Kubernetes ist nicht gleich Multi-Cloud

Der Einsatz von Kubernetes gilt oft als Synonym für Multi-Cloud, garantiert aber nicht automatisch die Portabilität von Anwendungen. Für eine erfolgreiche Multi-Cloud-Strategie gilt es Randbedingungen zu erkennen.

Eines der heißesten Themen im Cloud- und DevOps-Umfeld ist zurzeit Multi-Cloud. Doch auch wenn laut Gartner die meisten Unternehmen nach wie vor einen Hyperscaler für ihre Cloud-Workloads präferieren, verteilen gut drei Viertel diese mittlerweile auf zwei oder mehrere Hyperscaler (76 %). Viele Unternehmen sind der Überzeugung, containerisierte Ansätze und der Einsatz von Kubernetes wären der Heilige Gral, um einen Lock-in bei einzelnen Hyperscalern zu vermeiden und ihre Applikationen zwischen zwei Clouds zu verschieben. Den Zahlen von Gartner zufolge funktioniert das in der Realität aber in weniger als zwanzig Prozent der Fälle.

Dieser Artikel zeigt, warum Kubernetes nicht automatisch die Portabilität einer Applikation zwischen unterschiedlichen Clouds garantiert, und geht darauf ein, welche Argumente wann für und wann gegen ein Multi-Cloud-Szenario sprechen und wie sich echte Portabilität und weiterer Nutzen mit Multi-Cloud erzielen lässt.

Wie kommt man zur Multi-Cloud?

IT-Verantwortliche kommen zum Multi-Cloud-Szenario oft wie die Jungfrau zum Kind: sei es durch Developer in Digitalisierungsprojekten, die einfach die gängigsten Cloud-APIs einsetzen, oder durch Zukäufe. So entsteht „Multi-Cloud by Chance“, also unbeabsichtigt. Parallel dazu gibt es „Multi-Cloud by Strategy“: den gezielten Einsatz unterschiedlicher Clouds, um Ausfälle zu vermeiden, unterschiedliche Funktionalitäten oder Kostenprofile zu nutzen oder Compliance-Vorgaben zu erfüllen.

Heutzutage, wo drei Viertel der Unternehmen mehrere Cloud-Provider für dieselbe IT-Lösungskategorie einsetzen, ist es oft erschreckend, wie wenig sich IT-Infrastrukturverantwortliche, Softwarearchitekten oder -entwicklerinnen mit Kosten-Nutzen-Abwägungen von Multi-Cloud-Umgebungen befassen. Trotz immer ausgereifterer Tools von Microservice-Meshes über API-Management bis hin zu Cloud-Brokerage-Plattformen: Die schiere Komplexität von Multi-Cloud-Szenarien in großen IT-Landschaften ist überwältigend und nimmt stetig zu.

Der beste Ratschlag ist, Multi-Cloud für gleiche Workloads möglichst zu vermeiden und sich auf eine beschränkte, kuratierte Anzahl an Cloud-Providern, -Services und -APIs zu konzentrieren. Das Ziel: digitale Lösungen so zu designen, dass sie eine oder möglichst wenige Clouds einsetzen, die „good enough“ funktionieren und möglichst leicht zu managen sind.

Doch was ist, wenn das keine realistische Option ist, etwa weil das Unternehmen zu groß ist und zu viele unterschiedliche digitale Initiativen mit einer zu großen Vielfalt und Bandbreite bestehen? Dann bleibt nur, den Istzustand bestmöglich zu verwalten. Dabei wird Kubernetes häufig als einfache Lösung präsentiert. In der Praxis gibt es aber einiges zu bedenken.

Was bisher geschah …

Wir schreiben das Jahr 2008, erste Artikel über das Phänomen Cloud erscheinen in der deutschsprachigen Fachpresse. Begriffe wie Public und Private Cloud sind noch alles andere als etabliert, geschweige denn Hybrid- oder Multi-Cloud. AWS ist gerade sechs geworden und S3-Storage steht noch am Anfang seines zukünftigen Siegeszugs. Anfangs mit großer Skepsis betrachtet, setzen in den folgenden Jahren immer mehr Unternehmen auf die Innovationskraft der Cloud.

