



CLOUD-NATIVE

Unit:
Serverless Architectures

(1) Was ist das? Stärken + Schwächen



Urheberrechtshinweise

Diese Folien werden zum Zwecke einer praktikablen und pragmatischen Nutzbarkeit im Rahmen der **CCo 1.0 Lizenz** bereitgestellt.

Sie dürfen die Inhalte also kopieren, verändern, verbreiten, mit eigenen Inhalten mixen, auch zu kommerziellen Zwecken, und ohne um weitere Erlaubnis bitten zu müssen.

Eine Nennung des Autors ist nicht erforderlich (aber natürlich gern gesehen, wenn problemlos möglich).

Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

Die CCo Lizenz überträgt sich nicht auf zitierte Quellen. Hier sind bei der Nutzung natürlich die Bedingungen der entsprechenden Quellen zu beachten.

Die Quellenangaben finden sich auf den entsprechenden Folien.



KAPITEL 12

Serverless Architekturen



12.1 Eigenschaften von Microservices

12.2 Integrationsmuster für Microservices

12.3 Architekturelle Sicherheit

12.4 Skalierung von Microservices

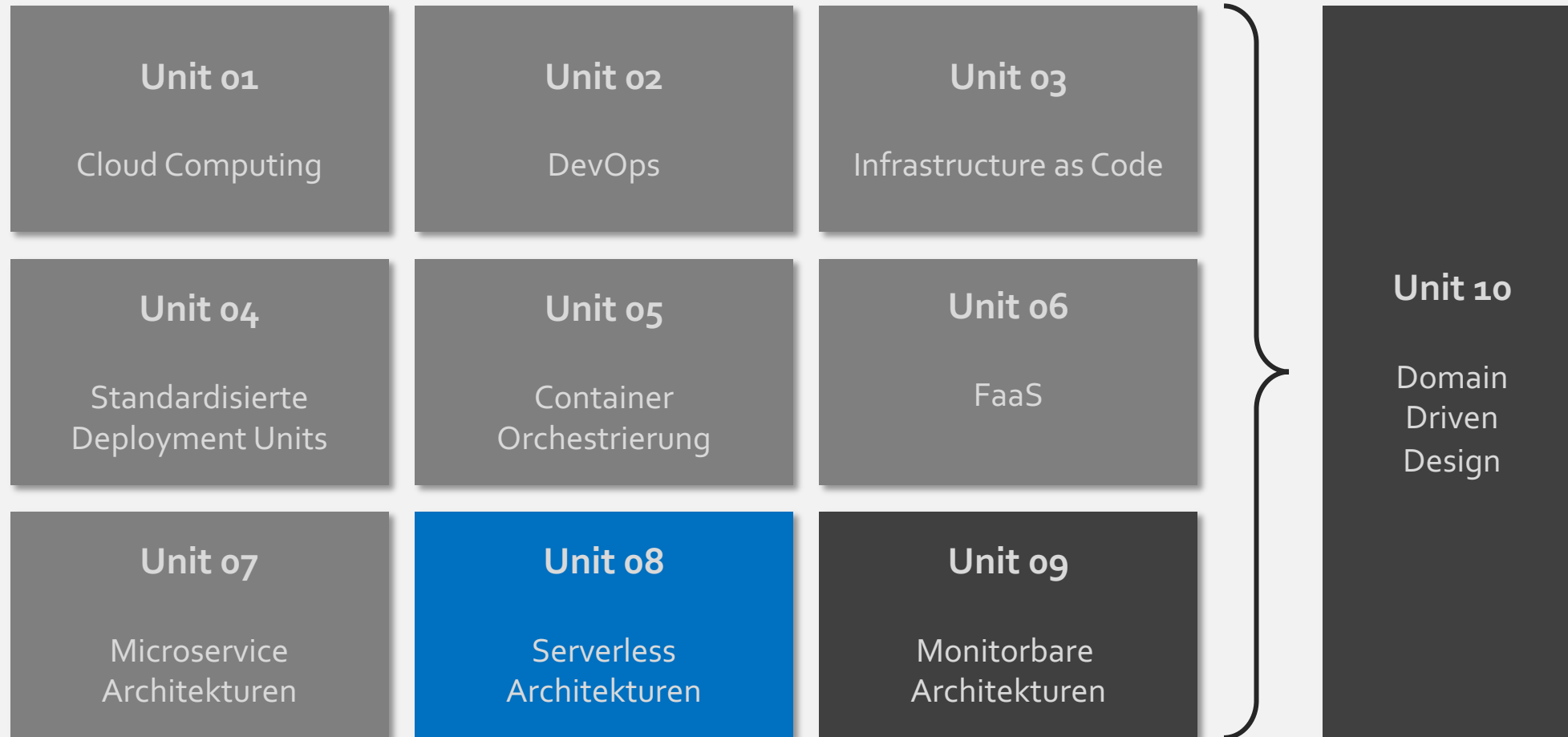
12.5 Prinzipien zur Entwicklung von Microservices

