CLOUD-NATIVE COMPUTING

Unit 06:

Function as a Service

FaaS

Stand: 27.03.23

KAPITEL 10

Function as a Service

Nane Kratzke

Cloud-native Computing

Software Engineering von Diensten und Applikationen für die Cloud

284 Seiten. E-Book inside

€ 59,99. ISBN 978-3-446-46228-1

Weitere Informationen unter: **www.hanser-fachbuch.de** HANSER

CLOUD NATIVE COMPUTING

creative commons

CLOUD NATIVE COMPUTING

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).

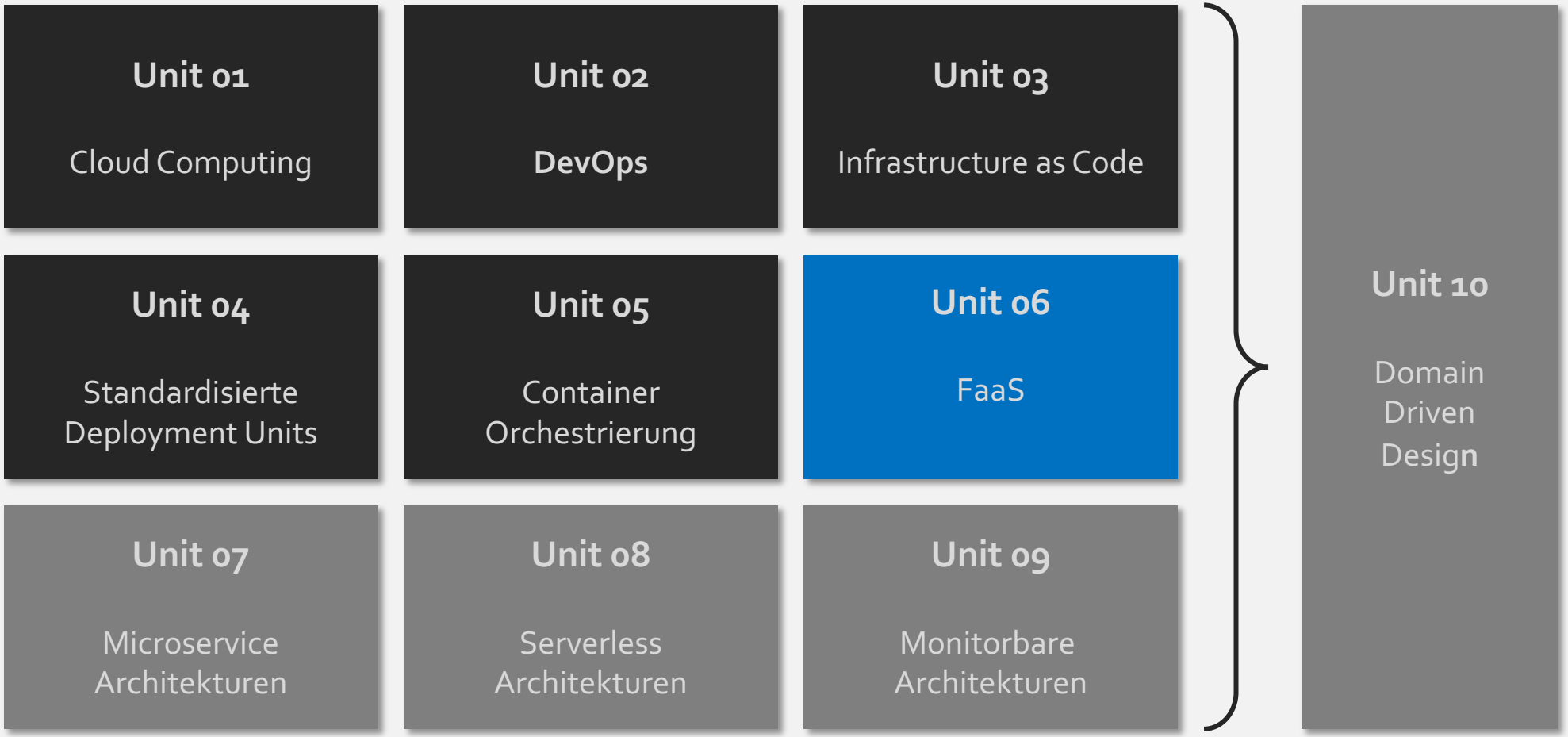
Diese Folien sind insb. für die Lehre an Hochschulen konzipiert und machen daher vom **§51 UrhG (Zitate)** Gebrauch.

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.

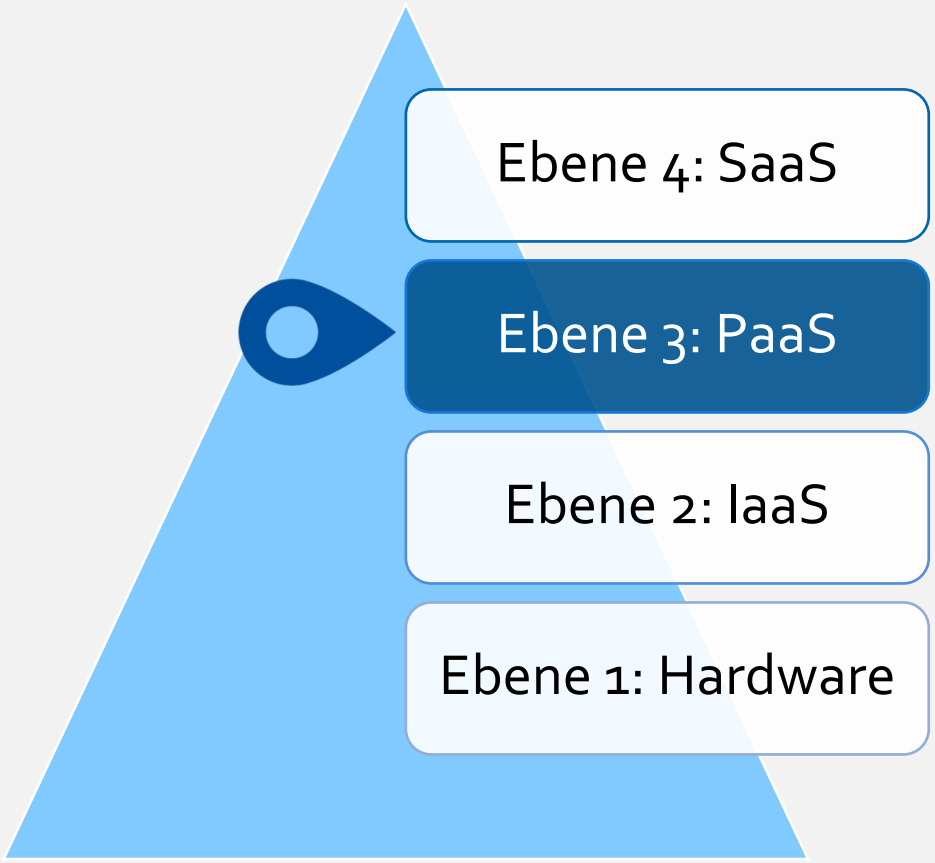
INHALTSVERZEICHNIS

Überblick über Units und Themen dieses Moduls



DAS SCHICHTENMODELL DES CLOUD COMPUTINGS

Wo sind wir?



Kunden, Endnutzer
- Anpassbare Software-Dienste
- XaaS (Everything as a Service)
- Transparente Updates

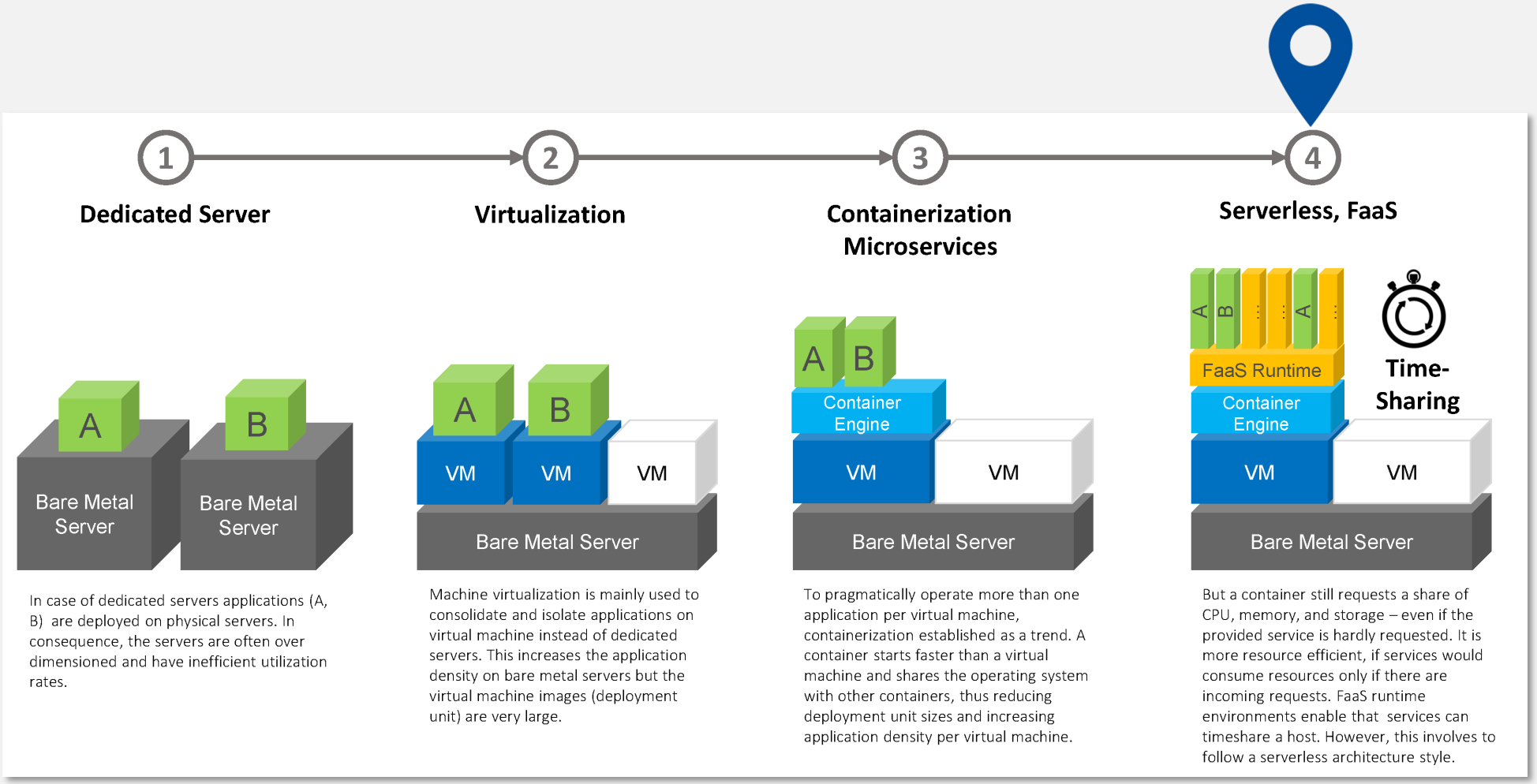
Entwickler
- **Programmierschnittstellen (APIs)**
- **Plattformdienste**
- **Abstraktion der technischen Infrastruktur**

Administratoren
- Elastizität
- Virtuelle Ressourcenpools
- Technische Infrastruktur (VM, Storage, Network)

Rechenzentrum
- Rechner
- Netzwerk
- Storage

EINE KURZE GESCHICHTE DER CLOUD

Wo sind wir?



In case of dedicated servers applications (A, B) are deployed on physical servers. In consequence, the servers are often over dimensioned and have inefficient utilization rates.

Machine virtualization is mainly used to consolidate and isolate applications on virtual machine instead of dedicated servers. This increases the application density on bare metal servers but the virtual machine images (deployment unit) are very large.

To pragmatically operate more than one application per virtual machine, containerization established as a trend. A container starts faster than a virtual machine and shares the operating system with other containers, thus reducing deployment unit sizes and increasing application density per virtual machine.

