

# A journey to serverless computing

BY  
MAIK BLÜMEL

# About me

---



**Maik Blümel**

Selbstständiger Softwareentwickler und -berater

- Webanwendungen
- FirstSpirit™ CMS
- Cloud – Themen

# Serverless?

---

**Serverless == Serverlos ???**

# Klassische Webanwendungen

---

- Verantwortung
- Kosten / Nutzen

# Klassische Webanwendungen

---

- Verantwortung

- Kosten / Nutzen

# Verantwortung

---

## Komponenten:

- Server (physik. oder virtuell)
- Betriebssystem
- Servlet-container / Applicationserver
- Anwendung (z.B. WAR, EAR, etc.)

# Verantwortung

---

## Komponenten:

- Server (physik. oder virtuell)
- Betriebssystem
- Servlet-container / Applicationserver
- Anwendung (z.B. WAR, EAR, etc.)



**Plattform / Infrastruktur**

**Anwendungspart**

# Verantwortung

---

## Komponenten:

- Server (physik. oder virtuell)
- Betriebssystem
- Servlet-container / Applicationserver
- Anwendung (z.B. WAR, EAR, etc.)



**Beschaffen, Aufsetzen, Aktualisieren, Warten !!!**

# Klassische Webanwendungen

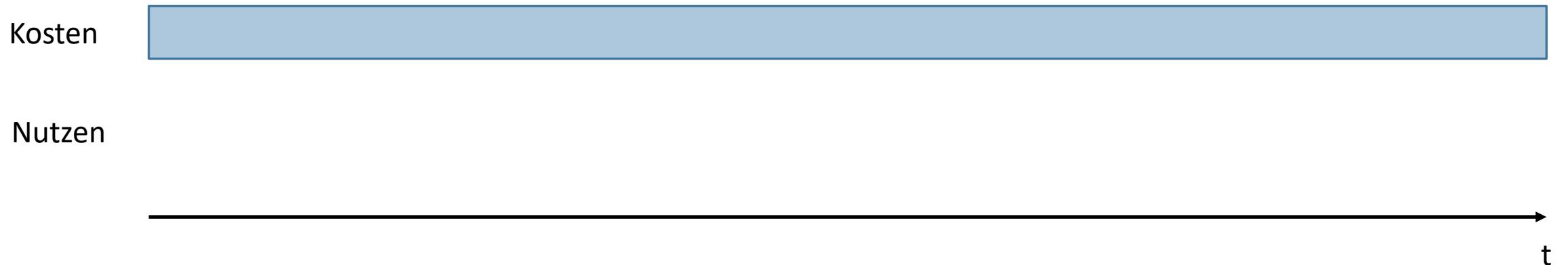
---

- Verantwortung
- Kosten / Nutzen

# Kosten / Nutzen

---

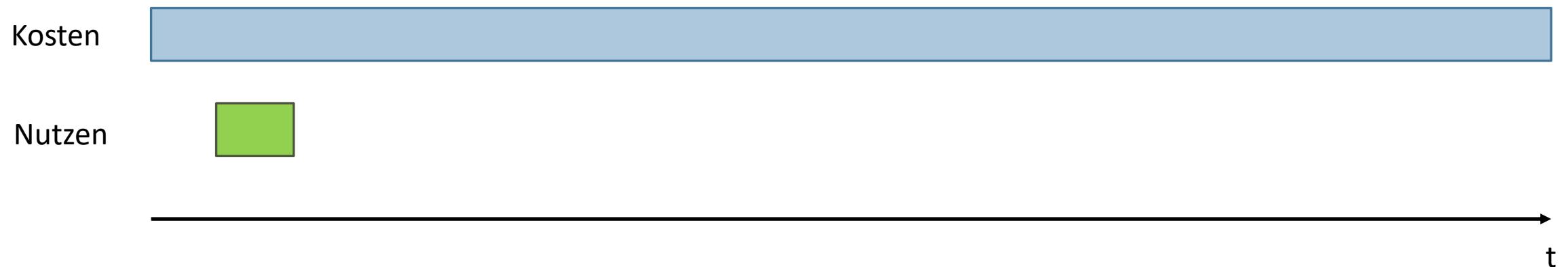
**Beispiel:** Täglicher Import aller Produkte in Webshop (Dauer ca. 30 Minuten).