12.6 Serverless-Architekturen

- Architekturelle Konsequenzen von Serverless-Limitierungen
- Das API-Gateway Pattern
- Abgrenzung zu Microservices

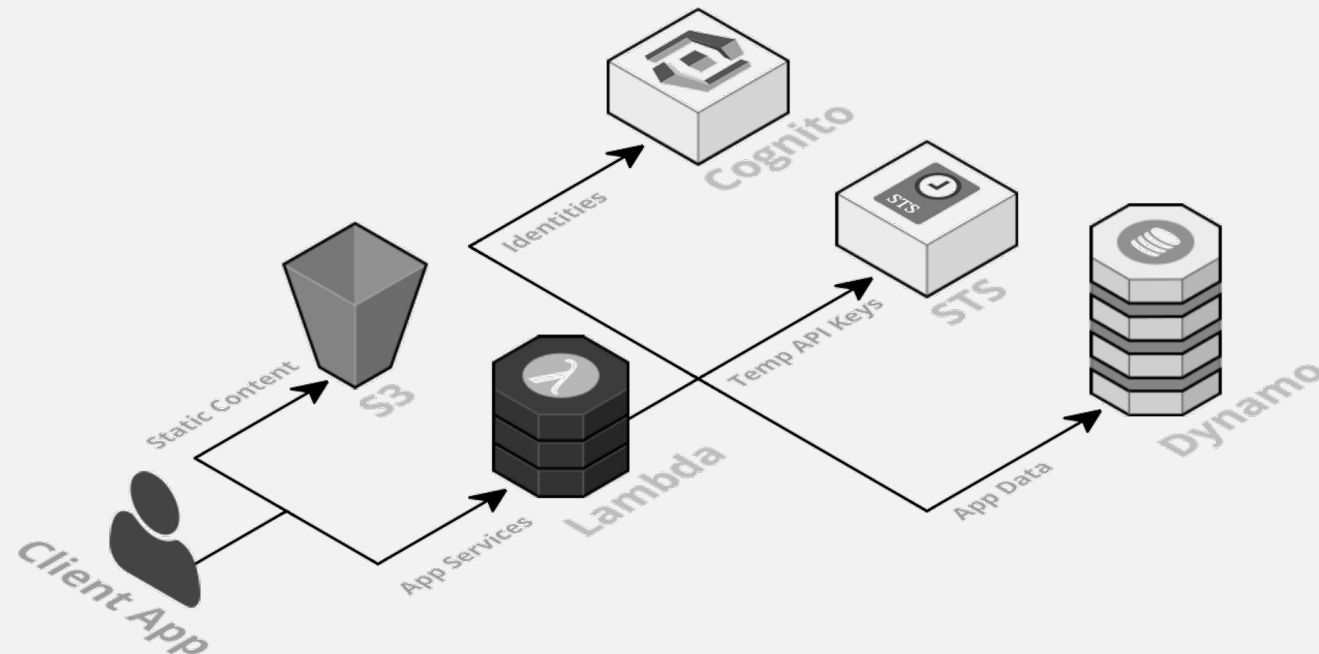
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



INHALTE

- Was ist Serverless?
- Limitierungen von Serverless
- Serverless Architekturen
- Abgrenzung zu Microservices