But a container still requests a share of CPU, memory, and storage – even if the provided service is hardly requested. It is more resource efficient, if services would consume resources only if there are incoming requests. FaaS runtime environments enable that services can timeshare a host. However, this involves to follow a serverless architecture style.

INHALTE

Serverless Computing

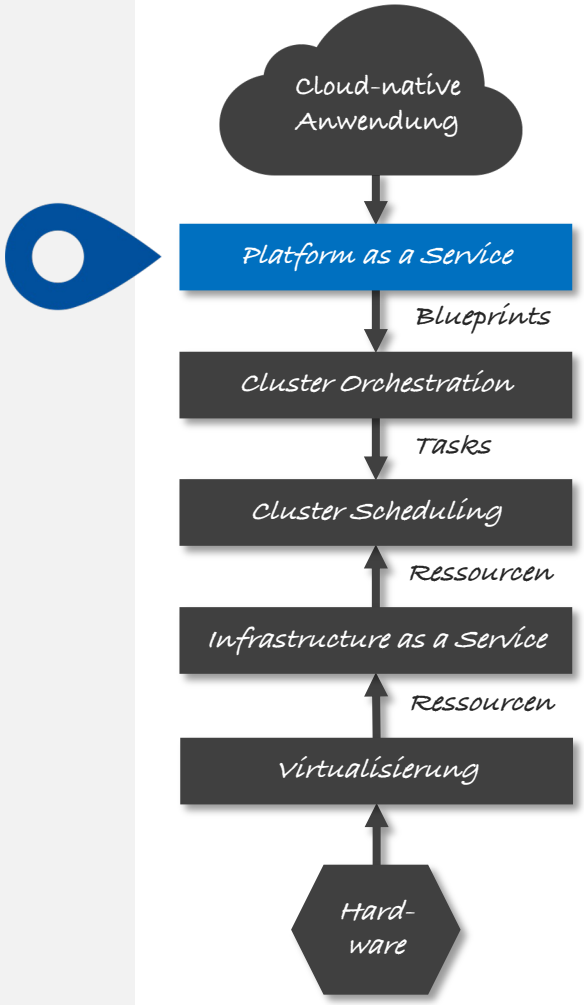
- Wie alles begann
- Serverless Definition
- A Berkley View on Serverless Computing (Simplified Cloud Programming)
- Limitierung von FaaS und Serverless

FaaS Plattformen

- Funktionen als bewährtes Serverless Application Programming Model
- Triggers und Actions
- Best Practices
- Überblick über FaaS-Provider, -Plattformen und -Frameworks

Inside Kubeless

- Typ-Vertreter (Kubernetes Deployable)
- kubeless CLI
- Funktions Interface
- Trigger (HTTP, Scheduled, PubSub/Kafka)



INHALTE

Serverless Computing

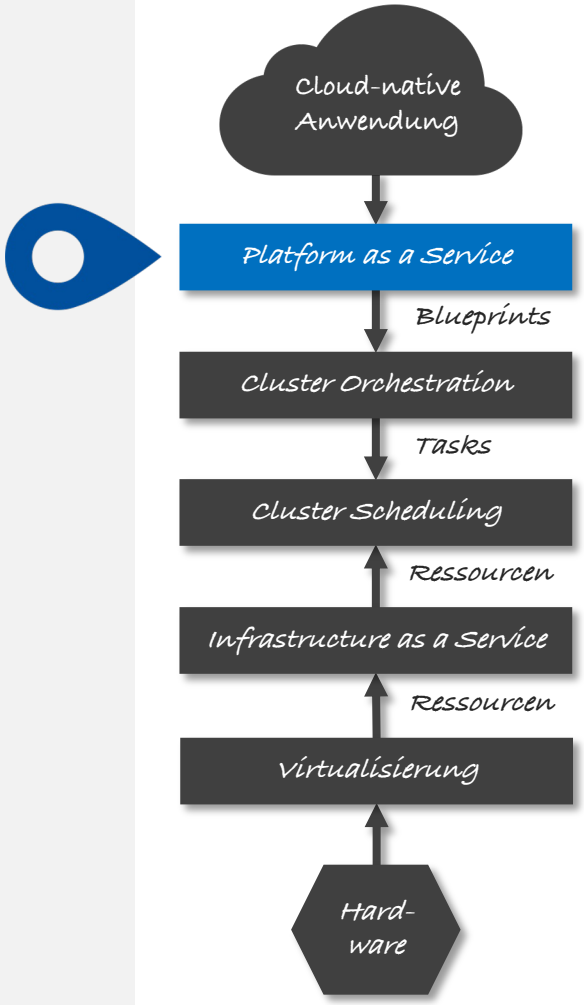
- Wie alles begann
- Serverless Definition
- A Berkley View on Serverless Computing (Simplified Cloud Programming)
- Limitierung von FaaS und Serverless

FaaS Plattformen

- Funktionen als bewährtes Serverless Application Programming Model
- Triggers und Actions
- Best Practices
- Überblick über FaaS-Provider, -Plattformen und -Frameworks

Inside Kubeless

- Typ-Vertreter (Kubernetes Deployable)
- kubeless CLI
- Funktions Interface
- Trigger (HTTP, Scheduled, PubSub/Kafka)



AWS LAMBDA

Function as a Service

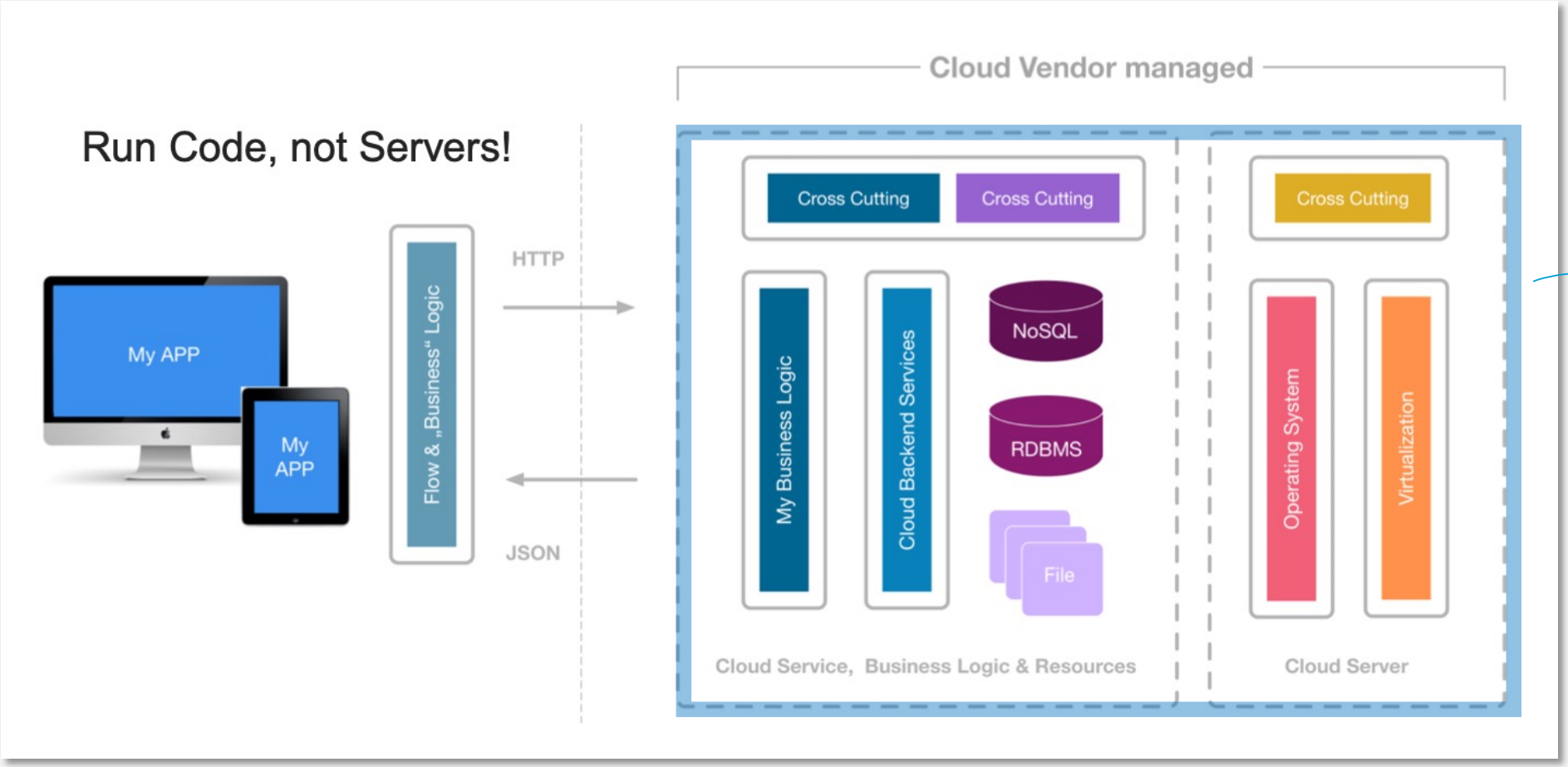
„Kein Server ist einfacher zu verwalten, als kein Server!“

Werner Vogels, CTO, Amazon

Erstes Release:
November 2014

SERVERLESS

Anwendungsarchitektur



- *High Availability*
- *Zero Administration*
- *Auto Scale to Zero*
- *Your Code*

CLOUD NATIVE COMPUTING

WAS IST SERVERLESS?

CNCF Definition

*„Serverless computing refers to a new model of cloud native computing, enabled by architectures that do **not require server management** to build and run applications. This landscape illustrates a **finer-grained deployment model** where applications, bundled as one or more **functions**, are uploaded to a platform and then executed, scaled, and billed in response to the **exact demand** needed at the moment.“*

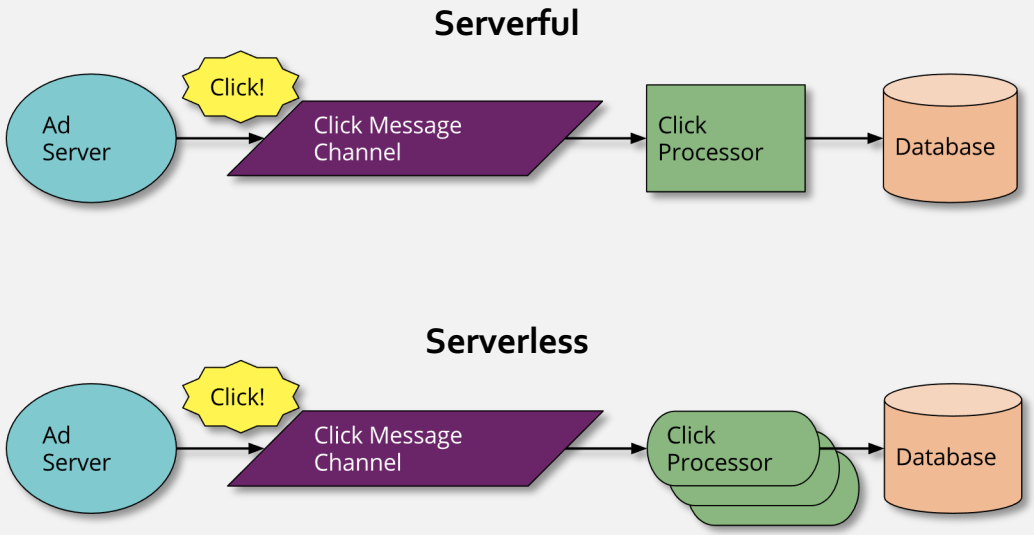
Cloud Native Computing Foundation, CNCF

WAS IST SERVERLESS?

A ThoughtWorks Point of View

„Serverless [...] mean applications where server-side logic is still written by the application developer, but, unlike traditional architectures, it's run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a third party."

Mike Roberts



Quelle: https://martinfowler.com/articles/serverless.html

CLOUD NATIVE COMPUTING

WAS IST SERVERLESS?

