The silence of the Lambda

Portabilität von Serverless / FaaS Implementierungen zwischen Cloud Providern unter Berücksichtigung regulatorischer Anforderungen

NEXGEN

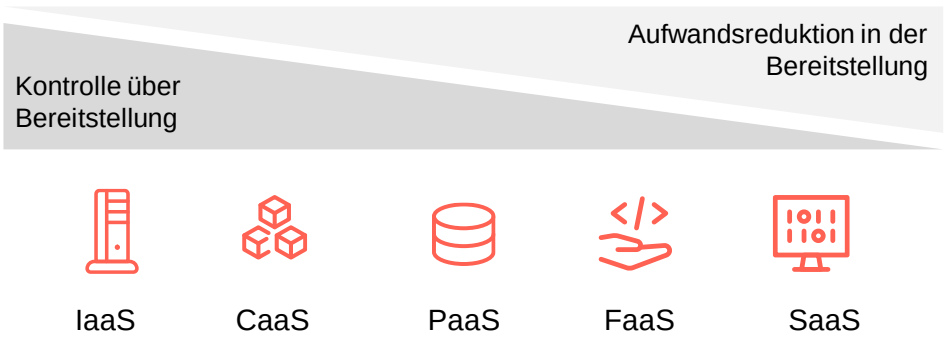
NEXGEN Whitepaper
April 2022

Tristan Poetzsch
André Rentschler

Einführung

Viel wurde sich in den letzten Jahren von den Möglichkeiten der Cloud versprochen – schnell und günstig sollte sie sein und dabei helfen, die technischen Altlasten in den Kernprozessen der Financial Services Industrie zu reduzieren. **Cloud Service Provider (CSP)** wie Google, Microsoft und Amazon und deren Dienste werden daher zunehmend relevanter für die Finanz- und Bankenwelt. Doch mit all den neuen Möglichkeiten kommen auch viele völlig neue Fragestellungen auf die Finanzwelt zu – allen voran die größte Frage im Outsourcing, nämlich die nach dem **Business Continuity Management (BCM).** Was machen wir, wenn einer der großen Anbieter entscheidet, seine aktuell den DSGVO-Anforderungen entsprechenden Datencenter doch wieder nach Amerika abzuziehen und wir die Dienste daher nicht mehr nutzen wollen und können? Was machen wir, wenn einer der großen Player entscheidet, seine Dienste einzustellen? Und weil der Regulator mit der MaRisk / BAIT auch seine Anforderungen mit einbringt – wie reagieren wir bei einer Cloud Transformation auf solche Anforderungen?

Die Antwort auf diese Frage ist bei vielen Finanzdienstleistern eine halbwegs pauschale Richtlinie: Möglichst viel der klassischen IT soll in portablen Containern gebaut werden, die einen relativ einfachen Umzug von einem Anbieter zu einem anderen ermöglichen. Dass es hier in der Praxis durchaus auch ganz eigene Herausforderungen gibt wie Versionskompatibilität oder unterstütze Container-Runtimes gibt, gerät dabei schnell aus dem Fokus.



Cloud-Betriebsmodelle im Vergleich

Paradox wird die Beziehung zu SaaS-Lösungen. Obwohl man kaum einen höheren Lock-In kaufen kann als SaaS, so wird doch neben der Anforderung an möglichst maximale Kompatibilität auch häufig die Losung ausgegeben, möglichst viel fertig einzukaufen. Hier merkt man, dass die hochtrabenden Ideen für die Cloud doch noch in die Brandung der Praxis krachen.

Zwischen diesen Möglichkeiten bieten Cloud-Anbieter aber noch andere Betriebsmodelle an, für welche die Bewertung aktuell noch schwerfällt: Die sogenannten **Serverless** und **Function-as-a-Service (FaaS)** Komponenten. Aber wie ist denn hier die regulatorische Einwertung und die Portabilität? Genau diese Frage beantworten wir in diesem Whitepaper, auch anhand ganz konkreter Use Case Implementierungen.

Was sind Serverless & Function-as-a-Service?

Serverless bedeutet nicht wirklich, dass kein Server mehr im Hintergrund läuft. Aber im Gegensatz zu anderen Betriebsmodellen hat man in Serverless Modellen tatsächlich überhaupt keinen Kontakt mehr zur Infrastruktur (wie es ja zum Beispiel noch bei Containern ist). Man hat wirklich nur noch seinen eigenen Code, der ausgeliefert wird und der Cloud Anbieter selbst kümmert sich um alles dahinter inklusive den Skalierungsfragen.