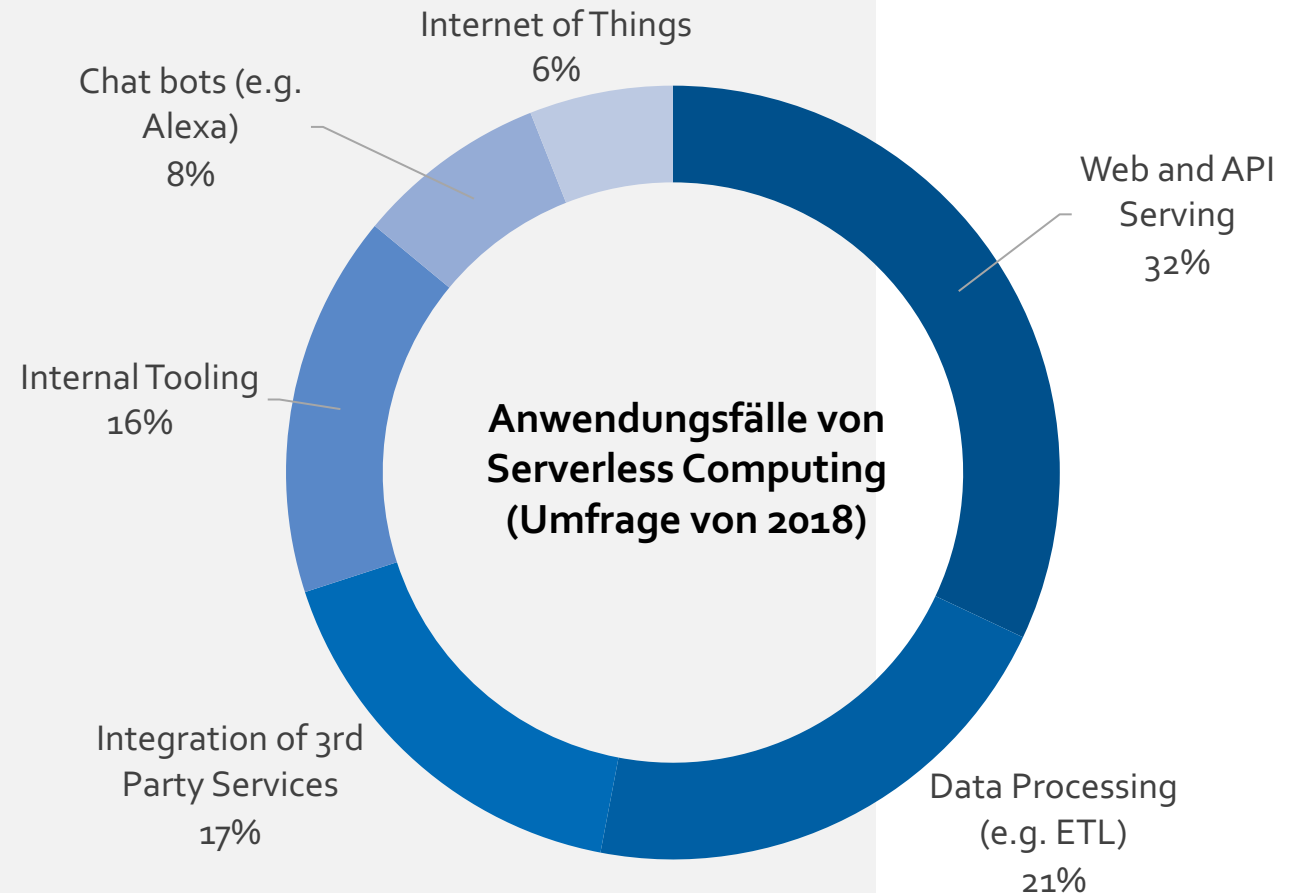


In dieser Unit betrachten wir vor allem die architekturellen Implikationen, die sich aus dem Faas Ansatz der Unit 06 ergeben.

WAS IST SERVERLESS?

A Berkley View on Serverless Computing

- Serverless Computing ist ein Cloud-Computing-Modell, bei dem Cloud-Provider **Ressourcen Request-basiert** zuweisen
- Entwickler von Serverless Anwendungen müssen sich daher nicht mit **Kapazitätsplanung**, Konfiguration, Verwaltung, Wartung, Betrieb oder Skalierung von Containern, VMs oder physischen Servern befassen (daher Serverless!)
- Serverless Computing ist **stateless**, d.h. es werden keine Zustände in Serverless Compute Units (Functions) zwischen Aufrufen gespeichert.
- Wenn Functions keine Requests erhalten, werden Functions auch keine Computing-Ressourcen zugewiesen (**Scale-to-Zero**)
- Die Preisgestaltung basiert auf dem Pay-as-you-go Prinzip basierend auf der Menge verbrauchter Computing Ressourcen zur Verarbeitung eingehender Requests