Unpacking Function as a Service (FaaS)

1. Fundamentally, FaaS is about running backend code without managing your own server systems or your own long-lived server applications.
2. FaaS offerings do not require coding to a specific framework or library. FaaS functions are regular applications when it comes to language and environment.
3. Deployment is very different from traditional systems since we have no server applications to run ourselves. In a FaaS environment we upload the code for our function to the FaaS provider, and the provider does everything else.
4. Horizontal scaling is completely automatic, elastic, and managed by the provider. The vendor handles all underlying resource provisioning and allocation.
5. Functions in FaaS are typically triggered by event types defined by the provider.
6. Most providers also allow functions to be triggered as a response to inbound HTTP requests.

Quelle: https://martinfowler.com/articles/serverless.html

CLOUD PROGRAMMING SIMPLIFIED

A Berkley View on Serverless Computing

CLOUD NATIVE COMPUTING

*„In any serverless platform, the user just writes a **cloud function in a high-level language**, picks the event that should **trigger** the running of the function—such as loading an image into cloud storage or adding an image thumbnail to a database table—and lets the **serverless system handle everything else**: instance selection, scaling, deployment, fault tolerance, monitoring, logging, security patches, and so on.“*

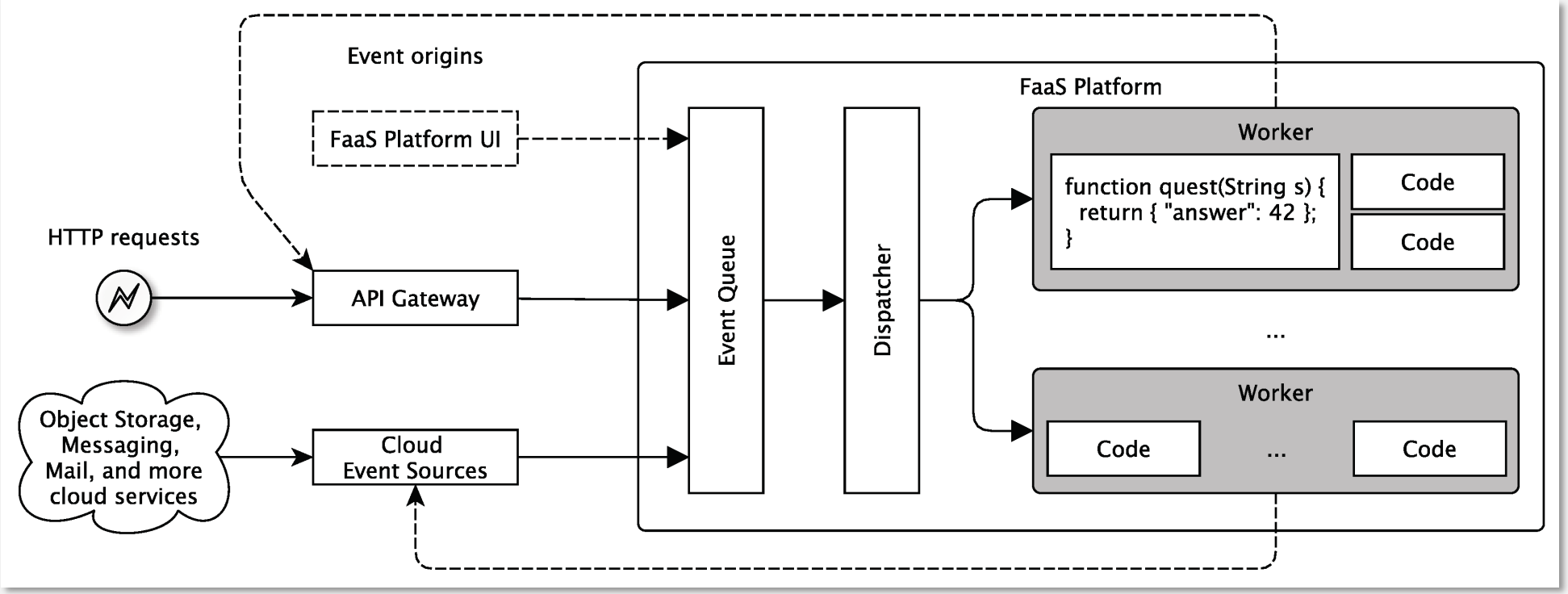
1. Redundancy for availability, so that a single machine failure doesn't take down the service.
2. Geographic distribution of redundant copies to preserve the service in case of disaster.
3. Load balancing and request routing to efficiently utilize resources.
4. Autoscaling in response to changes in load to scale up or down the system.
5. Monitoring to make sure the service is still running well.
6. Logging to record messages needed for debugging or performance tuning.
7. System upgrades, including security patching.
8. Migration to new instances as they become available.

Zu lösende Probleme, wenn man elastische Cloud Umgebungen für Nutzer einrichten möchte.

Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

FAAS - PLATFORMEN

Architektur



Quelle: Nane Kratzke: A Brief History of Cloud Application Architectures. In *Applied Sciences*; MDPI, 2018, https://doi.org/10.3390/app8081368

CLOUD PROGRAMMING SIMPLIFIED

A Berkley View on Serverless Computing

	Characteristic	*AWS Serverless Cloud*	*AWS Serverful Cloud*
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.0000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible

Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

CLOUD NATIVE COMPUTING

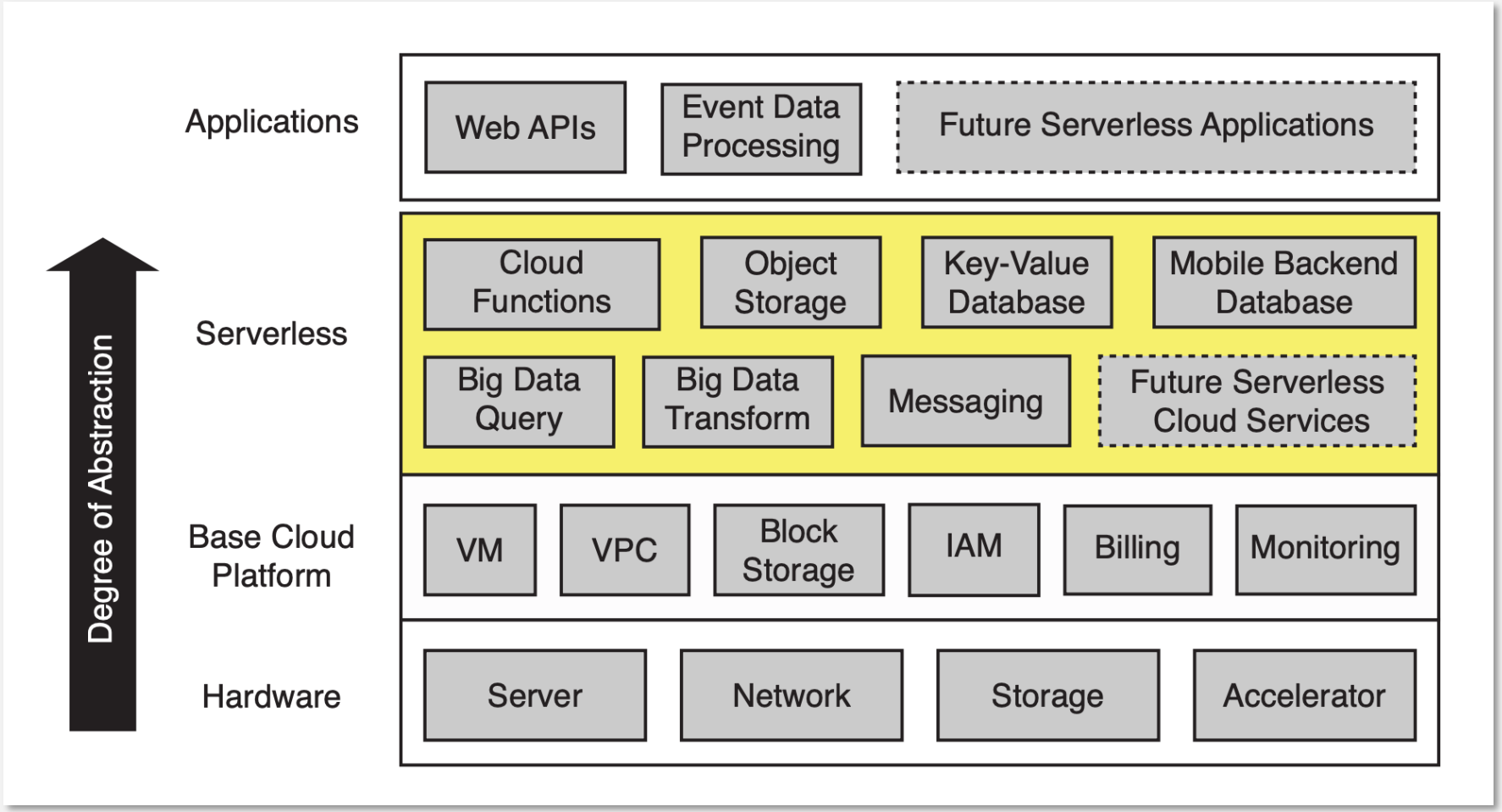
FaaS ermöglicht (wie Container) zeitliche Allokationen im Subsekunden-Bereich.

Anders als Container sind aber Funktionen keine Always-On Components, sondern Event-triggered.

Kein Event => keine Instanz (Scale to Zero)!

CLOUD PROGRAMMING SIMPLIFIED

A Berkley View on Serverless Computing



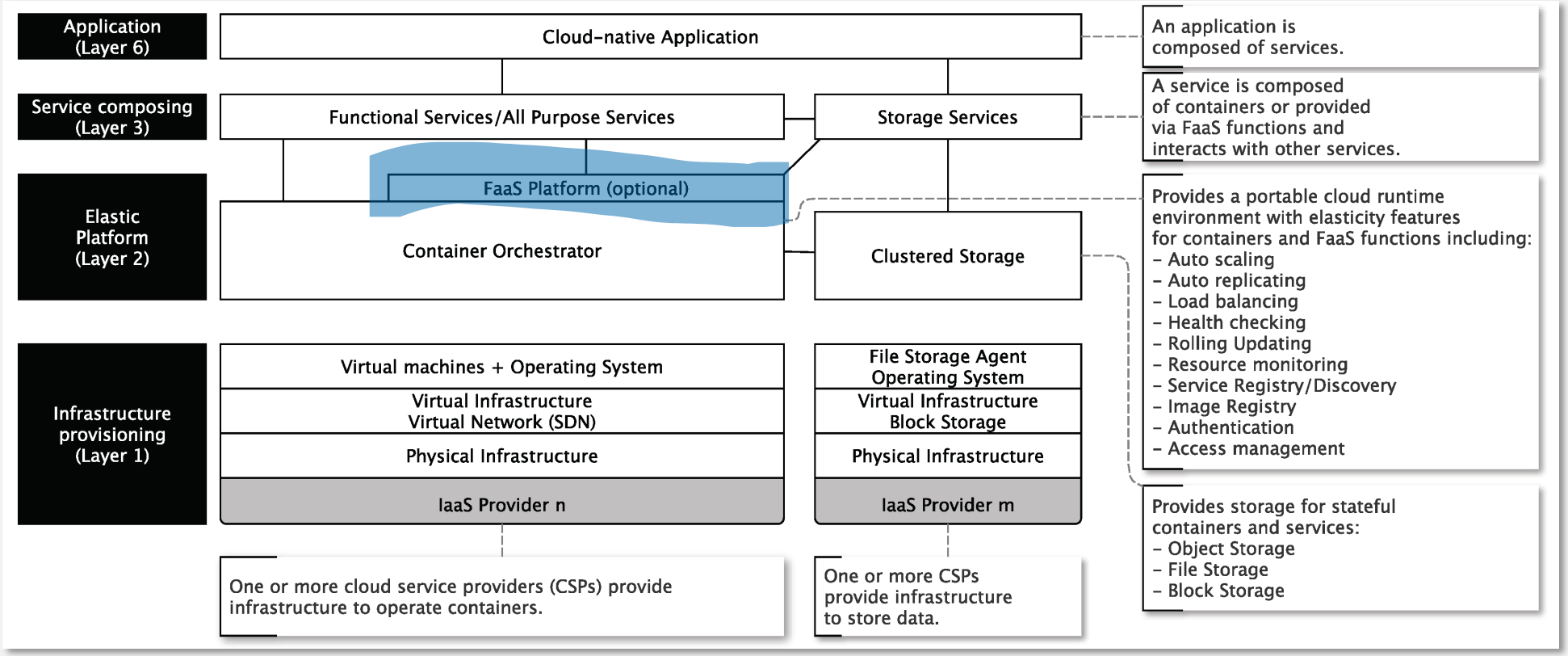
Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

CLOUD NATIVE COMPUTING

FaaS schiebt sich als eine Event-getriebene PaaS-Schicht zur Entwicklung von CNA-Komponenten zwischen Cloud-Plattformen und -Infrastrukturen.

