

Architekturarbeit in der Digitalen Ära

Cloud-native Architekt

Lothar Wieske

Architekten erarbeiten ein Architekturdokument; entstanden ist dieses Konzept vor ein bis zwei Jahrzehnten mit dem Übergang von technischen Projektleitern zu Architekten. Unified Process und Unified Modelling Language trugen das Ihre bei, um dem Rollenbild verbindliche Ergebnisse und Abläufe an die Hand zu geben. Und nun kommen Cloud-native und Agile. Oft fällt die Wahl auf Scrum – und das kennt doch gar keine Architektenrolle mehr. Was heißt das? Architektur im Team oder durch spezifische Arbeitszuweisungen – gegebenenfalls in Architektursprints?

Änderungen deuten sich allemal an. Was passiert aber, wenn man als Architekt drei Schritte zurücktritt und überlegt, wie Architektur im Zeitalter des Digitalen Wandels auch funktionieren könnte und sollte. Nicht mit dem einen neuen Konzept für Architektur. Davor gab es doch über die Jahre ganz viele – immer neue und immer andere. Sondern eher mit einigen Inspirationen und Kreationen, die zum Nachdenken und Weiterentwickeln einladen.

Architektenrolle und Architekturdokument

Die Architektenrolle in IT-Projekten gibt es eigentlich erst seit den 90er Jahren. Vorher oblagen entsprechende planerische und entwerfende Aufgaben einem technischen Projektleiter. Aber die Entwicklungen rund um objektorientierte Analyse und Design

im Allgemeinen und die Arbeiten der drei Amigos (Grady Booch, Ivar Jacobson, James Rumbaugh) rund um Rational Unified Process [RUP] und Unified Modelling Language [UML] im Besonderen lieferten einen kräftigen Schub in Richtung Softwarearchitektur. Und für die Architekten – zumindest der Software – war ein referenzierbarer Ordnungs- und Gestaltungsrahmen entstanden, mit dem sich trefflich modern arbeiten ließ.

Cloud-native Architecture – Irgendwas ist anders

Und nun Cloud-native [Dav19]. Wird da einfach nur die nächste Technologie-Sau durchs Dorf der Architekten getrieben und Arte-

Reinventing Organizations

Frédéric Laloux hat mit „Reinventing Organizations“ [Lal15] ein Grundlagenbuch für die integrale Organisationsentwicklung verfasst. Er liefert darin einen Überblick über die historische Entwicklung von Organisationsparadigmen, angefangen vom roten über das orange und gelbe hin zum grünen und türkisfarbenen Paradigma (s. Abb. 1). Neben der Unterscheidung liefert Laloux darin auch einen Leitfaden für den Weg hin zu einer ganzheitlich orientierten und sinnstiftenden (türkisen) Organisation.

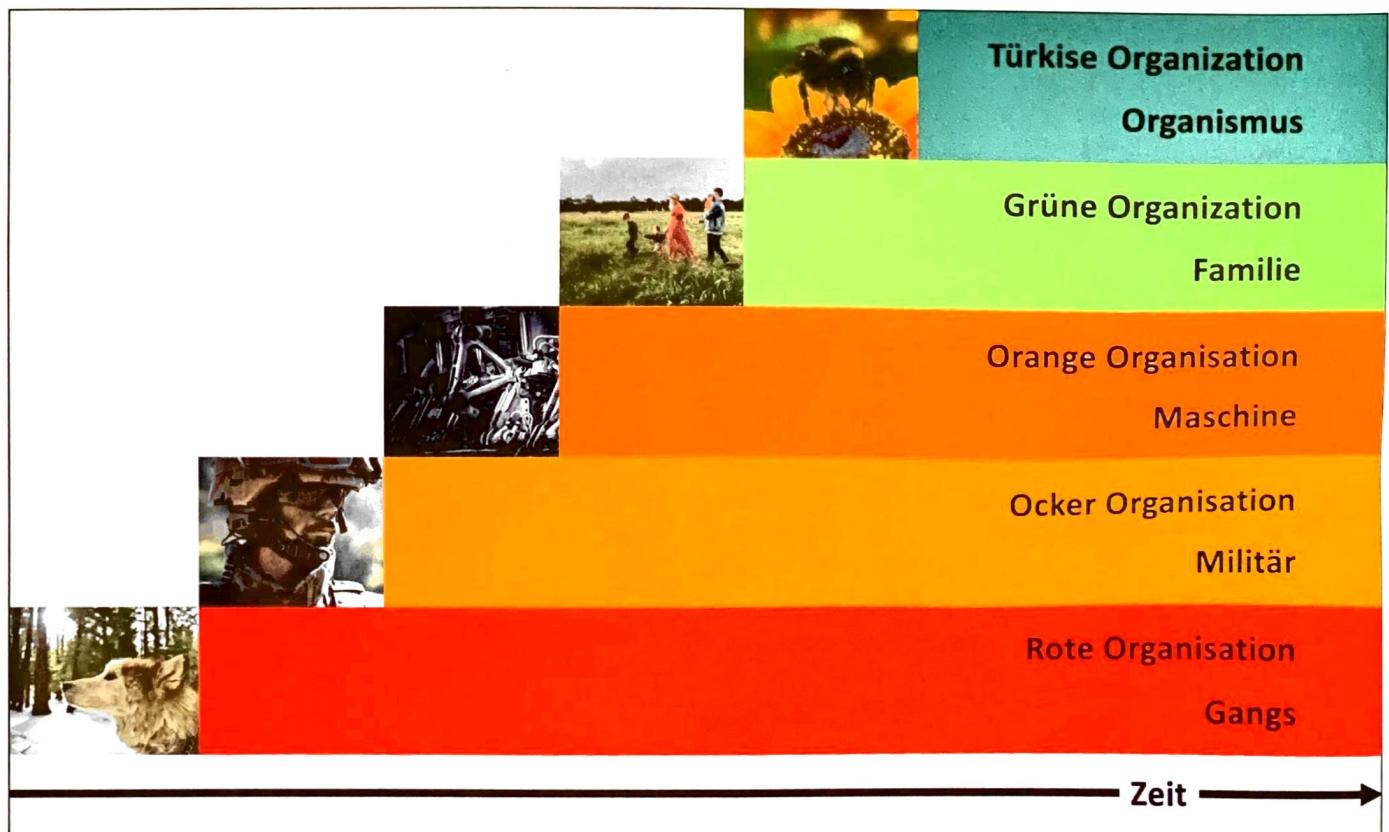
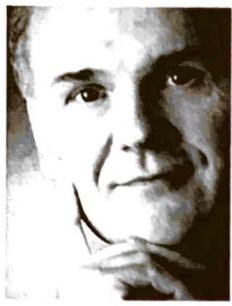


