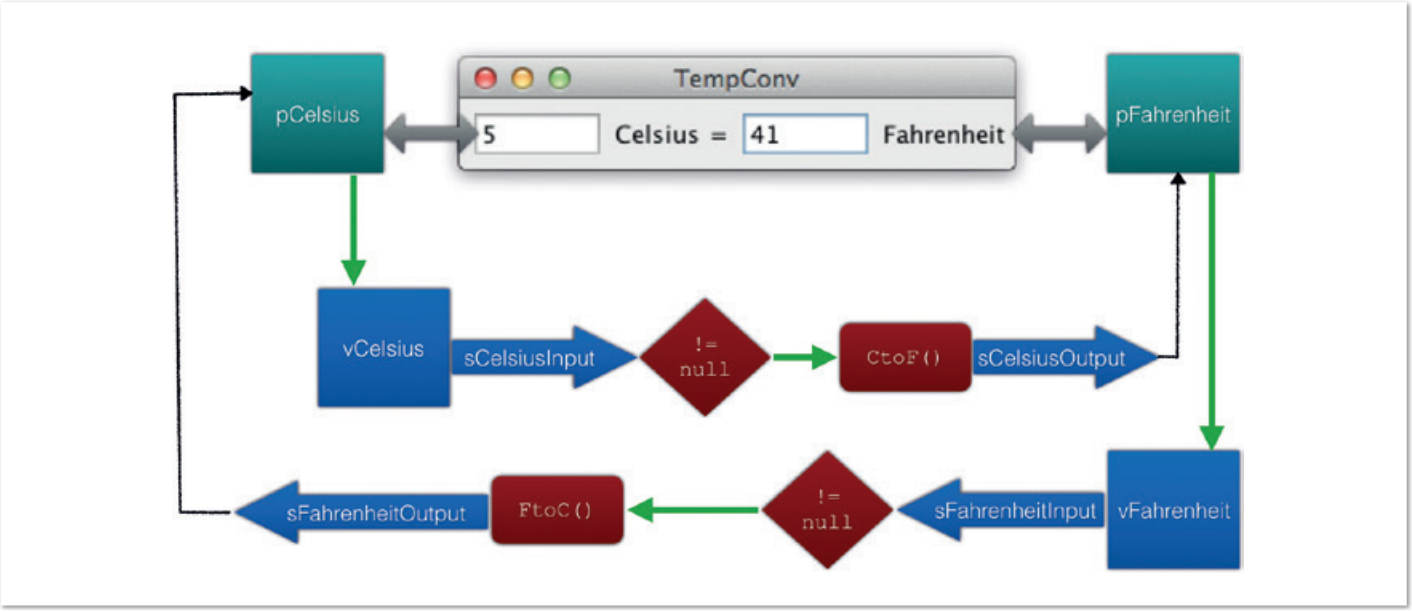


Abbildung 4: Darstellung des Graphen für die Temperatur-Umrechnung

jekts zurück, sodass die Loop später geschlossen werden kann. Die Anfangsbuchstaben „v" und „s" sind entscheidend, weil sonst nicht klar ist, ob eine „EventStreamLoop" oder eine „ValueLoop" erzeugt werden soll.

Zweite GUI: Temperatur-Umrechnung

Diese Aufgabe hat bestimmt jeder schon einmal in einer Lehrveranstaltung programmieren müssen. Es soll eine Temperatur von Celsius in Fahrenheit oder umgekehrt berechnet werden *(siehe Abbildung 4)*. Das Besondere an dieser Version ist, dass die Berechnung sofort beim Eintippen passiert, was einen vor eine Herausforderung stellt, wenn das programmseitige Schreiben in ein Textfeld ebenfalls als Eintippen interpretiert wird, wie es bei JavaFX der Fall ist.

Auch hier lässt einen Sodium nicht im Stich. Diese Art von Schleife wird nämlich erkannt und erzeugt eine „Exception", für die der Autor in seinem Groovy-Wrapper Fenja eine Methode gebaut hat, um sie zu ignorieren. Das Verhalten ist dann nämlich das Gewünschte, dass sofort beim Entdecken der Schleife die Änderungen nicht weiter propagiert werden. Es entsteht somit genau ein Rundgang und es wird nicht versucht, den umgerechneten Wert nochmal zurückzurechnen. *Listing 2* zeigt den Code dazu in Groovy.