CLOUD-NATIVE REFERENZARCHITEKTUR

Wo sitzen FaaS-Plattformen?

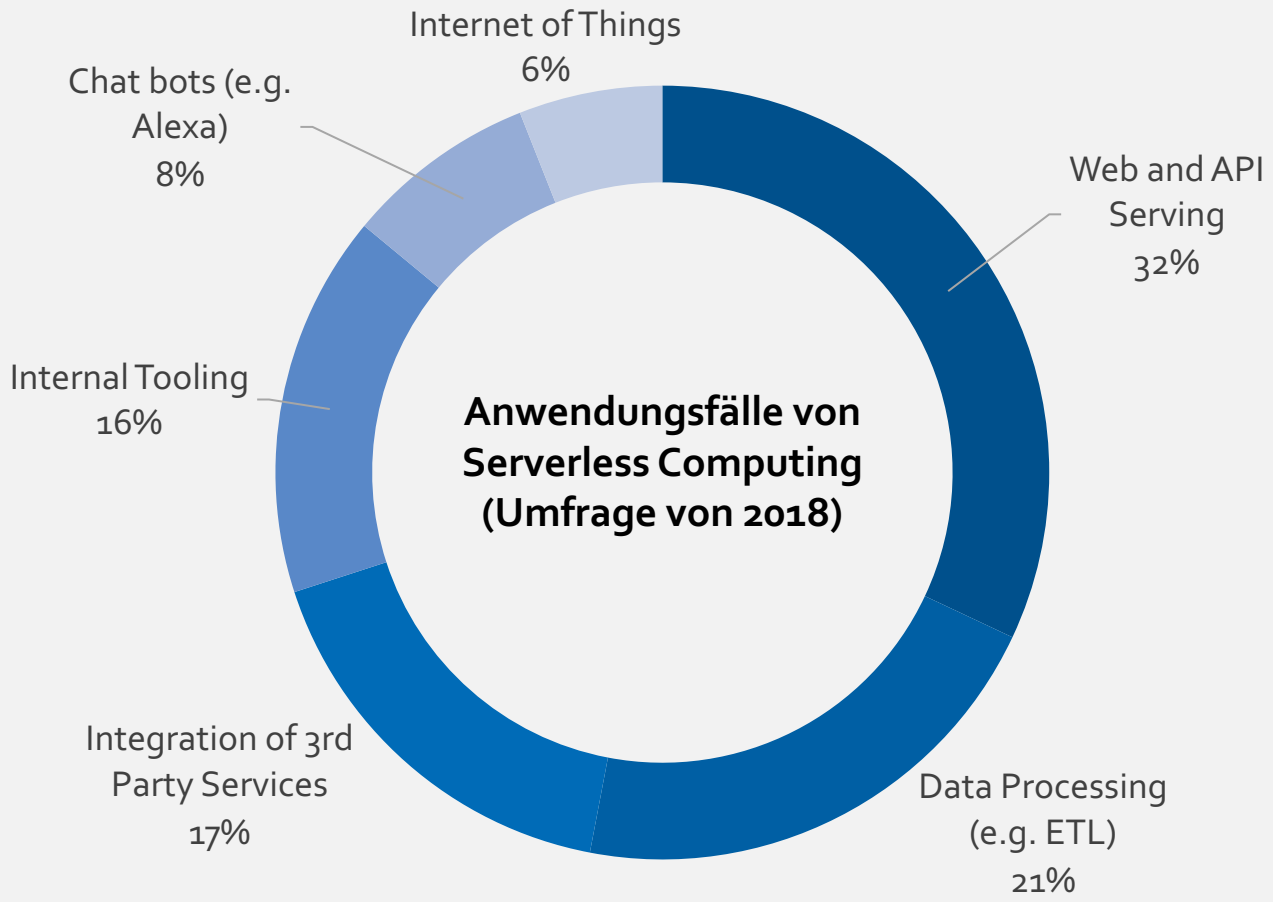


Diese Referenz-architektur hatten wir schon einmal in Unit 01!

Quelle: Nane Kratzke: A Brief History of Cloud Application Architectures. In *Applied Sciences*; MDPI, 2018, https://doi.org/10.3390/app8081368

CLOUD PROGRAMMING SIMPLIFIED

A Berkley View on Serverless Computing



Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

CLOUD NATIVE COMPUTING

FaaS wird überwiegend für Web und API-Serving, ETL-Data Processing und Integration (Glueing) von Drittanbieter-Diensten genutzt.

Alles Ereignis-gesteuerte Anwendungs-fälle.

LIMITATIONS OF SERVERLESS PLATFORMS

State, Execution Duration, Startup Latency and Cold Starts

CLOUD NATIVE COMPUTING

State

- Any state of a FaaS function that is required to be persistent needs to be externalized outside of the FaaS function instance.
- For FaaS functions that are a purely functional transformation of their input to their output—this is of no concern. Also the "Twelve-Factor app" concept has precisely the same restriction.
- State-oriented functions will typically make use of a database, a cross-application cache (like Redis), or network file/object store (like S3) to store state across requests

Execution Duration

- FaaS functions are typically limited in how long each invocation is allowed to run.
- Typical limits are several minutes (e.g. 5 – 10 minutes).
- So, certain classes of long-lived tasks are not suited to FaaS functions without re-architecture.
- You may need to create several different coordinated FaaS functions, whereas in a traditional environment you may have one long-duration task performing both coordination and execution.

Latency and Cold Starts

- It takes some time for a FaaS platform to initialize an instance of a function before each event.
- This startup latency can vary significantly, even for one specific function, depending on a large number of factors, and may range anywhere from a few milliseconds to several seconds.
- Initialization of a Function will either be a "warm start"—reusing an instance and its host container from a previous event—or a "cold start" —creating a new container instance, starting the function host process.

Quelle: https://martinfowler.com/articles/serverless.html

LIMITATIONS OF SERVERLESS PLATFORMS

A Berkley View on Serverless Computing

Ungeeignete Speicherformen für fein-granularen Betrieb

- Hohe Zugriffslatenzen für Object Storage
- Es fehlen autoskalierende Speicher Services mit niedrigen Latenzen

Fehlende Koordination für feingranulare Kontrollflüsse

- Double Spending Problem
- Bestehende Stand-Alone Notification Services kommen mit messbaren Latenzen einher

Schlechte Leistung ausgerechnet bei Standard-Kommunikationsmustern

- Funktionen können Daten nicht aggregieren/kombinieren, selbst wenn sie auf demselben Host ausgeführt werden.
- Vgl. folgender Exkurs

(Un-)Predicatable Performance Varying Performance

- Cold start latency
- Function start, Environment initialization, Application-specific initialization

Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

EXKURS:

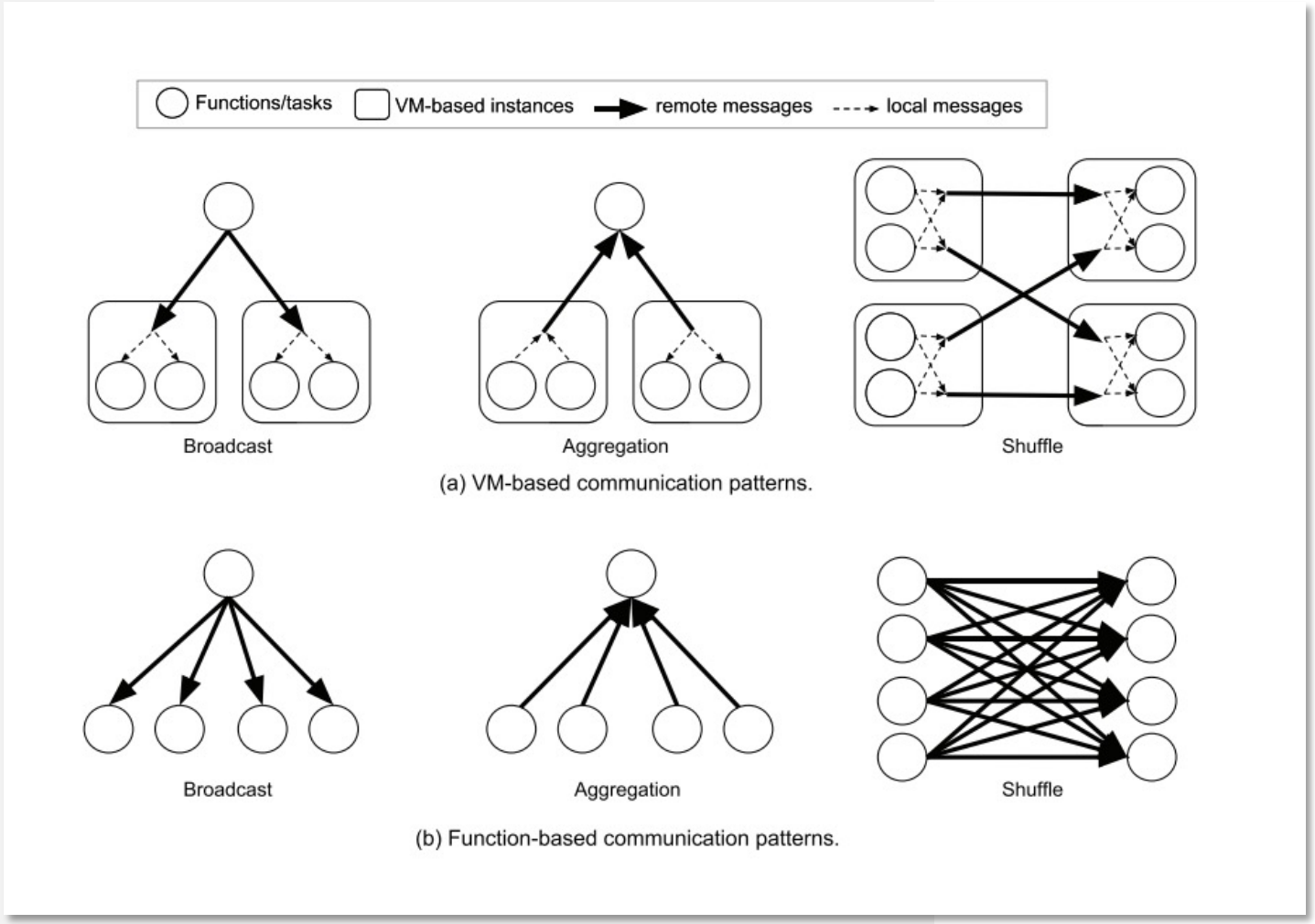
Kommunikations-Muster

Drei gängige Kommunikationsmuster für verteilte Anwendungen: Broadcast, Aggregation und Shuffle.

a) Zeigt diese Kommunikationsmuster für VM-Instanzen an, bei denen jede Instanz zwei Funktionen / Aufgaben ausführt.
b) Zeigt dieselben Kommunikationsmuster für Cloud-Funktionsinstanzen.

Beachten Sie die deutlich geringere Anzahl von Remote-Nachrichten für die VM-basierten Lösungen (a).

