SERVERLOSE DATENVERARBEITUNG IN DER CLOUD

Einschränkungen und Implikationen für eine IoT-Anwendung am Beispiel von AWS Lambda

Simon Rapp
HS Pforzheim
Tiefenbronnerstr. 65,
75175 Pforzheim
rappsimo@hs-pforzheim.de

Frank Morelli
HS Pforzheim
Tiefenbronnerstr. 65,
75175 Pforzheim
frank.morelli@hs-pforzheim.de

KEYWORDS

Cloud Computing, Serverlose Datenverarbeitung, Amazon Web Services (AWS), AWS Lambda Serviceverhalten und Einschränkungen, Internet of Things (IoT).

ABSTRACT

Ein populäres Alleinstellungsmerkmal der Cloud ist das Angebot der serverlosen Datenverarbeitung. Bei diesem bezahlen die Kunden nur für jene Ressourcen, die sie tatsächlich nutzen. Zudem erfolgt die Ressourcenallokation durch den Cloud-Anbieter dynamisch. Aus der Sicht des Kunden ist ein serverloser Cloud-Service somit automatisch hoch verfügbar und das Risiko einer Unter- oder Überprovisionierung wird minimiert.

Die Kehrseite dieses Ausführungsmodells ist jedoch, dass die Kunden nahezu alle Verwaltungsaufwendungen hinsichtlich der physischen und virtuellen Infrastruktur dem Cloud-Anbieter überlassen. Dies hat zur Folge, dass Kunden auf das Verhalten der von ihnen genutzten Services nur eine eingeschränkte Kontrolle ausüben können, was sich in konkreten Nachteilen äußern kann.

Der vorliegende Beitrag befasst sich mit den Gründen für diese Einschränkungen und zu welchen erkennbaren Nachteilen diese führen können. Im Mittelpunkt ist dabei der serverlose Cloud Service AWS Lambda, von dem das Serviceverhalten im Rahmen eines IoT-Use Cases genauer untersucht wird.

EINLEITUNG

Serverlose Datenverarbeitung, auch kurz als „serverlos“ bezeichnet, ist ein für die Cloud einzigartiges Ausführungsmodell. Hierbei führt der Cloud-Anbieter den Anwendungscode seiner Kunden durch dynamische Zuweisung von Rechenressourcen aus. Im Rahmen dieses Ansatzes sind nach wie vor Server, wie beispielsweise physische Server, virtuelle Maschinen oder Container, für die Ausführung von Code erforderlich. Im Vergleich zu serverbasierter Datenverarbeitung wird dabei jedoch eine weitere Abstraktionsschicht, oberhalb von der Cloud-Infrastruktur, hinzugefügt. Anwendungsentwickler müssen somit die zugrunde liegende Infrastruktur, die für die Ausführung von Code erforderlich ist, nicht selbst konfigurieren oder im laufenden Betrieb verwalten. Diese Aktivitäten werden durch den Cloud-Anbieter gemanagt, der in Abhängigkeit vom Bedarf die erforderliche Infrastruktur automatisiert bereitstellt und skaliert (Bahga und Madisetti 2019).

Ein weiteres Merkmal der serverlosen Datenverarbeitung ist das Abrechnungsmodell. So wird jede Ausführung separat berechnet, wobei die Dauer und die Menge der verwendeten Ressourcen den Preis determinieren. Dieser Ansatz ermöglicht eine Optimierung der Kosten: Bei der serverbasierten Datenverarbeitung müssen die zugrunde liegenden Ressourcen fix bereitgestellt werden, was die Gefahr einer Überprovisionierung begünstigt und damit zu Leerkosten bzw. ungenutzten Serverkapazitäten führen kann (Artasanchez 2021).

Das serverlose Konzept ist deshalb besonders für jene Anwendungstypen attraktiv, die stark schwankende oder zum Teil auch unvorhersehbare Datenströme verarbeiten sollen, wie zum Beispiel eine IoT-Anwendung. Hier ist es möglich alle zentralen Vorteile der serverlosen Implementierung zu nutzen, d.h. hohe Robustheit und Flexibilität des Systems bei begrenztem Verwaltungsaufwand und zugleich geringeren Kosten (Laszewski et al. 2018).

Mit der zunehmenden Popularität und Adaption der serverlosen Datenverarbeitung in der Cloud (Dremel und Herterich 2018), ist das Konzept allerdings auch kritisch zu würdigen. Ein wesentlicher Grund hierfür besteht darin, dass die Kunden eines serverlosen Cloud-Services die Kontrolle über die zugrunde liegende Infrastruktur nahezu vollständig dem Cloud-Anbieter überlassen. Im Einzelnen ergeben sich folgende Fragen aufgrund der eingeschränkten Kontrolle:

- Welche technischen Einschränkungen stehen den Vorteilen gegenüber?
- Wie lassen sich diese erkennen?
- Welche Auswirkungen haben sie auf das Verhalten einer Applikation?

Der vorliegende Beitrag beschäftigt sich mit diesen Fragestellungen am Beispiel des serverlosen Cloud-

Services „AWS Lambda“ von Amazon Web Services (AWS).

Zuerst werden hierzu die grundlegenden Servicefunktionen sowie bereits bekannte Einschränkungen von AWS Lambda charakterisiert. Danach erfolgt die Betrachtung eines industriellen Use Cases, am Beispiel einer IoT-Anwendung. In diesem Kontext wurde im Vorfeld eine vertiefende Analyse des Serviceverhaltens von AWS Lambda durchgeführt, deren Ergebnisse dieser Beitrag vorstellt und diskutiert. Abschließend wird die Thematik zusammengefasst und mit einem Ausblick gewürdigt.