Bei der Eingabe kommt die erwähnte Methode „valueOfSilent()" vor, die im Fall einer ungewollten Schleife den Fehler unterdrückt und die Schleife einfach nur durchbricht. Bei der Ausgabe wird kein Binding verwendet, da man sonst nichts mehr eingeben könnte. Stattdessen kommt die Methode „listen()" zum Einsatz, die einfach im Falle eines Events aufgerufen wird und somit den Setter für das Textfeld aufruft.

Dritte GUI: Flight Booker

Der Flight Booker ist von den sieben die Lieblingsaufgabe des Autors. Sie demonstriert nämlich am besten, welches Problem „Functional Reactive Programming" löst und wie man damit typische Fehler der GUI-Programmierung von vornherein unterbindet. Er hat sie deshalb auch schon in vielen seiner Trainings eingesetzt.

```
sClick = streamOf(btCountUp, ActionEvent.ACTION)
// -----
sNextCount = sClick.snapshot(vCount).map { count ->
count + 1 }
vCount = sNextCount.hold(0)
// -----
tfCount.textProperty() << vCount.
map(Integer.&toString)
```

Listing 1

```
sCelsiusInput     =  valueOfSilent(pCelsius).values();
sFahrenheitInput  =  valueOfSilent(pFahrenheit).values();
// -----
sFahrenheitOutput = sCelsiusInput
       .filter { it != null }
       .map { cToF(it.doubleValue()) };
sCelsiusOutput = sFahrenheitInput
       .filter { it != null }
       .map { fToC(it.doubleValue()) };
// -----
sFahrenheitOutput.listen(pFahrenheit.&setValue)
sCelsiusOutput.listen(pCelsius.&setValue)
```

Listing 2

Die Aufgabe ist leicht erklärt. Der Dialog besteht aus einer Combo-Box für die Auswahl, ob man nur Hinflug oder Hin- und Rückflug buchen möchte. Darunter sind zwei Textfelder, in die das Hin- beziehungsweise Rückreisedatum eingetragen wird. Das Textfeld für die Rückreise soll natürlich deaktiviert sein, wenn nur ein Hinflug gebucht wird. Das ist sicherlich noch leicht zu lösen, doch etwas schwieriger ist der „Buchen"-Button. Dieser soll genau dann aktiviert sein, wenn die Buchung so gültig wäre, die notwendigen Daten also gültig sind und bei der Buchung eines Rückflugs dieser zeitlich nicht vor dem Hinflug liegt *(siehe Abbildung 5)*.

Dieser Sachverhalt ist zwar auch mit herkömmlichen Mitteln recht einfach zu programmieren, doch tritt hier häufig der Fehler des „First Update" auf. Wer hat nicht schon mal einen Bug-Report auf den Tisch bekommen, der etwa so klang: „Wenn ich den Dialog das