VM-Instanzen bieten zahlreiche Möglichkeiten (z.B. Proxies, Caches) um Daten vor dem Senden oder nach dem Empfang lokal über Funktionen / Aufgaben hinweg gemeinsam zu nutzen, zu aggregieren oder zu kombinieren.



Quelle: Jonas et al., A Berkley View on Serverless Computing, 2019, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

EXKURS:

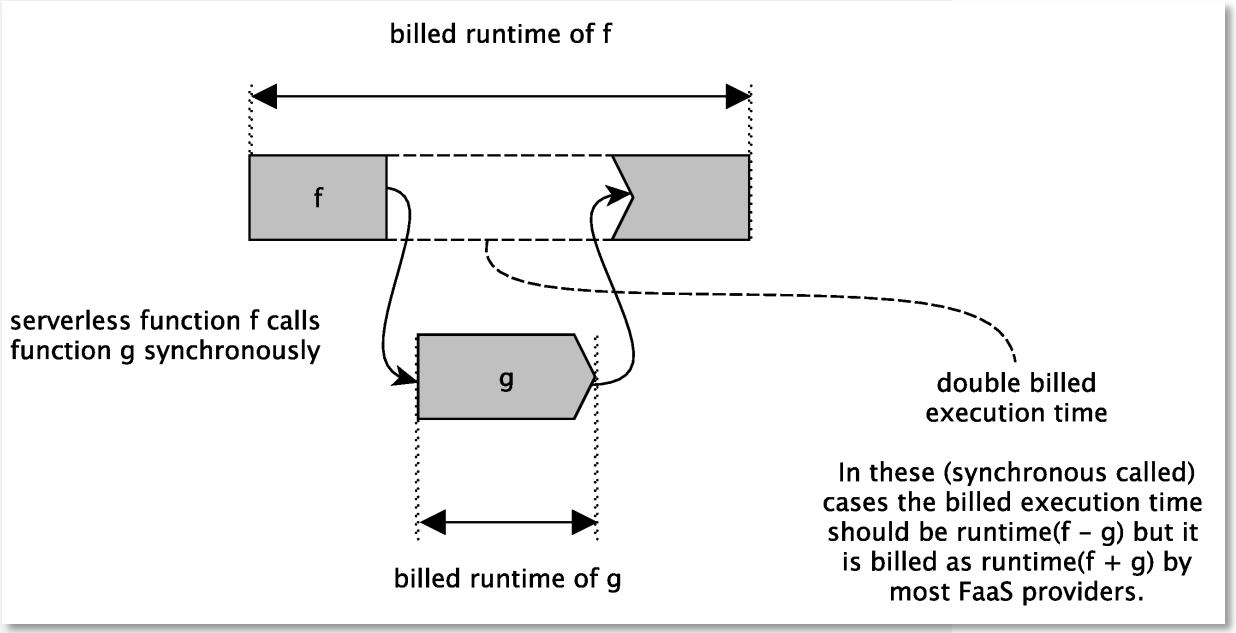
Double-Spending-Problem

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.



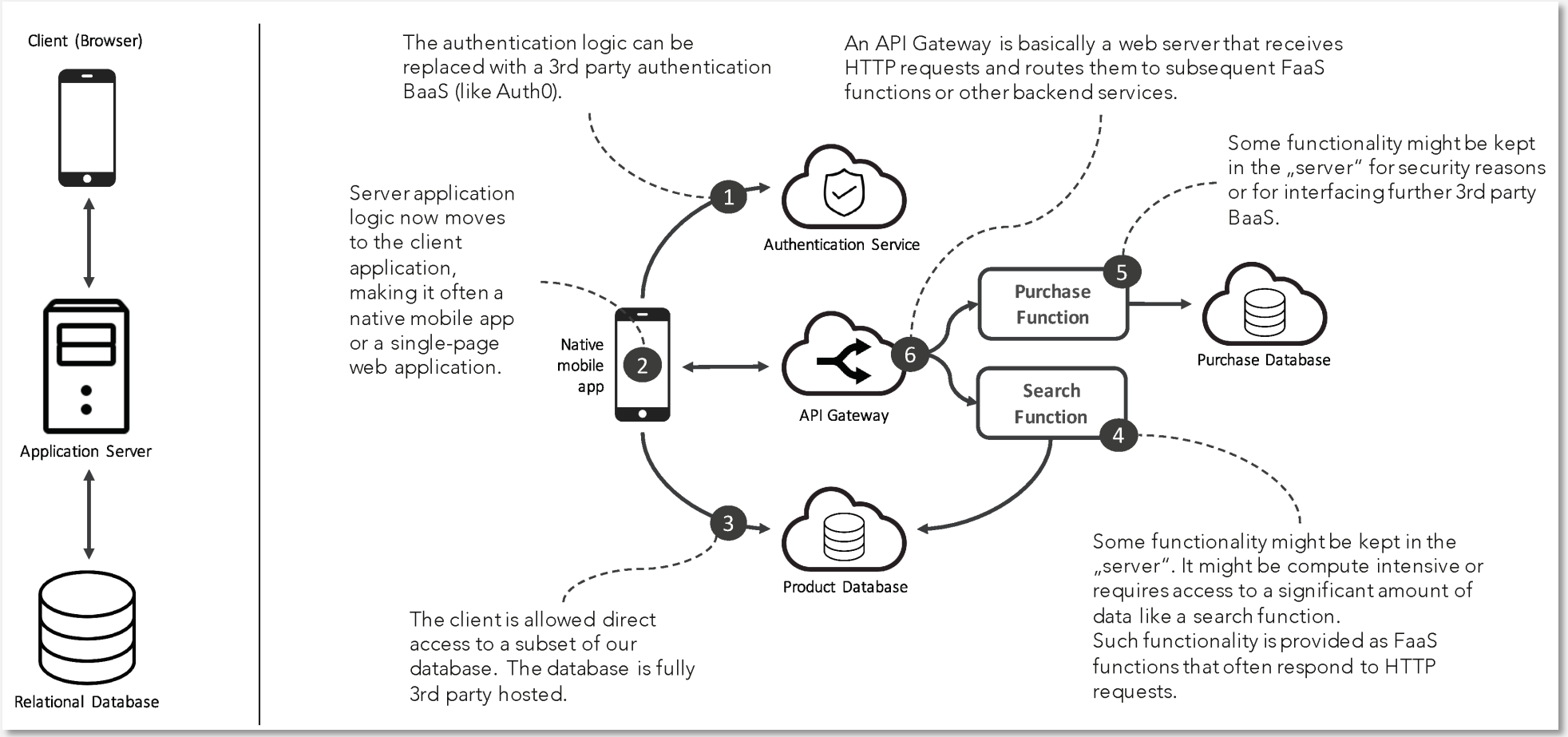
*Was man also **vermeiden** sollte ist, dass sich Funktionen **synchron untereinander aufrufen**!*

OpenWhisk addressiert dieses Problem bspw. mittels Sequences.

Quelle: Baldini et al. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*; Springer, 2017
https://arxiv.org/pdf/1706.03178.pdf

DER EFFEKT DES DOUBLE-SPENDING-PROBLEMS

Serverless-Architekturen



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

Quelle: Nane Kratzke: A Brief History of Cloud Application Architectures. In *Applied Sciences*; MDPI, 2018, https://doi.org/10.3390/app8081368

INHALTE

Serverless Computing

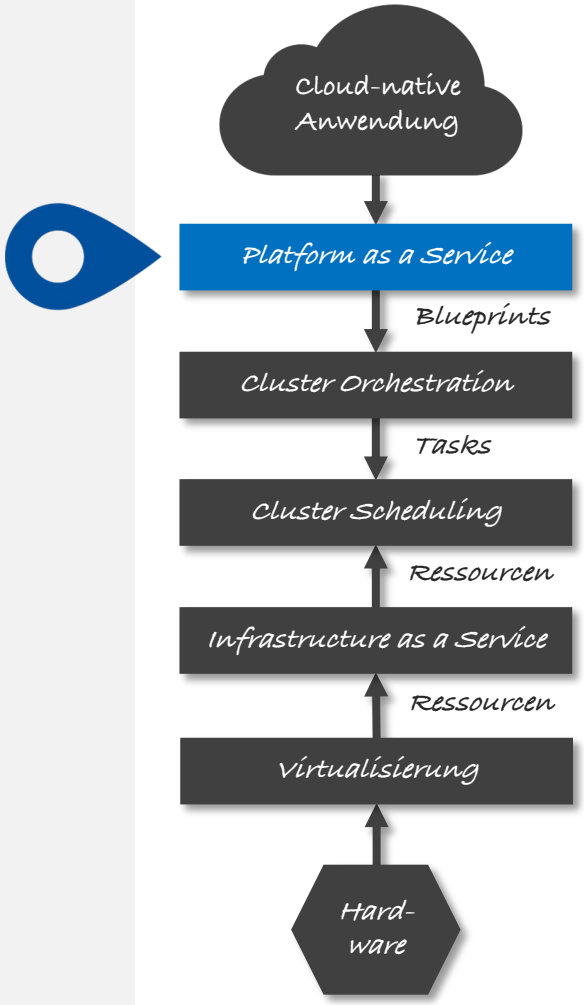
- Wie alles begann
- Serverless Definition
- A Berkley View on Serverless Computing (Simplified Cloud Programming)
- Limitierung von FaaS und Serverless

FaaS Plattformen

- Funktionen als bewährtes Serverless Application Programming Model
- Triggers und Actions
- Best Practices
- Überblick über FaaS-Provider, -Plattformen und -Frameworks

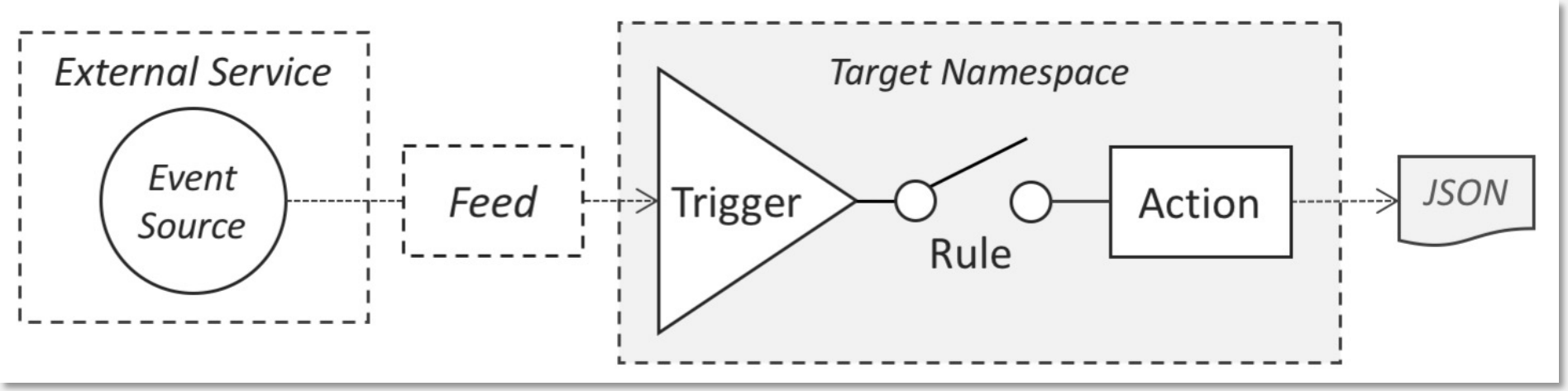
Inside Kubeless

- Typ-Vertreter (Kubernetes Deployable)
- kubeless CLI
- Funktions Interface
- Trigger (HTTP, Scheduled, PubSub/Kafka)



SERVERLESS

Funktionen als bewährtes Serverless Application Programming Model



Das Serverless Programming Model

- ist Event-driven und stateless.
- Events können aus unterschiedlichsten Quellen stammen.
- Z.B.: Datastores, Message Queues, Mobile and Web Applications, Sensors, Chat Bots, Scheduled Tasks (CRON)

