NOMAD-LEITFADEN SERVERLESS COMPUTING 2021

Von: Pavan Belagatti

JFrog Ltd. | www.jfrog.com

NOMAD-LEITFADEN SERVERLESS COMPUTING 2021

Einführung 2

Wie hat sich Serverless entwickelt? 2

Was ist Serverless überhaupt? 3

Warum ist Serverless notwendig? 5

Wie funktioniert Serverless? 6

Die zwei Grundformen serverloser Architektur 7

Sind Serverless und Container dasselbe? 8

Serverless Offerings by Different Cloud Platforms 8

Serverless auf AWS, Google Cloud und Microsoft Azure 9

AWS Lambda 9

Azure Functions 10

Google Cloud Functions 10

State of Serverless Report 2020 11

Beispiel einer Serverless-Architektur 14

Use Cases für Serverless 15

Realbeispiele aus der Praxis 16

LeapFrog: Wechsel auf Microservices, dann auf Serverless 16

ShoutOUT: Wechsel von Docker zu Serverless 17

LEGO.com: Umzug des Backends auf Serverless 18

Netflix: Codierung, Backups und Security 19

Coca-Cola: Deutlich reduzierte Automatenkosten 19

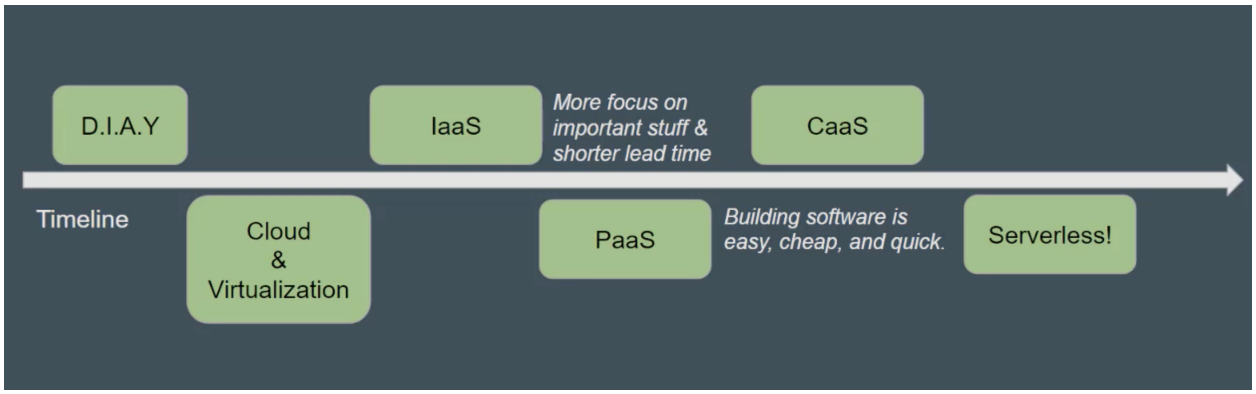
BBC: Neue Homepage auf Serverless 20

Fazit 22

Einführung

Mit zunehmender Beliebtheit und Verbreitung von Services wie AWS Lambda und Fargate richtet sich das Interesse der Wirtschaft insgesamt mehr auf serverlose Technologien im Stack. Der Wechsel zu Serverless ist eine willkommene Lösung für die besonderen Anforderungen moderner DevOps-Unternehmen. Serverless erleichtert insbesondere die Skalierung Cloud-gestützter Architekturen. Dem Themenreport 2019 von Forrester zufolge nutzen 49 % der Unternehmen bereits Serverless-Architekturen oder haben es in den nächsten zwölf Monaten vor.

Wie hat sich Serverless entwickelt?



Die Entwicklung bis zu Serverless.

Quelle: The TechCave

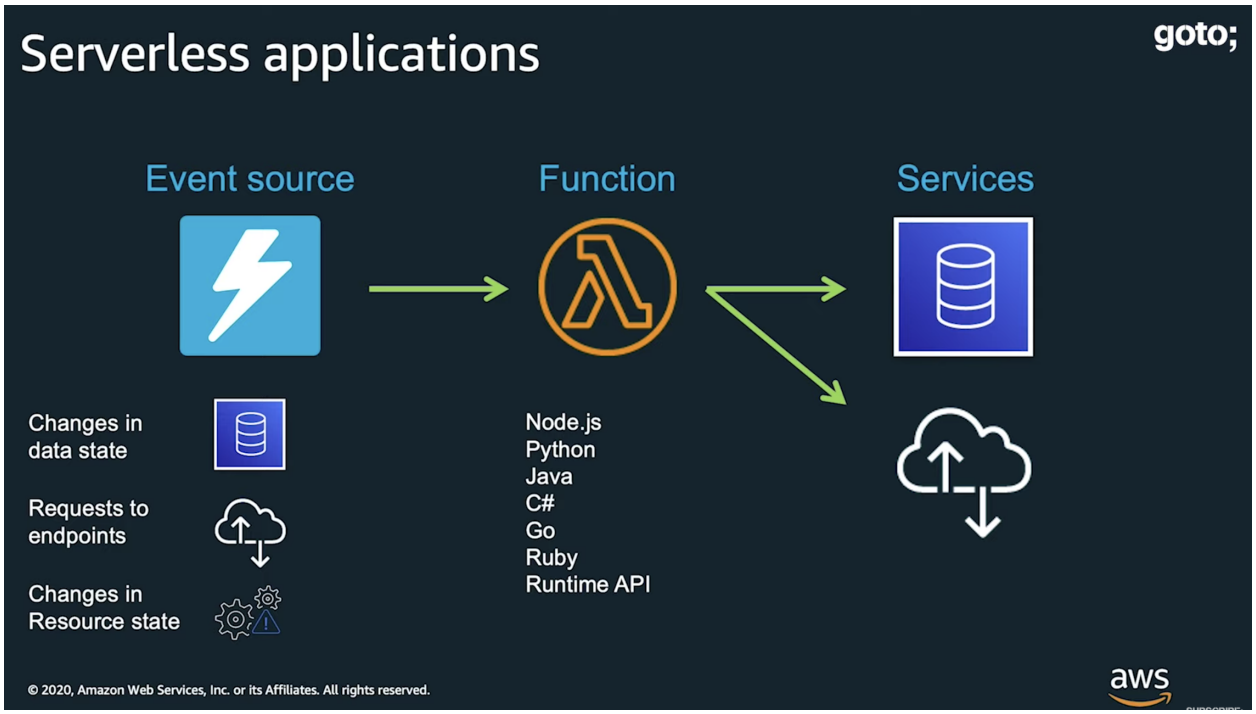
Es ist noch gar nicht lange her, dass sowohl Unternehmen als auch Einzelanwender ihre eigene Software und Hardware kauften und sich darum kümmerten, von der Netzwerkinfrastruktur über Speicher und Server bis hin zu anspruchsvollsten Aufgaben, für die man spezialisierte Teams und Experten einstellte. Es war eine Art Do-it-all-yourself-Ansatz (DIAY).
Dann begann man, bestimmte Aufgaben auszulagern. Dann kam die Cloud in Kombination mit Virtualisierungstechnologien und legte den Grundstein für Infrastructure as a Service (IaaS) und Platform as a Service (PaaS). Diese Technologien und Trends machten noch mehr Outsourcing möglich, und rückten damit auch die Business-Logik stärker in den Mittelpunkt. Die Vorlaufzeit neuer Software wurde kürzer, neue Features waren einfacher zu implementieren. Dann kam die Welle der Containerisierung, und es gab abermals neue Möglichkeiten, diesmal als Container as a Service (CaaS).
Vor dem Hintergrund dieser Entwicklung bzw. Revolution ist Serverless noch einmal ein Schritt nach vorne.
Bei all den Fortschritten im Laufe der Zeit war das Ziel immer, den Entwicklern etwas von der Müh und Not der Software-Programmierung abzunehmen. Allerdings gab es da immer noch die serverseitige Logik mit Code und Funktionalitäten, die weiterhin für Kopfschmerzen sorgte. Genau an diesem Punkt kam der Serverless-Trend – und damit sind die Schmerzen auf Serverseite fast verschwunden, denn diese Technologie kümmert sich um die gesamte serverseitige Logik, sodass sich Unternehmen ganz auf ihre Business-Logik konzentrieren können.

