Ausgabe 05/2020

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 05

Java aktuell

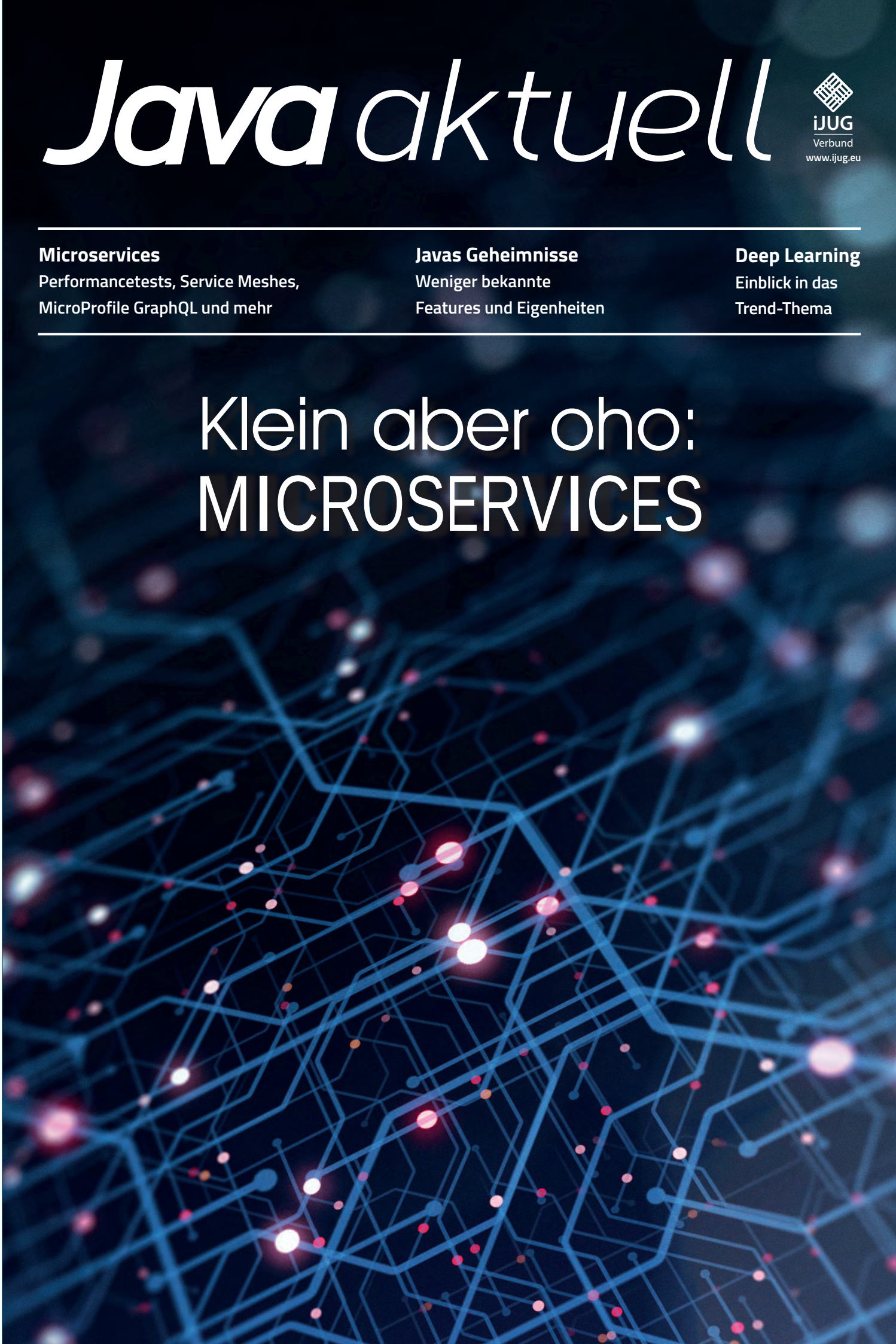
iJUG Verbund
www.ijug.eu

Microservices
Performancetests, Service Meshes, MicroProfile GraphQL und mehr

Javas Geheimnisse
Weniger bekannte Features und Eigenheiten

Deep Learning
Einblick in das Trend-Thema

Klein aber oho: MICROSERVICES



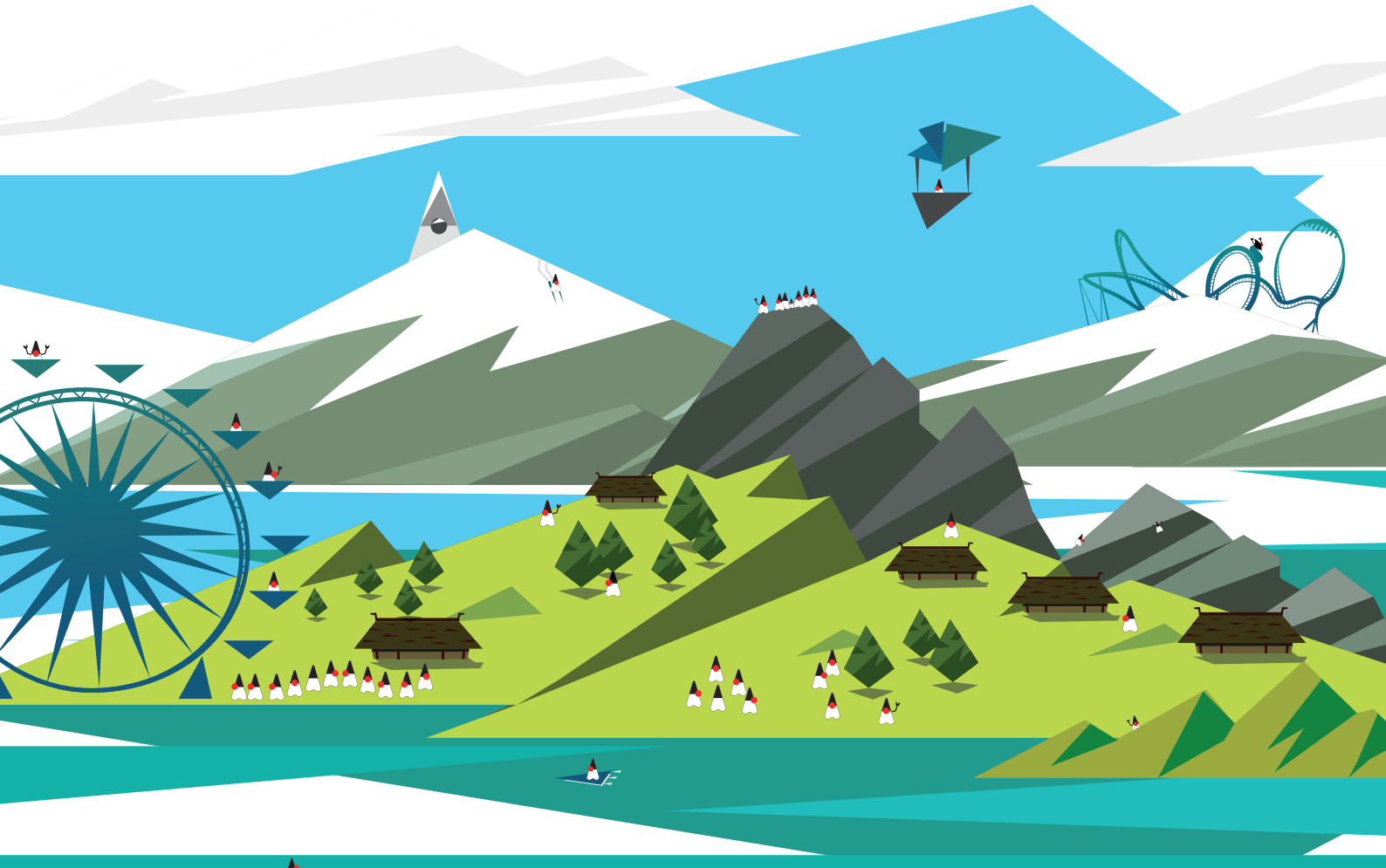
JavaLand

16. - 18. März 2021
in Brühl bei Köln

Save the Date

Hybride Veranstaltung

Was die JavaLand als Plattform für Wissentransfer und Networking ausmacht, kannst du im Phantasialand oder online erleben. Als Teilnehmer entscheidest du selbst, welche Variante du wählen möchtest.



Präsentiert von: DOAG Heise Medien

Community Partner: iJUG Verbund

www.javaland.eu

Drum prüfe, wer sich bindet – eine Microservice-Checkliste

Jürgen Lampe, S&N Invent GmbH

Microservices sind ein mächtiges, aber ebenso anspruchsvolles wie aufwendiges Architekturkonzept. Einmal begonnen, ist es nur unter erheblichem Aufwand möglich, auf eine monolithische Struktur zurückzuschwenken. Dieser Artikel versammelt in Form einer Checkliste wichtige Fragen, die man rechtzeitig überdenken und beantworten sollte.