# Kosten / Nutzen

---

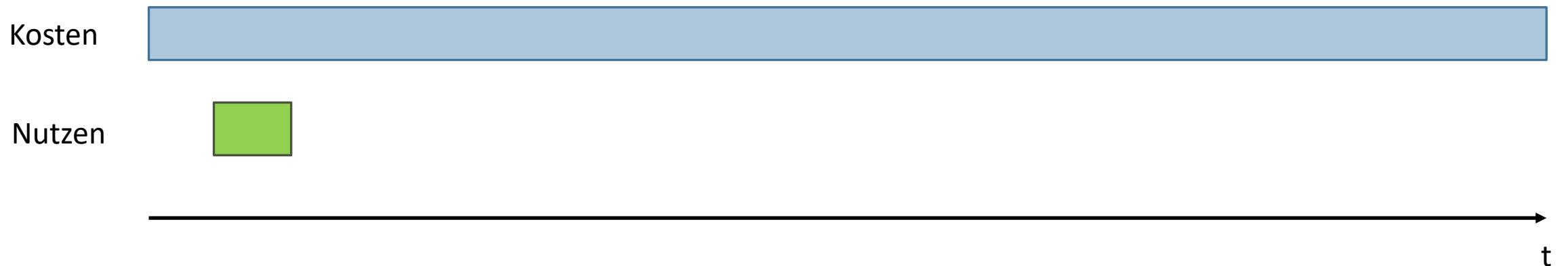
**Beispiel:** Täglicher Import aller Produkte in Webshop (Dauer ca. 30 Minuten).



# Kosten / Nutzen

---

Beispiel: Täglicher Import aller Produkte in Webshop (Dauer ca. 30 Minuten).



**Aufwendiger Betrieb für täglich 30 Minuten Nutzen**

# Klassische Webanwendungen

---



Ziel

---

**Anwendungslogik!!!**

# Optimierungsmaßnahmen

---

- Plattformaufwände minimieren => Automatisierung, Wiederverwendung, ...
- Laufzeiten optimieren => Plattformen wiederverwenden, ...
- Standardframework, APIs schaffen

# Optimierungsmaßnahmen

---

- Plattformaufwände minimieren => Automatisierung, Wiederverwendung, ...
  - Laufzeiten optimieren => Plattformen wiederverwenden, ...
  - Standardframework, APIs schaffen
- 
- *Betrieb macht jemand anderes?*

# Serverless?

---

**Serverless != Serverlos**

# Serverless?

---

**Serverless =**

**Ich konzentriere mich auf Anwendungslogik (Funktion);  
Server macht jemand anderes!!!**

**Function as a Service (FaaS)**

# Serverless Computing Manifest

---

- Functions are the unit of deployment and scaling
- No machines, VMs, Containers visible in the programming model
- Permanent Storage lives elsewhere
- Scales per request
- Never pay for idle
- Implicitly fault-tolerant because functions can run everywhere
- BYOC – Bring your own code
- Metrics and logging are a universal right

<https://de.slideshare.net/AmazonWebServices/getting-started-with-aws-lambda-and-the-serverless-cloud>

# Serverless Computing

---

- Diensteanbieter ist für komplette Laufzeitumgebung verantwortlich
  - Verfügbarkeit und Skalierfähigkeit regelt Diensteanbieter
- 
- Benutzer lädt (Funktion-)Code hoch und legt Ausführungsergebnisse fest