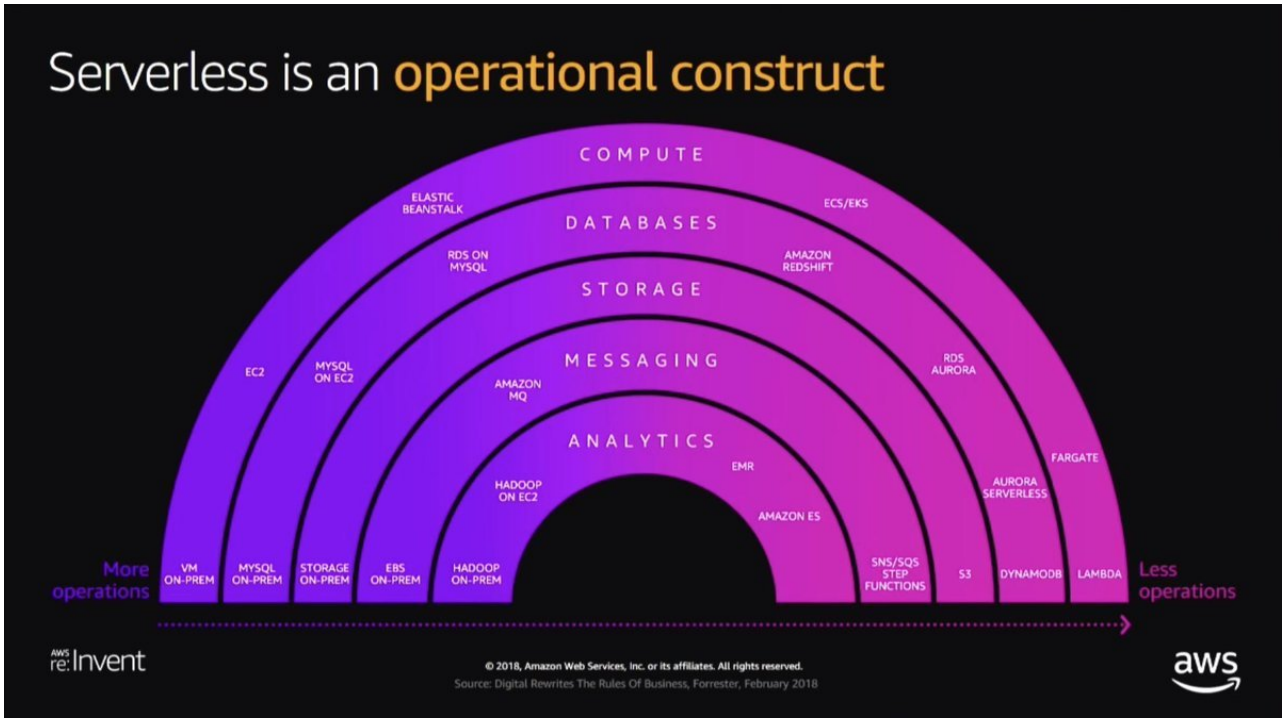
Was ist Serverless überhaupt?

Serverless bedeutet nicht weniger oder gar keine Server; es bezieht sich nicht auf Code, der ohne Server läuft, das hat mit Serverless nichts zu tun. Es heißt „serverlos“, weil die Unternehmen oder die Anwender, denen das System gehört, keine Server oder virtuellen Maschinen kaufen, mieten oder bereitstellen müssen, auf denen ihr Backend-Code läuft. Das Kernargument für Serverless lautet „Konzentration auf das Wesentliche“.

Serverless ist ereignisgesteuert und reagiert nur, wenn etwas geschieht.

Wenn Sie zu Hause zum Beispiel Alexa haben, dann nutzen Sie ein serverloses System. Sobald Sie Alexa ansprechen, wird die Sprachsteuerung aktiv, und das System verschickt die Information, damit ihre Anfrage umgesetzt wird. Wenn Sie sagen: „Alexa, mach das Licht an“, dann arbeitet Alexa serverlos für Sie.





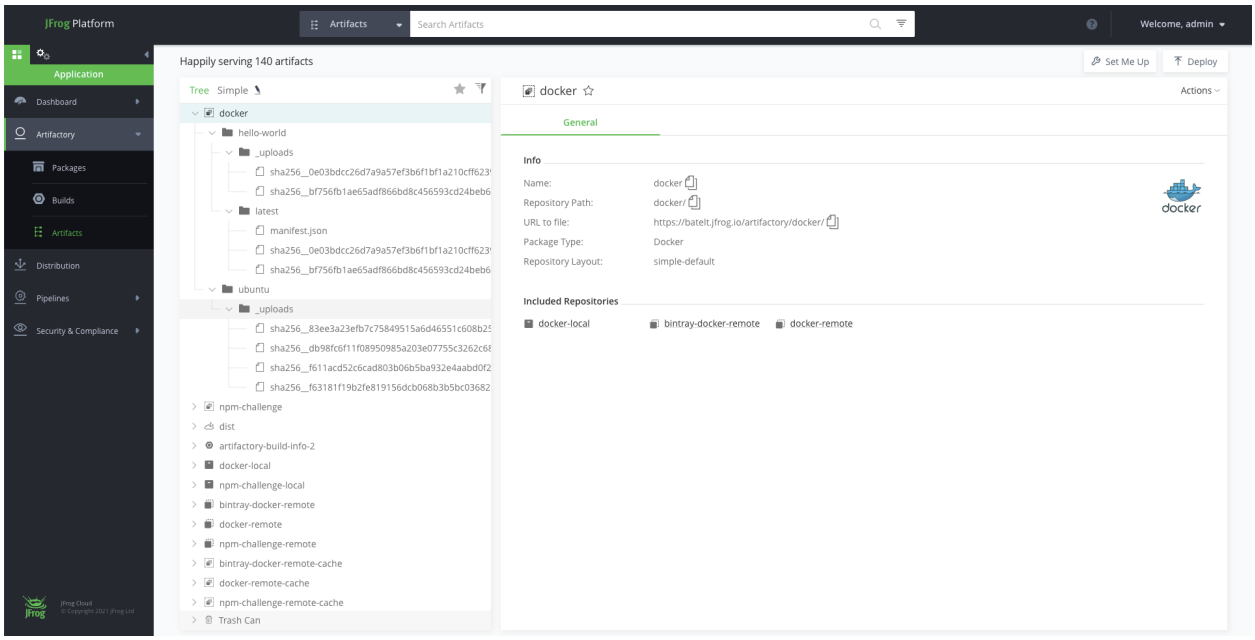
Serverless ist ein Betriebsmodell.

Quelle: [Kesley Hightower]

Serverless ist generell nach vier Prinzipien gebaut:

1. Keine Server, die zu warten oder zu managen wären.
2. Automatische Skalierung durch die Nutzung.
3. Entgelt nach Wert, nicht nach Servereinheiten.
4. Eingebaute Verfügbarkeit und Fehlertoleranz.

Wenn Sie über einen Umstieg auf Serverless nachdenken, geht es nicht nur um Container und Lambdas; auch Paketmanager wie Artifactory spielen eine wichtige Rolle.
Nehmen wir an, dass Sie an einem OpenFaaS-Projekt arbeiten, das serverlos auf Kubernetes läuft. An einem irgendeinem Punkt würden Sie gerne die Abhängigkeiten abrufen – und hier kommt Artifactory ins Spiel. Auch wenn Sie auf einer Serverless-Plattform bereitstellen, wird alles von Artifactory gefahren. Das bedeutet, dass Sie genau wissen, welche Version Sie haben und welche Lizenzen, und dass Sie genau wissen, was in Ihren Docker-Containern vor sich geht. Wenn Sie sich Ihre Build-Informationen bei Artifactory ansehen, wissen Sie genau, was gemacht ist, und haben dazu alle anderen Informationen, die Sie als Entwickler brauchen.



Umfassende Übersicht mit JFrog Artifactory.

Quelle: JFrog

Entwickler, die mit serverlosen Technologien arbeiten, müssen also immer noch wissen, was sie verwenden, welche Abhängigkeiten bestehen, wie die Lizenzen aussehen usw., denn letztendlich liegt es in der Verantwortung der Entwickler, dass sie das Richtige tun, wenn es um die Lizenzen und die vielerlei Compliance-Aufgaben geht.

Warum ist Serverless notwendig?