Jede Microservices betreffende Diskussion sieht sich mit der Unschärfe dieses Begriffs konfrontiert. Deshalb sei vorausgeschickt, dass es hier um das voll ausgeprägte Konzept mit konsequenter Trennung der Entwicklungsstränge geht. Die dazugehörende DevOps-Kultur mit umfassender Testautomatisierung und häufigen Deployments wird als unstrittig vorausgesetzt. Einige Bemerkungen zu Ansätzen, die das Konzept nur teilweise anwenden, finden sich am Ende. Wenn im folgenden Text von „Service" die Rede ist, so ist damit immer ein einzelner Microservice gemeint.

Eine folgenschwere Entscheidung

Der Entschluss, eine Microservices-Architektur aufzubauen, reicht in seinen Auswirkungen weit über andere Architekturentscheidungen hinaus, weil er die umgebende Organisationsstruktur unweigerlich mit einbeziehen muss. Das bedeutet, große Teile der Organisation sind mehr oder weniger davon betroffen und müssen diese Entscheidung in vollem Umfang unterstützen.

Ebenso wichtig ist es zu bedenken, dass ein solcher Entschluss nur unter erheblichem Aufwand rückgängig gemacht werden kann. Es ist im Zweifel günstiger, eine zunächst sauber modularisierte monolithische Applikation in Microservices zu zerlegen, weil dadurch nur geringe Mehrkosten entstehen. Beim umgekehrten Weg bleiben erhebliche, letztlich verschwendete Aufwände zurück (wenn man den Erfahrungsgewinn einmal ausklammert).

Deshalb ist es enorm wichtig, dass die im Folgenden aufgeworfenen Fragen gründlich bedacht und ehrlich beantwortet werden, bevor man eine so folgenschwere Entscheidung trifft. Natürlich wird es in vielen Fällen keine klare Ja-Nein-Antwort geben und mehr noch als bei anderen IT-Fragen spielt die Einbeziehung geschäftsstrategischer Gesichtspunkte eine wichtige Rolle. Schließlich ist nie zu vergessen, „dass Microservices kein Ansatz sind, um die Kosten der Software-Entwicklung zu minimieren. Es gehe vielmehr darum, die Reaktionszeiten – Stichwort „Time-to-Market" – sowie die Qualität der Systeme deutlich zu verbessern." *[1]*

Warum Microservices?

Diese erste, nur scheinbar triviale Frage sollte nicht zu leicht genommen werden. Der Aufbau einer Microservice-basierten Anwendung ist teuer. Dazu kommen, wie im Weiteren noch dargestellt wird, nicht zu ignorierende Folgekosten. Aber nicht nur deshalb sollte eine entsprechende Architekturentscheidung gut begründet sein, denn die Konsequenzen reichen weit über die IT hinaus und beeinflussen die Strategie über Jahre.

Alle wichtigen Vorteile von Microservices basieren auf drei Punkten:

1. **Schnelligkeit:** schnelle Entwicklung, schnelle Erweiterung der Ressourcen, schnelle Anpassung an den Markt
2. **Flexibilität:** einfacher Austausch von Komponenten und Technologien
3. **Skalierbarkeit:** gute Anpassbarkeit an stark wechselnde Bedürfnisse

Es gibt weitere positive Aspekte, deren Bedeutung jedoch geringer ist.

Die Vorteile bekommt man allerdings nicht geschenkt. Sie haben ihren Preis. Daher muss als Erstes geklärt werden, wie groß und wichtig der jeweils erzielbare Nutzen realistischerweise sein wird und welche Kosten man dafür akzeptieren kann oder will. Dieser Nutzen wird sich nicht immer monetär beziffern lassen, wenn es beispielsweise darum geht, eine Position im Markt zu besetzen oder zu behaupten. Trotzdem lässt sich die grundsätzliche Entscheidung, welchen Einsatz das angestrebte Ziel wert ist, nicht vermeiden, weil es um nennenswerte Beträge geht.

Sollen bestehende IT-Probleme gelöst werden?

Das wäre ein denkbar schlechter Ausgangpunkt, um mit dem Aufbau einer neuen komplexen IT-Struktur zu beginnen, weil in dieser Lage oft fachliche und organisatorische Probleme vorliegen, die besser vorher gelöst werden sollten. Wie bereits gesagt, lösen Microservices keine originären IT-Probleme, sondern können dabei helfen, bestimmten Anforderungen aus dem Geschäftsumfeld zu genügen. Eine Microservices-Architektur verlagert Komplexität, aber vermindert sie nicht. Und sie hilft ebenso wenig bei ungenügender Durchdringung oder Strukturierung des Anwendungsbereichs. Natürlich gibt es Ausnahmen, aber ob diese vorliegen, sollte sehr kritisch geprüft werden.