Im Zuge des AWS-Erfolgs treten weitere Anbieter auf den Plan, etwa Google, das Anfang der 2010er-Jahre bereits Datenzentren mit mehreren Tausend Servern besitzt und sein Know-how über Cloud-Computing zum Geschäftsmodell macht. Entscheidende Innovationen veröffentlicht der Suchmaschinenriese allerdings immer wieder als Open-Source-Projekte, so auch 2014 Kubernetes. Aus dem Wettbewerb der später als Hyperscaler bekannten Cloud-Provider entstehen weitere wegweisende Technologien, wegen ihrer elementaren Bedeutung auch „Commodity Services“ genannt. Häufig bleiben diese jedoch proprietär und somit nur bei bestimmten Anbietern verfügbar. Entwicklerteams beginnen deshalb mit den unterschiedlichen Cloud-Anbietern zu experimentieren, um verschiedene Features zu nutzen und Kosten zu optimieren.

Multi-Cloud-Herausforderungen

Bei Multi-Cloud-Szenarien mit Kubernetes-Infrastruktur gibt es viele Faktoren, die einer vollständigen Portabilität zwischen Cloud-Providern im Wege stehen können. Allgemein gilt: Je mehr eine Recheninstanz auf Dienste eines bestimmten Anbieters angewiesen ist, desto schlechter steht es um die Portabilität. Selbst wenn man alle anbieterspezifischen Funktionen vermeidet, kommt man nicht umhin, sich mit unterschiedlichen Provisionierungs- und Konfigurationsprozessen zu befassen.

Es gibt auch externe Dienste, die für den Betrieb eines Systems entscheidend sein können, wie eine gemanagte Datenbank, deren Funktionen nur mit erhöhtem Aufwand in einem Kubernetes-Cluster nachgeahmt werden können (etwa über einen Database Operator). Ein anderes Beispiel ist das Identitätsmanagement, das sich zwar über Services wie Dex (OpenID Connector) abstrahieren lässt, aber die Abhängigkeiten zu Diensten wie Azure AD nie vollständig auflöst.

Viele grundlegende Cloud-Services, die auf den ersten Blick vergleichbar und kompatibel erscheinen, können verborgene Inkompatibilitäten haben, etwa Unterschiede bei Performance, SLA oder Parametern. So lassen sich für persistenten Storage beispielsweise sowohl der Blockspeicherdienst Elastic Block Storage (EBS) bei AWS als auch sein Bruder GCE Persistent Disk bei GCP mithilfe der Storage Classes und des Container Storage Interface (CSI) in Kubernetes einbinden. Bei AWS können Nutzer jedoch die I/O-Performance IOPS pro Gigabyte festlegen, während das bei GCP nicht funktioniert. Das kann dazu führen, dass die notwendige Diskperformance für den Workload nicht zur Verfügung steht und die Applikation sich anders verhält oder sogar Fehler produziert.

Stateful-Systeme wie Datenbanken oder ein Data Warehouse erschweren die Portabilität zusätzlich. Abgesehen von den enormen Kosten, große Datenmengen zu transferieren, können kleine Unterschiede in der Performanz oder im Durchsatz entscheidend sein.

Ist ein Teil der Infrastruktur nicht containerbasiert und Teil des Kubernetes-Clusters, wird Reengineering für die Portierung nötig, was mit größeren Schwierigkeiten verbunden ist. Darauf sollten DevOps-Teams gefasst sein.

Für eine gelungene Multi-Cloud-Applikationsimplementierung gilt es nicht nur, diese Unterschiede zu identifizieren und ihre Auswirkungen auf die Applikation zu untersuchen, sondern sie im Zuge der Evolution der Anbieter kontinuierlich zu evaluieren. Wie das aus praktischer Perspektive aussieht, zeigt der nächste Abschnitt.

Gute Absichten, schlechte Aussichten

Die Verlockungen der Multi-Cloud sind zahlreich: Ausfälle umgehen, die Vorteile von Container- und Cloud-native-Technologien nutzen, Regionen anbinden, die vielleicht nur bei einem Hyperscaler verfügbar sind, oder gar der Aufbau eines Kubernetes-Clusters, der eine Control-Plane über mehrere Datenzentren hinweg schafft. Egal, welches Vorhaben den Anstoß für Multi-Cloud by Strategy gibt, ab irgendeinem Punkt gerät die Implementierung meist ins Stocken.