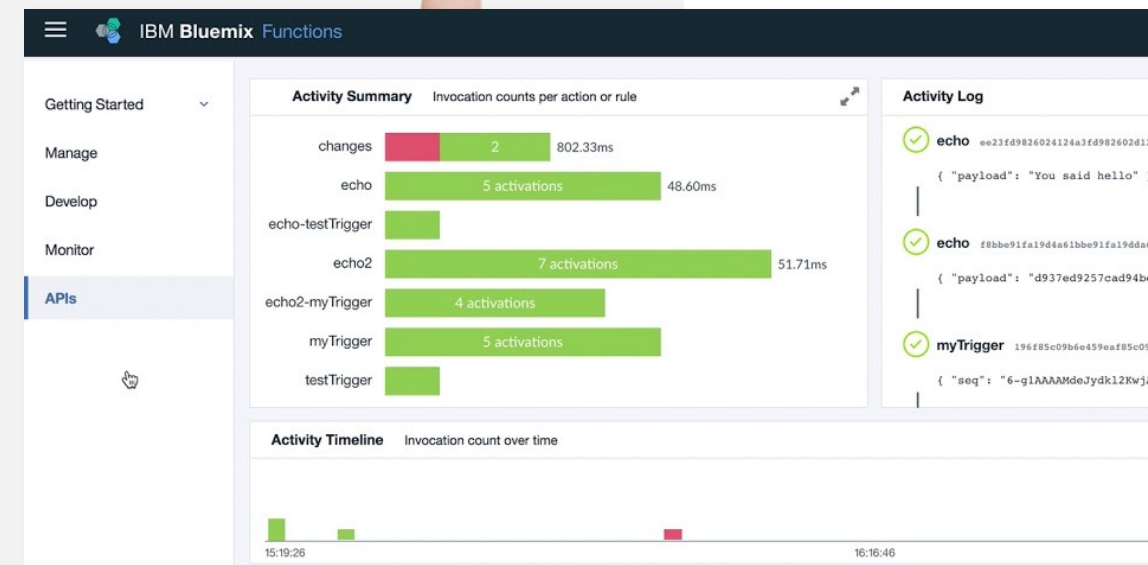


WAS IST SERVERLESS?

Reduktion des Time-to-Market

- Serverless Computing kann den Prozess der Entwicklung und Bereitstellung von Code in die Produktion vereinfachen und die **Time-to-Market** damit reduzieren.
- Serverloser Code kann in Verbindung mit anderen Ansätzen wie **Microservices** oder **SOA** (Monolithen) verwendet werden.
- Cloud-native Services können aber auch so geschrieben werden, dass sie **rein serverlos** sind und sich weder auf IaaS-Infrastrukturen oder PaaS-Container-Plattformen abstützen.
- Serverless Computing ist **kein dezentrales Processing-Modell** wie Peer-to-Peer (P2P) oder Volunteer Computing (VC)
- Serverless Provider bieten **Laufzeitumgebungen** (FaaS-Plattformen, Function as a Service) an die Anwendungslogik ausführen, jedoch keine Daten speichern.