Traditionelle Architekturen haben zwei Hauptprobleme:

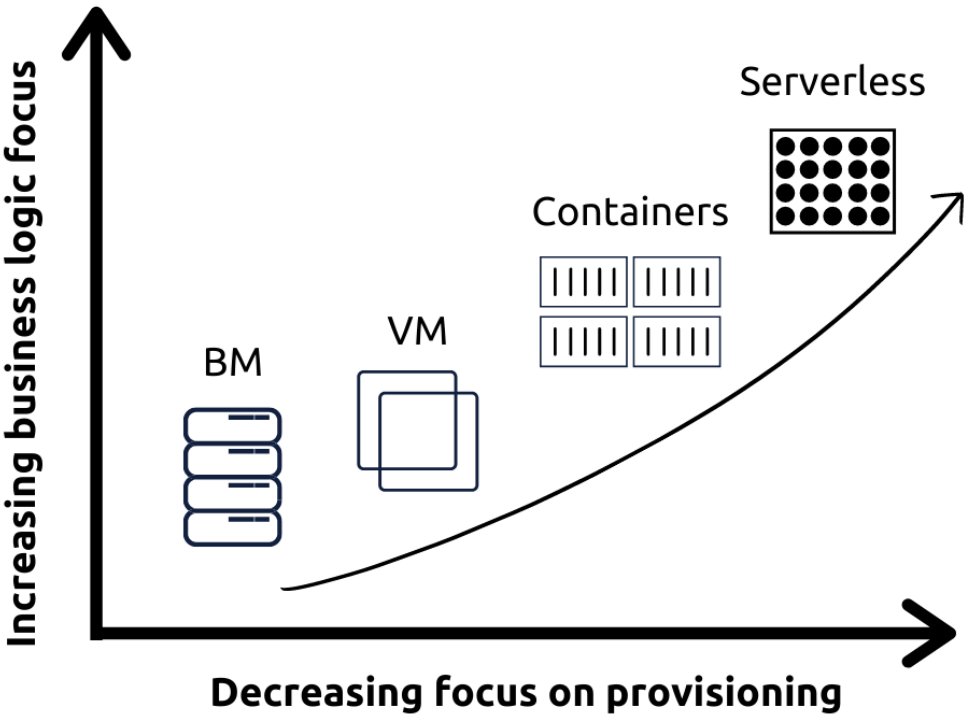
1. Single point of failure:

Wenn ein Server abstürzt, warum auch immer, dann stürzt die komplette Website ab und ist für die Kunden nicht mehr verfügbar, was sich negativ auf das Geschäft auswirkt. Erschwerend kommt hinzu: Wenn eine Produktionsdatenbank auf demselben Server wie die Website läuft, schlägt der Serverabsturz auch auf alle Verbindungen zur Datenbank!

2. Teurer Leerlauf:

„Always on“ heißt auf gut Deutsch, dass man immer zahlen muss. Ständig für die Servernutzung zu bezahlen, egal wie viele Benutzer auf die Website kommen, ist für die vielen Unternehmen, die wachsen wollen, aber nicht sonderlich hilfreich und wirft überflüssige Gemeinkosten auf.

Serverless ist eine Bewegung der Gegenwart, ein Trend, der aus Entwicklerperspektive denkt und auf besseren Code und reichhaltigere Funktionen aus ist, statt auf die Wartung von IT-Umgebungen.



Deployment zwischen Business-Logik und Provisionierung.

Quelle: [JAM]

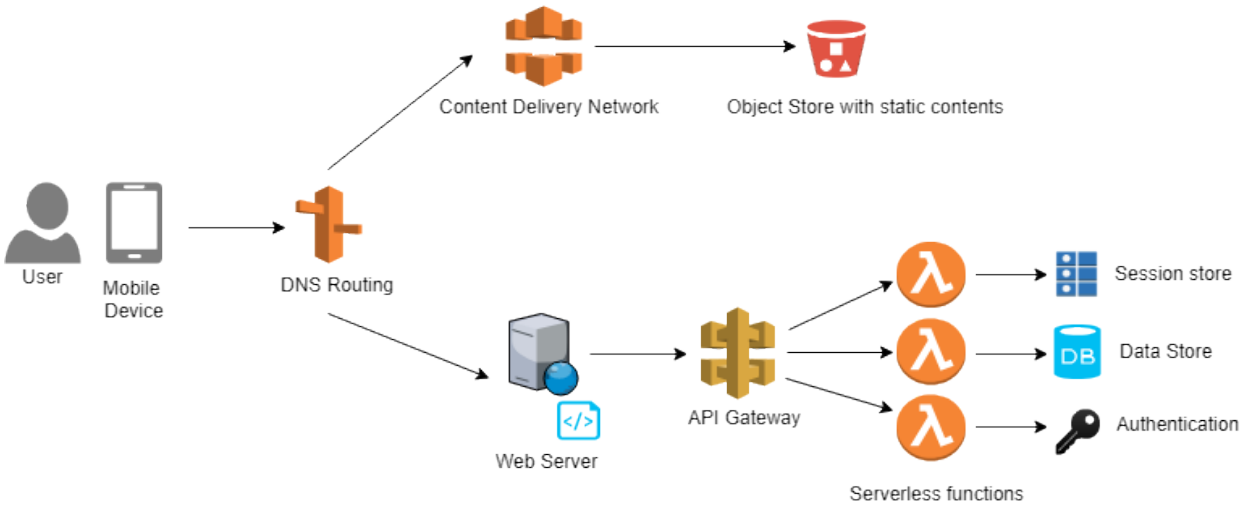
Wie funktioniert Serverless?

Praktisch müssen Sie nur Ihren Code auf den Service eines Cloud-Anbieters hochladen. Dieser stellt dann automatisch eine kurzlebige Umgebung bereit, ein sogenanntes Ephemeral Environment. Anders als traditionelle Architekturen kann sie umstandslos skalieren und Tausende von Anfragen gleichzeitig verarbeiten, Und: Sie zahlen nur, wenn Ihr Code ausgeführt wird.

Aus Entwicklerperspektive entfällt beim Serverless Computing die Notwendigkeit von physischen Infrastrukturen und Systemsoftware. Serverlose Architekturen sind extrem zuverlässig, hochgradig skalierbar und bringen starke Leistung. Bei Serverless Computing zahlen Sie nur für die Ressourcen, die Ihre Anwendung verbraucht, wenn sie läuft. Zwischen der Effizienz Ihres Codes und den Kosten seiner Ausführung besteht eine lineare Beziehung, die sich ganz einfach kontrollieren und prüfen lässt: Je schneller Ihr Code und Ihre Systeme laufen, desto weniger zahlen Sie.

Sie wissen bestimmt, wie es ist, wenn Site Reliability Engineers und DevOps-Verantwortliche sich konzentriert und rund um die Uhr um ihre Server kümmern müssen, auf denen die Anwendungen laufen, inklusive Überstunden lange nach Feierabend, an den Wochenenden und am frühen Morgen, alles wegen der unüberschaubaren Fehlermöglichkeiten und Probleme im laufenden Betrieb – ein ewiger Albtraum. Die gesamte Verantwortung für Wartung, Aktualisierung und Server-Monitoring liegt dann bei Ihnen. Serverless bedeutet dagegen, dass Sie sich nicht mehr um das Server-Management kümmern müssen; dafür ist dann der Cloud-Provider zuständig.

So sieht eine serverlose Architektur aus:



Serverlose Architektur.

Quelle: [Sumo Logic]

Und hier ein paar Serverless-Plattformen bzw. Serverless-Cloud-Services:

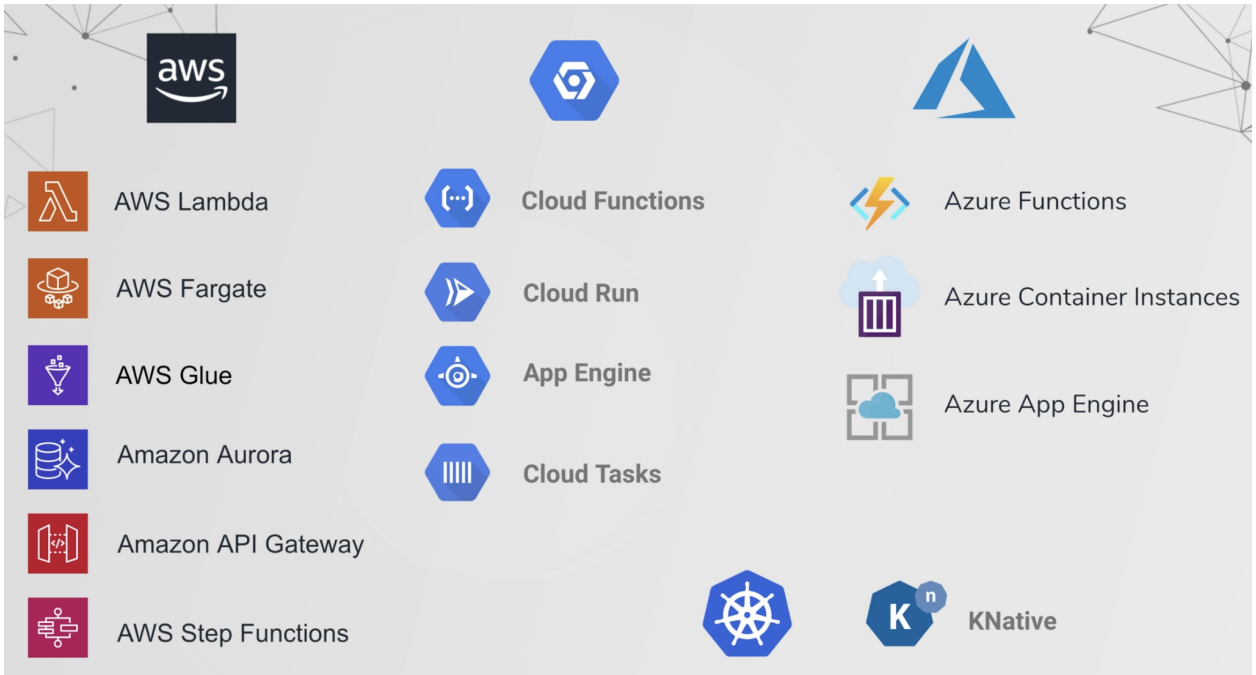
- **AWS Lambda**
- **Google Cloud Functions**
- **Azure Functions**
- **IBM OpenWhisk**
- **Alibaba Function Compute**
- **Iron Functions**
- **Auth0 Webtask**
- **Oracle Fn Project**
- **Kubeless**

1. **FaaS (Function as a Service)**
   FaaS bezieht sich auf zustandslose, ereignisgesteuerte Funktionen, die serverseitige Business-Logik ausführen und in unabhängigen Containern laufen.
2. **BaaS (Backend as a Service)**:
   Bei BaaS werden Dienste von Drittanbietern verwendet (zum Beispiel Firebase oder Auth0), um das nämliche Ziel zu erreichen. Bei BaaS besorgt der Client die meiste Business-Logik mit externen Diensten für Authentifizierung, Datenbanken, Benutzerverwaltung usw.