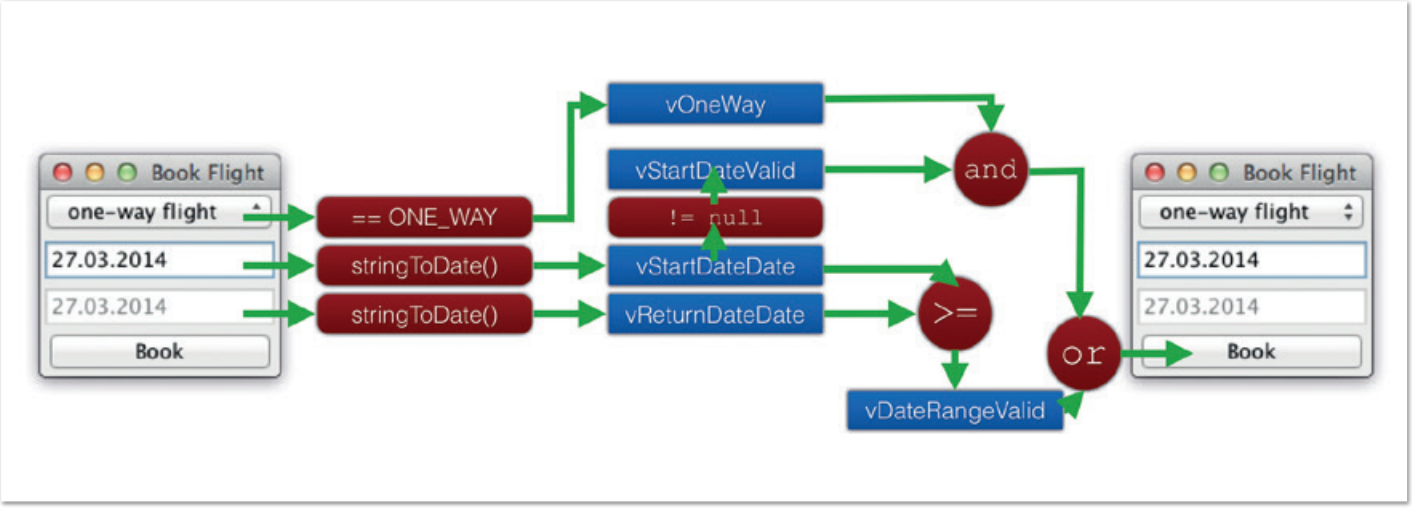


Abbildung 5: Darstellung des Graphen für den Flight Booker

```
vOneWay = vFlightType.map { v -> v == ONE_WAY_FLIGHT }

vStartDateAsDate = vStartDate.map { txt -> isDateString(txt)
                                        ? stringToDate(txt)
                                        : null }
vReturnDateAsDate = vReturnDate.map { txt -> isDateString(txt)
                                         ? stringToDate(txt)
                                         : null }

vStartDateIsValid = vStartDateAsDate.map { v -> v != null }
vReturnDateIsValid = vReturnDateAsDate.map { v -> v != null }

vDateRangeIsValid = (vStartDateAsDate**vReturnDateAsDate)
      { s, r -> s != null && r != null && s.compareTo(r) <= 0 }

vDatesValid = (vOneWay**vStartDateIsValid**vDateRangeIsValid)
              { o, s, r -> (o && s) || r }
```