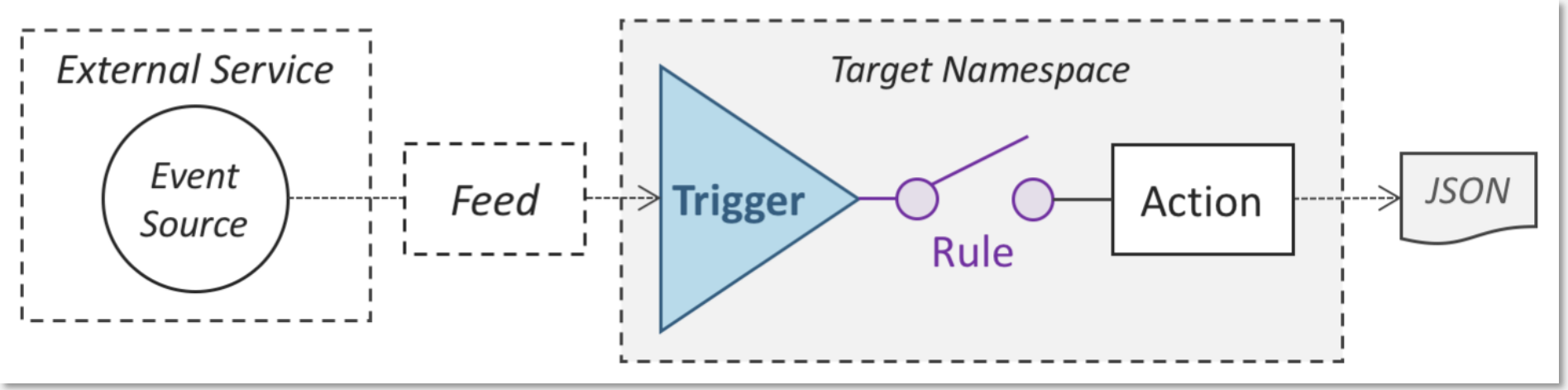
Das Serverless Programming Model

- ist polyglott und unterstützt somit meist eine Vielzahl von Language Runtimes.
- Häufig: Go, Java, JS, PHP, Python, Ruby, Swift, .NET
- Mittels Container-Images lassen sich häufig weitere Sprachen ergänzen.

Quelle: https://openwhisk.apache.org/documentation.html#programming-model

SERVERLESS

Triggers und Rules



Was sind Trigger?

Trigger sind benannte Kanäle für Klassen oder Arten von Ereignissen, die von Ereignisquellen gesendet werden.

Was sind Rules (Regeln)?

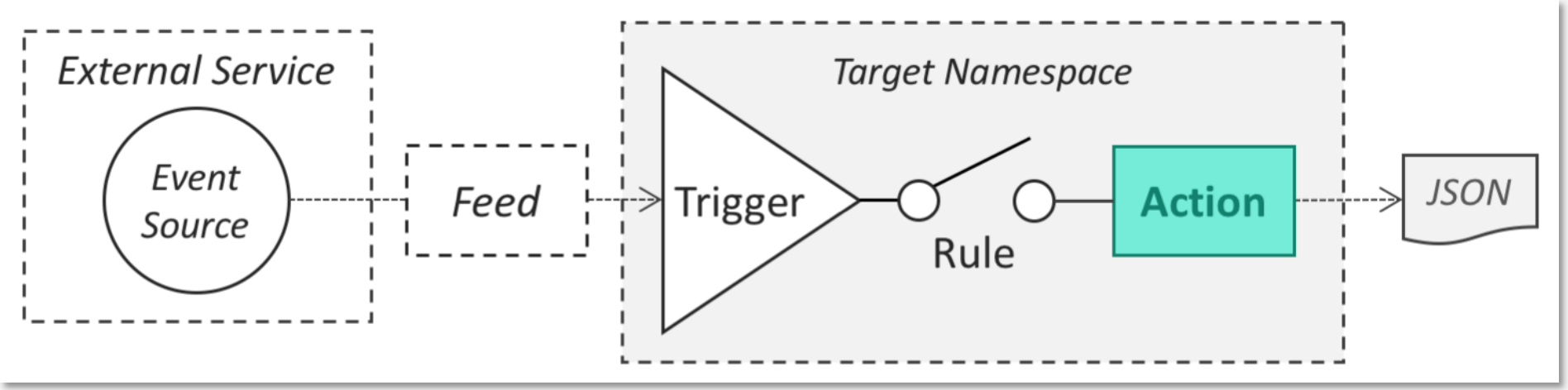
Regeln werden verwendet, um einen Trigger einer Aktion zuzuordnen. Durch diese Zuordnung, wird bei einem Auslöseereignis, die zugeordnete Aktion aufgerufen.

Was sind Ereignis-Quellen?

Hierbei handelt es sich um Dienste, die Ereignisse generieren. Solche Ereignisse weisen häufig auf Datenänderungen hin oder beinhalten die Daten selbst.

SERVERLESS

Actions



Was sind Actions?

- Actions (Aktionen) sind ***zustandslose Funktionen*** (Code-Snippets), die auf einer FaaS-Plattform ausgeführt werden.
- Aktionen kapseln Anwendungslogik, die als Reaktion auf Ereignisse ausgeführt werden soll.
- Aktionen können manuell über eine REST-API der FaaS-Plattform, einer FaaS-CLI, oder über Trigger automatisiert aufgerufen werden.

FUNCTIONS

Best Practices

CLOUD NATIVE COMPUTING

Single Purpose Functions

- Single-Responsibility Prinzip
- Funktionen sollten nur eine einzige Verantwortung haben.
- Dies macht Funktionen und deren Wechselwirkungen einfacher
- zu verstehen,
- zu testen und
- zu debuggen.

Do not chain functions

- Funktionen sollten lose gekoppelt sein.
- Funktionen sollten sich daher nicht dem **Antipattern** folgen und sich gegenseitig aufrufen (vgl. auch Double-Spending-Problem).
- Funktionen sollten vielmehr Nachrichten in Message-Queues oder in Datastores pushen, um andere Funktionen zu triggern.

FUNCTIONS

Best Practices

CLOUD NATIVE COMPUTING

Keep Functions lightweight and simple

- Jede Funktion sollte nur eine Aufgabe erfüllen.
- Jede Funktion sollte nur von einer minimalen Anzahl an externen Bibliotheken abhängen.
- Unnötiger Code macht Funktionen ggf. unnötig groß mit negativen Effekten auf die Start Time.

Design Stateless Functions

- Funktionen sollte keine Daten persistent speichern, die Funktionsaufrufe überdauern.
- Funktionen laufen normalerweise in isolierten Umgebungen und teilen nichts mit anderen Funktionsinstanzen (auch nicht mit Instanzen derselben Funktion).

FUNCTIONS

Best Practices

CLOUD NATIVE COMPUTING

Trenne Einstiegspunkte von Funktionslogik

- Funktionen haben einen Einstiegspunkt, der vom Funktionsframework aufgerufen wird.
- Der Framework-spezifische Kontext wird im Allgemeinen zusammen mit dem Aufrufkontext an diesen Einstiegspunkt übergeben.
- Wir die Funktion z.B. über ein HTTP-API-Gateway aufgerufen, enthält der Kontext HTTP-spezifische Details.
- Diese Einstiegspunktdetails sollten vom Rest des Codes getrennt werden.
- Dies verbessert die Verwaltbarkeit, Testbarkeit und Portabilität von Funktionen.

Vermeide lang laufende Funktionen

- Die meisten FaaS-Angebote (Function as a Service) haben eine Obergrenze für die Ausführungszeit pro Funktion.
- Langfristige Funktionen können daher zu Problemen wie längeren Ladezeiten und Zeitüberschreitungen führen.
- Wenn möglich, sollten Sie daher große Funktionen in kleinere Funktionen refaktorieren, die zusammen arbeiten.

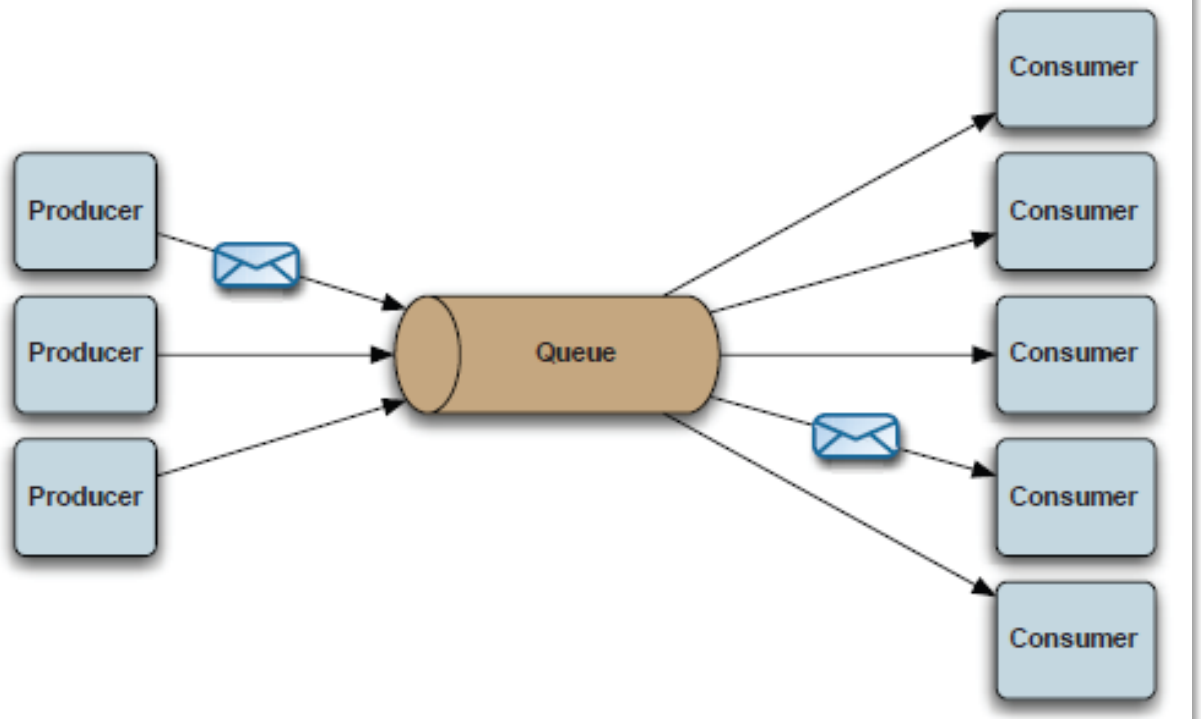
FUNCTIONS

Best Practices

CLOUD NATIVE COMPUTING

Verwende Warteschlangen für die funktionsübergreifende Kommunikation

- Anstatt Informationen untereinander weiterzugeben (enge Kopplung), sollten Funktionen eine Warteschlange verwenden (lose Kopplung), in die die Nachrichten gesendet werden.
- Andere Funktionen können basierend auf den Ereignissen in dieser Warteschlange ausgelöst und ausgeführt werden.
- Element hinzugefügt, entfernt, aktualisiert usw.



FAAS

Überblick über existierende Plattformen

Public (kommerzielle) FaaS Plattformen

AWS Lambda

Azure Functions

Google Cloud Functions

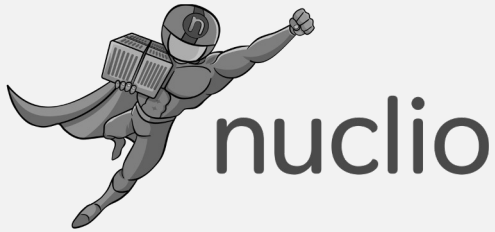
- Ähnlich wie bei PaaS-Diensten in der Anfangszeit ist FaaS allerdings nicht wirklich standardisiert.
- Funktionen des einen Dienstes sind also meist nicht 1:1 in andere Public Cloud Service Provider übertragbar.
- Funktionen sind allerdings weniger komplex (daher ist die Migration meist einfacher).
- Zudem wird mittlerweile meist die Bereitstellung von Funktionen mittels (OCI-konformen) Container-Images unterstützt.