Das folgende Szenario ist sicherlich für zahlreiche Unternehmen so oder so ähnlich typisch: IT-Verantwortliche nehmen sich vor, eine Plattform für Applikationsentwickler zu schaffen, damit diese ihren Code in Container-Images verpacken und zusammen mit einer kurzen YAML-Beschreibung deployen können, ohne sich mit den Feinheiten einzelner Clouds beschäftigen zu müssen. Das SecOps-Team hingegen ist naturgemäß bestrebt, für jede Cloud entsprechende Sicherheitsmechanismen, Backups und Monitoring-Möglichkeiten zu schaffen.

Das notwendige Bindeglied stellen dann häufig Plattformteams dar, auch SRE (Site Reliability Engineering) genannt, ein Ansatz von Google, der Überlegungen aus der Softwareentwicklung auf Infrastruktur- und Betriebsprobleme anwendet. Diese Teams sollen die Anfragen der Entwicklerinnen und Vorgaben seitens der Operations- oder Securityteams miteinander harmonisieren. Was zunächst nach einer gelungenen Arbeitsteilung klingt, entpuppt sich schnell als neues Problem. Die Zusammensetzung von Teams und die Zuständigkeiten können sich mit der Zeit ändern, was zwangsläufig zu Reibungsverlusten führt. Nicht zu vergessen die Perspektive des Managements, das Ansprüche im Hinblick auf die Kostenstruktur des Cloud-Betriebs oder des Rollouts neuer Funktionen stellt.

Alle diese unterschiedlichen Anforderungen lassen sich in der Regel nicht so leicht in Übereinstimmung bringen, da meist verschiedene Clouds betroffen sind. Von einheitlichen Schnittstellen bis hin zu Kompatibilitätsfragen gilt es viele Herausforderungen zu lösen, bis aus diesem Gewirr ein beherrschbares Ganzes wird, das den Namen Multi-Cloud verdient hätte.

Abgeleitet von den geschilderten Herausforderungen entstanden drei Guidelines, um Kubernetes als echten Multi-Cloud-Enabler zu nutzen (siehe Kasten „Wegweiser in eine bessere Multi-Cloud-Zukunft“). Diese sind architektur- und designagnostisch.

Diese Guidelines dürften auf der Multi-Cloud-Journey helfen, den richtigen Weg einzuschlagen. Gleichzeitig erhöhen sie jedoch auch die Komplexität einer Applikation. Ohne kontinuierliches und methodisches Testen wird die erfolgreiche Umsetzung von Multi-Cloud-Rekonfigurationen sonst nicht gelingen.

Praxisbeispiel: Petra und Klaus in der Multi-Cloud

Ein kleines praxisnahes Codebeispiel soll diese Punkte verdeutlichen. Es nutzt einige wohlbekannte und verbreitete Tools und Bibliotheken. Interessierte können es per GitHub-Repository nachstellen (siehe ix.de/zrrb). Terraform dient zur Provisionierung von Infrastrukturelementen über eine deklarative Beschreibung der Ressourcen, die mit vielen

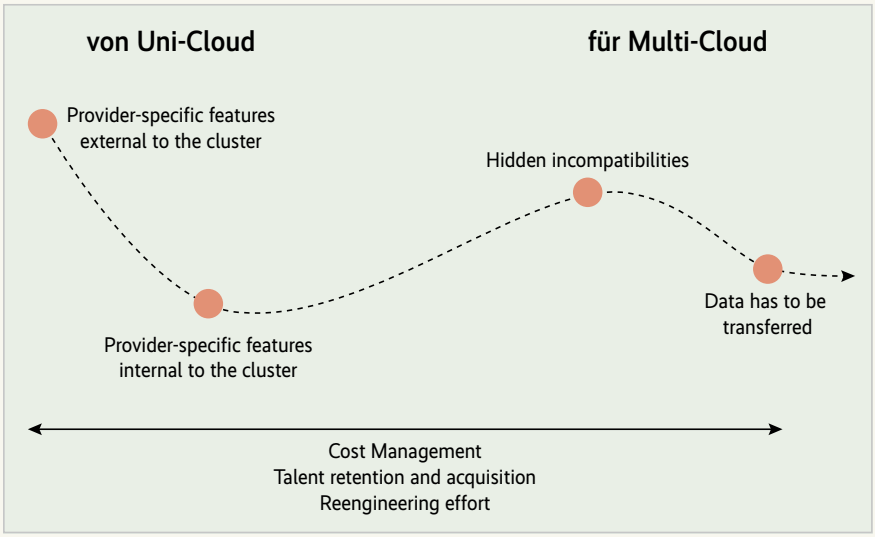
Kubernetes: Ein Haus mit festem Fundament?