Listing 3

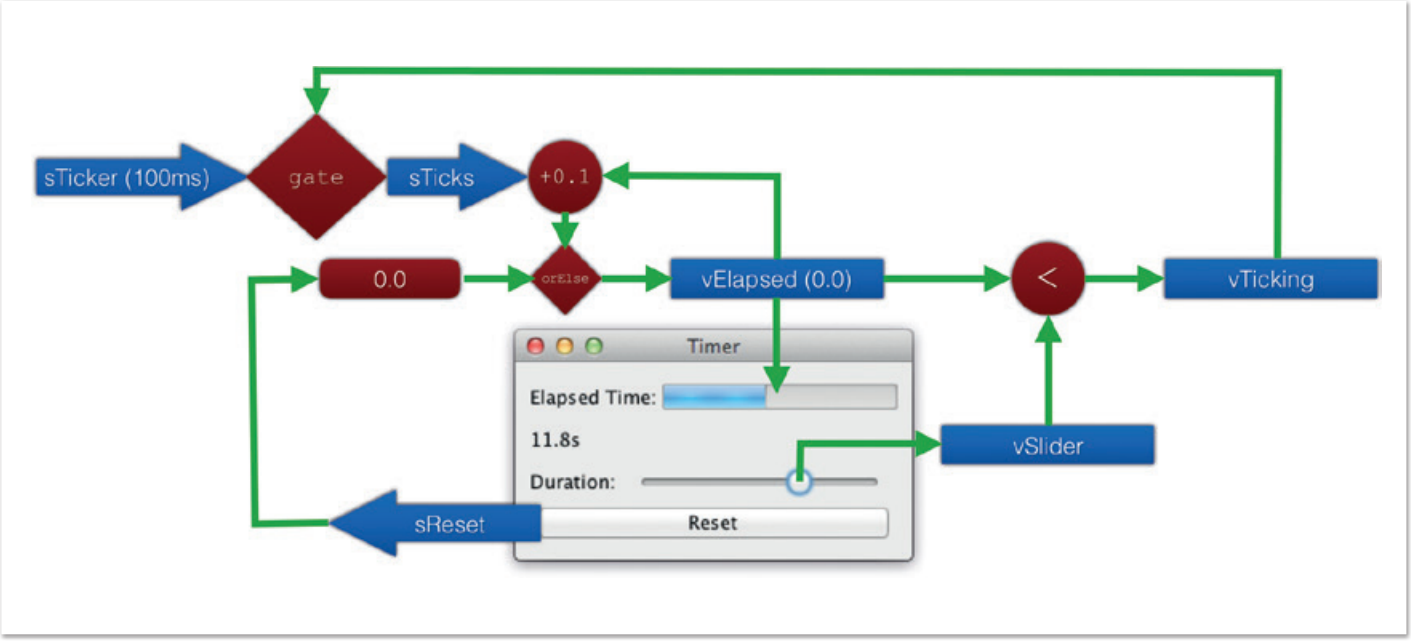


Abbildung 6: Darstellung des Graphen für die Timer-Aufgabe

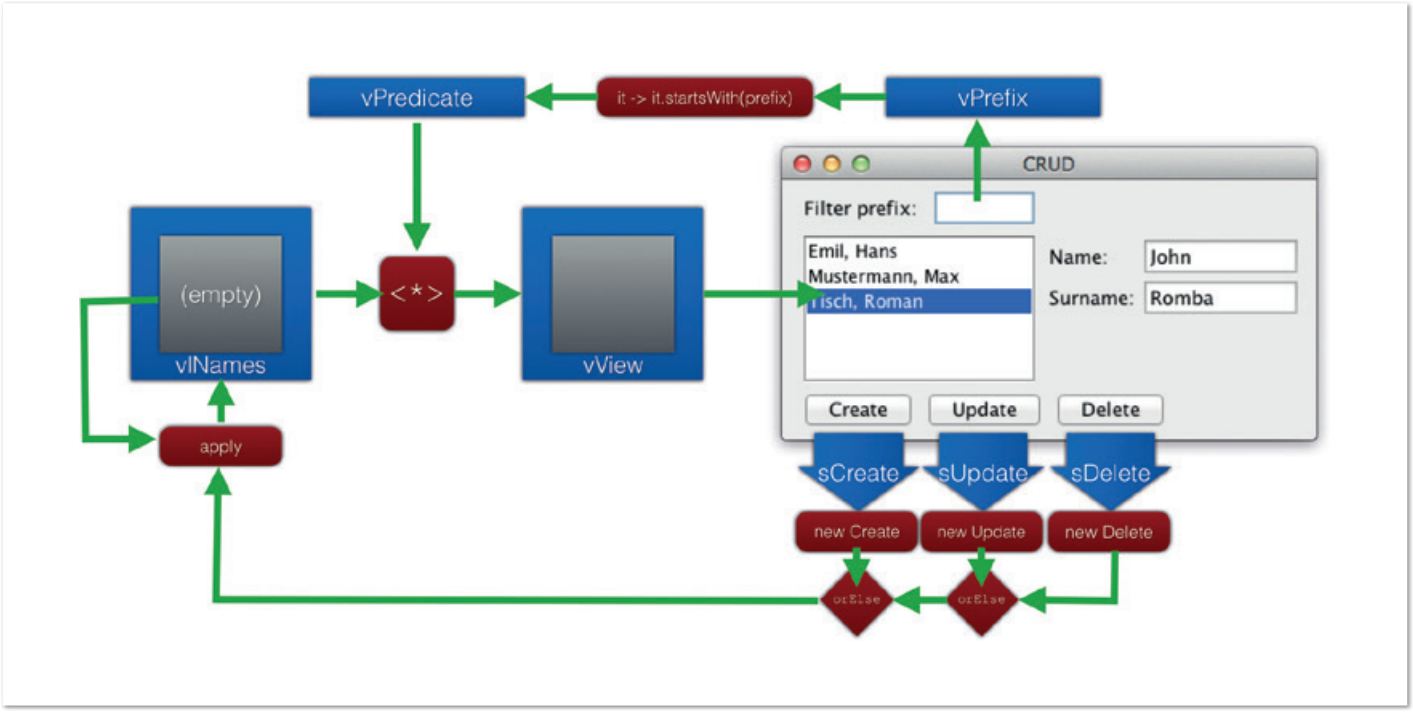


Abbildung 7: Darstellung des Graphen für die CRUD-Aufgabe

erste Mal starte, ist der „Buchen"-Button deaktiviert, obwohl alle Daten richtig sind. Ich muss erst das Datum verändern, damit der Button aktiv wird." Das liegt daran, dass die Berechnung, ob der Button aktiv ist, bei der herkömmlichen Methode oft nur bei Änderungen stattfindet und der Button am Anfang einfach fest aktiviert oder deaktiviert wird. Spätestens wenn sich am Initialisierungsprozess etwas ändert, könnte man sich für die falsche Option entschieden haben und es kommt zum beschriebenen Bug-Report.