Ist die Organisation vorbereitet?

Jeder Microservice verkörpert eine Anwendungssäule. Wenn die Vorteile der unabhängigen Entwicklung zum Tragen kommen sollen, muss die Organisationsstruktur entsprechend vorbereitet sein. Conways Gesetz „Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure." fordert sein Recht.

Daher ist es gefährlich, wenn versucht wird, Microservices als reines IT-Projekt einzuführen. In den seltensten Fällen wird die Organisationsstruktur schon ausreichend den Erfordernissen entsprechen. Je weiter die bestehende Struktur vom erforderlichen Zielbild entfernt ist, desto aufwendiger und riskanter wird das Projekt. Auch die gleichzeitige Umwandlung der Organisationsstruktur ist wegen der schwer beherrschbaren Komplexität kritisch und nur bei konsequenter Unterstützung durch das gesamte Management überhaupt umsetzbar. Manchmal funktioniert es, einen neuen Organisationszweig quasi parallel aufzubauen, der dann schrittweise Funktionen aus der „alten Welt" übernimmt. Im Vorteil sind in diesem Punkt Startups, die ihr Geschäftsmodell gerade erst entwickeln.

Ist die notwendige Unterstützung des Managements gegeben?

Eine für Microservices geeignete Organisationsstruktur erfordert zwangsläufig auch eine entsprechende Dezentralisation von Verantwortung und Entscheidungen. Das ist vor allem deshalb eine große Herausforderung, weil die gängige Praxis gewöhnlich anders aussieht. Unabhängige Entwicklung führt ganz natürlich zu vielen Redundanzen – das ist das genaue Gegenteil vom „Heben von Synergien". Auf allen Ebenen muss deshalb vorbehaltlos akzeptiert werden, dass jedes Entwicklungsteam frei entscheiden kann, beispielsweise auch über die zu verwendenden Arbeitsmittel. Ebenso müssen die einzelnen Geschäftsbereiche ihre Anforderungen ohne lange Abstimmung an das jeweilige Entwicklungsteam übergeben können.

Sind die übergreifenden Prozesse hinreichend stabil?

Bei aller Dezentralisierung wird es einige übergreifende Geschäftsprozesse geben – sonst wäre es ja nicht *ein* Anwendungsprojekt.

Häufig muss ein zentrales Buchungs- und Abrechnungssystem bedient werden. Es kann sich aber ebenso um eine gemeinsame Weboberfläche handeln. Und in jedem Fall gehört ein umfassendes Sicherheits- und Berechtigungskonzept dazu.

Alles, was bei Änderungen über einen einzelnen Service hinausreicht, ist kritisch. Wegen der dabei notwendigen Synchronisation verursachen solche Modifikationen enormen Aufwand und müssen sorgfältig geplant werden. Ihre Wahrscheinlichkeit sollte daher von vornherein minimiert werden. Aus diesem Grund ist ein möglichst stabiles (weiteres) Umfeld eine nicht zu vernachlässigende Voraussetzung. Das ist nun ein Punkt, an dem Startups eher im Nachteil sind, weil diese Stabilität nur durch Erfahrung erreicht werden kann.

Sind die Geschäftsprozesse ausreichend fehlertolerant?

Niemand sollte sich etwas vormachen: Fehler passieren. Wenn Services häufig und unabhängig produktiv gehen – und das ist ja eines der Kernprinzipien – wird es sich nicht vermeiden lassen, dass es, möglicherweise auch an ganz unerwarteten Stellen, zu Fehlern kommt, weil es unter anderem nicht möglich ist, alle Wechselwirkungen im Vorfeld zu testen. Damit muss man leben können, denn es gibt in diesem Paradigma keine Stabilisierungsphase wie bei herkömmlichen Projekten, an deren Ende alle schweren Fehler gefunden und beseitigt sind. Selbst umfassende Tests werden nie die vollständige Korrektheit des Codes garantieren können.

Die notwendige Fehlertoleranz im Geschäftsbereich ist deshalb ein entscheidendes Kriterium, das Microservices in ihrer konsequenten Umsetzung für bestimmte Anwendungen faktisch ausschließt. Denn während der Schaden, den ein abgebrochener Verkaufsvorgang oder eine Fehllieferung beim Kunden verursacht, durch einen Bonus beispielsweise relativ einfach behoben werden kann, ist das bei juristisch relevanten Vorgängen wie termingemäßen Überweisungen und Erteilungen oder Versagungen von Genehmigungen deutlich schwieriger oder sogar unmöglich.