Eines der Hauptmodelle für Serverless ist Function-as-a-Service (Faas). Hier werden selbst gecodete Funktionen in einer Umgebung des Cloud-Anbieters bereitgestellt und können durch definierte Events oder andere Funktionen aufgerufen und ausgeführt werden. Die Ergebnisse und Informationen der Ausführung von Funktionen sind nicht persistent, weshalb jede Durchführung wie zum ersten Mal von Grund auf neu passiert, was als zustandlose (stateless) Anwendung definiert wird. Wird eine Funktion zum ersten Mal seit längerer Zeit aufgerufen (ein sogenannter Kaltstart), ist die Laufzeit auch noch einmal unterschiedlich zu allen folgenden Aufrufen.

Die Function-as-a-Service Umgebungen bei den großen Cloud-Providern heißen AWS Lambda, Azure Function und Google Cloud Functions.

Was sagen die relevanten Regularien im Bereich Cloud-Outsourcing?

Geregelt werden Themen wie Informationssicherheit, Business Continuity und auch die Nutzung von Auslagerungen von IT an Dritte wie Microsoft, Google und Amazon vorrangig durch die Mindestanforderungen an das Risikomanagement der Banken (MaRisk AT 4.2, AT 7.2 sowie AT 9), das Kreditwesengesetz (KWG §25b) sowie die Spezifikation beider mit der bankaufsichtlichen Anforderungen an die IT (BAIT, hier besonders Kapitel 8).

Die Pflichten zum Umgang mit externer Software und Services wie Cloud-Diensten werden hier wie folgt konkretisiert:

1. Für Fremdbezüge ist eine angemessene Risikobewertung vorzunehmen
    - Es kann auf bereits vorhandene Bewertungen zurückgegriffen werden, sofern gleichartige IT-Dienstleistung vorliegt
    - Die für Informationssicherheit und Notfallmanagement verantwortlichen Funktionen des Instituts werden eingebunden
2. Bei Fremdbezug von IT-Dienstleistungen ist die Kompatibilität mit der IT-Strategie des Unternehmens zu berücksichtigen
3. Abgeleiteten Maßnahmen sind angemessen in der Vertragsgestaltung mit dem Provider zu berücksichtigen, wie zum Beispiel:
    - Vereinbarungen zum Informationsrisikomanagement, zum Informationssicherheitsmanagement oder zum Notfallmanagement
    - Ebenfalls berücksichtigt und dokumentiert sind diesbezügliche Exit- bzw. Alternativ-Strategie
4. Risikobewertungen sind regelmäßig und anlassbezogen zu überprüfen

BAIT vs. ESMA Leitlinien zur Auslagerung an CSP

Die ESMA Leitlinien zur Auslagerung an Cloud Service Provider sind europaweiter Natur. Sie sollen eine Standardisierung und Harmonisierung der Aufsichtspraktiken und des Risikomanagements für die Auslagerung an CSPs erwirken. Die Inhalte der BAIT und der ESMA Leitlinie decken sich weitestgehend.

Wie unterscheiden sich die FaaS-Angebot von den verschiedenen Cloud Service Providern?

Mit Rückblick auf die Spezifikationen der BAIT bezüglich Exit- und Alternativ-Strategien stellt sich die Frage, wie portabel Codes und Konfigurationen zwischen den einzelnen CSPs denn sind. Diese Frage beleuchten wir im Folgenden anhand der Aspekte *unterstützte Sprachen, Funktionalitäten, Leistung, Skalierung* sowie abseits der Regularien bezüglich *Preisgestaltung*.

Unterstützte Sprachen

Die erste Frage der Portabilität stellt sich schon bei den verwendbaren Programmiersprachen. Grundsätzlich unterstützen die FaaS Angebote der CSPs einen Großteil der Standard-Sprachen in den aktuellen Versionen. Auffällig ist aber zum Beispiel die fehlende Verfügbarkeit von Ruby und Go in Azure Functions, dafür wird hier als einziges TypeScript unterstützt.

Bei allen CSPs ist es möglich, benutzerdefinierte Sprachen hinzuzufügen, wobei im Fall von Amazon und Microsoft integrierte Services und bei Google benutzerdefinierte Docker Images genutzt werden.

Unterstützte Sprachen	Microsoft Azure Functions	Google Cloud Functions	Amazon Web Services Lambda
Node.js	v.14	v.16	v.14
Python	v.3.9	v.3.9	v.3.9
Ruby	-	v.2.7	v.2.7
Go	-	v.1.16	v.1.x
Java	v.11	v.11	v.11
Zusätzliche Sprachen	C#, F#, PowerShell, TypeScript	C#, F#, Visual Basic	C#, PowerShell
Benutzer-definierte Sprachen	Ja, mit Functions Custom Handler	Ja, mit Custom Docker Images	Ja, mit AWS Lambda Layers

Abzug vom 21.04.2021 – subject to ad-hoc-change

Funktionalität

FaaS ist ein eventgetriebener Cloud Service. Bei diesen Events kann es sich um geplante Durchführungen der Interaktion mit Cloud Services (z. B. der Upload einer Datei in einen Cloud-Speicher) oder Anfragen durch einen webbasierten (HTTP) Endpunkt handeln. HTTP-Events werden von allen drei CSPs zwar angeboten, sind aber nur bei Google und Azure nativ integriert. Bei AWS Lambda ist zusätzlich ein API Gateway zu konfigurieren und zu bezahlen. Azure Functions bietet im Gegensatz zu den anderen Anbietern direkt im Modul die Möglichkeit, einen getimten Trigger zu setzen.