# Serverless Computing

---

**Code schreiben, hochladen, ausführen  
Bezahlen nur für Ausführungszeit!**

# Cloud Anbieter

---

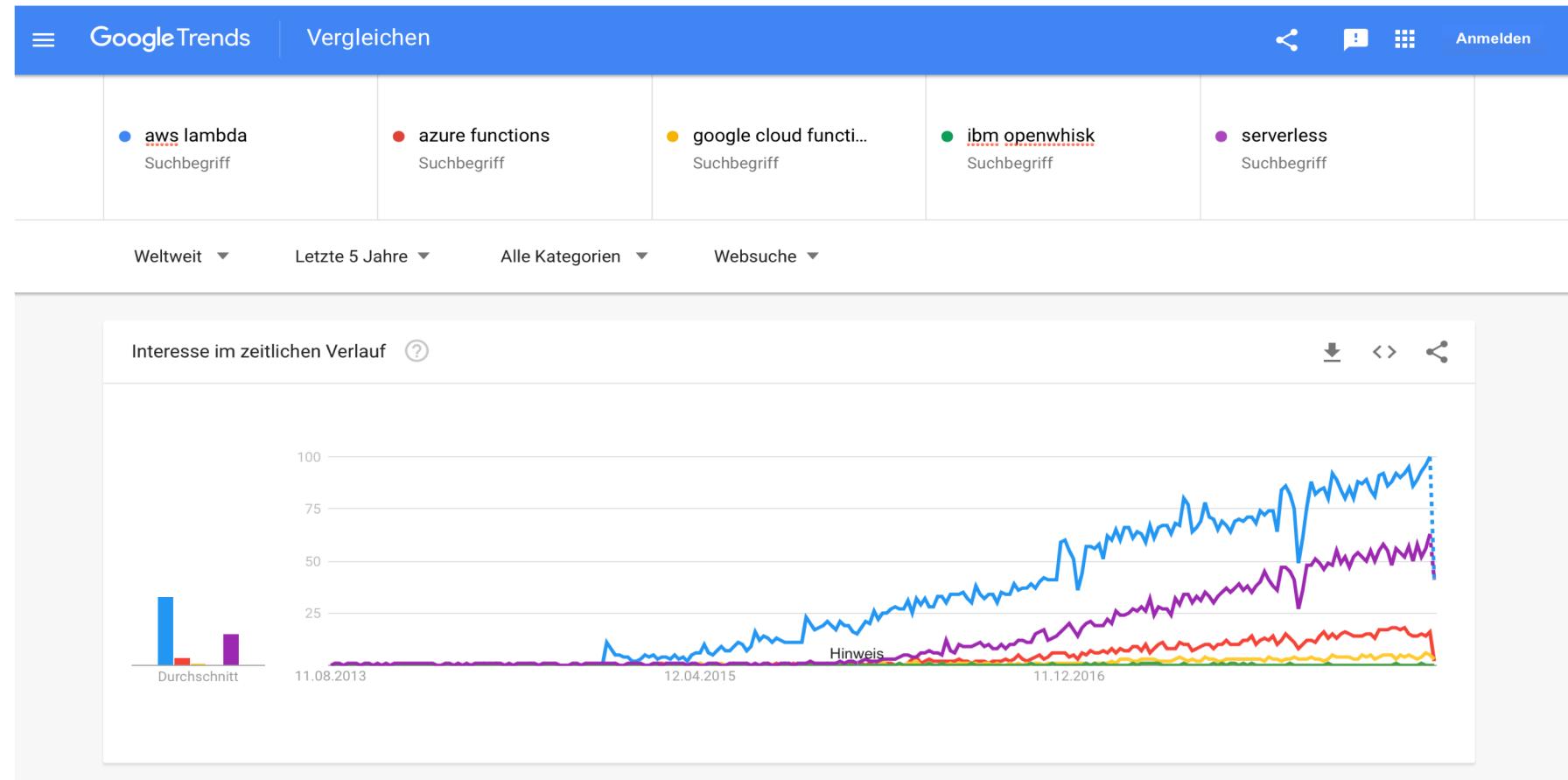
**AWS Lambda**

**Google Cloud Functions**

**Microsoft Azure Functions**

**IBM OpenWhisk**

# Cloud Anbieter



Quelle: <https://trends.google.de>

# AWS

---

- erster Cloud Provider mit Serverless Computing Angebot
- AWS Lambda seit 2014 (nur Node.js)

# AWS Serverless Dienste

---

- Compute
- APIs
- Storage
- Databases
- Interprocess messaging
- Orchestration
- Analytics

AWS Lambda 

Amazon API Gateway 

Amazon S3 

Amazon DynamoDB 

Amazon SNS / Amazon SQS  

AWS Step Functions / Cloudwatch Events



Amazon Kinesis 

# AWS Lambda

---

- Javascript (Node.js), Python, Java, .Net Core C#, Go
- Event basiert

# AWS Lambda - Limits

---

Anzahl der Prozesse und Threads (insgesamt)	1,024
Maximale Ausführungszeit pro Anfrage	300 Sekunden (5 Minuten)
Nutzlastgröße für den <a href="#">Invoke</a> -Anforderungstext (RequestResponse/synchroner Aufruf) HINWEIS: Die Nutzlast des Antworttexts muss dieses Limit ebenfalls einhalten.	6 MB
<a href="#">Invoke</a> Größe der Nutzlast für den Anforderungstext (Event / asynchroner Aufruf)	128 KB