Sind unsichere Antwortzeiten hinnehmbar?

Im Netz ist nichts garantiert. Wenn Services miteinander kommunizieren, dauert das nicht nur wesentlich länger als bei internen Prozeduraufrufen, es kann jederzeit auch zu Verbindungsabbrüchen oder extrem langen Wartezeiten kommen. Vorgegebene Antwortzeiten lassen sich also nur mit einer gewissen – und von vielen Einflüssen abhängigen – Wahrscheinlichkeit gewährleisten. Für eine Verkaufsanwendung wird das im Allgemeinen kein Problem sein. Ein Wertpapier-Hochfrequenzhändler sieht das aber möglicherweise anders.

Ist die Zielstellung angemessen?

Die gute Parallelisierbarkeit der Entwicklung auszunutzen, ist nicht billig. Praktisch verursacht jeder Service für sich den kompletten Verwaltungsaufwand eines (kleinen) Projekts. Es dauert darüber hinaus in der Regel einige Zeit, bis sich die Teams gefunden haben und voll leistungsfähig werden. Die danach beginnende produktive Phase sollte nicht zu kurz sein, das heißt, jeder Service muss genügend Entwicklungspotenzial für einen längeren Zeitraum aufweisen, um den hohen initialen Aufwand zu rechtfertigen. In Ausnahmefällen ist es allerdings denkbar, dass ein extern verursachter Zeitdruck (Time-to-Market) Kostenfragen zweitrangig erscheinen lässt.

Es ist auch zu bedenken, dass sich die Vorteile der häufigen Deployments nur dann wirklich nutzen lassen, wenn die Anwendung in sinnvoller Weise bereits mit einem kleinen Teil der vorgesehenen Funktionen produktiv genommen und in der Folge über einen längeren Zeitraum schrittweise erweitert werden kann.

Ist die fachliche Domäne gut genug verstanden?

Für die Microservices-Architektur muss die Anwendung in möglichst unabhängige *fachliche* Säulen oder Silos zerlegt werden. Das

gelingt nur dann erfolgreich, wenn die Fachdomäne ausreichend gut verstanden ist, weil die Abgrenzung der Säulen die logische Voraussetzung für die Zerlegung ist. Die Schnittstellen gehören natürlicherweise zur das Gesamtprojekt umfassenden Struktur. Damit gilt für sie ganz besonders die bereits getroffene Aussage, dass spätere Änderungen teuer und daher möglichst zu vermeiden sind. Das heißt, dass sie bereits frühzeitig klar gefasst werden müssen. Um qualitativ hochwertige Schnittstellen formulieren zu können, sind jedoch umfassende Kenntnisse nötig, und zwar mit Sicht über den aktuellen Stand der Verhältnisse hinaus – denn die Welt bleibt nicht stehen.

Diese Schnittstellen definieren auf der einen Seite die Architektur und geben der Anwendung ihre Struktur. Auf längere Sicht kann daraus aber auch ein Korsett erwachsen, das die weitere Entwicklung behindert. Da die Zukunft nicht vorhersehbar ist, lässt sich dieses Manko zwar nicht völlig vermeiden, aber ein möglichst tiefes Wissen hilft, den kritischen Zeitpunkt weiter in die Zukunft zu verschieben.

Nach diesen eher geschäftspolitischen Kriterien geht es im Folgenden vorrangig um technische Fragestellungen, die natürlich zumindest über die Kosten mit den ersteren verbunden sind.

Sind die technischen Voraussetzungen erfüllbar?

Microservices verlagern einen Teil der Komplexität aus dem Anwendungscode in die Struktur des Systems, wo sie durchaus nicht leichter zu beherrschen ist. Daraus resultieren entsprechend hohe Anforderungen beim Aufsetzen und bei der Verwaltung der technischen Infrastruktur:

- Heterogenität durch die (mögliche) Vielzahl der eingesetzten Technologien
- Eigene Backup- und Recovery-Prozesse für jeden Microservice
- Monitoring und Logging aller Services
- Skalierung und Resilienz (Konfigurationsautomatisierung)
- Sicherheit, zum Beispiel Zertifikatsverwaltung (siehe Sicherheitsanforderungen)

Diese Aufgaben sind nicht nur zeit- und kostenintensiv, sie erfordern auch spezielle Fähigkeiten und Erfahrungen. Zudem muss ein erheblicher Teil davon, zum Beispiel für Updates, Erweiterungen und Migrationen, über die gesamte Lebenszeit der Anwendung erbracht werden. Insbesondere können neu entdeckte Schwachstellen (Exploits) jederzeit ungeplante Aufwände verursachen. Die nicht unerheblichen Kosten der benötigten Verwaltungswerkzeuge dürfen ebenfalls nicht vergessen werden, denn die gratis angebotenen Funktionen sind für den produktiven Betrieb selten ausreichend. Hinzu kommt, dass Fachleute mit den benötigten Kenntnissen begehrt und rar sind.