Leistung

Ein großes Thema für Functions ist die maximale Laufzeit. Insbesondere bei größeren und geschachtelten Funktionen ist das ein wichtiger Faktor. Microsoft bietet mit dem Konsumplan 10 Minuten und bei höheren Abonnements sogar unbegrenzte Laufzeiten an. Im Gegensatz dazu bietet Google eine maximale Laufzeit von 9 Minuten (1st Generation) und bis zu 60 Minuten (2nd Generation bei HTTP Funktionen). Bei Amazon liegt die maximale Laufzeit immer bei 15 Minuten.

Bei der Konfiguration des Arbeitsspeichers ist überall ein Minimum 128 MB angesetzt. Bei Microsoft kann dieser Arbeitsspeicher mit dem Konsumplan auf 1.500 MB und mit höheren Abonnements auf 14 GB aufgestockt werden. Bei Google liegt der maximale Arbeitsspeicher der Funktionen bei 8 GB (1st Gen) und bis zu 16 GB (2nd Gen). Bei Amazon endet die Konfiguration des Arbeitsspeichers bei 10 GB.

Skalierung

Funktionen werden von den CSPs grundsätzlich skalierbar angeboten, jedoch gibt es Unterschiede in der maximalen Anzahl an gleichzeitigen Instanzen von Funktionen. Bei Microsoft gibt es die Möglichkeit, 100 – 200 Instanzen parallel zu betreiben. Google bietet ein maximales Instanzlimit von 3000 und Amazon von 1000.

Code-Level

Wichtig für die Interoperabilität und die Portabilität ist das Verständnis für die Unterschiede auf Code-Level. Bei allen CSPs und deren Functions gibt es Eigenheiten und kleine Unterschiede unter anderem was die Signatur der Functions sowie die Konfiguration der Events betrifft. Die Unterschiede in der Signatur können in an Beispiel einer Node.js Function eines HTTP-Requests betrachtet werden:

Google Cloud Functions

```
exports.myHandler = (req, res) => {
    res.status(200).send({
       message: `Hello ${req.body.myName}`
    } ); };
```

Azure Functions

```
module.exports = (context, req) => {
    const input = JSON.parse(req.body);
    context.res = {
        status: 200,
        body: {
            message: `Hello ${input.myName}`
        } };
    context.done(); };
```

AWS Lambda

```
exports.myHandler = async (event, context) => {
    const input = JSON.parse(event.body);
    return {
        statusCode: 200,
        body: {
            message: `Hello ${input.myName}`
        } }; }
```

Dabei ist anzumerken, dass die Unterschiede neben dem Cloud Provider ebenfalls von der Programmiersprache sowie der Art der Events abhängig sind. Bei der Integration von Events unterscheiden sich die Anbieter wie diese den Functions zugewiesen werden, daher, ob Events beispielweise in Themen verpackt sind oder direkt der Function zugeordnet werden. Weiterhin bietet jeder Anbieter für die Eingabe der Befehle hauseigene Kommandozeilen (CLI) und Befehle zur Publizierung und Verpackung der Functions an.

Preisgestaltung

Die Kosten für den Betrieb von Funktionen ergeben sich aus *Anfragen pro Monat* und der *Kosten für die Rechenzeit*, welche in Gigabyte-Sekunden (GB) gemessen wird. Alle drei CSPs bieten den Kunden ein kostenloses, monatliches Kontingent an Rechenzeit an. Ergänzt wird dies durch kostenlose Anfragen pro Monat.

	Microsoft Azure Functions	Google Cloud Functions	Amazon Web Services Lambda
Kostenlose monatliche GB-Sekunden	400.000	400.000	400.000
kostenlos Monatliche Anfragen	1 Million	2 Million	1 Million
Kosten für jede weitere 1 Million Anfragen	0,20$	0,40$	0,20$
Kosten für jede weitere GB-Sekunden*	0,000016$	0,0000025$	0,000016$

** Abhängig von der Menge des verwendeten Arbeitsspeichers*
Abzug vom 21.04.2021 – subject to ad-hoc-change

Wie baue ich typische Financial Services Use Cases als portable Functions konkret bei den CSPs auf?