Bei den bisher üblichen Architekturen sitzt die gesamte Business-Logik in einem einzigen dicken Server-Monolithen. Dagegen wird bei Serverless die monolithische Anwendung als FaaS in mehrere Serverkomponenten aufgeteilt. Dies zeigt ganz gut, welche Art von Gedankengang hinter Serverless in Kombination mit Microservices steht.

Sind Serverless und Container dasselbe?

Container und Serverless haben gemeinsam, dass sie die Art und Weise verändert haben, wie wir Software entwickeln und laufen lassen. Und beide Technologien sind das, was Entwickler lieben, weil sie sich damit auf ihren Code und ihre Kreativität konzentrieren können, anstatt sich mit der Infrastruktur herumzuschlagen. Und wenn Entwickler ungehindert programmieren und neue Funktionen entwickeln, steigt das Tempo der Software-Entwicklung. Sowohl Container als auch Serverless sind perfekt für Microservices und komponentenbasierte (Component-based) Architekturen geeignet. Mit Containern und Serverless im Tech-Stack werden Entwicklung, Bereitstellung und Skalierung schneller und kosteneffizienter als bei jeder klassischen, monolithischen Architektur.
Container und Serverless haben also viel gemeinsam. Dennoch unterscheiden sie sich in ihren Vor- und Nachteilen und je nach Anwendungsfall.



Serverless-Angebote verschiedener Cloud-Plattformen.

Quelle: [Tech Primers]

Serverless auf AWS, Google Cloud und Microsoft Azure

AWS Lambda

AWS Lambda ist die derzeit beliebteste Serverless-Computing-Plattform der Branche. Lambda war das allererste Serverless-Framework einer großen Public Cloud und hat viel zur Verbreitung von Serverless-Architekturen beigetragen.

Lambda kann Funktionen weitgehend unabhängig von der Programmiersprache ausführen, egal ob Node.js, Python, Java, C# oder Go. Mehrere simultane Ereignisse bewältigt Lambda mit entsprechend vielen Funktionskopien.

AWS Lambda umfasst drei Komponenten:

- **die Funktion**, also den Code, der die Aufgabe ausführt;
- **die Konfiguration** – sie definiert, wie die Funktion ausgeführt wird;
- **die Ereignisquelle**, also das Ereignis, das die Funktion auslöst. Ereignisquellen können diverse AWS-Services oder beliebige Drittanbieter-Services sein.

Azure Functions

Microsoft gibt sich viel Mühe, damit Entwickler diese ganzen neuen Serverless-Sachen auch mit der Azure-Cloud machen können. Der Konzern hat dafür eigene Basisfunktionen geschaffen – eben Azure Functions – und dazu noch ein paar Tools ausgetüftelt, die weniger versierten Programmierern die Arbeit erleichtern.

Microsoft Azure Functions fungieren als eine zeitgemäße, serverlose Architektur, die ereignisgesteuerte Cloud-Rechenleistung bereitstellt. Azure Functions sind auf Konformität mit der Anwendungsentwicklung ausgelegt. Sie können also Codeschnipsel in der Cloud ausführen, ohne dass Sie sich um die lästige Wartung der Webserver kümmern müssen.

Um Funktionen zu erstellen, benötigen Sie ein aktives Azure-Abonnement und ein Azure-Storage-Konto. Jede unabhängige Einheit verhält sich dabei wie ein Microservice. Entwickler, die eine Funktion erstellen, schaffen damit eine logische Einheit aus Verwaltung, Bereitstellung und gemeinsamer Ressourcennutzung.

Google Cloud Functions

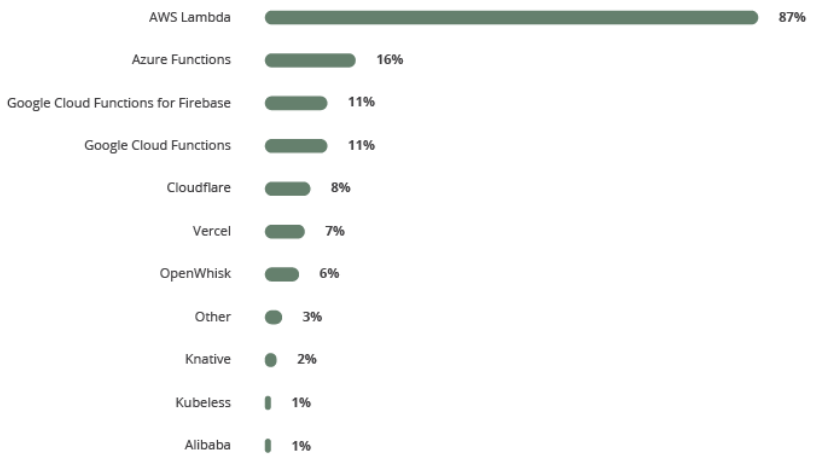
Google Cloud Functions unterstützt die Sprachen Golang, Node.js, Python, Java sowie .NET-Frameworks wie Visual Basic, C# und F#.
Google Cloud Functions lässt kleine Codeschnipsel ganz bestimmte, begrenzte Aufgaben ausführen, die in der Regel als Reaktionen auf Ereignisse in der wirklichen Welt oder auf Sofware-Ereignisse ausgelöst werden. Damit die Performance mithält, skaliert der Dienst die Ressourcen automatisch nach oben oder unten, je nach Bedarf der Funktion.

Als Unternehmen kann man Google Cloud Functions auf vielerlei Art nutzen. Ein Beispiel: automatisierte Log-Analyse. So könnte eine Funktion nach bestimmten Ereignissen in anderen Cloud-Diensten oder Anwendungen Ausschau halten – zum Beispiel nach einer Fehlermeldung –, damit eine Überprüfung der Logfiles auslösen und die benötigten Einträge als automatisierte Nachricht an ausgewählte Entwickler senden.

STATE OF SERVERLESS 2020 REPORT INSIGHTS

2020 hat Coding Sans 278 Fachleute zum Thema Serverless-Technologie und -Praxis befragt – mit einigen interessanten Ergebnissen.

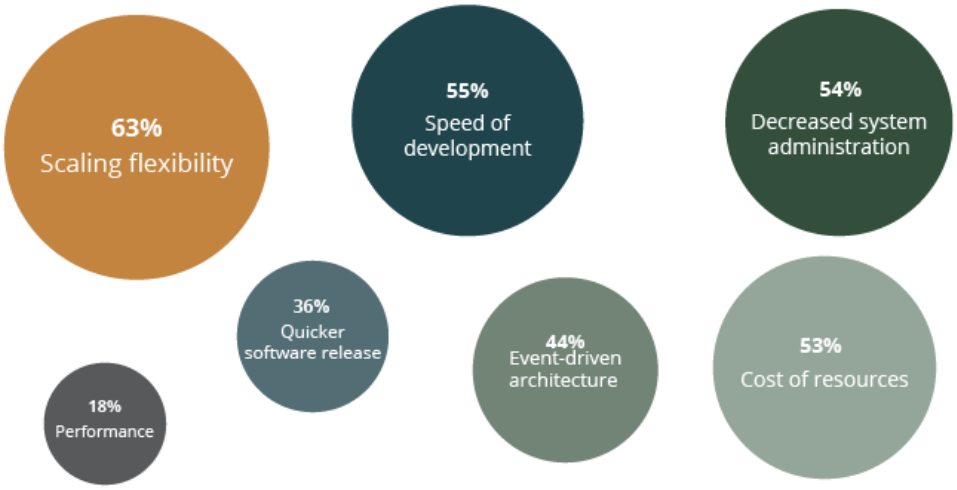
1. AWS ist derzeit der meistgenutzte Serverless-Anbieter, gefolgt von Azure und der Google Cloud.