Abb. 1: Organisationsparadigmen, Bildrechte: Wolf <https://unsplash.com/photos/8bGyWm9tvyQ>, Organism <https://unsplash.com/photos/kLluCDjgoPI>, Machine <https://unsplash.com/photos/BslSDcQww0M>, Family <https://unsplash.com/photos/5NLCAz2wJXE>



Lothar Wieske arbeitet als Architekt in Kundenprojekten mit vielen Technologien rund um Digitalisierung. Bis 2009 arbeitete er in Leuchttürmen wie einer Digitalen Radiologie, einer Wertpapierverwaltung sowie für Miles-and-More u. a. bei IBM, Microsoft und Sun Microsystems. Als Enterprise-Architekt im Innovationsmanagement der Bahn arbeitete er

ab 2009 praktisch mit den drei großen Public Clouds (Amazon/Google/Microsoft) und Analytics-Clustern (Hadoop/Spark/Kafka/Flink). Mit den Jahren kamen dann IoT, Blockchain, Augmented Reality hinzu und zuletzt auch KI und Deep Learning. Neben technologischen Aspekten haben für ihn zunehmend auch Fragen zur wirtschaftlichen, unternehmerischen und gesellschaftlichen Entwicklung und Umsetzung der Digitalisierung an Bedeutung gewonnen.

E-Mail: lothar.wieske@web.de

len- und Artefaktverständnis darf Architekturarbeit nach Laloux eher im Sinne eines traditionellen (gelb in Abbildung 1) oder modernen (ocker) Organisationsmodells verortet werden. Im Allgemeinen und im Umfeld von Cloud-native läuft es meist agil und oft genug läuft das dann auf Scrum hinaus – und das kennt doch gar keine Architektenrolle(n) mehr. Heißt das jetzt nun Architektur im Team oder ohne großes Aufheben durch jeweils spezifische Arbeitszuweisungen an spezifische Teammitglieder, wenn halt was anliegt oder in spezifischen Architektursprints, in denen dann das ganze Team sich ausschließlich der Architektur widmet und reihum jeder sein Stückchen Architekturarbeit abbekommt?

Nach der Einschätzung und den Erfahrungen des Autors läuft es in modernen Cloud-native Umgebungen meist auf „Architektur im Team“ hinaus, wobei ein Diskurs im Team verhandelt und ausgetragen wird, wie man es mit den Trade-offs zwischen kontinuierlichem Kundennutzen und bedarfsgtriebener Weiterentwicklung oder Überarbeitung von Architekturentscheidungen hält. Beim zweiten Ansatz leidet der kontinuierliche Kundennutzen schon viel zu sehr, insbesondere droht die Architektur aber auch im Dickicht der Backlog-Priorisierungen unterzugehen und irgendwann einfach zu spät zu kommen, um noch strukturgebend etwaige Aufwände für die unmittelbare Entwicklung wie auch die spätere Wartbarkeit reduzieren zu helfen.