TIME TO MARKET

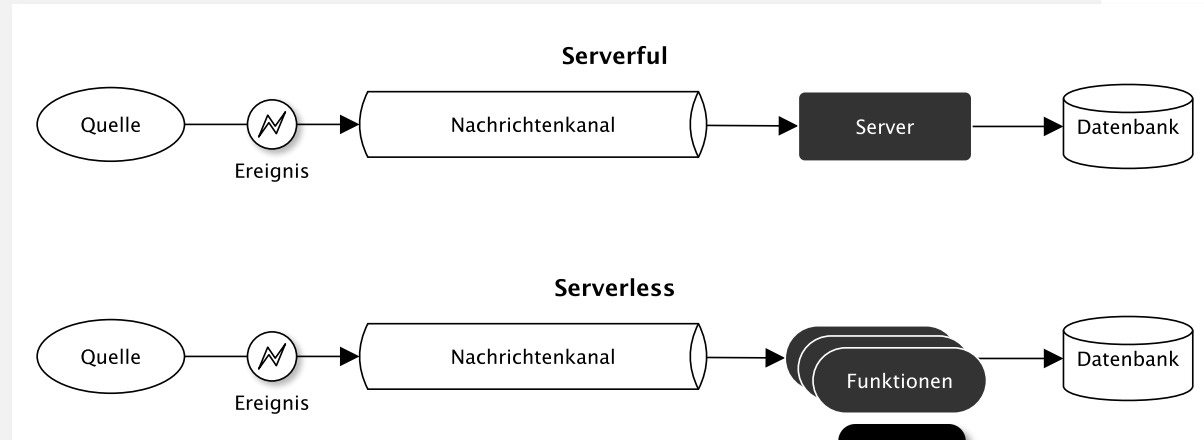


Quelle: IBM

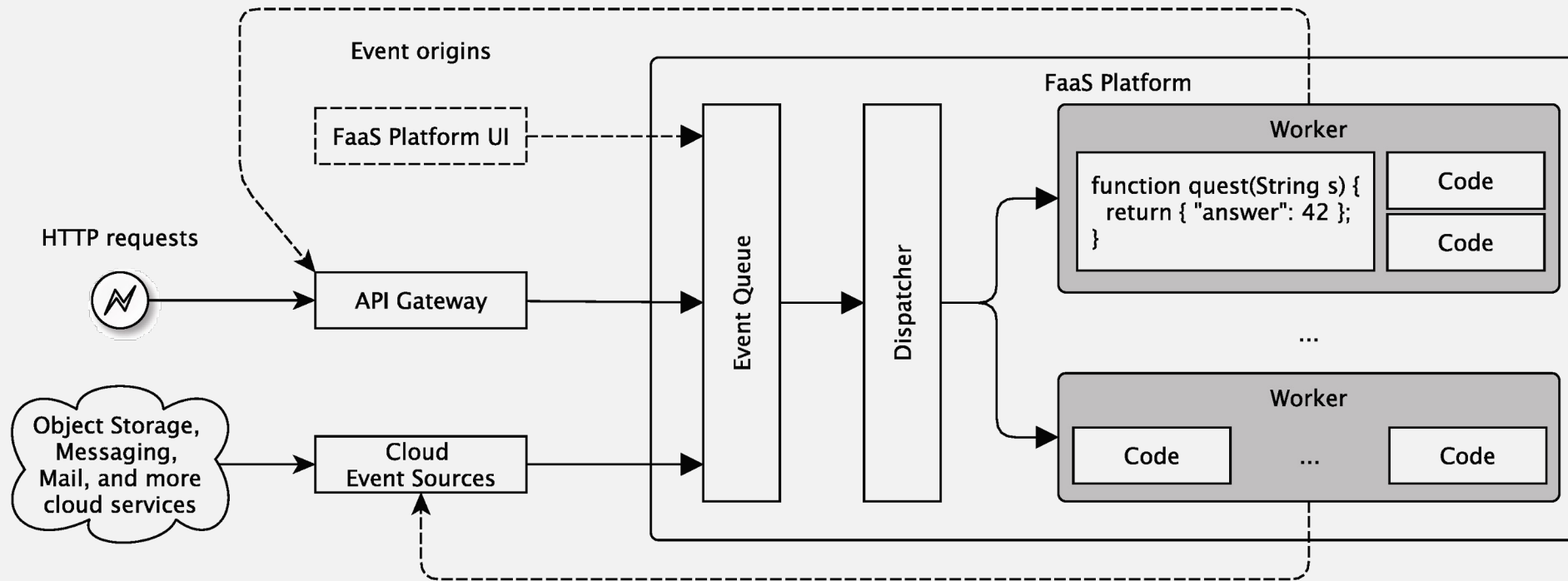
SERVERLESS COMPUTING

Scale-to-Zero vermeidet das teuerste Cloud-Nutzungsmuster

- Die Möglichkeit, Computing-Ressourcen auf null Instanzen zu skalieren (**Scale-to-Zero**), ist eines der entscheidenden **Unterscheidungsmerkmale** von Serverless Plattformen im Vergleich zu Container-Plattformen (PaaS) oder IaaS (virtuelle Maschinen).
- Scale-to-Zero ermöglicht die **Vermeidung von Always-On**-Komponenten und schließt daher das teuerste Cloud-Nutzungsmuster aus (vgl. Unit 01).
- Scale-to-Zero hat allerdings auch **architekturelle Auswirkungen**.



SERVERLESS PLATFORM ARCHITECTURE



Eine serverlose Plattform ist lediglich ein **Ereignisverarbeitungssystem**. Ereignisse können z.B. über HTTP gesendet oder von weiteren Ereignisquellen (aus Cloud-Infrastrukturen) empfangen werden. Die Plattform bestimmt dann, welche **Funktionen** für ein Ereignis registriert sind, sendet das Ereignis an eine dieser Funktionsinstanzen und wartet auf eine Antwort, um diese Antwort an den Requestor weiterzuleiten (im Falle von Request-Response-basiertem Triggern).

*vgl. Unit 06!
FaaS*

LIMITIERUNGEN VON SERVERLESS PLATTFORMEN

State, Execution Duration, Startup Latency and Cold Starts

Zustand

- Jeder **persistente Zustand** einer FaaS-Funktion, muss außerhalb der FaaS-Funktionsinstanz **ausgelagert** werden.
- Für nebeneffektsfreie Funktionen, ist dies nicht von Belang. Auch „**Zwölf-Faktoren-Apps**“ haben diese Einschränkung.
- Stateful Funktionen verwenden normalerweise externe Datenbanken, Cache-Systeme (wie Redis) oder einen Netzwerkdatei- / Objektspeicher (wie S3), um den Status über Anforderungen hinweg zu speichern.