Größe des Bereitstellungspakets für eine Lambda-Funktion (komprimierte ZIP-/JAR-Datei)	50 MB
Gesamtgröße aller Bereitstellungspakete, die pro Region hochgeladen werden können	75 GB
Größe des Codes / der Abhängigkeiten, die als Bereitstellungspaket in eine ZIP-Datei gepackt werden können (Größe der unkomprimierten ZIP-/JAR-Datei).	250 MB
Gesamtgröße der festgelegten Umgebungsvariablen	4 KB

[https://docs.aws.amazon.com/de\\_de/lambda/latest/dg/limits.html](https://docs.aws.amazon.com/de_de/lambda/latest/dg/limits.html)

# AWS Lambda - Kosten

---

## Kostenloses Kontingent

### 1 MIO. ANFORDERUNGEN

pro Monat

### 400 MB/SEK.

Datenverarbeitungszeit pro Monat.

Das kostenlose Kontingent für Lambda läuft nicht automatisch am Ende der 12-monatigen Laufzeit des kostenlosen Nutzungskontingents für AWS ab, sondern steht sowohl bestehenden als auch neuen AWS-Kunden unbearbeitet lange

## Anforderungen

### 1 MIO. ANFORDERUNGEN KOSTENLOS

Die ersten 1 Millionen pro Monat sind kostenfrei.

### DANACH 0,20 USD PRO 1 MIO. ANFORDERUNGEN

0,0000002 USD pro Anfrage.

## Dauer

### 400 000 GB/SEKUNDE PRO MONAT KOSTENLOS

Die ersten 400 000 GB/Sekunde pro Monat, also bis zu 3,2 Mio. Sekunden Verarbeitungszeit, sind kostenlos.

### DANACH 0,00001667 USD PRO GB/SEKUNDE

Der Preis ist abhängig von der Arbeitsspeichergröße, die Sie Ihrer Funktion zuweisen.

## Kalkulatoren:

<https://s3.amazonaws.com/lambda-tools/pricing-calculator.html>

<http://serverlesscalc.com> (beta)

# Beispiel-App: Pizzas from cloud

---

## User-Stories:

Als Kunde möchte ich aus den angebotenen Waren auswählen dürfen und diese über den Browser bestellen können.

Als Pizzabäcker möchte ich neue Bestellungen über den Webbrowser einsehen können, diese zur Bearbeitung markieren und als fertiggestellt kennzeichnen können.