AWS LAMBDA

AWS Lambda ist im Angebot von AWS ein häufig verwendeter Service für serverlose Implementierungen. Bei Lambda erfolgt die Strukturierung des Codes im Rahmen von Funktionen, die durch ein Ereignis eines anderen AWS-Service ausgelöst werden. Zugehörige Ereignisse sind zum Beispiel ein HTTP-Request an ein API Gateway, ein neu eingetragener Datensatz in einer Datenbank, eine neue hochgeladene Datei in einem Cloud-Speicher, eine neue Nachricht in einem Messaging-Queue, ein Überwachungsalarm oder ein terminiertes Ereignis (Gupta 2018). Jedes Mal, wenn eine Funktion ausgelöst wird, initialisiert AWS automatisch die Ressourcen einer Ausführungsumgebung und führt die Funktion innerhalb dieser aus (Amazon Web Services Inc. o.D. a).

Jede Funktion ist dabei zustandslos und hat keine Beziehung zur zugrunde liegenden Infrastruktur. Auf diese Weise kann Lambda schnell so viele Kopien einer Funktion starten, wie für die Skalierung erforderlich ist, um die Anzahl der eingehenden Ereignisse abzuarbeiten (Amazon Web Services Inc. o.D. b).

Bei der Erstellung einer Lambda-Funktion können Kunden verschiedene Konfigurationseinstellungen festlegen, wie zum Beispiel die Größe des Arbeitsspeichers, die Laufzeit oder die maximale Ausführungszeit. Lambda verwendet diese Informationen, um die Ausführungsumgebung einzurichten (Amazon Web Services Inc. o.D. c).

Grundlegendes Serviceverhalten

Das grundlegende Serviceverhalten von AWS Lambda beschreibt der typische Lebenszyklus einer Lambda-Ausführungsumgebung. Er besteht aus drei Phasen, die Abbildung 1 skizziert (Amazon Web Services Inc. o.D. c):

In Phase 1, der Init-Phase, erstellt Lambda gemäß der Kundenkonfiguration eine Ausführungsumgebung, lädt den Code für die Funktion herunter, initialisiert die Laufzeit und führt dann den Initialisierungscode der Funktion (d.h. der Code außerhalb des Haupthandlers bzw. der eigentlichen Hauptfunktion) aus.

Die Init-Phase, auch „Kaltstart-Phase“ genannt, wird bei der Standardbereitstellung von Lambda durch das „erste“ Ereignis gestartet, das eine Funktion erhält. Wenn der Kunde jedoch gegen einen Aufpreis eine sogenannte „Provisioned Concurrency“ hinzubucht, erfolgt der Start dieser Phase direkt im Anschluss an die Erstellung bzw. nach dem Deplyoment der Funktion in AWS.

In der zweiten Phase „Invoke“ ruft Lambda auf Basis des erhaltenen Ereignisses die Hauptfunktion auf und führt diese aus. Nachdem die Ausführung vollständig beendet wurde, lässt sie sich durch ein weiteres Ereignis erneut starten.

Phase 3, die Shutdown-Phase, wird eingeleitet, wenn die Funktion für eine gewisse Zeit keine Aufrufe erhält. Lambda fährt dann die Laufzeit herunter und entfernt die Ausführungsumgebung. Die Verarbeitung eines neuen Ereignisses beginnt bei der Standardbereitstellung somit wieder mit Phase 1.

Bekannte Einschränkungen des Serviceverhaltens

Die bekannten Einschränkungen des Serviceverhaltens von Lambda werden durch die dynamische Bereitstellung des Services verursacht. Die Auswirkungen daraus zeigen sich in der Ausführungsdauer der Funktion, wobei zwei Parameter sich in diesem Zusammenhang als relevant erweisen (Beswick 2021):

Der erste Parameter beinhaltet die Ressourcenallokation, welche durch den Ressourcenbedarf der Kunden von Lambda bestimmt wird. AWS weist Kunden immer die Anzahl an Ressourcen zu, die sie zum aktuellen Zeitpunkt benötigen. Ändert sich der Bedarf, wird neu allokiert. Folglich weist AWS permanent neue Ressourcen zu bzw. entzieht diese bei fehlendem Bedarf.

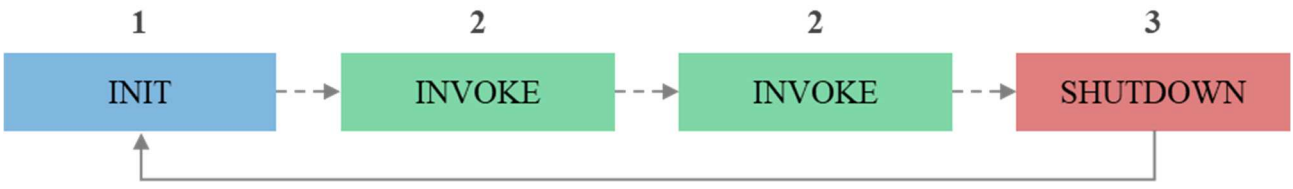

Abbildung 1: Lebenszyklus einer Lambda-Ausführungsumgebung (Quelle: Eigene Darstellung in Anlehnung an Amazon Web Services Inc. o.D. c)