Execution Timeouts

- Die **Ausführungsdauern** von FaaS-Funktionen sind normalerweise **begrenzt**.
- Typische Grenzwerte sind mehrere Minuten (z. B. 5 bis 10 Minuten).
- Daher sind insb. **langlebige Aufgaben** für FaaS-Plattformen **ungeeignet**.
- In diesen Fällen muss man langlaufende Aufgaben in kürzere Aufgaben dekomponieren und deren Ausführung (z.B. mittels Message-Queues) koordinieren.

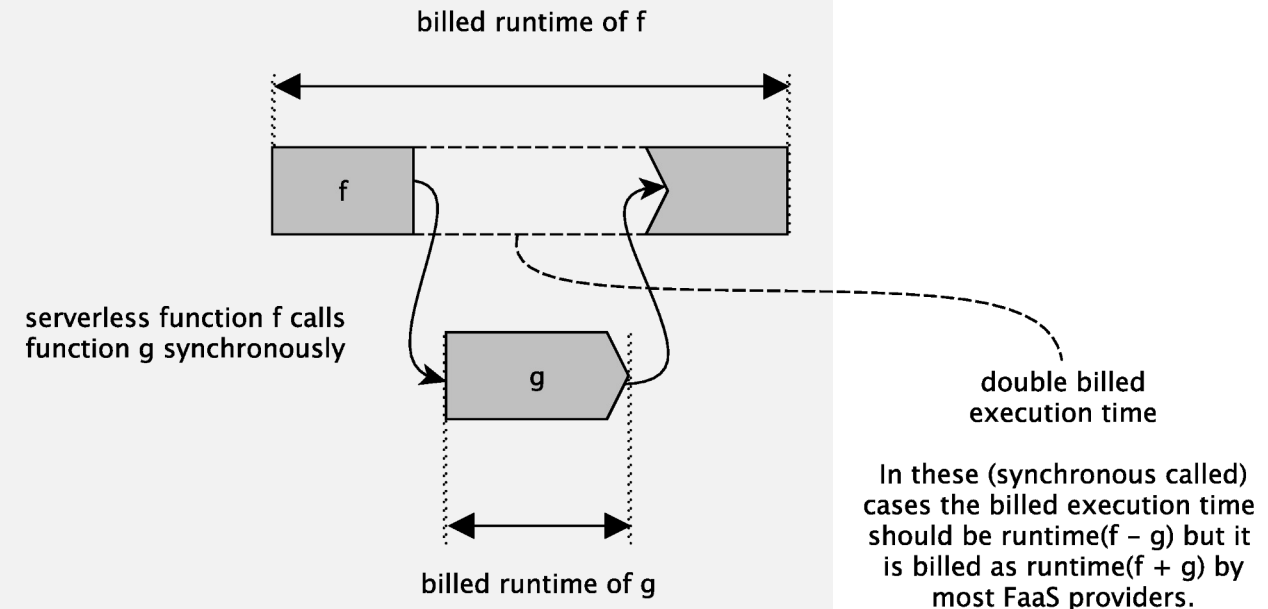
Latenz und Cold Starts

- FaaS-Plattform müssen vor einem Event (Request) eine Funktionsinstanz erst einmal **initialisieren**.
- Diese **Startlatenz** kann selbst für eine bestimmte Funktion in Abhängigkeit von einer Vielzahl von Faktoren erheblich variieren und zwischen einigen Millisekunden und mehreren Sekunden liegen.
- Die Initialisierung einer Funktion ist entweder ein „**Warmstart**“, bei dem eine Instanz und ihr Hostcontainer aus einem früheren Event wiederverwendet werden, oder ein „**Kaltstart**“, bei dem eine neue Containerinstanz erstellt und der Funktionshostprozess gestartet wird.