Kann die zusätzliche Komplexität beherrscht werden?

Verteilte Programme machen auch den eigentlichen Anwendungscode komplizierter. Im Gegensatz zu einem einfachen Prozeduraufruf kann beim Zugriff auf einen Service einiges schiefgehen, was sich nicht allein auf der technischen Ebene abfangen lässt. Wie die Reaktion auf einen Timeout oder einen unberechtigten Zugriffsversuch aussehen muss, ist oft eine fachliche Frage. Und je nachdem, ob der Zugriff synchron oder asynchron erfolgt, muss die Anwendungslogik anders aufgebaut werden. Die Berücksichtigung dieser Kommunikationsanforderungen macht die Programmierung schwieriger.

Können die Sicherheitsanforderungen gewährleistet werden?

Einen einzelnen Computer kann man einschließen – eine Microservices-Anwendung nicht. Jeder Service steht im Netz und stellt damit einen potenziellen Angriffspunkt dar. Auch eine eventuelle Begrenzung aufs Intranet reduziert das Risiko nur graduell. Ein Teil der Sicherheitsanforderungen kann bereits auf der Ebene der technischen Infrastruktur gewährleistet werden, aber es bleiben Fragen, die rechtzeitig geklärt werden müssen:

1. Welche Rechte und Rollen brauchen die Services und wie müssen diese unter Umständen abgestuft werden? Wo werden diese Rechte und Rollen verwaltet?
2. Wie werden die Berechtigungen durch die Services geprüft? Wie erfolgt die Weitergabe von Rechten?
3. Wie kritisch sind die übertragenen Daten? Ist eventuell eine zusätzliche Verschlüsselung erforderlich?
4. Wie wird gegebenenfalls verhindert, dass bei der Arbeit für einen Mandanten im Namen eines anderen auf Services zugegriffen werden kann?
5. Ist es möglich, allein durch Analyse der Aufruffolgen, Nachrichtenlängen oder Ähnlichem sensible Informationen zu gewinnen? Muss das verhindert werden?

Die Liste ist nicht vollständig, sie soll nur zeigen, dass die Anforderungen an Berechtigungs- und Sicherheitskonzepte nicht unterschätzt werden dürfen.

Als übergreifender Gesichtspunkt muss ein Sicherheits- und Berechtigungskonzept mit hoher Priorität und so früh wie möglich erstellt werden, um rechtzeitig prüfen zu können, ob die zu gewährleistende Sicherheit mit vertretbaren Mitteln überhaupt erreicht werden kann beziehungsweise welche Punkte bei Verträgen mit externen Dienstleistern zu regeln sind.

Sind die Leistungsanforderungen realisierbar?

Microservices skalieren zwar gut mit der Anzahl der Anfragen, aber es wird leicht übersehen, dass ein einzelner Service für sich genommen fast immer längere Antwortzeiten aufweist, als das bei einer monolithischen Anwendung der Fall wäre. Dafür gibt es zwei Gründe. Der erste ist die bereits erwähnte Netzwerklatenz beim internen Aufruf weiterer Services. Dazu kommen jedoch auch noch zusätzlich notwendige Arbeitsschritte vor beziehungsweise nach jedem Serviceaufruf:

- Datenkonvertierung in oder aus einem meist textbasierten Format (JSON, XML etc.) beim Empfangen und Senden
- Ver- und Entschlüsselung, falls notwendig
- Überprüfung der Gültigkeit der Anfrage und der übermittelten Rechte
- Validierung der erhaltenen Daten

Bereits daraus ergeben sich Grenzen für die Antwortzeit, die selbst bei optimaler Netzwerkleistung nicht unterschritten werden können. Und

natürlich schlagen sich diese zusätzlichen Aufgaben auch im Hauptspeicherbedarf und der beanspruchten Prozessorleistung nieder.

Ist die mehrfache Realisierung von Funktionen akzeptabel?

Bei der parallelen und unabhängigen Entwicklung der Services besteht immer das Risiko, dass einzelne Funktionen nicht rechtzeitig oder nicht in der notwendigen Qualität fertig werden. Wenn Funktionen für die Gesamtanwendung essenziell sind, kann es sinnvoll sein, diese mehrfach implementieren zu lassen.