Dies zeigt sich beispielsweise darin, dass die Ausführungsumgebung einer Lambda-Funktion, sobald initiiert, nicht dauerhaft aktiv bleibt. Zwar bleibt eine Funktion nach der Verarbeitung eines Ereignisses für kurze Zeit in einem sogenannten „warmen“ Zustand. In dieser Situation lässt sich beim Eintritt eines neuen zugehörigen Ereignisses Phase 1 überspringen und die Verarbeitung in der „Invoke“-Phase unmittelbar in der Ausführungsumgebung durchführen. Ansonsten erfolgt nach einer bestimmen Zeitdauer im Rahmen von Phase 3 durch AWS eine Löschung der Ausführungsumgebung, damit die zugrunde liegenden Ressourcen wieder frei für andere Kunden sind. Kundenseitig kann im Rahmen der Standardbereitstellung auf die Dauer dieses Zeitraums kein Einfluss genommen werden.

Erfolgen die Aufrufe der Funktion lediglich sporadisch ist davon auszugehen, dass die Verarbeitung bei einem hinzukommenden Ereignis erneut mit Phase 1 beginnt. Diese Kaltstart-Phase benötigt zusätzlich Zeit, die aufgrund der Initialisierungsaktivitäten oftmals signifikant länger dauert als die eigentliche Verarbeitung in Phase 2.

Dies gilt ebenso für parallele Funktionsaufrufe, da hierbei jedes Ereignis eine separate Ausführungsumgebung benötigt. Entsprechend erfahren sämtliche parallele Verarbeitungsvorgänge, für die keine „warme“ Umgebung bereitsteht, ebenfalls einen Kaltstart.

Der zweite Parameter im Rahmen der bekannten Einschränkungen des Serviceverhaltens von Lambda ist die Ressourcenverfügbarkeit, welche in der Verantwortung von AWS liegt.
AWS ist bestrebt diese auf möglichst hohem Niveau zu halten und permanent zu optimieren. Aktuell wird für Lambda z. B. ein Service Level Agreement (SLA) mit einer Verfügbarkeitsgarantie von 99,95% den Kunden zugesprochen (Amazon Web Services Inc. o.D. d).

Um dies zu gewährleisten, wird der Service und damit auch die Ausführungsumgebungen der Kunden über mehrere Rechenzentren hinweg verteilt und verwaltet. Dabei werden verschiedene Sicherheitsmechanismen aktiv. Sie sorgen beispielsweise in Abhängigkeit vom erzeugten Datenvolumen durch die Kunden und der daraus resultierenden Auslastung der einzelnen Rechenzentren dafür, dass ein Lastausgleich für eine Funktion vorgenommen werden kann. Das bedeutet, dass bei einer Überlastung oder bei einer ungleichen Lastverteilung zwischen den Rechenzentren Ressourcen umverteilt und neu initialisiert werden.

Deshalb ist es möglich, dass eine Funktion innerhalb eines kurzen Zeitraums zweimal aufgerufen wird und beide Ausführungen dennoch aufgrund dieser Lastausgleichsaktivitäten einen Kaltstart erfahren.

USE CASE IOT-ANWENDUNG

Der folgende Use Case handelt von einem Industrieunternehmen, das im Rahmen der AWS Cloud eine IoT-Anwendung entwickelt, die Maschinendaten aus der Produktion verknüpft und analysiert. Auf dieser Basis sollen zu ergreifende Maßnahmen definiert und automatische Workflows, zum Teil in Echtzeit, initiiert werden.

In der Umsetzung ist AWS Lambda ein wichtiger Bestandteil des Dateneingabe-Moduls der Anwendung. Dieses veranschaulicht Abbildung 2 anhand eines Architekturdiagramms. Hierbei werden die Maschinendaten, nach ihrer Übertragung in die Cloud, zunächst durch einen SQS-Queue zwischengepuffert.

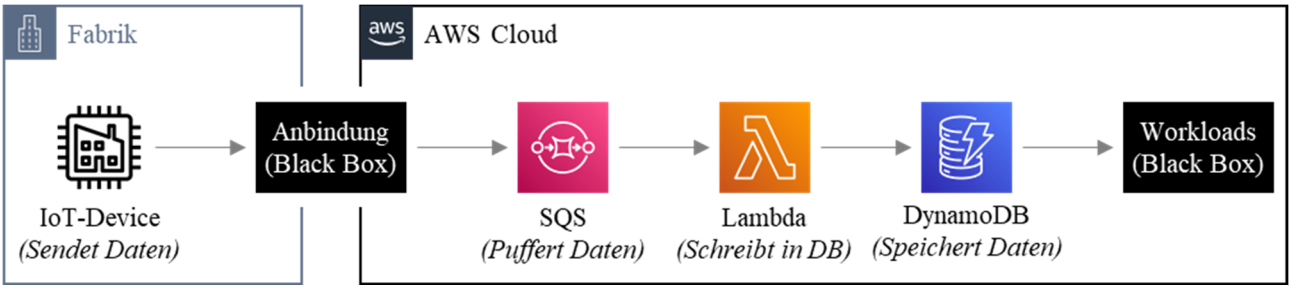

Abbildung 2: Use Case Architektur (Quelle: Eigene Darstellung)

Sobald neue Datensätze vorliegen, sendet der Queue ein Ereignis an eine Lambda-Funktion, welche daraufhin die Daten aus dem Queue entnimmt und in eine DynamoDB Datenbank schreibt. Dieser Datenhub fungiert als zentraler Ausgangspunkt für weitere Verarbeitungsschritte.