Sichten und Ergebnisse aufeinander beziehen

Und dann ist da ja noch DevOps – doch auch eher ein KulturtHEMA als ein Prozess- oder TechnologietHEMA; mit den letzten beiden Themen hat es sicher zu tun, aber den dominierenden Erfolgsfaktor bildet doch viel eher die kulturelle Ausgestaltung gemeinsamen Arbeitens und Lernens. In diesen Tagen bringen die meisten Mitarbeiter (noch) eine Entwicklungsgeschichte in einer der beiden Teildisziplinen mit und wachsen lehrender und/oder lernender Weise gemeinsam in eine verzauberte Arbeitsweise hinein.

Die unterschiedlichen Entwicklungslinien unterscheiden sich schon sehr deutlich. Das findet seinen deutlichen Niederschlag in den gängigen Prozessmodellen CMMI für die Entwicklung und ITIL für den Betrieb. Gut integriert waren beide doch eigentlich nie. Wie denn auch? CMMI als Reifegradmodell macht keine direkten Vorgaben oder Vorschläge, sondern sagt nur, nach welchen Kriterien der Reifegrad bewertet wird. ITIL ist dagegen ein Prozessmodell im eigentlichen Sinne und gibt „Good Practices“ vor.

ITIL geht eigentlich davon aus, dass viele der Prozessvorgaben gar nicht manuell abgearbeitet werden und über viele Stufen mit komplizierten Übergaben und Freigaben von Mitarbeiter zu Mitarbeiter laufen. Über all die Jahre wurde oft übersehen, dass man im ITIL-Sinne gut und gerne 80 Prozent der Prozesse durch entsprechende Vorgenehmigungen hätte automatisieren können. Mit einem solchen Automatisierungsgrad käme man bei der Bereitstellung von (virtuellen) Servern schnell in den Bereich weniger Stunden. Der tatsächliche Vorlauf lag in vielen größeren Unternehmen jedoch routinemäßig im Bereich einiger Wochen. In gewissem Sinne entsprechen die heutigen Verhältnisse in Public Clouds viel eher dem eigentlich gewollten Zielbild von ITIL als die langjährige Praxis in den Rechenzentren der Unternehmen.

Weiter oben war schon die Rede von der Bedeutung von Diagrammen für die Interaktion und Kommunikation rund um Architektur. UML hatte seinen Platz bei den Entwicklern. Aber bei den Betrieblern? Da war UML weder besonders bekannt noch eigent-

fakte und Rollen bleiben wie gehabt? Oder sind da Gelegenheit und Erfordernis zum Nachdenken über IT-Architektur in der Digitalisierung? In der Reihenfolge ihrer Dringlichkeit und Wichtigkeit werden für Kultur, Prozess und Technologie je zwei beispielhafte Inspirationen und Kreationen vorgestellt. Leider muss es damit in diesem Artikel – der Länge geschuldet – genug sein.

Architekturarbeit im New-Work-Stil

Um dem kulturellen Anspruch um Cloud-native gerecht zu werden, lohnt es sich, wichtige Entwicklungsstufen der kulturellen Arbeits- und Unternehmensorganisation nachzuzeichnen und damit architektonische Arbeit und ihre Organisationsformen entsprechend einordnen und aufeinander beziehen zu können.

Der Wirtschaftsphilosoph und Experte für New Work, Frédéric Laloux, veröffentlichte in seinem Buch „Reinventing Organizations“ [Lal15] aufrüttelnde Management-Ansätze und dachte Führung und Organisation jenseits eines „Im Prinzip haben wir im Management alles verstanden und im Griff“ neu (s. Kasten). Motiviert durch anregende Beispiele von Unternehmen, die es anders angehen, stellt Laloux in seinen Betrachtungen Fragen nach Sinn und Zweck von Unternehmen in den Vordergrund.

Frédéric Laloux konstatiert, dass Unternehmen jenseits des Geldes und der Steigerung von Umsatz, Gewinn und Marktanteilen Sinn stiften sollten. Traditionelle Konzepte der Zielvorgaben nehmen in seinem Denksatz unhinterfragt an, dass Unternehmen als leblose und passive Gestalten betrachtet werden können und müssen; Zielvereinbarungen sind für ihn schlicht Instrumente, mit denen das schlängernde Etwas gesteuert werden muss. Wenn man aber beginnt, Unternehmen als lebendige Organismen wahrzunehmen, die über Sinn reflektieren wollen und sich selbst orientieren können, kann Neues entstehen.

Komplexe Systeme funktionieren ohne hierarchische Bindung an eine zentrale Instanz. Das menschliche Gehirn besteht beispielsweise aus vielen Milliarden Zellen und keine einzelne beziehungsweise eine kleine Gruppe sagt den anderen Zellen, was sie zu tun haben. Komplexe Systeme – auch Organisationen und Menschen – funktionieren (besser) mit Selbstführung, und das Mehr an Echtheit führt zu mehr Lebendigkeit quer durch die gesamte Organisation.