Eine ähnliche Situation ergibt sich, falls eingesetzte Fremdsoftware oder eine Programmiersprache zukünftig nicht mehr verwendet werden können oder sollen. Dieser Mehraufwand muss toleriert werden. Gleichzeitig ist es Aufgabe einer geschickten Personalführung, dafür zu sorgen, dass durch das Ersetzen von Services deren Entwickler nicht demotiviert werden.

Ist der notwendige Testumfang realisierbar?

Microservices benötigen umfangreichere Tests als monolithische Anwendungen. Das liegt einmal daran, dass jeder Service unabhängig geprüft werden muss. Da es keine umfassenden Integrationstests gibt, entfällt das implizite „Mitprüfen" von Teilfunktionen. Jeder Service braucht seinen eigenen, möglichst gut abdeckenden Test.

Zum anderen müssen alle im Zusammenhang mit der Kommunikation stehenden Funktionen abgedeckt werden. Dazu gehören die sicherheitsrelevanten Aufgaben ebenso wie Datenkonvertierungen und -verschlüsselungen. Außerdem muss die korrekte Reaktion auf Verbindungsfehler und -störungen getestet werden. Automatisierte Tests dieser zweiten Gruppe sind meist nicht trivial zu erstellen und verursachen erheblichen Entwicklungsaufwand.

Im dritten Abschnitt geht es nun um die entwicklerbezogenen Voraussetzungen, die ebenfalls nicht leichtgenommen werden sollten.

Sind die benötigten Entwickler in überschaubarer Zeit verfügbar?

Software wird von Menschen gemacht. Um schnell parallel entwickeln zu können, werden ausreichend Entwickler gebraucht. Dabei ist zu berücksichtigen, dass für diese Arbeit einige besondere Voraussetzungen erfüllt sein müssen. Gerade in kleinen und selbstständig arbeitenden Teams spielen Persönlichkeitseigenschaften und Erfahrungen eine große Rolle. Einerseits sollten nicht zu viele Neulinge (mit häufig unrealistischen Vorstellungen) vertreten sein, andererseits darf die Flexibilität und Freude am Neuen nicht völlig durch gewohnten Trott verschüttet werden. Insbesondere letztere Gefahr macht es bisweilen schwierig, die benötigten Entwickler einfach durch Umsetzung aus einer hierarchisch organisierten Projektorganisation, die Eigeninitiativen nur bedingt fordert, zu gewinnen.

Sind die Entwickler bereit, als Full-Stack-Entwickler zu arbeiten?

Microservices brauchen Full-Stack-Entwickler, die von der Erfassung und Formulierung der Anforderungen bis zur Sicherstellung des Betriebs alle Tätigkeiten beherrschen. Vor allem die Auseinandersetzung mit der Fachlichkeit und den fachlichen Auftraggebern kann sich als schwierig erweisen. Dabei sind gute kommunikative

Fähigkeiten gefragt. Darüber hinaus gilt es, sprachliche Hürden zu überwinden, nicht nur bei verschiedenen Muttersprachen, sondern auch im Hinblick auf das Verständnis des Anwendungsgebiets. Je komplexer das Anwendungsgebiet ist, desto langwieriger wird die notwendige fachliche Einarbeitung und damit wächst möglicherweise die Schwierigkeit, Entwickler für diese Spezialisierung zu motivieren. Dazu kommt als eher organisatorische Aufgabe die Sicherung einer effektiven Zusammenarbeit mit den jeweils betroffenen Fachbereichen.

Kann ein ausreichend motivierendes Umfeld gesichert werden?

Die Freiheiten bei der Entwicklung und schnelle Rückmeldungen zu den erreichten Ergebnissen wirken für sich bereits stark motivierend. Trotzdem sind auf Dauer weitere Maßnahmen zur Sicherung der Motivation nötig, denn man darf nicht vergessen, dass den Entwicklern auch einiges abverlangt wird. Neben der notwendigen Vertrautheit mit der Fachlichkeit muss der produktive Betrieb durch Bereitschaftsdienste oder Ähnliches abgesichert werden. Solche Fragen müssen durch angepasste und flexible Vergütungs- und Arbeitszeitmodelle angemessen gelöst werden. Eine übermäßige Fluktuation wirkt sich in den meist kleinen Teams besonders nachteilig aus.

Zum Abschluss noch eine Betrachtung, die auch eine Warnung an alle ist, die denken, dass das doch auch mit viel weniger Aufwand realisierbar sei. Natürlich lässt sich einiges einsparen – aber nicht ohne Folgen.