Weitere Dienste:

- *IBM Cloud Functions (OpenWhisk)*
- *Oracle Functions (Fn Project)*
- *Alibaba Cloud Functions*
- *…*

FAAS

Überblick über existierende Open-Source Plattformen



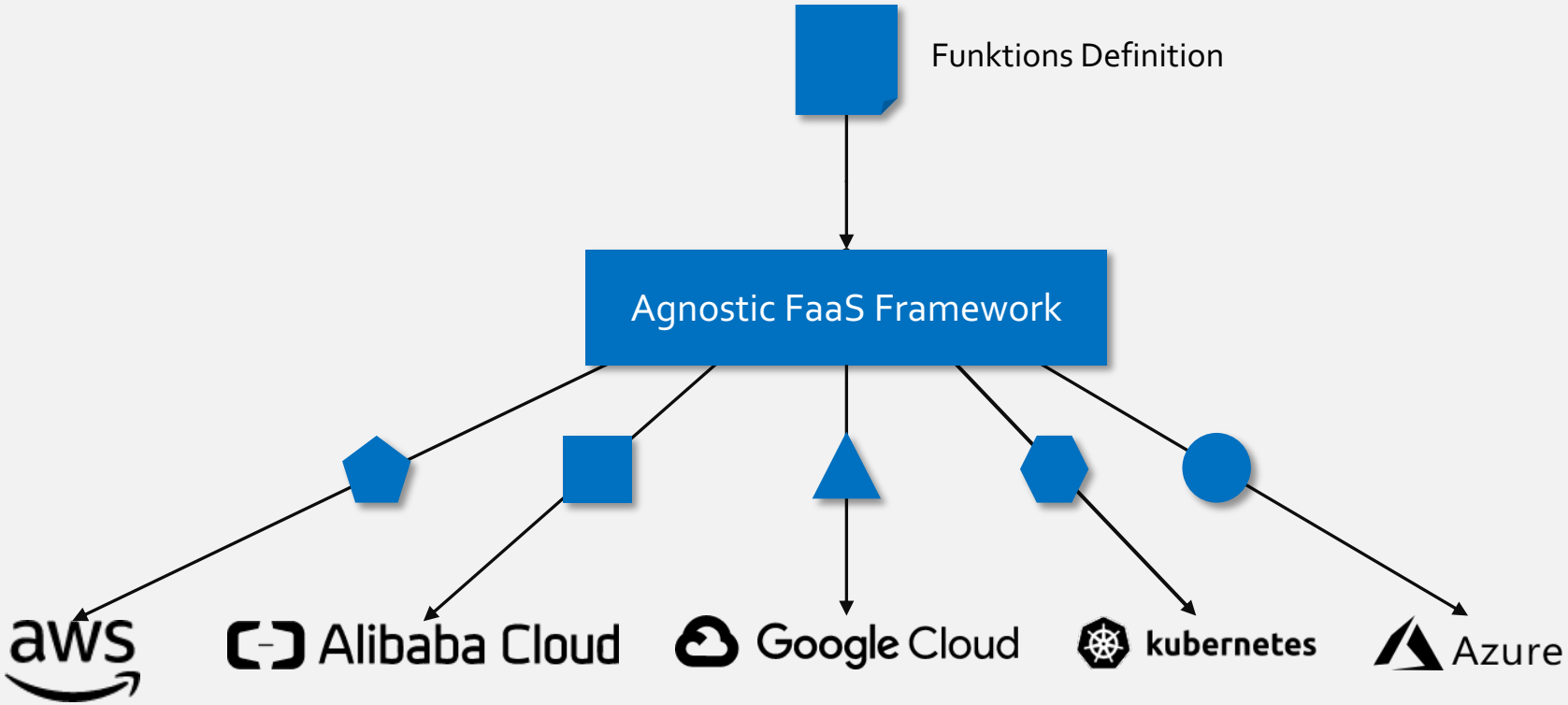
Project FN und OpenWhisk werden als kommerzielle Managed FaaS Dienste von Oracle und IBM betrieben.

Als Private FaaS ist man natürlich gezwungen, die Infrastruktur selbst zu betreiben.

Die meisten dieser Plattformen lassen sich hierfür auf Kubernete-Clustern deployen.

FAAS

Plattform-agnostische Frameworks



CLOUD NATIVE COMPUTING

Agnostische Frameworks definieren Funktionen FaaS-Plattform unabhängig und „übersetzen" diese agnostischen Funktionen für spezifische FaaS-Plattformen oder FaaS-Provider.

Beispiel:

serverless

INHALTE

Serverless Computing

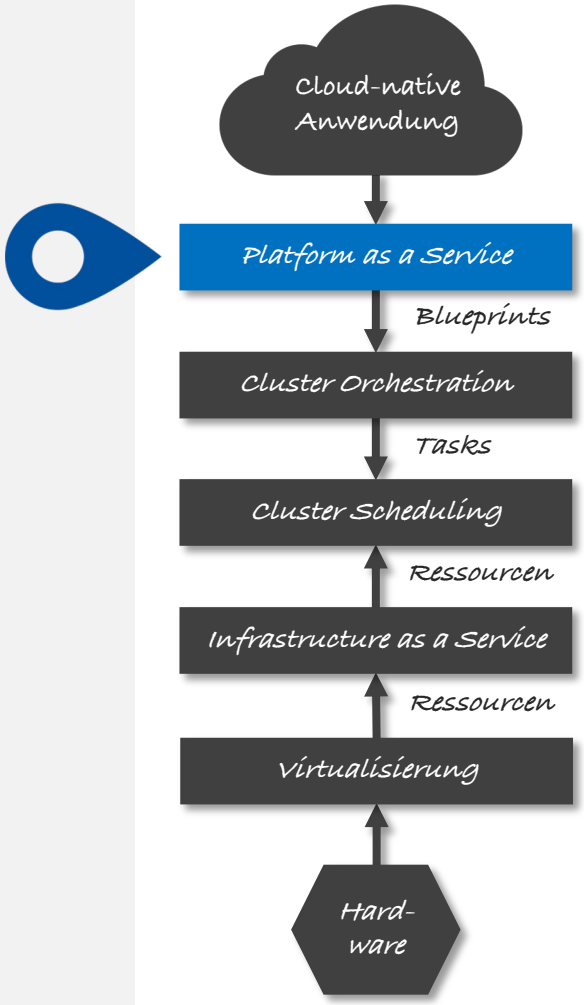
- Wie alles begann
- Serverless Definition
- A Berkley View on Serverless Computing (Simplified Cloud Programming)
- Limitierung von FaaS und Serverless

FaaS Plattformen

- Funktionen als bewährtes Serverless Application Programming Model
- Triggers und Actions
- Best Practices
- Überblick über FaaS-Provider, -Plattformen und -Frameworks

Inside Kubeless

- Typ-Vertreter (Kubernetes Deployable)
- kubeless CLI
- Funktions Interface
- Trigger (HTTP, Scheduled, PubSub/Kafka)



INSIDE

Kubeless

„Kubeless is a Kubernetes-native serverless framework that lets you deploy small bits of code (functions) without having to worry about the underlying infrastructure. It is designed to be deployed on top of a Kubernetes cluster and take advantage of all the great Kubernetes primitives. If you are looking for an open source serverless solution that clones what you can find on AWS Lambda, Azure Functions, and Google Cloud Functions, Kubeless is for you!“

kubeless.io

- Support for Python, Node.js, Ruby, PHP, Golang, .NET, Ballerina and custom runtimes
- CLI compliant with AWS Lambda CLI
- Event triggers using Kafka messaging system and HTTP events
- Prometheus monitoring of functions calls and function latency by default
- Serverless Framework plugin

INSIDE KUBELESS

Quick Start

Funktionen in Kubeless haben unabhängig von der Sprache der Funktion oder der Ereignisquelle immer das gleiche Format.

- Funktionen empfangen ein `event` als ersten Parameter. Dieser Parameter enthält alle Informationen zur Ereignisquelle. Insbesondere sollte der Schlüssel `data` den Hauptteil der Funktionsanforderung enthalten.
- Der zweite Parameter `context` mit allgemeinen Informationen zur Funktion.

Funktionen geben eine Zeichenkette oder Objekt (JSON) zurück als Antwort für den Aufrufer.

```
def hello(event, context):
  print event
  return event['data']
```

INSIDE KUBELESS

Quick Start

Funktionen können mittels des kubeless CLI ausgebracht werden:

- `hello`: Name der Funktion
- `--runtime python2.7`: Laufzeitumgebung in der die Funktion laufen soll.
- `--from-file test.py`: Datei mit dem Funktionscode (max. Größe 1 MB, etcd. Limitierung)
- `--handler test.hello`: Dies gibt die Datei und die exponierte Funktion an, die beim Empfang von Anforderungen verwendet werden. Hier: Funktion `hello` aus der Datei `test.py`

Anlegen der Funktion:

```
> kubeless function deploy hello \
  --runtime python2.7 \
  --from-file test.py \
  --handler test.hello
```

Aufrufen der Funktion per CLI (meist nur zu Testzwecken):

```
> kubeless function call hello \
  --data 'Hello world!‘

Hello world!
```

Löschen der Funktion:

```
> kubeless function delete hello
```

Überblick über alle Laufzeitumgebungen:
`https://kubeless.io/docs/runtimes/`

INSIDE KUBELESS

Functions Interface

Jede Funktion erhält zwei Argumente: Ereignis und Kontext.

Das erste Argument enthält Informationen zur Quelle des Ereignisses, das die Funktion empfangen hat.

Das zweite Argument allgemeine Informationen über die Funktion, wie den Namen oder das maximale Zeitlimit.

Funktionen müssen eine Zeichenfolge zurückgeben, die als HTTP-Antwort für den Aufrufer verwendet wird. Einige Laufzeiten unterstützen auch weitere Typen (z. B. Objekte) für die zurückgegebenen Werte.

```yaml
event:
  data:                                          # Event data
    foo: "bar"                                   # The data is parsed as JSON
  event-id: "2ebb072eb24264f55b3fff"             # Event ID
  event-type: "application/json"                 # Event content type
  event-time: "2009-11-10 23:00:00 +0000 UTC"    # Timestamp of the event sou
  event-namespace: "kafkatriggers.kubeless.io"   # Event emitter
  extensions:                                    # Optional parameters
    request: ...                                 # Reference to the request r
    response: ...                                # Reference to the response
                                                 # (specific properties will

context:
    function-name: "pubsub-nodejs"
    timeout: "180"
    runtime: "nodejs6"
    memory-limit: "128M"
```

Darstellung eines Kafka-Ereignisses in YAML

INSIDE KUBELESS

FunctionsTimeouts, Scheduled Functions, Triggers

CLOUD NATIVE COMPUTING

Timeouts:

- Für Runtimes wird ein maximales Zeitlimit durch die Umgebungsvariable `FUNC_TIMEOUT` festgelegt.
- Diese Umgebungsvariable kann mit der CLI-Option `--timeout` festgelegt werden.
- Der Standardwert ist **180 Sekunden**.
- Wenn eine Funktion mehr als die Ausführung benötigt, wird der Prozess beendet.

Scheduled Functions:

- Es ist möglich, Funktionen nach einem bestimmten Zeitplan auszulösen.
- Der Zeitplan wird im Cron-Format angegeben.
- Geplante Funktion werden mittels K8S-CronJobs ausgeführt, die HTTP-GET-Anforderung gegen die Funktion ausführen.

Triggers:

- HTTP
- CronJob (siehe Links)
- PubSub
- Kafka Trigger

Weitere Trigger können gem. Darstellung von Bitnami einfach ergänzt werden.