Was heißt das für die Architekturarbeit – nicht nur, aber auch und besonders im Cloud-native Umfeld? Nun, im klassischen Rol-

lich beliebt – auch aufgrund der steilen Lernkurven und hohen Lizenzkosten von leistungsfähigen UML-Werkzeugen. Entsprechend schwierig waren in der Vergangenheit Großprojekte mit Chefarchitekten verschiedener Disziplinen, die sich erstmal über ihren Zugang zu Diagrammen verständigen mussten. UML war das nie. (Wie schwierig diese Verständigung sein kann, hat der Autor als Lead Architect eines großen Projekts zur Desktop-Virtualisierung noch in lebhafter Erinnerung. Damals halfen Flipchart, Marker und Pair-Drawing und anschließend: Einsatz der Digitalkamera. Damit wurde es interaktiv und lebendig.)

So auch hier die klare Empfehlung. Wenn andere Zugänge fehlen oder länger dauern: Einfach mit Flipchart/Marker/Handy loslegen und an das grafische Gedächtnis appellieren.

Wer als Architekt der Entwicklung mit Architekten im Betrieb zusammenarbeitet(e), wird schon mit dem stärkeren Interesse des Betriebs auch und gerade an dynamischen Darstellungen in Berührung gekommen sein. In moderner Cloud-native Denke geht es dann um Observability – in Lastfällen, aber auch bei Fehlerfällen – und um die Verhältnisse beim anschließenden Wiederanlauf. Neben kurzfristigen Betrachtungen gewinnen bei modernen Anwendungen auch längerfristige Betrachtungen beispielsweise bezüglich der Lastverteilungen oder (skalierten) Ressourcenbereitstellungen auf der Ebene vergangener Monate oder Jahre. Neben den dynamischen Sichten haben diese Darstellungen noch einen anderen Aspekt. Die „Vorwärts“-Betrachtung der Software-Visualisierung wird aufgegeben und zu einer „Rückwärts“-Betrachtung dessen, was schon ist, und gemessen und vermessen werden kann.

Das lässt sich dann natürlich auch wieder für die Software anwenden. Anstelle eines Diagramms der Pakete/Module/Komponenten/You-Name-It und der Hoffnung, dass in der Entwicklung das Vorkonzipierte auch so umgesetzt wird, tritt dann eine Generierung des Diagramms aus dem aktuellen Stand im Repository. Aktueller und stimmiger geht es nicht.

Zwei Herausforderungen ergeben sich aus dem gerade Gesagten.

- Architektur lebt auch und gerade von der Abstraktion. Es wird also nicht genügen, einfach 1:1 Grafiken aus dem Repository oder dem System zu generieren. Geeignete Abstraktionsschichten und Abstraktionssichten zu entwickeln, wird zur gemeinsamen Herausforderung für Entwicklung und Betrieb.
- Architektur lebt zudem auch in der Zeit. Die Evolution des Repository und/oder Systems wäre nach Ansicht des Autors auch ein darzustellender Aspekt der Architektur.

Die entstehenden vorwärts- und rückwärtsgerichteten Grafiken und/oder Tabellen führen in natürlicher Weise zu Pipelines auch für die Architekturdokumentation. Nicht immer ist das Repository online verfügbar. Um auch eine gewisse Offline-Fähigkeit der Dokumentation der Architektur oder von Architekturausschnitten (im Sinne von Mitnehmen und Lesen) zu schaffen, sind weitere Pipelines zur Generierung sinnvoll und notwendig. Damit wird die gesamte Architekturdokumentation auch Teil von Continuous Integration und Continuous Delivery und ist aktuell und konsistent. Ebenso werden dann auch in Verlängerung der Konzepte der Entwicklung und für Code entsprechende Verifizierungen und Validierungen von Architekturentscheidungen möglich.

Auch wenn die letzten Ausführungen eher technische Aspekte beleuchtet haben, zentral bleibt die Interaktion von Kollegen beim inkrementellen Herantasten an geeignete Darstellungsformen für Architekturaspekte, angefangen von Flipchart/Marker/Handy bis hin zu datengetriebenen Diagrammen des oben dargestellten Zuschnitts.

Die Dynamik von Diskussionen und Entscheidungen

Die Definition dessen, was informationstechnologische Architektur eigentlich kennzeichnet und ausmacht, hat über die Jahre zu einer Vielzahl von Ansätzen und Schwerpunkten geführt. Sie reichen vom einfachen Elemente/Beziehungen-Ansatz über Modelle oder Sichten und zuletzt immer mehr hin zu Architekturentscheidungen als solche getroffenen Entwurfsentscheidungen, deren Rücknahme oder Veränderung signifikanten Einsatz von zusätzlichen Aufwänden (Zeit, Geld, Komplexität) erfordern und damit schwierig wird.

Es ist in Projekten gängige und gute Praxis, solche Architekturentscheidung in einem Logbuch (Architecture Decision Records) einzutragen und damit den Überblick zu wahren, zumal einzelne Architekturentscheidungen nicht isoliert nebeneinander, sondern in einem Beziehungsgeflecht stehen.