Besonderheiten und Methodik

Im vorliegenden Use Case gibt es kaum Abweichungen bei der Eingaberate der Maschinendaten. Weiterhin verarbeitet Lambda die SQS-Ereignisse in Batches, wobei dies im Sekundenrhythmus erfolgt oder wenn 10 Stück in weniger als einer Sekunde aufgelaufen sind. Bei kleineren Lastspitzen muss die Funktion folglich nicht sofort skalieren.

Entsprechend ist aufgrund der geschilderten Ausganssituation davon auszugehen, dass es bei diesem Setup nur dann zu einem verzögernden Kaltstart von Lambda kommen kann, wenn AWS einen Lastausgleich zwischen seinen Rechenzentren vornimmt.

Im Rahmen einer vertiefenden Analyse wird dieser angenommene Sachverhalt überprüft. Ein weiteres Ziel in diesem Kontext besteht darin, einen Überblick über die Häufigkeit solcher erkennbaren Lastausgleichsaktivitäten innerhalb eines bestimmten Zeitraums zu erhalten.

Die Vorgehensweise orientiert sich an der Auswertung von CloudWatch Logs, die jede Lambda-Funktion standardmäßig für alle Ausführungen im Betrieb generiert. Anhand der Logs lässt sich zum Beispiel ablesen wie lange die Ausführung gedauert hat oder ob es zu einem Kaltstart gekommen ist und wie viel Zeit dieser in Anspruch genommen hat.

Die Auswertung der Logs einschließlich Visualisierung und Filterung erfolgt im Rahmen der CloudWatch Konsole und der Abfragesprache von CloudWatch Logs Insights.

Ergebnisse der AWS CloudWatch Log Analyse

In einem ersten Schritt der Log-Analyse wurde ein Betrachtungszeitraum von einem vollen Tag gewählt, der um 00:00:00 Uhr beginnt und um 23:59:59 Uhr endet. Die IoT-Anwendung ist dabei durchgehend in Betrieb. Abbildung 3 veranschaulicht dazu den zeitlichen Verlauf der Funktionsaufrufe und der jeweiligen Ausführungsdauer für die Verarbeitung der Batches.

Ausführungsdauer anstatt wie im Durchschnitt wenige Millisekunden, mehr als 750 Millisekunden dauert.

Der Grund hierfür liegt nicht in der Anwendung, da

- der Umfang der zu verarbeitenden Datenpakete nur geringfügig variiert,
- die Anzahl an verwendeten Ausführungsumgebungen für die Batchverarbeitung nicht verändert wurde, denn sie lag konstant bei eins, und
- grundsätzlich keine Fehler aufgetreten sind, welche zu einer Neuinitialisierung der Ausführungsumgebung geführt hätten.

Wie die Auswertung der CloudWatch Logs zeigt, wurde aber jede der zu erkennenden Abweichungen durch die Neuinitialisierung der Lambda-Ausführungsumgebung, also einen Kaltstart, verursacht. Die in Tabelle 1 dargestellten Ergebnisse verdeutlichen dies.

Den Ergebnissen der Log-Analyse zufolge wurden alle Ausführungen, die länger als 143,27 Millisekunden dauerten, durch einen Kaltstart verzögert. Für alle 11 erheblichen Abweichungen von der durchschnittlichen Ausführungsdauer, die in Abbildung 3 zu erkennen sind, trifft das zu.

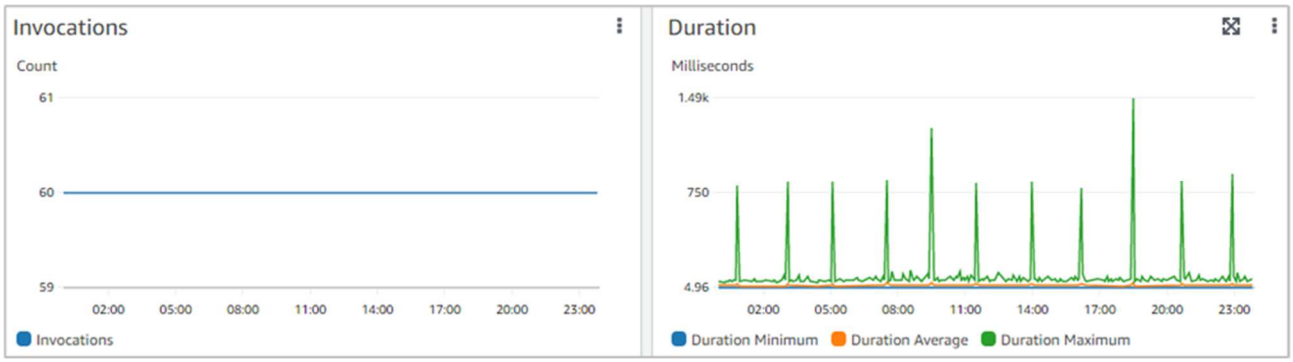

Abbildung 3: Zeitlicher Verlauf der Funktionsaufrufe und der jeweiligen Ausführungsdauer im Zeitraum 00:00:00 Uhr bis 23:59:59 Uhr am 23.03.2022 (Quelle: Eigene Darstellung)

Wie dargestellt, sind die Aufrufe (Invocations) der Lambda-Funktion über den gesamten Zeitraum konstant. Dennoch kommt es bei der Ausführungsdauer (Duration) aber wiederholt zu Abweichungen, wobei die Ursache hierfür liegt nicht bei dem Verhalten oder einem Fehler der Anwendung. Sie muss demnach bei einem internen Prozess des Lambda-Service bzw. bei AWS liegen.