Das Dilemma der Multi-Cloud sei hier mit einer kleinen Erzählung verdeutlicht: der Geschichte von den drei kleinen Schweinchen (Cloud-Architekten).

Der erste Cloud-Architekt baute seine Mono-Cloud aus Stroh, doch der böse große DDoS-Angreifer pustete sie einfach weg. Der zweite wusste es besser und baute seine Hybrid-Cloud aus Holz, in der Überzeugung, dass dieses Set-up standhafter sei. Doch auch hier dauerte es nicht lang und der große, böse und hungrige Workload brachte das Haus zum Einsturz. Beide Cloud-Architekten flüchteten sich schlussendlich in die Multi-Cloud zu ihrer Kollegin, die ihre Plattform auf ein Kubernetes-Fundament gebaut hatte.

Sie lachten erleichtert über ihr Erlebnis und wiegten sich in Sicherheit, doch die nächste unangenehme Überraschung in Gestalt des erbosten CFO ließ nicht lange auf sich warten. Als dieser, fuchsteufelswild über die ausufernden Kosten des Spaßes, durchs Fenster hereinschaute, bangten sie um ihre Karriere. Von da an beschlossen sie, guten Mutes gemeinsam an einer nachhaltigen und sicheren Lösung für ihr Cloud-Haus zu arbeiten.

Die Moral von der Geschichte: Viele Unternehmen sind der Überzeugung, sie seien mit Kubernetes und einer Multi-Cloud-Umgebung gut aufgestellt und könnten – da sie ja Cloud-native sind – ganz einfach zwischen Services und Providern hin- und herwechseln. Doch Kubernetes heißt nicht immer „Ende gut, alles gut“.



Bei einem Wechsel von ausschließlich einem Provider hin zu einem Multi-Cloud-Ansatz kommen mehrere Faktoren ins Spiel (Abb. 1).

Wegweiser in eine bessere Multi-Cloud-Zukunft

Guideline 1:

- Nutzen und entwickeln Sie Module und Bibliotheken, die von vornherein mit mehreren Cloud-Providern kompatibel sind.
- Gibt es keine fertigen Module für anbieterunabhängiges Arbeiten, empfiehlt sich das Dependency-Inversion-Prinzip: Man baut ähnliche Module für alle Cloud-Anbieter und injiziert sie über ein einheitliches Interface in seine Lösung, etwa auf der Ebene Infrastructure as Code.
- Providerspezifische Informationen übergibt man dem Modul entweder zur Laufzeit als Parameter oder das Modul kann sie selbstständig aus der Umgebung ableiten, in der es gerade läuft.
- Es kann verlockend sein, für jeden Hyperscaler individuelle Infrastruktur-Stacks zu pflegen. Das führt jedoch zwangsläufig zu technischen Silos, die die Wiederverwendbarkeit der eigenen Lösungen reduzieren.
- Diese Guideline ist stark am Entwurfsmuster „Ports and Adapters“ angelehnt und lässt sich analog implementieren. Es schlägt vor, dass Zugriffe auf externe Dienste über austauschbare Konnektoren erfolgen sollten, die die Anbieterlogik abstrahieren.

Guideline 2:

- Wo ein Dienst deployt wird, sollte für andere, darauf angewiesene Dienste keine Rolle spielen. Gleichzeitig sollte seine Schnittstelle immer gleich bleiben.
- Die Kommunikationslogik innerhalb des Ökosystems eines Anbieters sollte identisch mit der zwischen den verschiedenen Anbieter-Ökosystemen aufgebaut sein.
- Mit anderen Worten: Setzen Sie auf API-basierte Kommunikation zwischen unterschiedlichen Diensten.
- Fälle, in denen Zugriffsberechtigungen für Applikationen erteilt werden, die ausschließlich in einer spezifischen Ressource verfügbar sind (wie gemanagte Identitäten in Azure), sind zu vermeiden. Stattdessen sollte sich jeder Dienst explizit authentisieren, wenn er mit anderen Diensten kommuniziert.