# Beispiel-App: Pizzas from cloud

---

## API:

/offers - **GET**

- List all offers

/orders - **GET/POST**

- List all orders / Create new order

/orders/{orderID} - **PUT**

- Update specific order

# Pizzas from cloud

---

*Microservice mit Serverless-Komponenten*

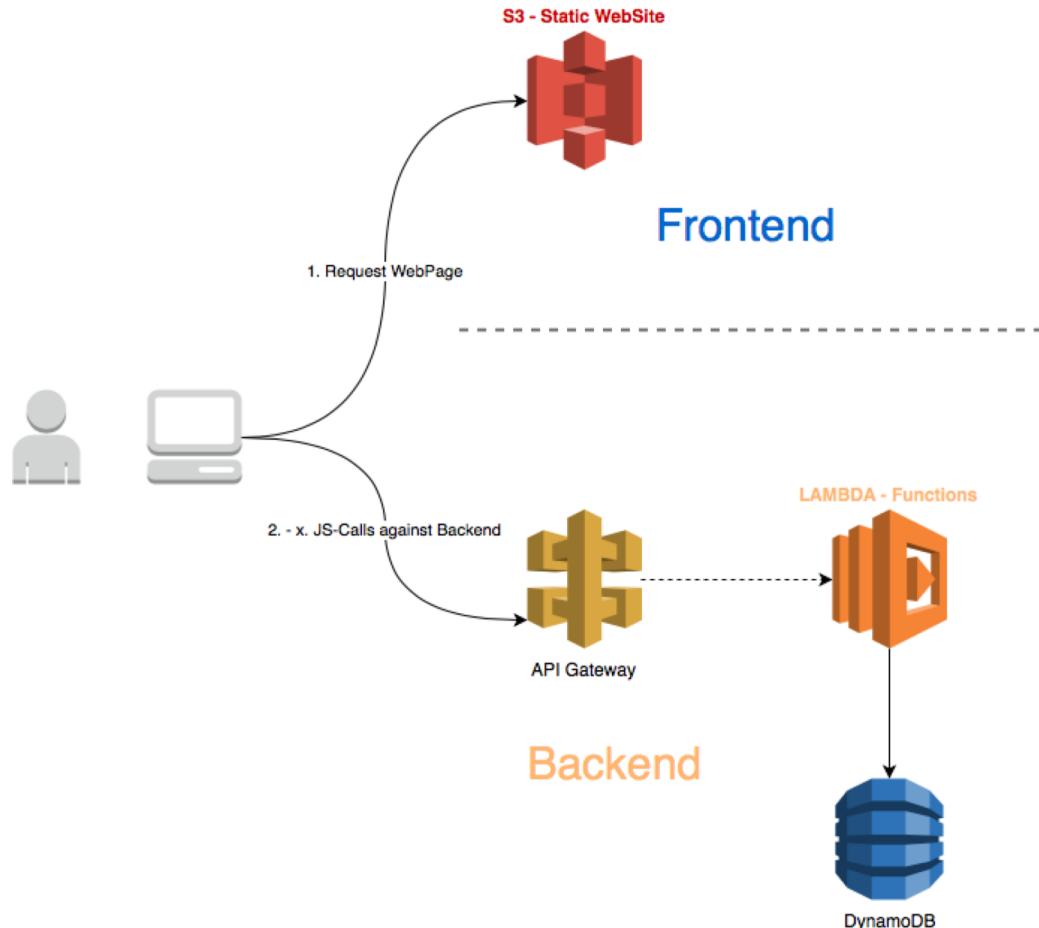
## **Benötigte Komponenten:**

- Frontend (HTML, JS)
- Backend

- Gateway           =>       API – Gateway
- Business-Logic   =>       Lambda
- Datenbank         =>       DynamoDB
- Logging           =>       Cloudwatch

# Pizzas from cloud

---



# Live part – AWS Lambda

---

- Vorstellung AWS Console
- Wie richte ich eine Lambda-Funktion über die Konsole ein
- Test einer Lambda-Funktion
- Cloudwatch - Logs

# Erstes Resümee

---

## Pro

- einfaches Einrichten mittels AWS Console
- transparent und sehr gut dokumentiert

# Erstes Resümee

---

## Pro

- einfaches Einrichten mittels AWS Console
- transparent und sehr gut dokumentiert

## Contra

- Änderungen nicht dokumentiert
- Aufsetzen neuer Stage aufwendig und fehleranfällig, da manuell
- Vendor Lock-In

# Erstes Resümee

---

## Pro

- einfaches Einrichten mittels AWS Console
- transparent und sehr gut dokumentiert

## Contra

- Änderungen nicht dokumentiert
  - Aufsetzen neuer Stage aufwendig und fehleranfällig, da manuell
  - Vendor Lock-In
- 
- Besser → Infrastructure-as-Code

# AWS Cloudformation

---

- Standardwerkzeug für AWS Ressourcen
- Beschreibung jeder Ressourcen mittels JSON / YAML – Templates
- Zusammengehörende Ressourcen bilden einen Stack
- Cloudformation erkennt Änderungen an Templates und erstellt Changeset

# AWS Cloudformation

---

- Standardwerkzeug für AWS Ressourcen
- Beschreibung jeder Ressourcen mittels JSON / YAML – Templates
- Zusammengehörende Ressourcen bilden einen Stack
- Cloudformation erkennt Änderungen an Templates und erstellt Changeset

## Nachteil:

- Templates werden recht schnell recht umfangreich

# SAM – Amazon Serverless Application Model

---

- kompaktere Ressourcendefinition und -konfiguration
- voll kompatibel zu Cloudformation Templates
- wird mit Cloudformation – Service ausgeführt

# SAM – Amazon Serverless Application Model

---

- kompaktere Ressourcendefinition und -konfiguration
- voll kompatibel zu Cloudformation Templates
- wird mit Cloudformation – Service ausgeführt

## Nachteile:

- bisher nur API Gateway, DynamoDB und AWS Lambda mittels SAM-Syntax verwaltbar  
(zusätzliche Ressourcen im selben Template mittels Cloudformation)
- Rollen müssen selber verwaltet werden