Bei FRP kann dieser Fehler nicht passieren, da einmal dauerhaft definiert ist, unter welchen Bedingungen der Button aktiviert ist. Je nachdem, ob diese am Anfang vorherrschen, wird der Button aktiviert oder eben nicht. Für diese GUI reicht der Verarbeitungsteil als Code *(siehe Listing 3)*. Hier kommt wieder die Möglichkeit vor, Groovy-Operatoren zu überladen, um mithilfe des „**"-Operators mehrere „Values" zu kombinieren.

Vierte GUI: Timer

In der vierten Aufgabe soll getestet werden, wie gut zeitabhängige Events funktionieren. Hier hat der Autor mithilfe von JavaFX einen „EventStream" erzeugt, der alle hundert Millisekunden feuert. Dieser läuft durch einen „filter()", der nur dann die Events durchlässt, wenn die „ProgressBar" ihren durch den Slider vorgesehenen Maximalwert noch nicht erreicht hat *(siehe Abbildung 6)*.

Schließlich ist da noch der Reset-Button, der einfach ein „EventStream" erzeugt, das mit dem gefilterten Timer-„EventStream" kombiniert wird. Hier kommt die Funktion „orElse()" zum Einsatz, die zwei „EventStreams" auf eine Weise kombiniert, dass ihr Verhalten immer deterministisch ist. Hier tut sich Sodium wieder mit seinen Transaktionen hervor. Treten nämlich innerhalb der gleichen Transaktion zwei Events auf, die in den gleichen „EventStream" fließen sollen, ist genau festgelegt, welches Event durchkommt und welches nicht. Bei anderen reaktiven Frameworks kann das Verhalten in so einer Situation durchaus zufällig sein.

Fünfte GUI: CRUD

Bei der CRUD-Aufgabe wird es schon etwas komplizierter. Es soll ein übliches Create-Read-Update-Delete-Interface aufgebaut werden. Hierfür musste man eine spezielle Datenstruktur bauen, die eine reaktive Liste verwalten kann. Diese Listen-Datenstruktur wird mit den Events aus den drei Buttons am unteren Rand gefüttert und befindet sich jeweils in dem grauen Kasten in der Grafik *(siehe Abbildung 7)*.

Diese Datenstruktur ist notwendig, weil es in dieser Aufgabe möglich sein sollte, die Liste live zu filtern, Listen in Sodium jedoch bisher noch nicht vorgesehen sind. Listen sind überhaupt noch ein schwieriges Thema in FRP. Deshalb arbeitet der Autor gerade an einer neuen Version von Fenja, die ohne Sodium auskommt und auch reaktive Listen beinhaltet.

Sven Reinck
sreinck@fluxparticle.de

Sven Reinck hat von 2003 bis 2007 Informatik an der Fachhochschule in Wedel nahe Hamburg studiert und mit Diplom und Master of Computer Science abgeschlossen. Anschließend hat er in verschiedenen Firmen gearbeitet und dabei vor allem in der Spielebranche wertvolle Praxiserfahrung gesammelt. Seit dem Jahr 2012 unterrichtet Sven Reinck als freiberuflicher IT-Trainer und bietet seit dem Jahr 2016 auch eigene Seminare an.

Die Zukunft von Java mitbestimmen

Sebastian Daschner, Sebastian Daschner – IT-Beratung

Mit dem JavaLand Lobby Day haben die Organisatoren direkt vor der JavaLand 2018 eine Java-Unkonferenz veranstaltet. Eine Unkonferenz bezeichnet ein Treffen im Open-Space-Format, bei der es keine feste Agenda gibt, sondern die Themen spontan beschlossen und anschließend diskutiert werden.

Im ersten JavaLand Lobby Day wurde hauptsächlich die Zukunft von Open Source Java-Frameworks besprochen. Die Hauptthemen waren „Jakarta EE", „EE4J", „MicroProfile", deren mögliches Zusammenspiel in der Zukunft sowie Java bei der Eclipse Foundation generell. Daneben wurde auch intensiv über JavaFX, die Zukunft von Desktop-Java und die neuen Java-Releasezyklen diskutiert.

Ebenfalls wichtig war, wie sich die Community und jeder Einzelne in die Weiterentwicklung der Java-Plattform einbringen kann. Für den Autor außerdem interessant, weil es thematisch sehr nah an seinem JavaLand-Vortrag lag, wie Java EE beziehungsweise Jakarta EE sich im modernen Umfeld von Cloud-Native-Technologien wie Docker, Kubernetes oder Istio zurechtfinden kann.