Wer jemals vor einem inneren Dialog „Was für ein Unsinn! Wie konnte es dazukommen?“ stand, wird schnell darauf gekommen sein, dass alle Architekturentscheidungen einen Kontext besitzen. Und um den besser greifbar zu machen, reicht eine reine Ergebnisdokumentation nicht aus; eine Verlaufsdocumentation ist hilfreich, um das Hin und Her und die Abwägungen jenseits des Ergebnisses zu verstehen.

In vielen Projekten bietet eine Chat-Software (Slack, Teams) eine Drehscheibe für die Kommunikation. Wer hier einen eigenen Kanal für Architektur und noch ein wenig in eine Indizierung über Tags investiert, hat eine solche Verlaufsdocumentation in modernem digitalem Gewand ohne wesentliche zusätzliche Aufwände. Laufend fortgeschrieben und hochaktuell mit ein wenig Selbstdisziplin aller Beteiligten.

Minimal Viable Architecture und Minimal Viable Platform

In agilen Projekten wird viel Wert auf das Minimal Viable Product (MVP) gelegt, um frühest möglich die Kunden praktisch und konstruktiv einbeziehen zu können. Natürlich sollte diese frühe Erstlieferung an den Kunden auch auf einer Minimal Viable Platform erfolgen und nicht auf irgendeinem Fantasiekonstrukt. Damit wird eine Minimal Viable Architecture begonnen, die dann im Projekt iterativ und inkrementell fortgeschrieben werden kann.

Die meisten Cloud-native Architekturen kreisen ja doch um Kubernetes als Epizentrum. Da wird die Umsetzung mit einem ersten leeren Kubernetes-Cluster besonders einfach. Alle drei großen Public Clouds (Amazon, Google, Microsoft), und noch einige mehr, bieten Managed Kubernetes. Mit den entsprechenden Diensten EKS, GKE und AKS kann ein erstes leeres Cluster mit einem Einzeler auf der Kommandozeile oder einer kleinen Konfigurationsdatei im Repository aufgesetzt werden. Keine filigrane Feinarbeit ist hier gefragt, sondern ein pragmatischer Ansatz im Sinne „Just Enough Infrastructure“, um das erste Befüllen mit Funktionalität für den Kunden zu ermöglichen.

Deklarative Spezifikationen und evolutionsfähige Plattformarchitektur

Kubernetes automatisiert und orchestriert Container. Dafür abstrahiert es die Infrastruktur, indem es mit eigenen Ressourcen-Definitionen eine eigene Sicht darauf anbietet und mit dem zugehörigen API deren Manipulation ermöglicht. Mittlerweile kann die

statische Sicht auf die von Kubernetes vorgegebenen Ressourcen dynamisiert und durch eigene Definitionen von Ressourcen und deren Automatisierung erweitert werden.

Das besondere Merkmal dieses Ressourcen-API ist seine Möglichkeit zur deklarativen Verwendung. Zwar bietet Kubernetes keine eigene Konfigurationssprache beziehungsweise ein Konfigurationssystem. Aber das deklarative API kann von beliebigen Formen deklarativer Spezifikationen angesprochen werden. Im Kern läuft das auf die Unterscheidung zwischen „kubectl create“ und „kubectl apply“ hinaus:

- *kubectl create* steht für den imperativen Ansatz; damit bekommt das Kubernetes-API explizit gesagt, welche Ressourcen es erstellen, ersetzen oder löschen soll.
- *kubectl apply* läutet den deklarativen Ansatz ein; damit bekommt das Kubernetes-API nur gesagt, wie das Cluster nachher aussehen soll. Das API hat dann selbst die Aufgabe, dafür einen Plan zu entwickeln, und umzusetzen, was es dafür tun muss. Damit bekommt das Kubernetes-API nur noch implizit gesagt, welche Ressourcen es erstellen, ersetzen oder löschen soll.

Deklarative Ansätze bringen enorme Effizienz- und Produktivitätsgewinne. Auf diesen Vorteilen beruht ja auch der Erfolg der J2EE. Im Applikationsserver konnten Anteile der Entity- und Session-Beans deklarativ in Manifesten vorgegeben werden. Das reduzierte den vorherigen Aufwand für notwendige Systemprogrammierung in vorher ungekannten Ausmaß. Nun erfasst der deklarative Ansatz mit den deklarativen Spezifikationen von Kubernetes in einem enorm vergrößerten Geltungsbereich jenseits von zwei Software-Komponenten nunmehr für Infrastruktur, Plattform und gegebenenfalls Anwendung (Sidecars).

Diese Leichtigkeit, verbindliche Entscheidungen einfacher und schneller zu implementieren, schafft Freiheitsgrade für die Evolution der Architektur. Mit der leichteren Kopplung und Entkopplung von Aspekten können einerseits Entscheidungen zeitlich aufgeschoben werden, bis eine verbindliche und endgültige Festlegung wirklich unausweichlich geworden ist. Andererseits können so auch unterschiedliche Szenarien und Experimentalumgebung auf- und abgebaut werden, ohne die Entwicklung zu behindern, und mit dem entscheidenden Vorteil auch auf Plattformebene.