Welchen Serverless-Anbieter nutzen Sie?

Quelle: Coding Sans

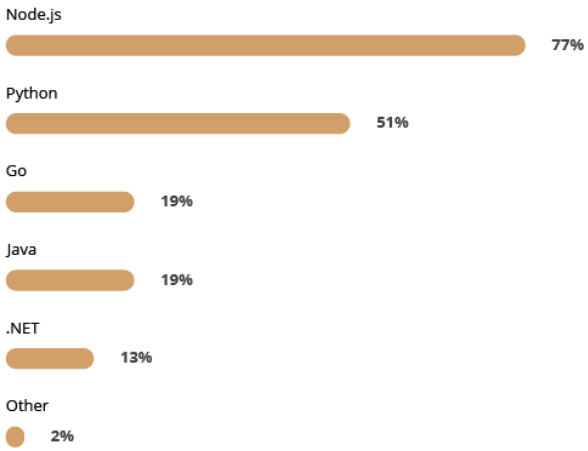
2. Die Flexibilität der Skalierung und das Tempo der Software-Entwicklung sind die beiden wichtigsten Faktoren, die bei der Wahl einer serverlosen Architektur in Betracht gezogen werden.



Was sind für Ihr Unternehmen die größten Vorteile von Serverless?

Quelle: [Coding Sans]

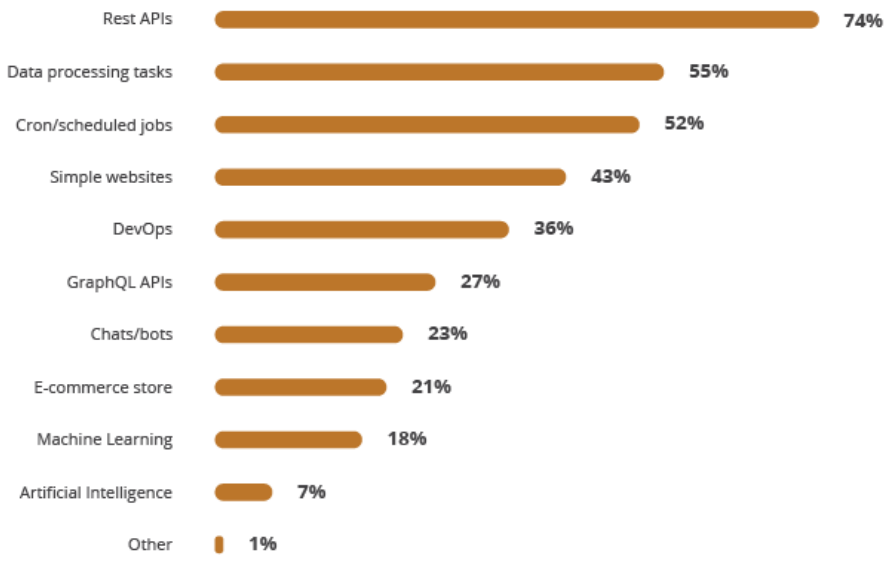
3. Die beliebteste Programmiersprache bei serverlosen Frameworks ist **Node.js,** gefolgt von **Python.**



Welche Programmiersprachen verwendet Ihr Unternehmen zur Entwicklung serverloser Funktionen?

Quelle: [Coding Sans]

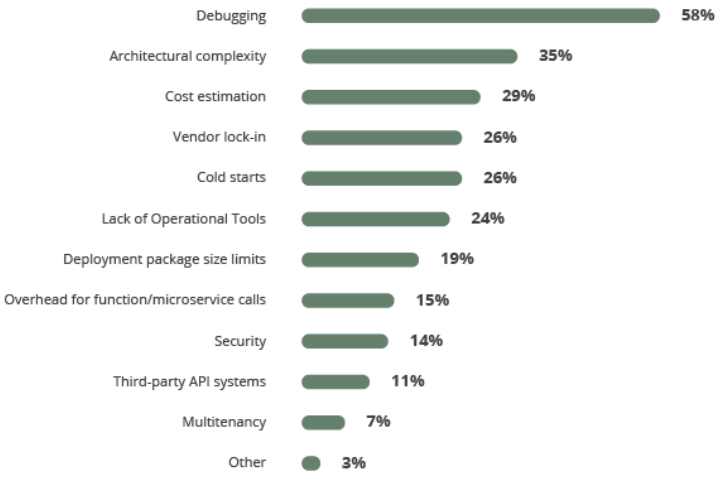
4. **REST-APIs** sind der häufigste Serverless-Anwendungsfall, noch vor **Datenverarbeitungsaufgaben.**



In welchen Fällen nutzen Sie eine serverlose Architektur?

Quelle: Coding Sans

5. Die schwierigste Entwickleraufgabe bei Serverless ist offenbar das **Debugging.**

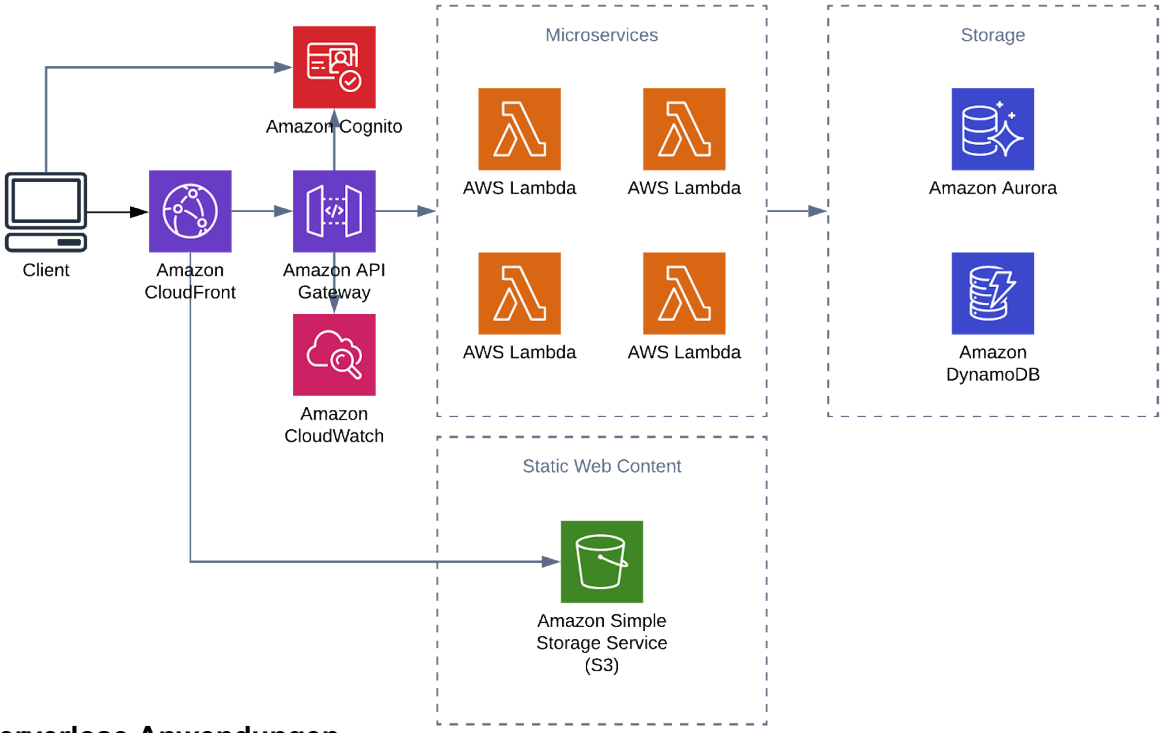


Was sind für Ihr Unternehmen die größten Herausforderungen bei Serverless?

Quelle: Coding Sans

Weitere Ergebnisse der Umfrage finden Sie direkt bei den Serverless Trends von Coding Sans.