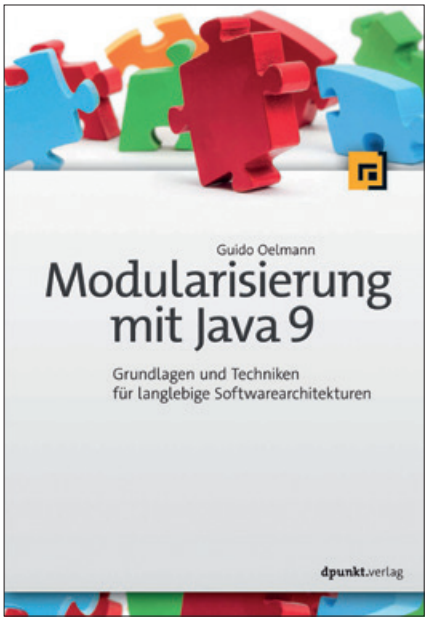
Die für den Autor am positivsten Eindrücke: der Enthusiasmus und konstruktive Ansatz aller Teilnehmer, die Zukunft von Java aktiv zu gestalten. Das kann er nur an alle weitergeben: Folgt den Mailing-Listen und Entwicklungen der Themen, an den ihr interessiert seid, beispielsweise EE4J oder MicroProfile, und im Idealfall gebt Feedback, stellt Fragen und kollaboriert in den Open-Source-Projekten.

Wichtige Links zum Thema

- EE4J: *https://projects.eclipse.org/projects/ee4j/charter*
- Jakarta EE Working Group Charter: *https://www.eclipse.org/org/workinggroups/jakarta_ee_charter.php*
- MicroProfile: *http://microprofile.io*
- Future of JavaFX, Oracle Blog: *https://blogs.oracle.com/java-platform-group/the-future-of-javafx-and-other-java-client-roadmap-updates*

Sebastian Daschner
mail@sebastian-daschner.com

Sebastian Daschner arbeitet als freiberuflicher Java-Consultant, Software-Entwickler und -Architekt und programmiert begeistert mit Java (EE). Er nimmt am Java Community Process teil, ist in der JSR 370 Expert Group vertreten und entwickelt an diversen Open-Source-Projekten. Er ist Java Champion und arbeitet seit mehr als sieben Jahren mit Java. Sebastian Daschner evangelisiert Java- und Programmier-Themen unter „https://blog.sebastian-daschner.com" sowie auf Twitter unter „@DaschnerS". Wenn er nicht gerade mit Java arbeitet, bereist er auch gerne die Welt – entweder per Flugzeug oder Motorrad.



Modularisierung mit Java 9

gelesen von Heiko Sippel

Die wichtigste Neuerung in der nun auch nicht mehr ganz so frischen 9. Version von Java ist sicher das Modularisierungskonzept, das unter dem Namen „Jigsaw" bekannt ist. Dies ist allerdings wichtig genug, um dafür ein eigenes Buch zu schreiben und es nicht nur als großes Kapitel in einem Java-9-Buch abzuhandeln. Da Modularisierung bereits in mittelgroßen Projekten unabdingbar ist, wird man als Entwickler oder Architekt kaum darum herumkommen, sich mit Jigsaw zu beschäftigen, auch wenn man bereits mit anderen Lösungen arbeitet.

Das Buch von Guido Oelmann nimmt sich daher ausschließlich dieses Themas an. Es zerfällt deutlich in mehrere Teile. Der erste behandelt ganz allgemein Sinn und Zweck einer Modularisierung und bietet eine gute Einführung in dieses Thema sowie eine Erläuterung der wichtigsten Begriffe. Wer schon weiß, warum man komplexe Software besser in kleine Teil zerlegt, kann diese vierzig Seiten getrost überschlagen und direkt zum praktischen Teil II des Buchs übergehen. Hier wird zunächst das neue Modul-Konzept von Java 9 vorgestellt, die verschiedenen Modul-Typen und das Service-Provider-Consumer-Modell. Dabei wird gleich an einem praktischen Beispiel ein einfaches Projekt umgesetzt, wobei zum Bauen der Module die Kommandozeile verwendet wird. Nach diesem Kapitel hat man schon einen sehr guten Überblick, wie das Ganze funktioniert, und kann im nächsten Kapitel verfolgen, wie das JDK selbst modularisiert wird.