INSIDE KUBELESS

Exponieren von HTTP Triggers

Standardmäßig stellt Kubeless eine Funktion als Service mit ClusterIP als K8S-internen Service bereit. Aus diesem Grund steht der Befehl kubeless trigger http zur Verfügung, mit dem eine Funktion öffentlich mittels einem K8S Ingress verfügbar gemacht werden kann.

```
> kubeless function deploy test \
  --runtime python2.7 \
  --handler test.foobar \
  --from-file test.py
```

```
def foobar(event, context):
  print event['data']
  return event['data']
```

```
> kubeless trigger http create get-test \
  --function-name test \
  --path echo \
  --hostname example.com \
  [--enableTLSAcme]                  # Bei konfiguriertem Cert-manager für HTTPS
```

```
> kubectl get ing                    # Listet dann den get-test Ingress
```

Erzeugt einen über Ingress erreichbaren HTTP(S)-Trigger

INSIDE KUBELESS

PubSub Triggers (Kafka)

Neben HTTP kann jede Kubeless-Funktion über einen PubSub-Mechanismus ausgelöst werden. Funktionen werden dabei durch Ereignisse aus einem vordefinierten Topic eines Messagingsystems wie bspw. Kafka oder NATS getriggert.

```
> kubeless function deploy test \
  --runtime python2.7 \
  --handler test.foobar \
  --from-file test.py
```

```
def foobar(event, context):
  print event['data']
  return event['data']
```

```
> kubeless trigger kafka create test \
  --function-selector created-by=kubeless,function=test \
  --trigger-topic test-topic
```

Erzeugt einen Publish-Subscribe-Trigger für ein Topic

```
> kubeless topic create test-topic
> kubeless topic publish --topic test-topic --data "Hello World!"
```

Erzeugt das Topic und triggert ein Ereignis

INSIDE KUBELESS

> kubeless --help

```
Serverless framework for Kubernetes


Usage:
  kubeless [command]


Available Commands:
  autoscale         manage autoscale to function on Kubeless
  completion        Output shell completion code for the specified shell.
  function          function specific operations
  get-server-config Print the current configuration of the controller
  help              Help about any command
  topic             manage message topics in Kubeless
  trigger           trigger specific operations
  version           Print the version of Kubeless


Flags:
  -h, --help   help for kubeless


Use "kubeless [command] --help" for more information about a command.
```

CLOUD NATIVE COMPUTING

Tipp:
Die kubeless CLI ist ziemlich umfangreich dokumentiert.

FAAS

Nur Versuch macht kluch ...

Klonen Sie dieses Repository:

```
git clone https://git.mylab.th-luebeck.de/cloud-native/lab-faas.git
```

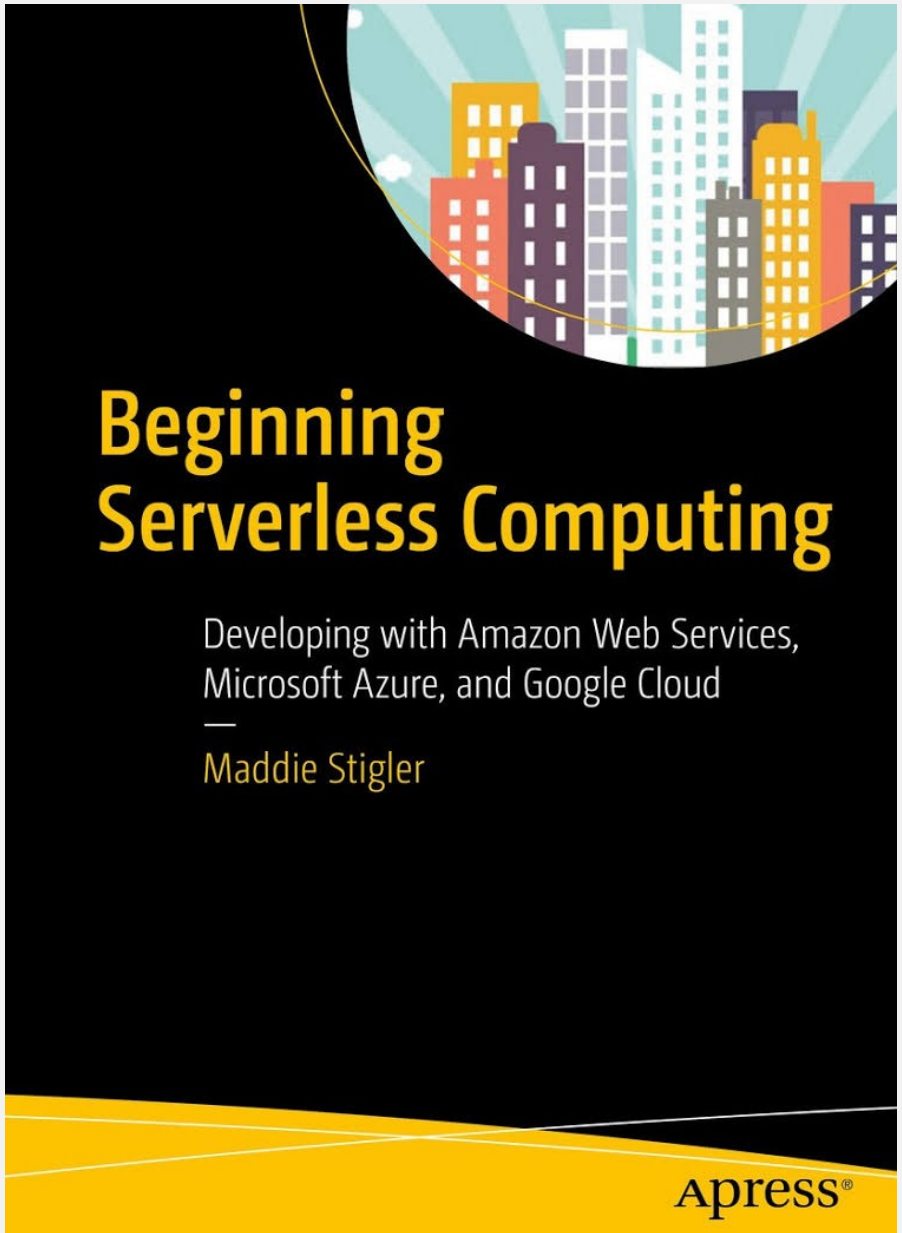
Private FaaS

> Hello World
> HTTP-Trigger
> Nanoservice für COVID-19 Plots
> CRON-Trigger: ISS-Monitor

Public/Managed FaaS

> Portierung des Nanoservice für COVID-19 Plots auf GCE

ZUM NACHLESEN



Chapter 1: Understanding Serverless Computing
- What Is Serverless Computing?
- How Is It Different?
- Benefits and Use Cases?
- Limits to Serverless Computing?

Chapter 2: Getting Started
- Provider Overview, Triggers + Event
- Tools + Environments

Chapter 3: Amazon Web Services

Chapter 4: Azure

Chapter 5: Google Cloud

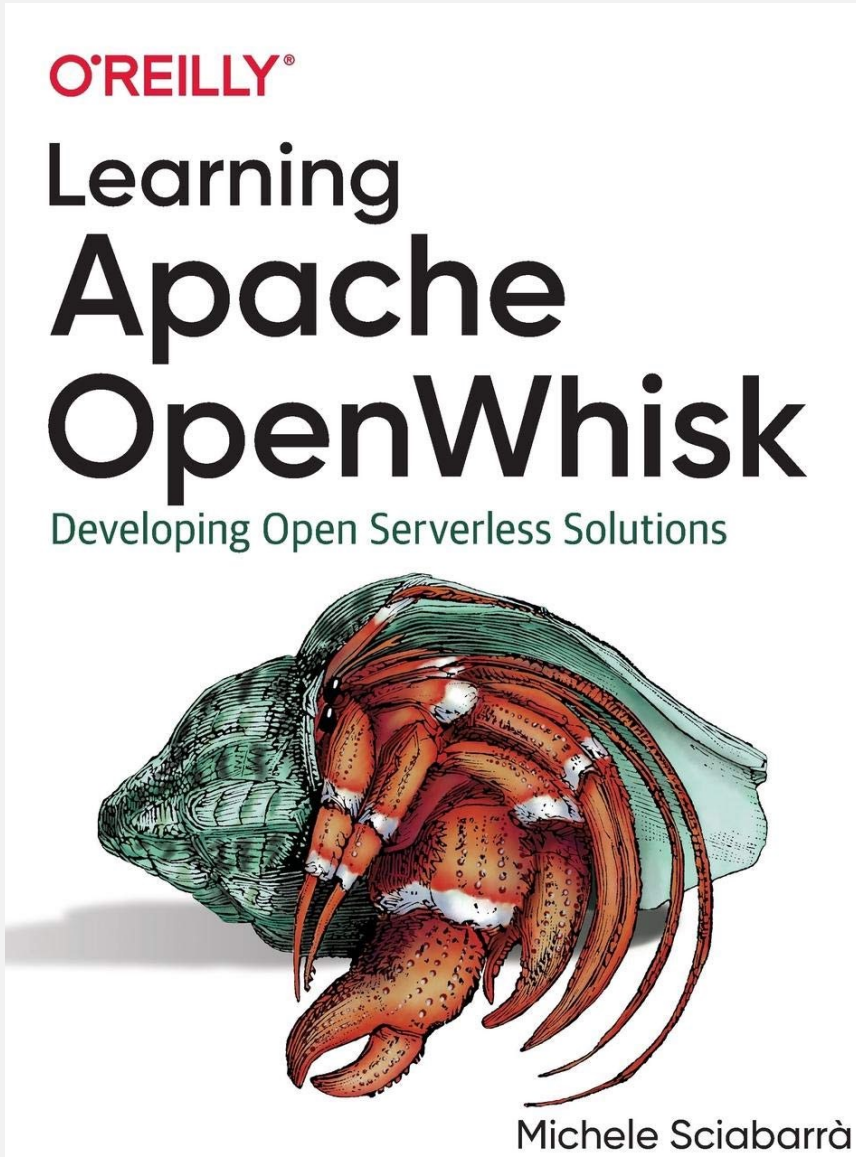
Chapter 6: An Agnostic Approach
- Need for Agnostic Solutions
- Agnostic Approaches
- Code and Examples

CLOUD NATIVE COMPUTING

Tatsächlich ist es aktuell recht schwer gute Literatur zu finden, die FaaS Provider-unabhängig aber dennoch Anwendungs-bezogen behandelt.

Dies ist für Public FaaS eine der wenigen brauchbaren Quellen, die ich gefunden habe.

ZUM NACHLESEN



Part I: Introducing OpenWhisk Development

1. Serverless and OpenWhisk Architecture
2. A Simple OpenWhisk Application
3. The OpenWhisk CLI and JavaScript API
4. Common Design Patterns in OpenWhisk
5. Integration Design Patterns in OpenWhisk
6. Unit Testing OpenWhisk Applications

Part II: Advanced OpenWhisk Development

7. Developing OpenWhisk Actions in Python
8. Using CouchDB with OpenWhisk
9. An OpenWhisk Web Application in Python
10. Developing OpenWhisk Actions in Go
11. Using Kafka with OpenWhisk
12. Deploying OpenWhisk with Kubernetes

Apache OpenWhisk ist leider (wie so viele Apache Projekte) eine im Vergleich recht komplex einzurichtende Plattform.

Daher haben wir es in diesem Modul nicht als Typvertreter behandelt.

Dennoch sei diese Quelle als umfassende Literatur für self-hosted FaaS-Plattformen empfohlen.

ZUM NACHLESEN

Paper

- Baldini et al. **Serverless Computing: Current Trends and Open Problems**. In *Research Advances in Cloud Computing*; Springer, 2017
  https://arxiv.org/pdf/1706.03178.pdf
- Nane Kratzke: **A Brief History of Cloud Application Architectures**. In *Applied Sciences*; MDPI, 2018
  https://doi.org/10.3390/app8081368
- Jonas et al., **A Berkley View on Serverless Computing**, 2019 (Technical Report, UC Berkley)
  https://www2.eecs.berkeley.edu/Pubs/TechRpts/2019/EECS-2019-3.pdf

KONTAKT

Disclaimer

Nane Kratzke

+49 451 300-5549

nane.kratzke@th-luebeck.de

kratzke.mylab.th-luebeck.de