Microservices „im Kleinen“

Die genannten Herausforderungen sind selbstverständlich nicht unbemerkt geblieben. Deshalb gibt es viele Versuche, den Aufwand zu verkleinern, indem auf Redundanzen verzichtet wird. Das ist jedoch nur in beschränktem Umfang sinnvoll möglich. In vielen Situationen sind Abstriche an der freien Wahl von

- Programmiersprachen,
- Versionsverwaltung,
- Build-Werkzeug oder
- Datenbank-Software akzeptabel.

Schwieriger wird es, wenn in größerem Ausmaß standardisierte Bibliotheken eingesetzt werden, weil dadurch abgestimmte Versionswechsel notwendig werden. Es gibt weitere Möglichkeiten, die Unabhängigkeit der Service-Entwicklungen zu verringern. Allen gemeinsam ist, dass dabei die eingangs genannten Vorteile von Microservices zunehmend verlorengehen, und zwar ohne den Aufwand in vergleichbarem Umfang zu senken. Denn der größte Kostenblock, der durch die in die Struktur verlagerte Komplexität verursacht wird, bleibt davon unberührt. Alles, was mit dem Netzwerk zusammenhängt, muss in gleicher Weise eingerichtet und verwaltet werden. Allein deshalb ist es ein kostspieliger Ansatz, Microservices nur zu verwenden, um eine saubere Modularisierung zu erzwingen.

Das spricht nicht grundsätzlich gegen kleine Microservices-Anwendungen. Vor allem zum Erfahrungsgewinn sind sie nützlich. Der wirkliche Nutzen entsteht jedoch erst bei konsequenter Umsetzung, und das bedingt fast immer einen größeren Rahmen.

Fazit

Die aufgeführten Punkte stellen eine sicher unvollständige Liste von Kriterien dar, die bei einer Entscheidung für Microservices berücksichtigt werden sollten. Es geht nicht darum, diese Technologie als solche zu bewerten. Vielmehr soll diese Zusammenstellung helfen, die Chancen und Risiken des Einsatzes in konkreten Situationen realistisch einzuschätzen. Ein beeindruckendes Beispiel dafür, wie man es (unter den richtigen Voraussetzungen) richtig macht, wird auf GitHub *[2]* vorgestellt.

Quellen

[1] Röwekamp, Lars (2015): Microservices: (noch) keine einheitliche Definition. *https://jaxenter.de/microservices-keine-einheitliche-definition-16683*

[2] Gauder, Sebastian (2019): 5 years of food retail e-commerce. A microservice success story. Präsentation DEVelopers Köln Meetup, *https://github.com/devk-insurance/devk-meetups/blob/master/microservices-bright-vs-dark/2019-11-20-devk-meetup-microsevices-brightside.pdf*

Jürgen Lampe
S&N Invent GmbH Eschborn
juergen.lampe@sn-invent.de

Dr. Jürgen Lampe ist IT-Berater bei der S&N Invent GmbH. Er ist promovierter Mathematiker und war unter anderem als Hochschuldozent tätig. Seit mehr als 20 Jahren befasst er sich mit Design und Implementierung von Java-Anwendungen, hauptsächlich im Bankenumfeld. Sein spezielles Interesse gilt effizienten anwenderorientierten Softwarearchitekturen und domänenspezifischen Sprachen. Neben seiner Tätigkeit als Senior-Berater schreibt er Artikel für Fachzeitschriften und spricht auf Konferenzen. Er ist Autor des Buches „Clean Code für Dummies“.

2. + 3. DEZEMBER 2020

BERLINER EXPERTENSEMINAR

DOAG

NICOLAI PARLOG

Java after eight

KURSÜBERBLICK

In diesem Kurs werden die Java-Versionen 9 bis 15 beleuchtet. Dabei liegt der Fokus auf neuen Sprachfeatures wie Sealed Classes, Records, Text Blocks, switchExpressions und var, aber auch neue und erweiterte APIs sowie JVM- und Performance-Verbesserungen werden theoretisch vorgestellt und mit praktischen Übungen untermauert. Darüber hinaus werden der Migrationspfad von Java 8 zu 11 bzw. 15, der neue Release-Zyklus und die aktuelle Situation um Distributionen und Support besprochen.

ZIELGRUPPE

Erfahrene Java-Entwickler, die sich theoretisch und praktisch auf die neuen Java-Versionen vorbereiten möchten.

VORAUSSETZUNGEN

Einige Jahren praktische Erfahrung mit Java-Entwicklung und sicherere Java-8-Kenntnisse.

www.doag.org/go/expertenseminar_parlog