Die folgenden Kapitel orientieren sich ganz am praktischen Einsatz, etwa die Einbindung in Entwicklungsumgebungen (Eclipse, IntelliJ) und Build-Systeme (Ant, Maven, Gradle). Ein etwas längeres Beispielprojekt eines einfachen Hotelreservierungssystems, in dem zwei verschiedene Modularisierungskonzepte implementiert werden, rundet die Einführung ab.

Das Buch lässt sich sehr gut lesen und kommt immer auf den Punkt. Angenehmerweise verzichtet der Autor während der Vorstellung der Konzepte auf kritische Bemerkungen der Art „Dies ist gut", „Das hätte man besser machen können", was in anderen Büchern oft den Lesefluss stört. Stattdessen findet eine Kritik des Jigsaw-Konzepts und ein Vergleich mit anderen Lösungen – insbesondere OSGi und Microservices – in eigenen Kapiteln statt, die man lesen oder auch überspringen kann (was man aber nicht tun sollte, da auch sie sehr interessant sind). Eventuell wäre es besser gewesen, für diesen eher theoretischen Bereich einen Teil III hinzuzufügen.

Fazit: Das Buch ist eine sehr tiefgehende und gleichzeitigt kompakte Einführung in Jigsaw. Dank des Aufbaus kann man die für sich wichtigen Themen herauspicken. Die einzelnen Kapitel sind allein sehr gut lesbar, sodass man auch Teile überspringen kann, für die man sich nicht interessiert, während andere Kapitel beim praktischen Arbeiten zum Nachschlagen dienen.

Heiko Sippel
heiko.sippel@infaktum.de

Autor: Guido Oelmann
Titel: Modularisierung mit Java 9
Verlag: dpunkt.verlag
Umfang: 330 Seiten
Preis: 32,90 Euro, eBook 25,99 Euro
ISBN: 978-3-86490-477-6

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 © Sira Anamwong / *123RF.com*
S. 8: Original © Freepik / *freepik.com*
S. 12-15: © Javaland GmbH
S. 16: © Leo Wolfert / *123RF.com*
S. 21: © Jan Hruby / *123RF.com*
S. 25: © Alvaro German Vilela / *123RF.com*
S.28: © scyther5 / *123RF.com*
S.37: © ndul / *123RF.com*
S.43: © Jenkins project / *https://jenkins.io*
S.48: © Bakhtiar Zein / *123RF.com*
S.53: © Carlos Baraza, Web Assembly
S.58: © aurielaki/ *123RF.com*
S.64: © Timur Arbaev/ *123RF.com*
S.65: © dpunkt.verlag

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

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

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

Inserentenverzeichnis


ORAWORLD
Das e-Magazine für alle Oracle-Anwender!
EOUC
EMEA ORACLE USERGROUP COMMUNITY
• Spannende Geschichten aus der Oracle-Welt
• Technologische Hintergrundartikel
• Leben und Arbeiten heute und morgen
• Einblicke in andere User Groups weltweit
• Neues (und Altes) aus der Welt der Nerds
• Comics, Fun Facts und Infografiken
Jetzt Artikel einreichen oder Thema vorschlagen!
Bis 9. August 2018
Jetzt e-Magazine herunterladen
www.oraworld.org

Would you like to develop **Android apps** for more than 8 million unique visitors per month?

JOIN OUR TEAM!

"Hi, I'm Sami and I work for 1&1 as an Android Software Developer. My team and I are responsible for developing Android applications for our brands GMX and WEB.DE, which are rated as "top developer" apps in the Google Play Store. My work consists of maintaining a clean code, building new features and being up to date with all the technologies of our apps.

Applying agile methods, we mainly use Java and Kotlin to develop our features, which means we can easily react to customer requests. The strong relationships between team members and the knowledge we share with each other is what makes working in our team so special. Are you interested in working with us? We are looking forward to meeting you." **sami@1and1.com**

MEMBER OF united internet

1&1, GMX and WEB.DE are brands of the United Internet AG – a stock-listed company with more than 9,000 employees and an annual sales volume of approximately 4 billion Euro.

DESIGN YOUR CAREER

top EMPLOYER GERMANY 2018 CERTIFIED EXCELLENCE IN EMPLOYEE CONDITIONS

0721 91374-6891 · www.design-your-career.com