Tabelle 1: Auswertung CloudWatch Logs der Lambda-Funktion im Zeitraum 00:00:00 Uhr bis 23:59:59 Uhr am 23.03.2022 (Quelle: Eigene Darstellung)

	Ausführungen mit Kaltstart-Phase	**Kaltstart-Phase**	**Ausführungen ohne Kaltstart-Phase**
Minimale Dauer (in MS)	787,92	449,27	4,96
Durchschnittliche Dauer (in MS)	932,51	590,35	21,77
Maximale Dauer (in MS)	1.494,95	1.143,81	143,27
Anzahl	11	11	17.269

Es erscheint aber unwahrscheinlich, dass es sich um Auswirkungen handelt, die aufgrund von intern durchgeführten Lastausgleichsaktivitäten herrühren. So sind anhand der Datenlage regelmäßige Zeitabstände zwischen den einzelnen Abweichungen erkennbar, die darauf schließen lassen, dass AWS den Austausch der Ausführungsumgebungen geplant bzw. terminiert ansteuert.

Eine mögliche Ursache hierfür könnte in der Absicherung der Service-Verfügbarkeit liegen, wenn davon ausgegangen wird, dass das Setup einer Ausführungsumgebung mit der Zeit an Stabilität verliert. Die genaue Ursache für die regelmäßigen Neuinitialisierungen lässt sich jedoch nicht aus der Auswertung ableiten.

Durch dieses Verhalten muss der Kunde Performanceschwankungen tolerieren, über die er keine Kontrolle hat. Es handelt sich somit um eine Einschränkung der Servicequalität.

Um die beobachtete Regelmäßigkeit zu untermauern, wurde der Betrachtungszeitraum anschließend auf drei Tage erweitert, von 00:00:00 Uhr am ersten Tag bis um 23:59:59 Uhr am dritten Tag. Abbildung 4. veranschaulicht hierzu wieder den zeitlichen Verlauf der Funktionsaufrufe und der Ausführungsdauer für die Verarbeitung der Batches.

Das gleiche Resultat ergibt die Analyse der Log-Files, wonach jede Ausführung, die länger als 784,68 Millisekunden, oder ohne diesen Ausreißer, länger als 148,84 Millisekunden dauerte durch einen Kaltstart verzögert wurde. Bei den meisten Abweichungen, die in Abbildung 4 zu erkennen sind, tritt der beschriebene Sachverhalt somit auf. Tabelle 2 zeigt hierzu die Ergebnisse der Auswertung.

In Summe kommt es im betrachteten Zeitraum zu 39 Kaltstartes, obwohl auch hier der Umfang der zu verarbeitenden Datenpakete nahezu immer gleich war, die Anzahl an verwendeten Ausführungsumgebungen konstant bei 1 lag und es auch zu keinem Ausfall gekommen ist.

Diskussion und Implikationen

Die Ergebnisse der Log-Analyse zeigen, dass die Durchgängigkeit einer Lambda-Ausführungsumgebung auch bei permanent aktiver Umgebung nicht gewährleistet ist. Im dauerhaften Betrieb wird sie immer wieder ausgetauscht und neu initialisiert, was einen Kaltstart zur Folge hat, der die Ausführung um bis zu 1,37 Sekunden verzögert.

Es fällt auf, dass der Austausch der Ausführungsumgebungen in einer gewissen Regelmäßigkeit stattfindet. Dieser Sachverhalt wurde im

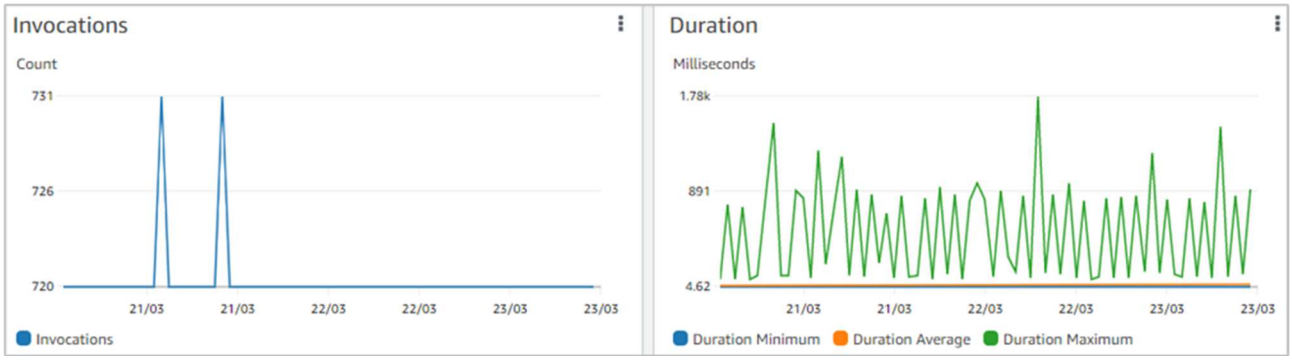

Abbildung 4: Zeitlicher Verlauf der Funktionsaufrufe und der jeweiligen Ausführungsdauer im Zeitraum 21.03.2022 um 00:00:00 Uhr bis 23.03.2022 um 23:59:59 Uhr (Quelle: Eigene Darstellung)

Wie dargestellt, setzt sich die zuvor beobachtete Regelmäßigkeit in den Abweichungen der Ausführungsdauer (Duration), trotz überwiegend konstanter Aufrufe, fort.