Observability und die Dynamik von Cloud-native Architektur

Es hat sich herumgesprochen, dass im Umfeld von dynamischen und verteilten Cloud-native Architekturen die traditionellen Ansätze von Monitoring, Logging und Alerting an ihre Grenzen stoßen. Spätestens im Problemfall muss das System vorbereitet sein, das Innere zu offenbaren und das eventuell aus dem Ruder gelaufene Systemverhalten verstehen und gegensteuern zu können.

Klassische Architekturansätze, beispielsweise mit dem oben angesprochenen UML, hatten eine ausschließlich qualitative Ausrichtung. Im Problemfall müssen in der Architektur schnell und zuverlässig abgreifbare Messpunkte vorgesehen und aktivierbar sein. Damit aber nicht erst im Problemfall eine hektische Suche nach „Wie denn, wo denn, was denn?“ losgeht, wird es zur Anfangsorientierung und Eingrenzung notwendig sein, auf aussagekräftige quantitative und abstrahierte Informationen zum Systemverhalten im Normalfall zurückgreifen zu können. Solche Informationen entsprechend sorgfältig abzuleiten, auszuwerten und darzustellen, gibt der Architekturarbeit eine neue Dimension und stellt eine bislang eher weniger geübte Praxis dar.

Zusammenfassung

Was passiert, wenn man als Architekt mal drei Schritte zurücktritt und überlegt, wie Architektur im Zeitalter des Digitalen Wandels und mit Cloud-native Architektur auch funktionieren könnte?

Der Artikel hat einige Inspirationen und Kreationen für Cloud-native Architekturen vorgestellt, die Kultur in den Vordergrund stellen und ebenso auf den Einsatz digitaler Methoden setzen.

Zentrale Ideen waren das kulturelle Klima im Architekten-team, das Entstehen einer Minimal Viable Architecture, Evolution dieser Architektur durch verzögerte und leicht änderbare Deklarationen.

Viel Spaß bei angeregten Diskussionen im Team und dem Weiterentwickeln und Verfeinern der hier skizzierten Ideen.

Literatur und Links

[App19] J. Appelo, Management 3.0: Leading Agile Developers, Developing Agile Leaders, Addison-Wesley, 2010

[CMMI] https://de.wikipedia.org/wiki/Capability_Maturity_Model_Integration

[Dav19] C. Davis, Cloud Native Patterns – Designing change-tolerant software, Manning, 2019

[For17] N. Ford, R. Parsons, P. Kua, Building Evolutionary Architectures: Supporting Constant Change, O'Reilly, 2017

[ITIL] https://de.wikipedia.org/wiki/IT_Infrastructure_Library

[Lal15] F. Laloux, Reinventing Organizations, Ein Leitfaden zur Gestaltung sinnstiftender Formen der Zusammenarbeit, Franz Vahlen, 2015

[New15] S. Newmann, Building Microservices, O'Reilly, 2015

[Rub12] K. S. Rubin, Essential Scrum: A Practical Guide to the Most Popular Agile Process, Addison-Wesley, 2012

[RUP] https://de.wikipedia.org/wiki/Rational_Unified_Process

[Sut15] J. Sutherland, J. J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, Random House, 2015

[UML] https://de.wikipedia.org/wiki/Unified_Modeling_Language



Schwerpunkt Cloud Native Architektur & Integration

SCHWERPUNKTTHEMEN

Architekturarbeit in der Digitalen Ära

10 Cloud-native Architekt

Lothar Wieske

Architekten erarbeiten ein Architekturendokument; entstanden ist dieses Konzept vor ein bis zwei Jahrzehnten mit dem Übergang von technischen Projektleitern zu Architekten. Und nun kommen Cloud-native und Agile. Oft fällt die Wahl auf Scrum – und das kennt doch gar keine Architektenrolle mehr. Was heißt das? Architektur im Team oder durch spezifische Arbeitszuweisungen?

Ein sagenhaftes Muster

14 Cloud-native und Business-Transaktionen

Ralph Soika

Was bedeutet Cloud-native eigentlich für die Geschäftswelt? Wie können in der Cloud, jenseits der großen Technologieversprechen, Geschäftsprozesse in Unternehmen und Organisationen erfolgreich umgesetzt werden?

EDITORIAL

3 Attacke der Cloud-Natives

AUS DER SZENE

6 Interview mit Maximilian Hille, Senior Analyst bei Crisp Research, über Low Code Development

EFFECTIVE JAVA

Im Dutzend billiger

53 Garbage-Kollektoren im Überblick

Michael Hunger

In Java bewegt sich was! Nicht nur die kommenden Sprachänderungen, wie Projekte Panama (Native Interface), Loom (Continuations) und Valhalla (Value Types), sind im Kommen, auch die schon beschriebenen Entwicklungen des Graal-Compilers und der GraalVM können sich sehen lassen. Sogar im allseits beliebten Bereich der Speicherbereinigung hat sich einiges getan.