Guideline 3:

- Die Funktionalität des Systems sollte gleich bleiben, unabhängig von der internen Organisation.
- Man sollte das System kontinuierlich im Hinblick auf Kriterien wie Verfügbarkeit, Latenz et cetera überwachen. Außerdem ist eine bewusste Entscheidung darüber zu fällen, wie eine ideale Organisation des Systems aussieht, ob man ausschließlich auf einen bestimmten Anbieter setzt oder ob es einen Anbietermix geben soll.
- Diese Entscheidungen sind im gesamten System durchzusetzen, indem Services zerstört und neu aufgebaut werden. Letzteres erfolgt gemäß dem bevorzugten Deployment-Muster, sei es Canary (inkrementelles Rollout), Blue-Green (partielles Testen neuer Releaseversionen mit verringertem Traffic beziehungsweise verringerter Nutzung) oder ein beliebiges anderes.
- Die Autoren bevorzugen den Begriff Reorganisation anstelle einer vollständigen Migration, da man sich jederzeit entscheiden kann, für eine bestimmte Funktion immer auf einen konkreten Anbieter zu setzen. Es kann beispielsweise sinnvoll sein, einen einzelnen Anbieter für das Identitäts- und Zugriffsmanagement zu nutzen.
- Der Begriff Entscheidung wird hier genutzt, um das Event zu beschreiben, das eine Reorganisation von Modulen triggert. Doch es wurde noch nicht näher festgelegt, wie und durch wen diese Entscheidung getroffen wird. Die Implementierungen können von einem vollständig automatisierten und dezentralisierten Prozess bis hin zu manuellen und zentralisierten Handgriffen reichen. Beim erstgenannten Ansatz managt jeder Teil des Systems sein eigenes Deployment bei dem am besten geeigneten Provider, während weiterhin die Kommunikationsfähigkeit mit anderen Services sichergestellt wird. Beim zweiten entscheidet ein SRE-Team über die bestmögliche Konfiguration des Systems und nutzt eine zentrale Registry, um den Standort jedes Dienstes zu speichern.

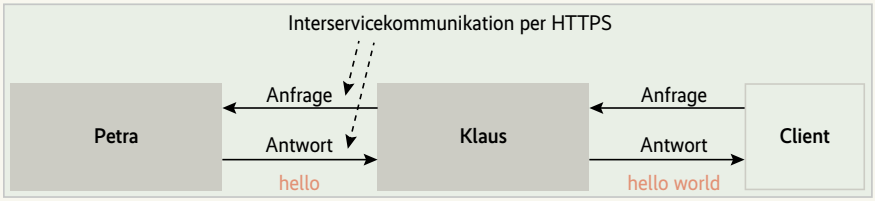
Cloud-Providern kompatibel ist. Die Bibliothek Terratest stellt die Muster und Helper-Funktionen für Infrastrukturtests bereit. Das Python-Webframework FastAPI dient im Beispiel als Mikroserviceapplikation. Das Deployment erfolgt auf den Managed-Kubernetes-Plattformen von Azure und AWS, Azure Kubernetes Service und Amazon Elastic Kubernetes Service.

Es gibt zwei Webservices, genannt Petra und Klaus. Petra besitzt nur einen Endpunkt, der stets `hello` ausgibt. Auch Klaus besitzt nur einen Endpunkt, der Petra aufruft und der Antwort das Wort `world` hinzufügt. Doch woher weiß Klaus, wo Petra sich befindet? Petras IP-Adresse ist als eine Umgebungsvariable von Klaus gesetzt. Dies kann beispielsweise über einen zentralen Konfigurationsmanager erfolgen.

Beide Webservices werden als Container in einem eigenen Kubernetes-Cluster deployt. Der Cluster ist im Terraform-Modul definiert, das so doppelt instanziiert wird. Das Modul erhält den String `provider` als Parameter, der entscheidet, ob der Cluster auf AWS oder Azure laufen soll. Für die Zwecke dieser Demonstration ist die Architektur des Systems so gestaltet, dass Klaus stets auf Azure läuft. Für diese Entscheidung kann es viele gute Gründe geben. In einer realen Multi-Cloud-Umgebung fällt diese Entscheidung situativ von Fall zu Fall.

Das gesamte System wird auf seine Multi-Cloud-Fähigkeit hin getestet. Es existieren zwei automatisierte Terraform-Tests: Der für die Module prüft, ob diese laufen, der für das System testet, ob der Webserver von Klaus `hello world` ausgibt. Das Unit-Testing für Klaus wird auf Azure, das für Petra sowohl auf Azure als auch auf AWS durchgeführt. Der Systemtest berücksichtigt zwei Konfigurationen – Konfiguration 1: beide Cluster laufen auf Azure; Konfiguration 2: Petras Cluster läuft auf AWS, der von Klaus auf Azure.