Rahmen der ersten Log-Analyse mit einem Betrachtungszeitraum von einem Tag beobachtet und durch eine erweiterte Analyse mit einem Beobachtungszeitraum von drei Tagen bestätigt. Aus

Tabelle 2: Auswertung CloudWatch Logs der Lambda-Funktion im Zeitraum 21.03.2022 um 00:00:00 Uhr bis 23.03.2022 um 23:59:59 Uhr (Quelle: Eigene Darstellung)

	Ausführungen mit Kaltstart-Phase	Kaltstart-Phase	Ausführungen ohne Kaltstart-Phase
Minimale Dauer (in MS)	685,27	381,36	4,61
Durchschnittliche Dauer (in MS)	930,82	579,03	22,1
Maximale Dauer (in MS)	1.780,11	1.371,38	(784,68) 148,84
Anzahl	39	39	51.823

diesem Grund erscheint es wenig plausibel, dass ein interner Lastausgleich zwischen den Rechenzentren von AWS dafür verantwortlich ist. Anzunehmen ist vielmehr, dass der Austauschprozess terminiert ist und nach einer gewissen Einsatzdauer einer Lambda-Ausführungsumgebung initiiert wird.

Im behandelten Use Case und dem dabei betrachtetet Ausführungszeitraum werden die Umgebungen in einem durchschnittlichen Rhythmus von 1,85 Stunden ausgetauscht. Der relative Anteil der Ausführungen, die dabei durch einen Kaltstart verzögert werden, ist zwar gering – bei 51.862 Ausführungen beträgt dieser 0.0752% –, allerdings ist zu beachten, dass in größeren Anwendungsszenarien oftmals mehrere Lambda-Funktionen voneinander abhängen bzw. miteinander verkettet sind, wobei jede Funktion vom dargestellten Sachverhalt betroffen sein kann.

Wenn der zeitliche Rhythmus der Neuinitialisierungen für jede Funktion versetzt erfolgt, steigt als Konsequenz mit jeder Funktion, welche ein Datensatz durchläuft, die Wahrscheinlichkeit, dass die Verarbeitung durch einen Kaltstart verzögert wird. Anhand der Use Case Daten veranschaulicht Tabelle 3 diese Auswirkungen im Detail. Als Grundlage der Berechnung dient die folgende Formel:

$$P = 1 - (1 - 0{,}000752)^n$$

Ferner gibt die in Abbildung 5 skizzierte Architektur, welche sich am Setup des Use Case orientiert, ein praktisches Beispiel für einen Workload, bei dem mehrere Funktionen miteinander verkettet sin: Die erste Lambda-Funktion, welche die IoT-Datensätze aus dem SQS-Queue entnimmt, ruft über eine zweite Funktion weitere Informationen aus einer weiteren Datenbank ab, um die IoT-Daten mit statischen Attributen anzureichern. Anschließend wird durch die DynamoDB eine dritte Funktion getriggert, die anhand der aktualisierten Datensätze eine Reihe von Kennzahlen in einem Dashboard aktualisiert oder einen weiteren Prozess anstößt.

Alle drei Funktionen weisen demzufolge Abhängigkeiten voneinander auf, wobei die Ausführungsrate von jeder Funktion jeweils durch die Eingaberate der IoT-Daten bestimmt wird. Sie sind somit dauerhaft in Verwendung. Die Wahrscheinlichkeit, dass bei einer dieser drei Funktionen der Verarbeitungsprozess der IoT-Daten verzögert wird, liegt demnach bei 0,2254%.

Tabelle 3: Wahrscheinlichkeit für Kaltstart in Abhängigkeit der Anzahl an verketteten Funktionen (Quelle: Eigene Darstellung)

Verkettete Funktionen (n)	Wahrscheinlichkeit (P) für prozessverzögernden Kaltstart
1	0,0752%
2	0,1503%
3	0,2254%
4	0,3005%
5	0,3754%
6	0,4504%

Bei einem Volumen von 1 Millionen Batchverarbeitungsvorgänge, die von jeder Funktion zu leisten sind, wären folglich 2.254 Verarbeitungsvorgänge von einem verzögernden Kaltstart betroffen.

Die einzige Möglichkeit, um den Lambda-Service zu verwenden und dieses Problem zu umgehen, besteht im Hinzubuchen einer Provisioned Concurrency für jede Lambda-Konfiguration. Es ist allerdings zu beachten, dass in der Folge die Nutzungskosten von AWS Lambda erheblich steigen können.