Cluster One

18 Cloud Native Enterprise Architecture

Johannes Weigend, Johannes Siedersleben,

Mario-Leander Reimer

Cloud-native Entwicklung hat den Zenit des Gartner-Hype-Zyklus überschritten. Die erste Welle der Ernüchterung ist eingetreten. Wie kann Cloud-native Softwareentwicklung auf Unternehmensebene erfolgreich funktionieren und welche Rolle spielt die Softwarearchitektur dabei?

Aus Erfahrung gut

24 Den Betrieb automatisieren mit Kubernetes-Operators

Sophie Kuna, Daniel Bornkessel

Kubernetes hat sich zur Standardlösung für Container-Management entwickelt. Vor allem zustandslose Anwendungen, die lokal keine Daten speichern, sind so einfach zu betreiben. Datenbanken und andere Komponenten der Persistenzschicht werden jedoch oft weiterhin klassisch betrieben oder als Service hinzugebucht. Operatoren bieten unter anderem die Möglichkeit, auch zustandsbehaftete Software verlässlich auf Kubernetes zu betreiben.

JDK/JVM

Kaffee schwarz

56 Delegation über Vererbung in Java

Sven Ruppert

In dieser Kolumne sehen wir uns an, was der Unterschied zwischen Vererbung und Delegation sein kann. Oder besser ausgedrückt, warum ich Delegation bevorzuge und ein eher selten eingesetztes Feature in Java hervorheben möchte.

DER PRAKTIKER

Der dynamische Stellvertreter

59 Dynamische Proxys mit dem JDK umsetzen

Michael Vitz

In dieser Kolumne sehen wir uns eine der unbekannteren Programmierschnittstelle an, nämlich das Dynamic Proxy Class API.

Kubernetes AppOps Security

30 Security Context – Teil 1: Good Practices

Johannes Schnatterer

Werden Anwendungen auf managed Kubernetes-Clustern betrieben, ist auch der Betreiber des Clusters für die Sicherheit zuständig, oder? Nicht ganz! Dieser Artikel zeigt, welche Einstellungen im Security Context von Pods und Containern empfehlenswert sind.

Raus aus dem YAML-Tal!

34 Individualisierte Kubernetes-Deployments mit kustomize

Sebastian Sirch

Eine erste Version der Anwendung steht und nun soll sie auf ein Kubernetes-Cluster verteilt werden? Dann ist die Beschreibung des Deployments im YAML-Format der nächste Schritt. Dieser Artikel stellt kustomize als eine leichtgewichtige Lösung und interessante Alternative zu Helm vor, um mit dem daraus resultierenden Chaos umzugehen.

Es muss nicht immer Kubernetes sein

38 Microservice-Architekturen on-premise betreiben

Stephan Kaps

Dieser Artikel betrachtet die sich kontinuierlich entwickelnde Architektur einer Anwendungslandschaft hin zu Cloud-native. Dabei werden Werkzeuge für die schrittweise Anpassung der On-premise-Infrastruktur vorgestellt, ganz ohne Kubernetes.

FACHTHEMEN

Kap des guten Codes

44 Interview mit den Gründern von Cape Of Good Code

Michael Stal

Die deutsche Firma Cape Of Good Code wurde vor nicht mehr als einem Jahr gegründet. Sie beschäftigt sich mit der Analyse von Softwarearchitekturen und beweist, dass es auch hierzulande möglich ist, innovative Ideen kommerziell umzusetzen. JavaSPEKTRUM hat sich mit den beiden Gründern Egon Wuchner und Konstantin Sokolov unterhalten.

Spielend Testen lernen

48 LEGO® MINDSTORMS®-Lernlabor für IoT-Testing

Thomas Auer, Michael Felderer

Durch die Möglichkeit, physische und virtuelle Gegenstände fast beliebig miteinander zu vernetzen, hat die Bedeutung des Internet of Things (IoT) in den letzten zehn Jahren stark zugenommen. Damit wird auch die Testbarkeit von IoT-Projekten immer wichtiger. Dieser Artikel stellt eine spielerische Herangehensweise mit LEGO® MINDSTORMS® Education vor.

TOOL TALK

63 Best Practice Patterns (nicht nur) für Microservices

Thomas Ronzon

MAP, die Kollektion der Microservice API Patterns, hat absoluten Praxisbezug.

RÄTSELHAFTES JAVA

65 Denken Sie auch einmal an die Welt da draußen!

Thomas Ronzon

Wenn Sie versuchen, Ihre JVM zum Absturz zu bringen, erfahren Sie mehr über „die Welt da draußen“.