LIMITIERUNGEN VON SERVERLESS PLATTFORMEN

Double-Spending-Problem

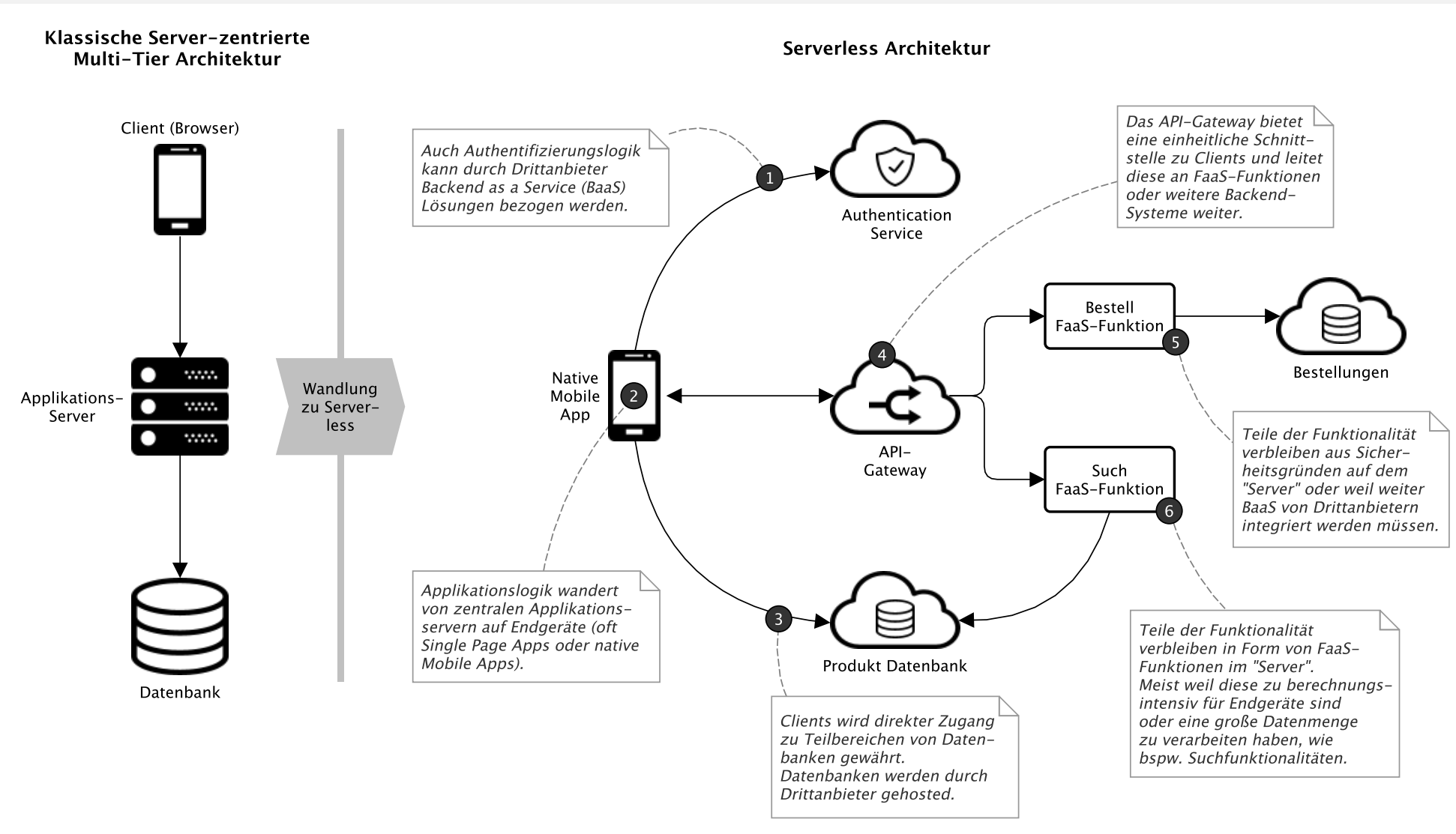
- Dieses Problem tritt auf, wenn eine Funktion f **synchron** eine andere Funktion g aufruft.
- In diesem Fall wird dem Verbraucher die Ausführung von f und g in Rechnung gestellt - obwohl nur g Ressourcen verbraucht, weil f auf das Ergebnis von g wartet.
- Um dieses Problem der doppelten Ausgaben zu vermeiden, **delegieren** viele Serverless-Anwendungen die Komposition von Funktionen (also die **Steuerung des Workflows**) an Clientanwendungen und Edge-Geräte außerhalb des Bereichs von FaaS-Plattformen.
- Dieses Kompositionsproblem führt somit zu neuen, verteilteren und dezentralisierteren Formen von Cloud-nativen Architekturen.



Was man also **vermeiden** sollte, ist dass sich Funktionen **synchron untereinander aufrufen!**

OpenWhisk adressiert dieses Problem bspw. mittels Sequences.

DER EFFEKT DES DOUBLE-SPENDING-PROBLEMS



Serverless Architekturen führen zu weniger zentralisierten Kompositionen von Anwendungs-komponenten und Backend-Diensten im Vergleich zu klassischen (schichten-basierten) Anwendungs-architekturen.