Dies kann zum Beispiel anhand einer Rechnung mit dem Online frei verfügbaren AWS Pricing Calculator demonstriert werden (Amazon Web Services Inc. o.D. d). Dabei ist eine mögliche Konfiguration für Lambda eine

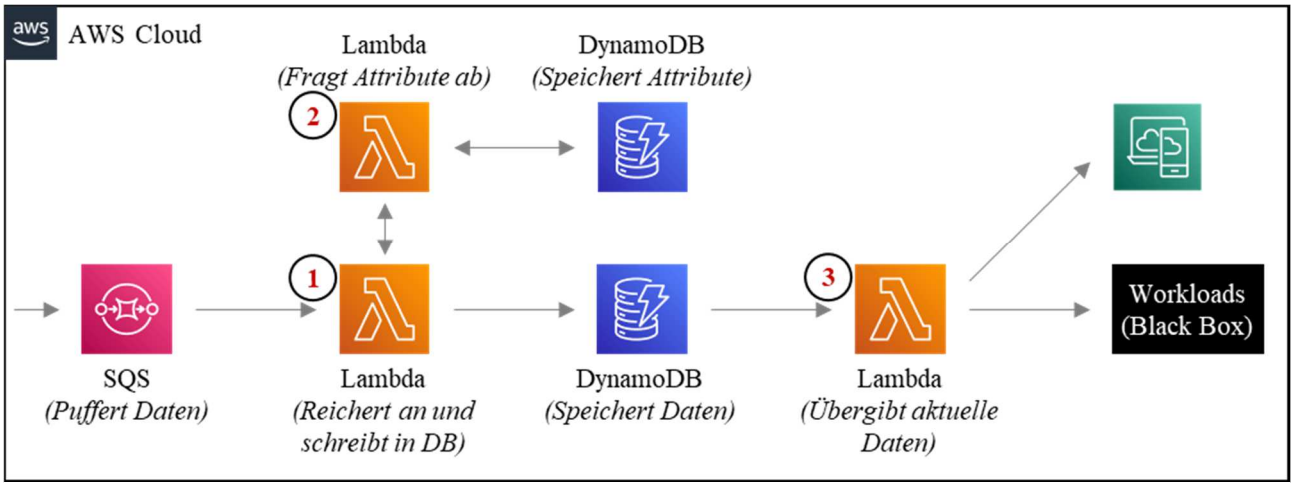

Abbildung 5: Erweiterte Architektur von Use Case (Quelle: Eigene Darstellung)

x86 Architektur mit einem Arbeitsspeicher von 1024 MB. Die bereitstellende Region, in welcher die Funktion gehostet wird, ist für dieses Beispiel Irland. Die Provisioned Concurrency wird auf 1 gesetzt und soll für einen Tag aktiviert sein – das bedeutet, dass eine Ausführungsumgebung nach Erstellung der Funktion für 24 Stunde, sofern sie nicht gerade ein Ereignis verarbeitet, permanent in einem warmen Zustand gehalten wird.

Die Kosten dafür betragen 0,40 USD. Hinzu kommen die Kosten für die eigentliche Datenverarbeitung in dieser Zeit. Bei einer Anzahl von 1 Millionen Anfragen an die warme Umgebung, belaufen sich diese auf 0,25 USD. Dabei wird impliziert, dass ein Verarbeitungsvorgang im Durchschnitt 5 Millisekunden dauert. Somit beträgt der Preis für eine Funktion mit einer Provisioned Concurrency von 1 pro Tag in Summe 0,65 USD.

Wenn keine Provisioned Concurrency für die verwendete Umgebung aktiviert ist, liegen die Kosten für die Datenverarbeitung etwas höher bei 0,28 USD. In diesem Fall handelt es sich aber um den einzigen Kostenfaktor. Die Differenz zu einer Lambda-Konfiguration mit Provisioned Concurrency von 1 beträgt demnach pro Tag 0,37 USD.

Auf ein Jahr gerechnet, summiert sich der Kostenunterschied in der Bereitstellung für diese eine Funktion und Ausführungsumgebung auf 135,05 USD. Die Kosten für die Funktion ohne Provisioned Concurrency würden dabei 102,2 USD betragen, während für die Funktion mit einer Provisioned Concurrency von 1 insgesamt 237,25 USD anfallen.

Für jeden vergleichbaren Use Case stellt sich daher die Frage, ob sich zur Lösung der herausgearbeiteten Kaltstart-Problematik eine Provisioned Concurrency als sinnvoll und ob die hierfür zusätzlich anfallenden Kosten tragbar sind. Dies hängt von den Anforderungen an den jeweiligen Use Case bzw. den individuellen Rahmenbedingungen ab.

Bei der Entscheidungsfindung ist davon auszugehen, dass sich die drei nachfolgenden Fragen als relevant erweisen:

1. Gefährden regelmäßige Kaltstarts einer Lambda-Funktion die Effizienz der Anwendung?
2. Erweist sich eine serverlose Umsetzung mit AWS Lambda als gerechtfertigt, auch wenn das kostenpflichtige Hinzubuchen einer Provisioned Concurrency erforderlich ist, um die Performance-Anforderungen an die Applikation erfüllen zu können?
3. Ist eine alternative Bereitstellung, trotz höherer Verwaltungsaufwände, im Vergleich zu AWS Lambda mit aktivierter Provisioned Concurrency kostengünstiger? Im Falle einer Kostengleichheit gilt es zu bedenken, dass kundenseitig mehr Einflussnahme über die zugrunde liegende Infrastruktur ausgeübt werden kann.

FAZIT UND AUSBLICK

Serverlose Cloud-Services sind dafür bekannt, dass sie hoch verfügbar sind und ein attraktives Kostenmodell bieten. Dieses Verhalten gilt es jedoch kritisch zu würdigen.