# Serverless - Framework

---

- noch einfachere Verwaltung von Cloud Ressourcen
- Open Source Framework
- nicht auf einen Cloud-Anbieter beschränkt
- Paketiert Programmcode und lädt diesen selbstständig mit hoch

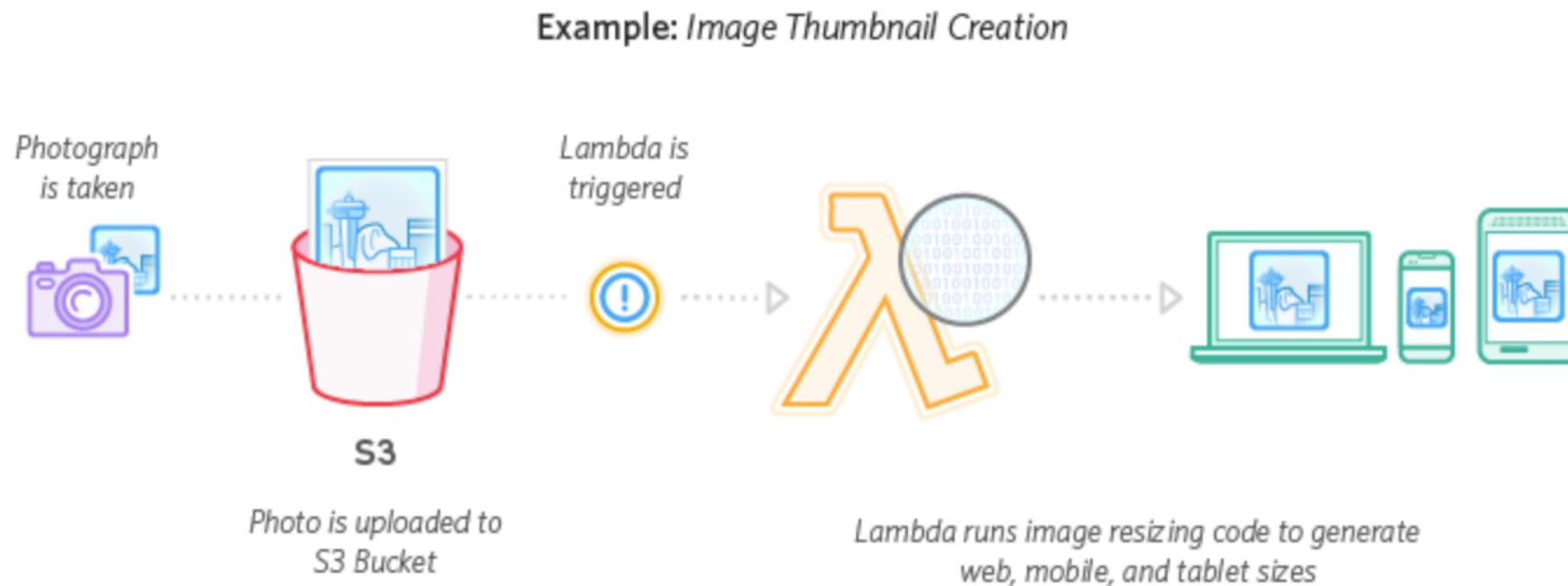
# Beispielanwendung mit *Serverless*

---

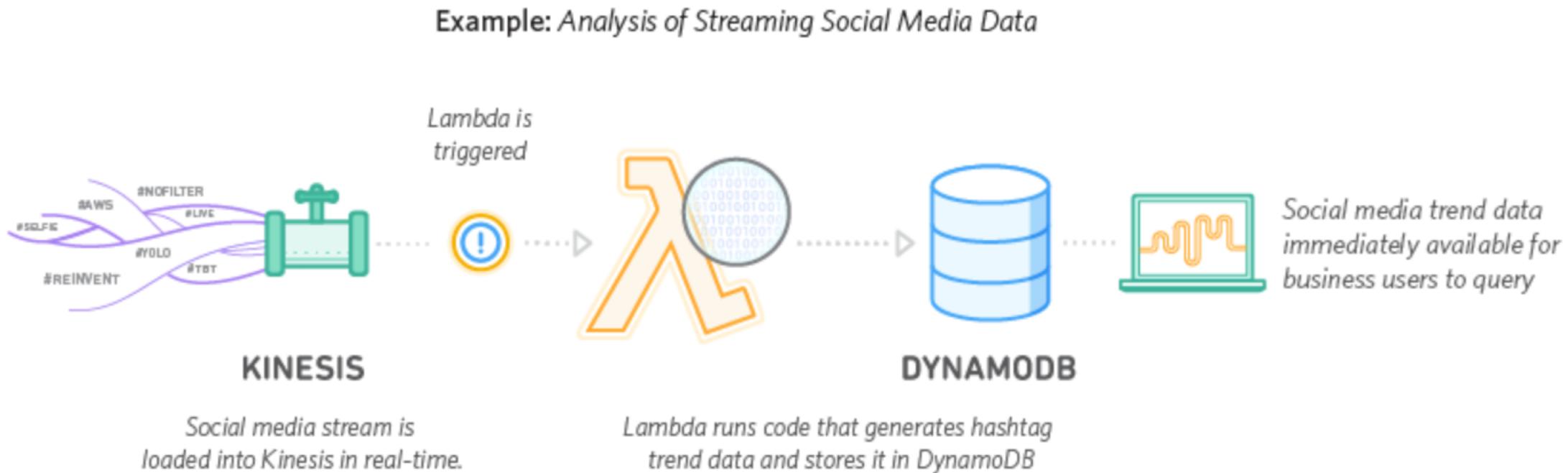
Zeigen

# Weitere Beispiele - Datenverarbeitung

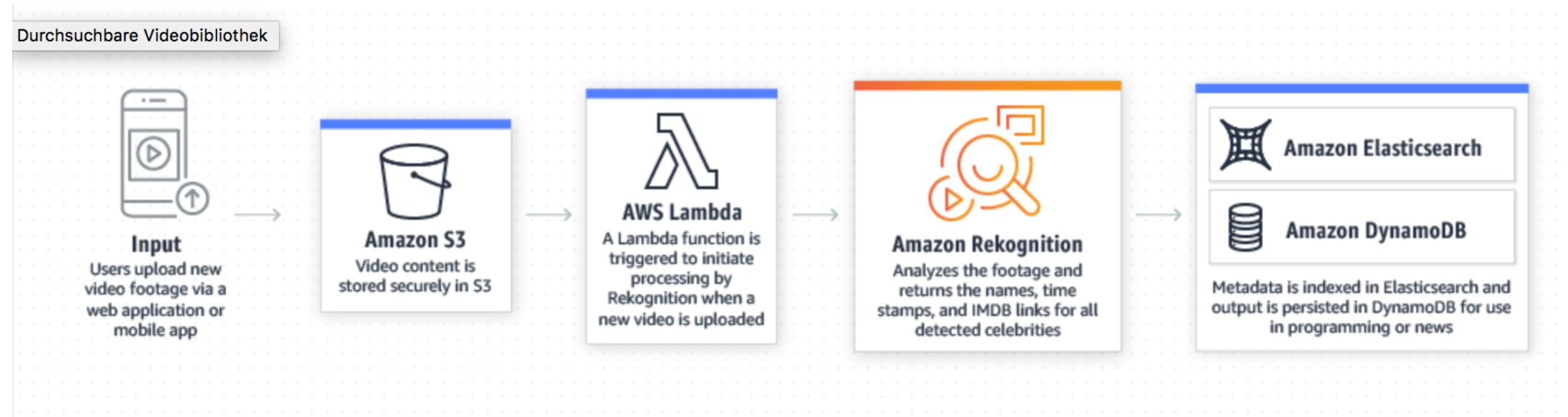
---



# Weitere Beispiele - Datenverarbeitung



# Weitere Beispiele – Videobibliothek durchsuchen



# *Best Practices*

---

- Anwendungslogik entkoppeln von Infrastruktur-Code
- Frühzeitig Gedanken wegen Limits machen und in Infrastrukturgedanken einfließen lassen
- Rechtzeitig Gedanken bzgl. Monitoring machen

## **Lauftzeit**

- Dependencies auf minimale beschränken
- Fail Fast
- warm containers
- interessanter Blogartikel: <https://blog.newrelic.com/technology/aws-lambda-cold-start-optimization/>



**Fragen ???**

# Quellen

---

- <https://serverless.com>
- <https://aws.amazon.com/de/serverless/>
- [https://aws.amazon.com/de/rekognition/?nc2=h\\_a1](https://aws.amazon.com/de/rekognition/?nc2=h_a1)
- Buch: Serverless Computing in der AWS Cloud (Niko Köbler) – ISBN: 978-3-86802-807-2



Vielen Dank