Beispiel einer Serverless-Architektur



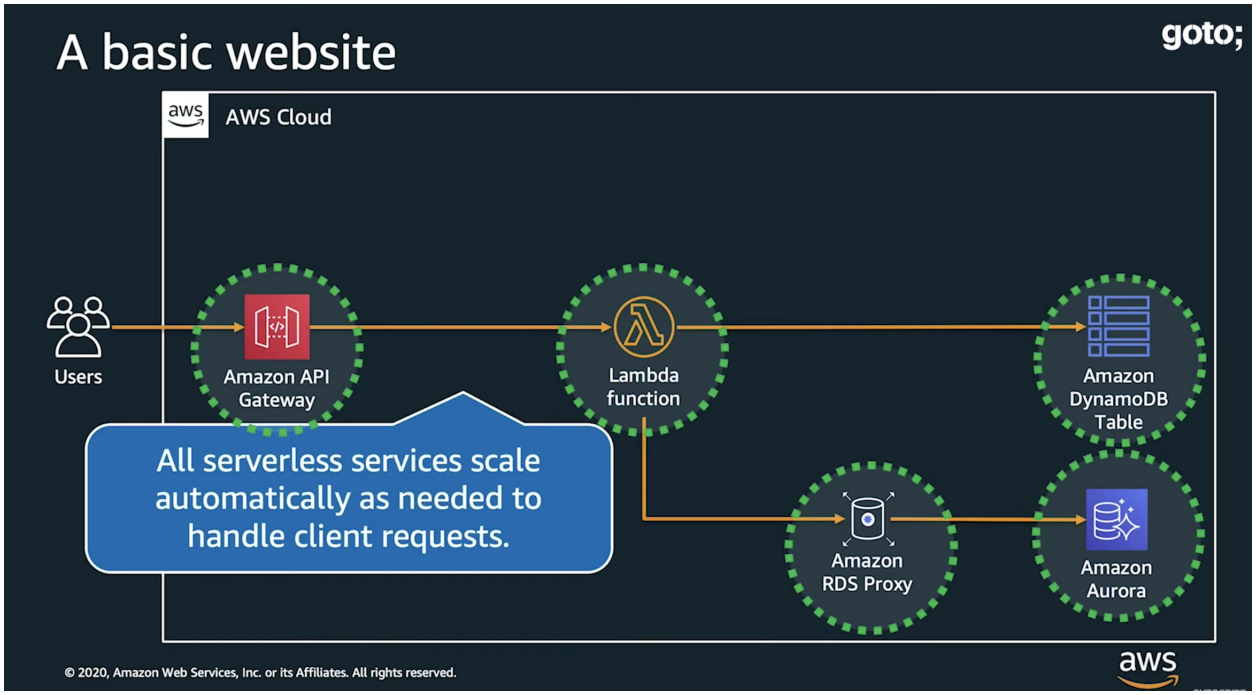
Serverlose Anwendungen.

Quelle: [NordCloud]

Das Beispiel zeigt eine serverlose SaaS-Anwendung, die Folgendes verwendet:

- S3-Buckets für statische Webinhalte,
- Lambda als serverlose Microservices,
- ein API-Gateway als REST-API,
- Cognito zur Benutzerverwaltung und Authentifizierung,
- Amazon Aurora Serverless als SQL-Datenbank und
- DynamoDB als NoSQL-Datenbank.

Die Lambda-Funktionen können sich gegenseitig triggern, sodass es recht leicht ist, selbst komplizierte Anwendungen mit einfachen Funktionen zu gestalten. Es empfiehlt sich sehr, keine Serverless-Monolithen zu bauen, sondern die Lambda-Funktionen möglichst unabhängig zu entwickeln. Der ereignisgesteuerte Ansatz, bei dem die Lambda-Funktionen voneinander getrennt sind und durch Ereignisse ausgelöst werden, ist Best Practice.

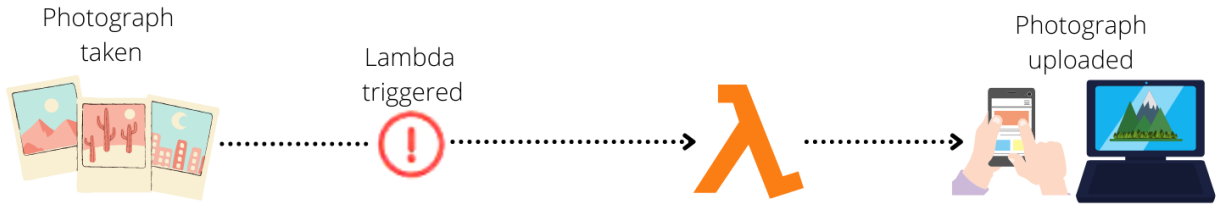


Eine einfache Website mit serverloser Architektur.

Quelle: AWS

Use Cases für Serverless

In unserer rasanten Software-Welt gibt es eine ganze Menge von praktischen Serverless-Anwendungsfällen. Stellen Sie sich zum Beispiel vor, dass Sie der VP of Engineering oder CTO von Instagram sind: Eine der entscheidenden Leistungen von Instagram ist es, den Benutzern das Hochladen ihrer Stories, Fotos und Videos zu ermöglichen. Instagram selbst ist eine gigantische Plattform; der Betrieb einer solchen Plattform und ihre Skalierung für über zwei bis drei Milliarden Nutzer erfordert definitiv eine kolossale Infrastruktur mit sehr vielen Servern. Hier kann serverlose Technologie glänzen: Die Handhabung und Verwaltung so vieler Server wird mit einem serverlosen Framework nahtlos bewältigt.



Serverless-Beispiel Instagram.

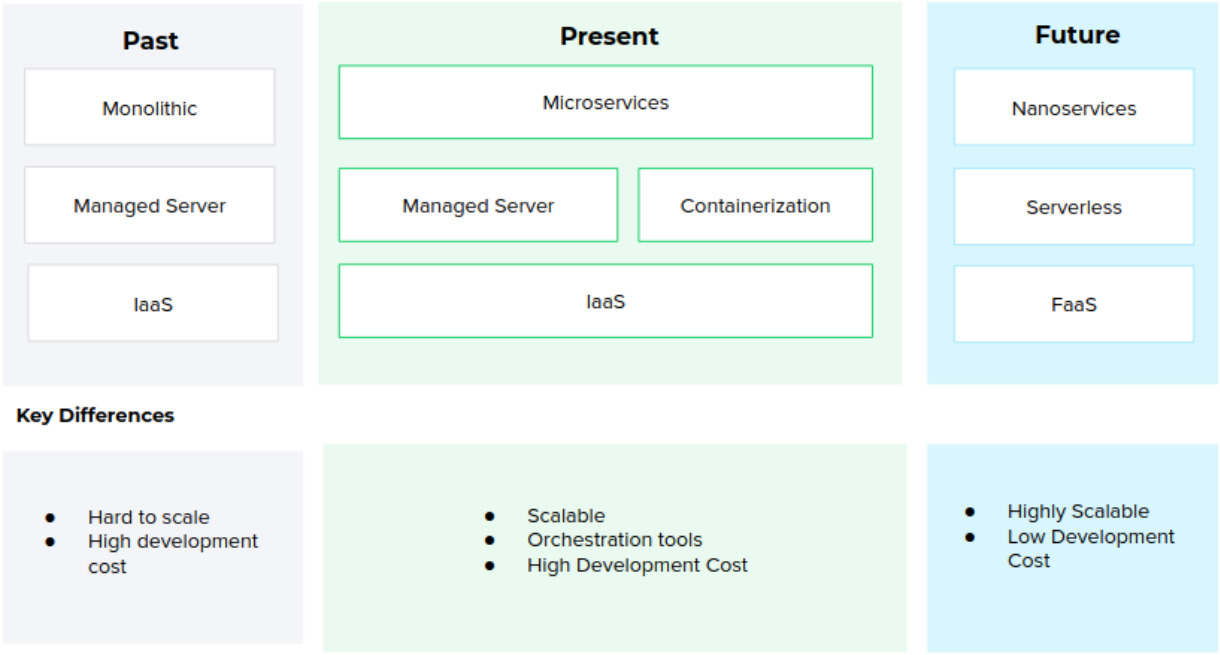
Quelle: Serverless360

Mit Amazon Lambda, Google Cloud Functions oder Azure Functions kann jede ein solches Serverless-Setup implementieren. Bei diesem Setup bekommen die Anwender automatisch einen HTTP-Endpunkt vorgesetzt; über diesen Endpunkt können sie ihre Inhalte dann einfach aus der Frontend-Web-Anwendung oder der mobilen App hochladen. Gespeichert werden die Daten im angebundenen Backend. Das ist eine stark vereinfachte Darstellung – im Idealfall werden Entwickler, die an einem Serverless-Setup arbeiten, die HTTP-Endpunkte noch um ein API-Gateway und etwas Security ergänzen. Dies kann entweder per Amazon API Gateway oder Azure API Management geschehen.

Der kritische Punkt, den man hier im Kopf behalten muss, ist, dass dieses spezielle Setup für sehr viele Benutzer skalieren muss, die jede Minute Millionen und Milliarden von Fotos und Videos hochladen. Ohne serverlose Technologie würde es sehr viel länger dauern, diese Plattformfunktionalität zu entwerfen und zu implementieren, außerdem lägen die Kosten deutlich höher.

LeapFrog: Wechsel auf Microservices, dann auf Serverless

Im Blog berichtet Pravash Raj Upreti, wie und warum sein Unternehmen auf Serverless Computing umgestiegen ist. Das folgende Diagramm zeigt den Wandel der Deployment-Architektur.



LeapFrog-Architekturwandel.