Am Beispiel des serverlosen Cloud-Services AWS Lambda zeigt der vorliegende Beitrag auf, dass Kunden aufgrund der eingeschränkten Kontrolle über die zugrunde liegende Infrastruktur auch technische Einschränkungen hinnehmen müssen. Diese äußern sich im Performanceverhalten des Service.

So werden die zugrunde liegenden Ressourcen einer Lambda-Funktion durch AWS dynamisch optimiert. Den Kunden werden somit immer nur so viele Kapazitäten bereitgestellt, wie aktuell benötigt werden. Wenn die Anwendungslast steigt, ist zwar durch die SLAs abgesichert, dass AWS den Bedarf deckt, jedoch kann es zu Kaltstartzeiten kommen. Dies liegt daran, dass neu zugeteilte Ressourcen vor der Nutzung immer zuerst hinsichtlich der Kundenkonfiguration initialisiert werden müssen.

Des Weiteren zeigt eine vertiefenden Use Case-Analyse, dass AWS dauerhaft aktive Laufzeitumgebungen einer Lambda-Funktion regelmäßig austauscht. Dies wirkt wie ein interne Lastausgleich und führt ebenfalls zu Kaltstarts und damit einhergehenden Verzögerungen.

Die von AWS Lambda angebotene Konfigurationsmöglichkeit „Provisioned Concurrency“, mit der es möglich ist, die Kaltstart-Phase für eine definierte Anzahl an Ausführungsumgebungen zu überspringen, wird ebenfalls betrachtet. Es zeigt sich allerdings, dass im Vergleich zu den Preisen, welche typischerweise für die eigentliche Datenverarbeitung anfallen, mit erheblichen Mehrkosten zu rechnen ist. Dies gilt insbesondere dann, wenn diese Lösung dauerhaft adaptiert wird. Es ist daher in Abhängigkeit vom Anwendungsfall und dem dahinterstehenden Business Case zu entscheiden, ob die Einrichtung einer Provisioned Concurrency sinnvoll und wirtschaftlich tragbar ist.

Entsprechend kann es sich als sinnvoll erweisen, Alternativen zu AWS Lambda hinsichtlich Kosten und Leistung zu evaluieren. Wenn Kaltstartzeiten zwingend zu vermeiden sind, kann eine serverbasierte Lösung im Vergleich zum serverlosen Lambda-Service mit Provisioned Concurrency kostengünstiger sein und gleichzeitig mehr Kontrolle über die zugrunde liegende Infrastruktur bieten.

Weitere Studien zu diesem Thema können hier ansetzen. Unternehmen dürften in diesem Kontext vor allem an der

Frage interessiert sein, ab welchem Datenvolumen eine serverbasierte Lösung gegenüber AWS Lambda, ein besseres Kosten und Leistungsverhältnis bietet.

Für eine ganzheitlichen Überblick erweist es sich ebenfalls von Interesse, wie sich die Kaltstartthematik bei anderen führenden Cloud-Anbieter, z.B. Microsoft Azure oder Google Cloud, verhält und welche Optionen diese anbieten.

LITERATUR

Amazon Web Services Inc. o.D. a. *"AWS Lambda Developer Guide. Stichwort: What is AWS Lambda?"* https://docs.aws.amazon.com/lambda/latest/dg/welcome.html (besucht am 17.09.2022).

Amazon Web Services Inc. o.D. b. *"AWS Lambda Developer Guide. Stichwort: AWS Lambda foundations."* https://docs.aws.amazon.com/lambda/latest/dg/lambda-foundation.html (besucht am 17.09.2022).

Amazon Web Services Inc. o.D. c. *"AWS Lambda Developer Guide. Stichwort: AWS Lambda execution environment."* https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html (besucht am 17.09.2022).

Amazon Web Services Inc. o.D. d. *"AWS Service Level Agreement (SLA)."* https://aws.amazon.com/de/legal/service-level-agreements/ (besucht am 17.09.2022).

Amazon Web Services Inc. o.D. e. *"AWS Pricing Calculator. Die Kosten für Ihre Architektur-Lösung schätzen."* https://calculator.aws/#/ (beucht am 17.09.2022).

Artasanchez, A. 2021. *"AWS for Solutions Architects. Design your cloud infrastructure by implementing DevOps, containers, and Amazon Web Services."* Packt Publishing, Birmingham.

Bahga, A und V. Madisetti. 2019. *"Cloud Computing Solutions Architect – A Hands-On Approach. A Competency-based Textbook for Universities and a Guide for AWS Cloud Certification and Beyond."*

Beswick, J. 2021. *"Operating Lambda. Performance Optimization – Part 1."* https://aws.amazon.com/de/blogs/compute/operating-lambda-performance-optimization-part-1/ (besucht am 17.09.2022).

Dremel, C. und M. Herterich. 2018. *„Digitale Cloud-Plattformen als Enabler zur analytischen Nutzung von operativen Produktdaten im Maschinen- und Anlagenbau." In Cloud Computing – Die Infrastruktur der Digitalisierung 2018, Reinheimer, S. (Hrsg.).* Wiesbaden, 73–88.

Gupta, M. 2018. *„Serverless Architectures with AWS – Discover how you can migrate from traditional deployments to serverless architectures with AWS."* Packt Publishing, Birmingham.

Laszewski, T.; K. Arora; E. Farr und P. Zonooz. 2018. *„Cloud native architectures. Design high-availability and cost-effective applications for the cloud."* 1. Aufl. Packt Publishing, Birmingham.