Auch wenn das Beispiel im Vergleich zu einer reellen Applikation extrem vereinfacht ist, dient es dazu, die praktische Bedeutung



Das Verhalten des Systems im Praxisbeispiel ist unabhängig von seiner internen Organisation konstant. Alle Dienste sind unabhängig und können bei unterschiedlichen Providern laufen (Abb. 2).

In iX extra 10/2022: Security: Neue Trends und Produkte zur it-sa

Immer perfidere Angriffsstrategien erfordern auch angepasste Gegenmaßnahmen. Hier hat sich das noch relativ junge Sicherheitskonzept Zero Trust als vielversprechend erwiesen. Es behandelt Zugriffe von außen wie von innen gleich, ohne internen Nutzern oder Geräten einen Vertrauensvorschuss zu gewähren. Die leider schlagkräftigsten Angriffe zielten jüngst auf die Supply Chain. Sie verursachten maximalen Schaden durch das Verteilen kompromittierter Software etwa als Updates. Ein geringer Aufwand für das Infizieren hatte katastrophale Auswirkungen. Hier sind Experten gefragt, die die Angriffe analysieren und Anleitungen zur Abwehr liefern. Einen anderen Schutz bieten Security Operations Center, die mit KI / maschinellem Lernen enorme Mengen an Netzverkehr auf Angriffe und Bedrohungen durchforsten.

All diese Ansätze sollen Unternehmen und Organisationen dabei unterstützen, aktuelle Herausforderungen und Risiken zu bewältigen. Orientierungshilfe geben auch die begleitenden Marktübersichten.

Erscheinungsdatum: 22.09.2022

Die weiteren iX extras

Ausgabe	Thema	Erscheinungsdatum
11/2022	**Storage: Neue Storage-Techniken**	**20.10.2022**
12/2022	**Hosting: Colocation**	**24.11.2022**

der vorgeschlagenen Guidelines zu erkunden. Die Konstruktion von Terraform-Modulen folgt aus der ersten Guideline, da das Modul den Provider abstrahiert, der für das Deployment der Ressource zum Einsatz kommt. Wie die Services miteinander kommunizieren, ist ein Beispiel für Guideline 2: Klaus interessiert es nicht, wo Petra läuft. Die Funktion des Systems, also die Ausgabe des altbekannten `hello world`, blieb gleich, wie in Guideline 3 vorgeschlagen. Die Entscheidung fiel in diesem Fall für einen manuellen Prozess, den die erneute Anwendung von Terraform umsetzt.

Jenseits von fortlaufendem und methodischem Testen sind sinnvolle Tooling- und Abstraktionslayer entscheidend, um die Komplexität zu reduzieren. Beispielhaft seien die folgenden Projekte genannt:

- **AI Sprint** stellt einen anbieterunabhängigen Abstraktionslayer für Edge-AI sowie ein Toolset für föderales Lernen und eingebettete Anonymisierung nebst anderen Anwendungen zur Verfügung.
- **Gaia-X:** Initiative für eine zukünftige Generation von Cloud-Infrastruktur, die Kompatibilität, Transparenz und europäische Datenhoheit verbessert.
- **Open Application Model:** portierbarer Standard für das Deployment von Applikationen mit klarem Fokus auf eine Auslieferung in hybriden Umgebungen.

Fazit

Bei Multi-Cloud-Deployments sollte man sich bewusst sein, dass allein Kubernetes keinesfalls die Portabilität einer Anwendung garantiert. Wenn dann nach wie vor mehr Gründe für die Multi-Cloud als dagegensprechen, sollten Systeme so konstruiert sein, dass sie echte Portabilität sicherstellen. Die Tools und Architekturmuster können eine Umgebung in ein komplexes Durcheinander verwandeln. Allein disziplinierte und methodische Arbeit wird langfristig zum Erfolg einer Multi-Cloud-Strategie führen. (avr@ix.de)

Quellen

Links zum GitHub-Repository und zu weiterführenden Informationen und Onlineartikeln: ix.de/zrrb

Leonardo Benitez
ist Data Scientist und Machine Learning Developer bei Skaylink.
Silvio Kleesattel
ist Technology und Innovation Lead bei Skaylink, einem führenden europäischen Cloud-Managed-Service-Provider.
Maximilian Zinke
ist Expert Consultant bei Skaylink im Bereich Kubernetes und Cloud-Native Solutions.