66 Vorschau und Impressum

Anzeige

DEIN ATlassian-PARTNER

- Deutschsprachige Lizenzberatung
- Installation und Konfiguration
- Ablösung von Altsystemen
- Einführungsstrategie
- Erweiterungs- und Integrationslösungen
- Prozess- und Managementberatung zu agilen Softwareprozessen und Softwaredokumentation



Unsere Schulungen

- Jira Plattform – fachliche Administration
- Jira Software für agile Projekte
- Confluence für Anwender
- Scrum Jumpstart
- Kanban Jumpstart



Trivadis Germany GmbH | Weinheimer Str. 68
68309 Mannheim | Tel. +49 621 71839-0
info@oio.de | oio.de | braintime.de

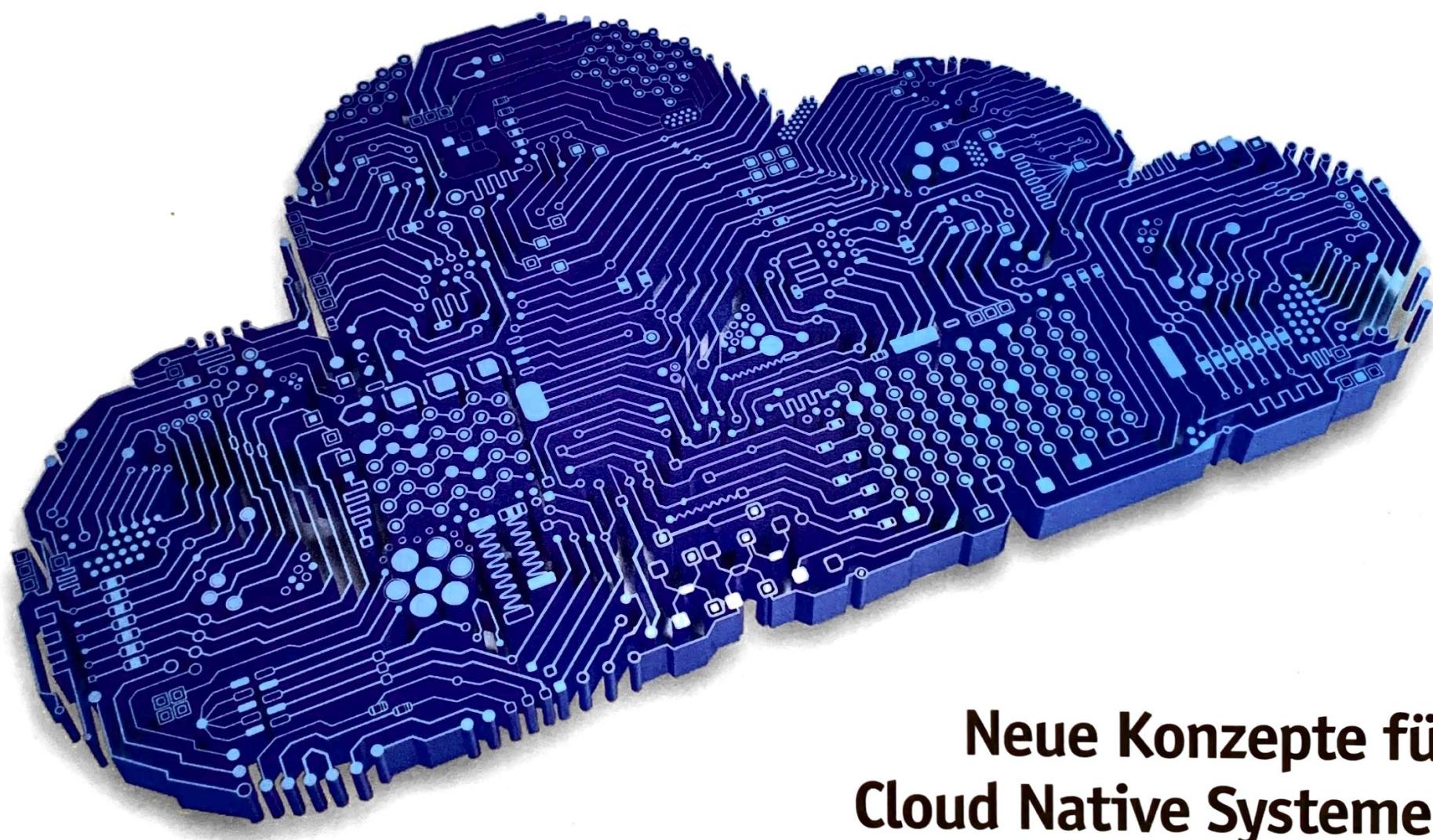


Platinum
Solution Partner
ENTERPRISE

JavaSPEKTRUM

Magazin für professionelle Entwicklung und digitale Transformation

Cloud Native – Architektur & Integration



**Neue Konzepte für
Cloud Native Systeme**

**Mit Kubernetes-Operators
den Betrieb automatisieren**



Interview

Maximilian Hille von Crisp Research
über beschleunigte Softwareentwicklung
und mehr Agilität für Unternehmen
durch Low Code Development

Fachthemen

**LEGO® MINDSTORMS®-Lernlabor –
spielend Testen lernen**
**Wie Stakeholder mit Analysetools
besser zusammenarbeiten**