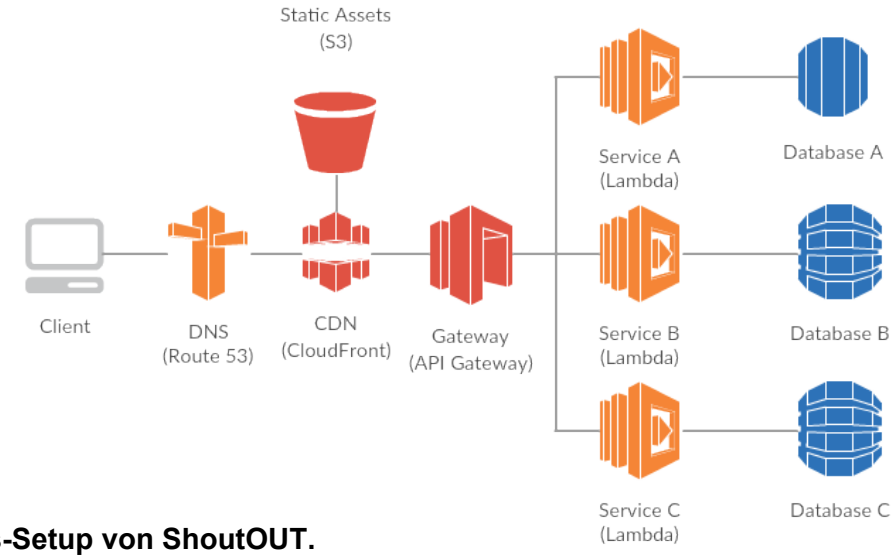
Quelle: LeapFrog

Die Lernplattform LeapFrog versuchte es zunächst mit einem als Microservice entwickelten Authentifizierungsmodul, das dann zwecks Rapid Application Development im SAM-Framework (Serverless Application Model) bei AWS Lambda-Services gehostet wurde. Es gab dabei zwar ein paar kleinere Schwierigkeiten, aber man verlor nicht den Mut, weil man sicher war, dass das Resultat Früchte tragen würde.

Die serverlose App-Architektur wurde kontinuierlich weiterentwickelt. Außerdem wollte LeapFrog sicherstellen, dass das Team eine ordnungsgemäße Deployment-Pipeline bekam, die in der Regel drei Phasen umfasst: Entwicklung (Dev), Qualitätssicherung (QA) und Produktion. Nach ein paar Experimentierwochen stellten sie ihre App auf der serverlosen Architektur bereit, und jetzt können sich die Entwickler ganz auf Code und Logik konzentrieren, anstatt sich um die Infrastruktur zu kümmern.

ShoutOUT: Wechsel von Docker zu Serverless

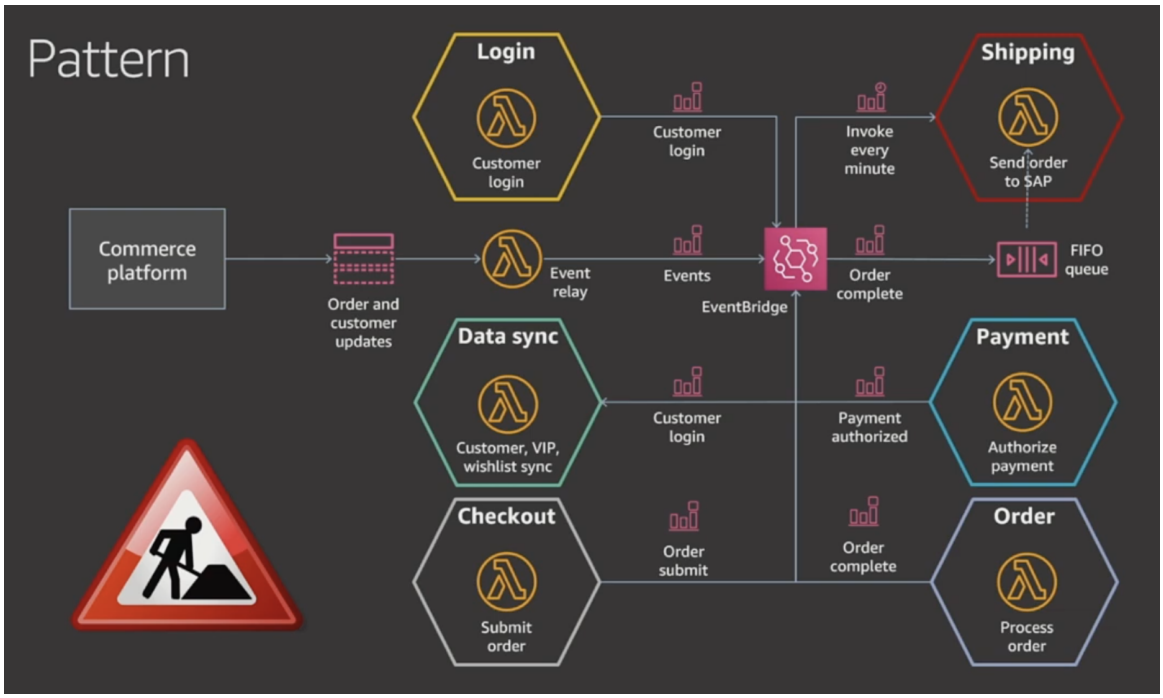
Die Customer Engagement Platform nutzte ursprünglich Docker und Amazon ECS. Die Gründe dafür lagen im Komfort und in der guten Unterstützung bei der Verwaltung von Microservices. ShoutOUT konnte die Services separat halten und die Skalierung individuell handhaben. Als die Services aber an Zahl und Komplexität zunahmen, wurde klar, dass man eine Möglichkeit brauchte, für mehr Rechenleistung zu sorgen.



Serverless-Setup von ShoutOUT.

Quelle: [serverless blog]

LEGO.com: Umzug des Backends auf Serverless



Serverless-Setup von LEGO.com.

Quelle: [Lego Engineering – AWS re:Invent 2019]

Die Backend-Business-Services von LEGO.com laufen serverless in der AWS-Cloud. LEGO.com migrierte im Juli 2019 von einer klassischen monolithischen E-Commerce-Plattform zu Serverless. Die Implementierung umfasste eine Vielzahl von Microservices und über 100 Lambda-Funktionen sowie mehrere weitere AWS-Services. Der Spielzeughersteller brauchte eine Technologie, die Geschwindigkeit und Agilität bot, und da erschien Serverless einfach sinnvoller.

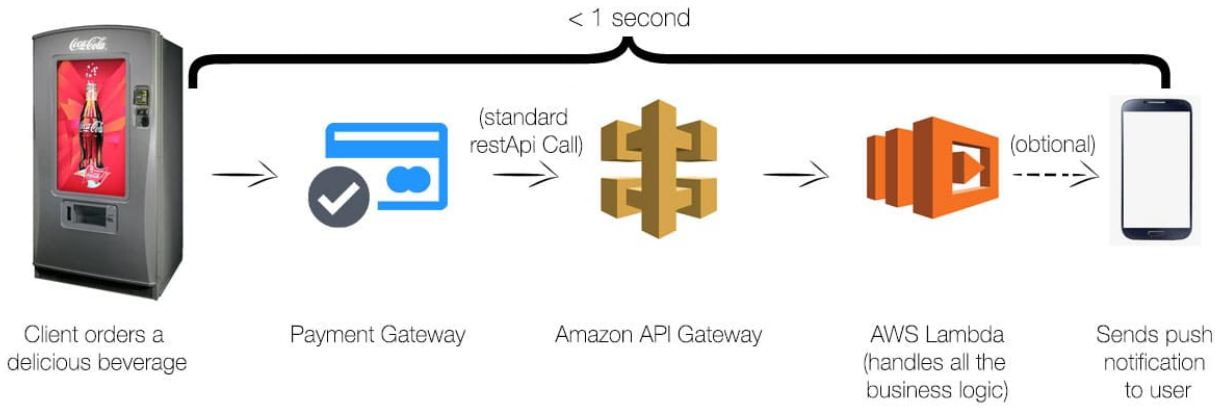
Netflix: Codierung, Backups und Security

Publisher laden auf Netflix täglich Tausende von Dateien hoch, und jede Minute dieser Dateien muss kodiert und sortiert werden, bevor sie letztlich an die Nutzerbasis gestreamt werden. Nachdem die Dateien auf S3 hochgeladen worden sind, löst Amazon ein Ereignis aus, das eine AWS-Lambda-Funktion aufruft, die das Video in Fünf-Minuten-Teile portioniert, die in die 60 separaten, parallelen Streams codiert werden, die Netflix benötigt. Sobald der letzte Teil des Videos verarbeitet ist, wird das Ganze aggregiert und mithilfe systematischer Regeln und Ereignisse bereitgestellt.