SERVERLESS ARCHITEKTUREN

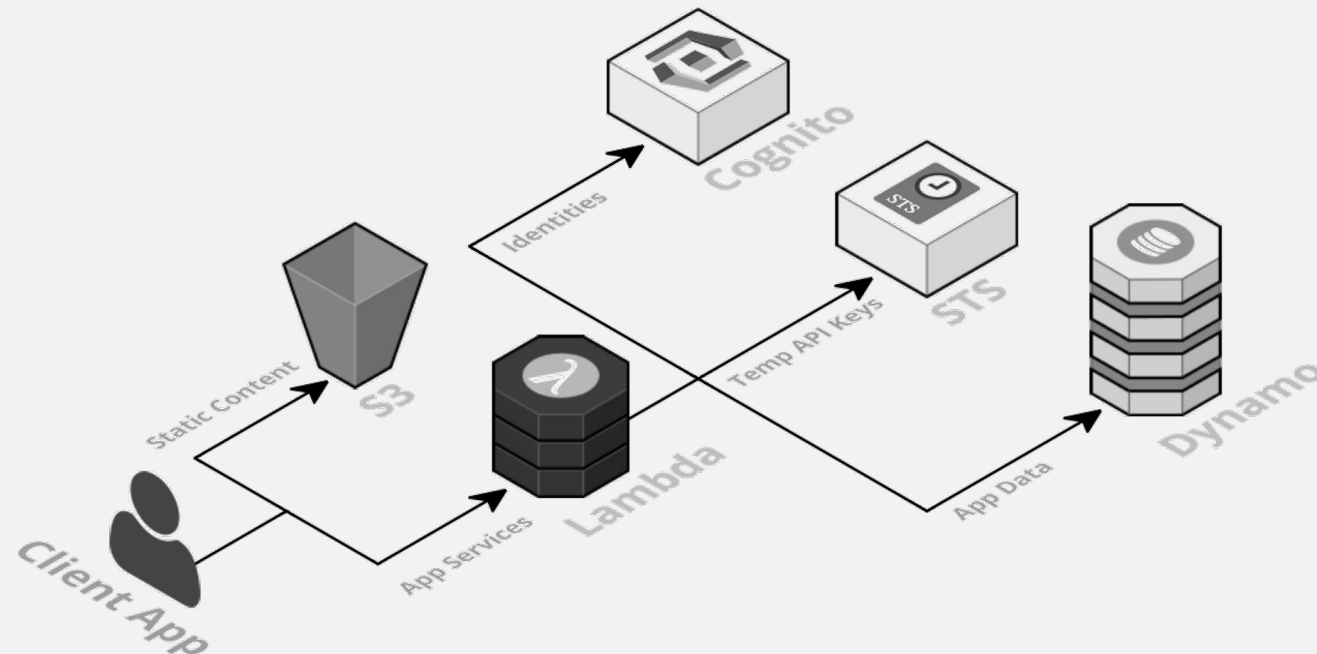
Konsequenzen für Cloud-native Anwendungsarchitekturen

- Insbesondere Serverless Architekturen erfordern im Vergleich zu herkömmlichen E-Commerce-Anwendungen ein Redesign der Cloud-Anwendungsarchitektur, die oft nachhaltig bis zum Kontrollfluss durchschlägt.
- Serverless Architekturen gehen häufiger weiter als Microservice-Architekturen und integrieren konsequenter **Backend-Services von Drittanbietern** wie Authentifizierung oder Datenbankdienste.
- Die Entwicklung von Funktionen für FaaS-Plattformen werden häufig auf sehr Anwendungsspezifische, sicherheitsrelevante oder rechenintensive Funktionen beschränkt.
- Funktionen, die klassisch auf einem zentralen Anwendungsserver bereitgestellt worden wären, werden als Vielzahl isolierter Micro- oder sogar Nanoservices (Funktionen) bereitgestellt.
- Die **Steuerung des Kontrollflusses** all dieser isolierten und autarken Services als sinnvolle Endbenutzerfunktionalität wird an Endgeräte delegiert (sehr häufig in Form von nativen mobilen Anwendungen oder progressiven Webanwendungen).



AUSBLICK

- Was ist Serverless?
- Limitierungen von Serverless
- **Serverless Architekturen**
- **Abgrenzung zu Microservices**



KONTAKT

Disclaimer

Nane Kratzke

☎ +49 451 300-5549

✉ nane.kratzke@th-luebeck.de

🔗 kratzke.mylab.th-luebeck.de