Netflix nutzt AWS Lambda noch auf eine weitere Weise: als Backup-System. Da tagtäglich Tausende von Dateien verändert werden, prüfen Lambdas, welche Files gesichert werden müssen, sie kontrollieren die Gültigkeit und Integrität der Dateien, und falls etwas fehlschlägt, können sie an die Quelle des Problems zurückgehen und den gesamten Prozess neu starten. Im Bereich Security hat Netflix Tausende oder gar Hunderttausende von Prozessen, die ständig Instanzen starten und stoppen. Netflix braucht nichts als Lambda, um zu validieren, ob jede Instanz gemäß den Regeln und Vorschriften des Systems konstruiert und konfiguriert ist. Lambdas steuern hier sogar die Ausgabe von Alerts und das Herunterfahren bei nicht autorisierten Zugriffen.

Coca-Cola: Deutlich reduzierte Automatenkosten

Die Coca-Cola-Automaten auf der ganzen Welt kommunizieren über ein integriertes System mit der Zentrale. So erfährt Coca-Cola sofort, wenn ein Getränkeschacht leer wird oder etwas kaputt ist.

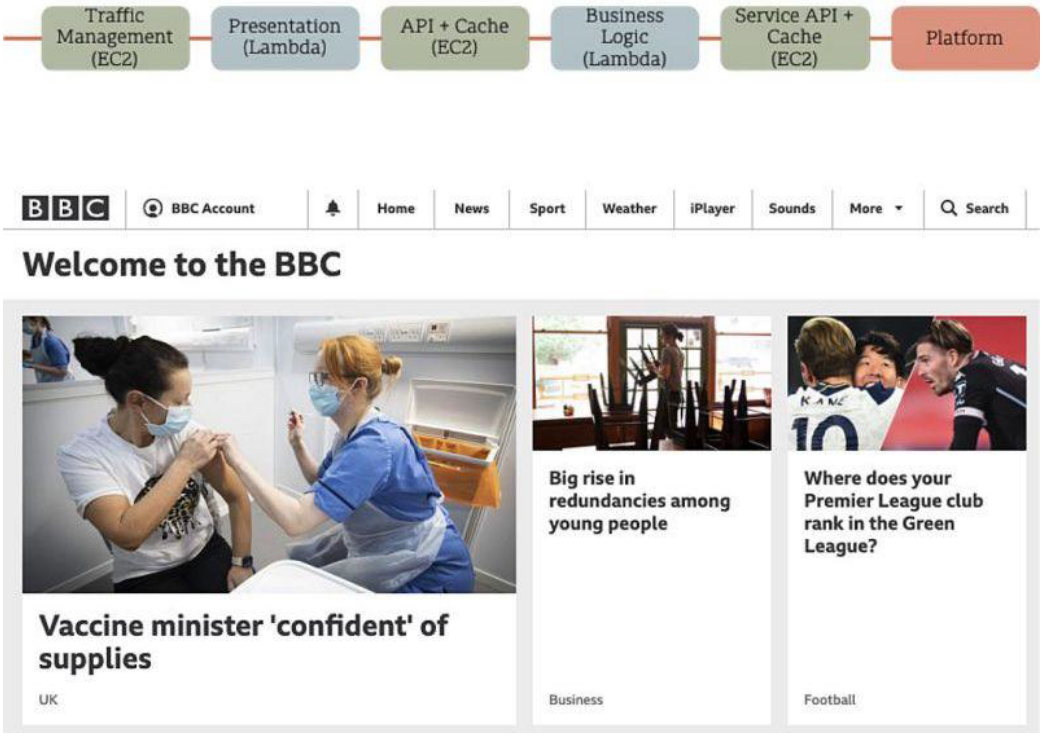


Die ältesten Automaten (mit der genannten Funktionalität) sind etwa zehn bis zwölf Jahre alt. Bis 2016 nutzte Coca-Cola zur Datenverarbeitung sechs EC2-T2.Medium-Maschinen, was jährliche Wartungs- und Betriebskosten von 12.864 Dollar aufwarf. Dies umfasste alles von der Automatisierung über Elastic Load Balancing bis zur Sicherheit.

Nach der Migration auf ein Serverless-Framework sanken die Kosten auf 4490 Dollar pro Jahr, bei rund 30 Millionen Anfragen zum damaligen Stand. So konnte Coca-Cola durch den Umstieg auf AWS Lambda an diesem Punkt fast zwei Drittel der Kosten einsparen.

BBC: Neue Homepage auf Serverless

Das Technikteam der BBC hat die Website des Senders komplett neu aufgebaut. Mit dem Umzug in die Cloud wurde dabei auch die zuvor in eigenen Rechenzentren gehostete Site durch eine Cloud-native Neufassung ersetzt. Dabei verwendet die BBC moderne DevOps-Ansätze und Technologien wie Serverless, um die Produktivität ihres Technologie-Stacks insgesamt zu steigern.



Serverless Setup der BBC.

Quelle: [BBC Technology Blog]

Die BBC fand, dass es zu nervig wäre, virtuelle Maschinen bzw. Container zu warten, zu verwalten und abzusichern. Und dass es ziemlich mühsam und enorm zeitaufwendig wäre, das Ganze verfügbar und skalierbar zu halten. Diese Probleme hat die BBC mit Serverless gelöst. Jawohl, etwa die Hälfte der BBC-Website wird jetzt mit AWS Lambda serverlos gerendert.

Derzeit laufen bei der BBC etwa 2000 Lambdas pro Sekunde und bauen die BBC-Website auf – die Engländer rechnen aber damit, dass diese Zahl noch steigen wird. Der Sender profitiert davon, dass keine Wartungs- und Betriebskosten anfallen und dass das Setup so schnell wie möglich skaliert. Genau das brauchen Medienunternehmen auch, denn jedes Mal, wenn wichtige Eilmeldungen online gehen, kann es sein, dass der Traffic erratisch und unvorhersehbar nach oben ausschlägt. Lambda kann mit so etwas viel besser umgehen als jede andere Technologie.

Fazit

Serverless ist der Weg nach vorn, und zwar auf absehbare Zeit. DevOps ist in der Welt der Software-Entwicklung in den Mittelpunkt gerückt; parallel dazu wächst der Anteil von Serverless-Computing-Technologien, weil Unternehmen damit die Gemeinkosten für die Wartung der serverseitigen Infrastruktur und andere Ärgernisse reduzieren, die Entwickler sonst von ihrer eigentlichen Arbeit abhalten. Serverless ist zu einem Segen für die DevOps-Branche geworden. Es empfiehlt sich, jetzt die Vorteile von Serverless zu nutzen und auf diese Weise Ihre Software-Entwicklung zu beschleunigen.

Wie wir an den Fallbeispielen gesehen haben, ermöglicht Serverless Unternehmen eine rasche Skalierung, die Umsetzung kann aber einige Haken haben; daher sollten Sie sorgfältig auswählen, was zu Ihren Anforderungen passt. Es gibt viele Anbieter, die Ihnen auch bei der Verwaltung der Abhängigkeiten und Lizenzen helfen, sodass Sie die Kosteneinsparungen von Serverless voll ausschöpfen können.

Das Erstellen von serverlosen Anwendungen ist ein mehrstufiger Prozess. Einer der wichtigsten Schritte in diesem Prozess ist das Verpacken der Serverless-Funktionen, die Sie bereitstellen möchten, für die FaaS-Plattform Ihrer Wahl.

Wir empfehlen dringend die Verwendung von Paketmanagern wie JFrog Artifactory bei Ihren Serverless-Prozessen, weil Sie bei jeder Serverless-App, die Sie erstellen, immer noch die Abhängigkeiten und Builds berücksichtigen müssen. Artifactory unterstützt Sie bei der Verwaltung dieser Abhängigkeiten mithilfe von Proxy-Repositories mit schnellem Cache, damit sichergestellt bleibt, dass Sie stets mit der Version arbeiten, die Sie erwarten. OpenFaaS lässt Ihnen große Freiheit bei der Wahl ihrer Programmiersprache, und Artifactory unterstützt rund 30 Pakettechnologien, sodass Sie damit ganz auf der sicheren Seite sind.

Starten Sie Ihre Serverless-Reise mit einem Universal-Paketmanager – jetzt kostenfrei!

MEHR ÜBER JFROG

JFrog ist auf einer "Liquid Software"-Mission, um den Fluss von Software-Updates nahtlos und sicher von den der Entwicklung bis zur Produktion zu ermöglichen. JFrogs universelle, hybride end-to-end DevOps-Plattform bietet die Werkzeuge und die Transparenz, die moderne Software-Entwicklungsorganisationen benötigen, um die Möglichkeiten von DevOps voll auszuschöpfen. JFrog's Plattform ist als Open-Source, selbstverwaltet und als SaaS-Service (mit kostenlosem Tier) auf AWS, Microsoft Azure und Google Cloud verfügbar. JFrog genießt das Vertrauen von Millionen von Entwicklern und Tausenden von Kunden, darunter die Mehrheit der Fortune-100-Unternehmen, die sich beim Management ihrer DevOps-Pipelines auf JFrog-Lösungen verlassen. Erfahren Sie mehr unter jfrog.com.