IT-Infrastruktur im Banking bietet an vielen Stellen das Potenzial, erneuert oder ausgebaut zu werden. Oftmals sind kleine Programme oder Programmteile in riesigen HOST-Modulen eingebettet, welche nicht selten sechs-stellige Codezeilenanzahl knacken. An vielen Stellen handelt es sich hier nur um Konvertierungs- oder Weiterleitungsaufgaben, welche problemlos von einer Function ersetzt werden könnte. Aber auch ganze Abschnitte der IT-Landschaft im Banking könnten von der Verwendung von Functions profitieren. Nachfolgend wurden hierzu Use Cases sowohl mit als auch ohne Datenbankzugriffe betrachtet und über die drei CSPs AWS, Azure und GCP hinweg Referenzarchitekturen verglichen.

Besonderes Augenmerk wurde hier auf die Komplexität der Migration in die Cloud verwendeten Module, Besonderheiten der CSPs und eine Bewertung der Portabilität zwischen den CSPs gelegt. In den folgenden Beispielen wurden bestehende Prozesse und ganze Verarbeitungsstrecken unter die Lupe genommen und anhand einer auf FaaS basierenden Architektur neu aufgestellt. Außerdem wurden explizit die Architekturen in den CSPs GCP, Azure und AWS betrachtet. Explizit nicht weiter vertieft werden Code-Level Unterschiede.

Insbesondere für Referenzarchitekturen gilt es dabei ein scharfes Auge für die jeweiligen Eigenheiten der CSPs zu behalten und Vendor Lock-Ins in dem jeweilig spezifischen Fall soweit möglich zu reduzieren.

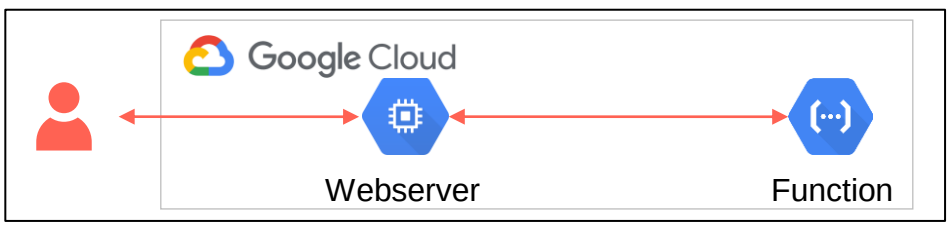
Was ist ein Vendor Lock-in?

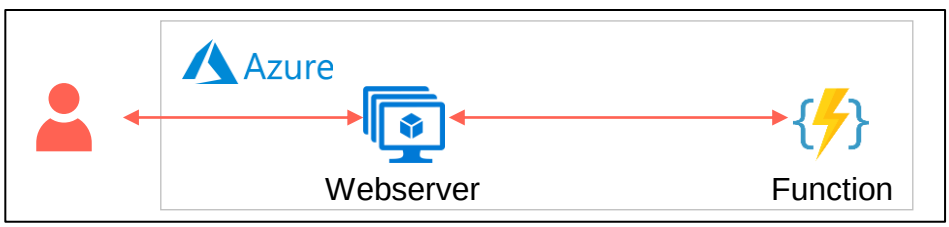
Der Vendor Lock-in beschreibt das Problem, dass ein Kunde nur mit hohem Aufwand oder hohen Kosten einen genutzten Service oder ein verwendetes Produkt durch eine gleichwertige Lösung eines anderen Anbieters austauschen kann, da diese z. B. proprietäre Technologien verwenden, welche mit anderen Anbietern inkompatibel ist. Ein Vendor Lock-in kann jedoch auch aus vertraglichen oder prozessualen Abhängigkeiten entstehen.

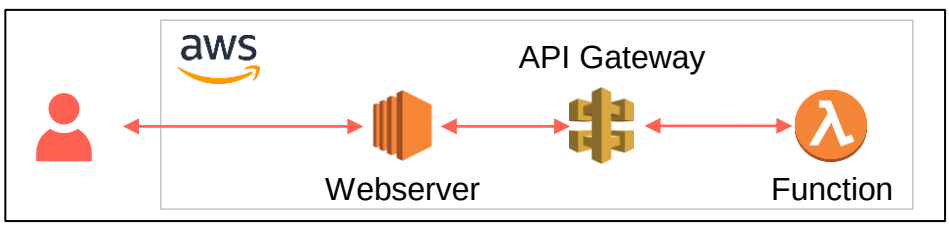
Use Case 1: Konvertierung von Identifiern in neue Nummernkreise

In diesem Use Case geht es um eine Anwendung, die einen bestehenden Identifier, z. B. eine Kundennummer mit einer Länge von 12 Zeichen in eine mit 13 stellen konvertiert. Aktuell wird ein solche Konvertierung in einem Web-Service durchgeführt, welcher dauerhaft auf einem Applikation-Server läuft. Dieser Fall eignet sich hervorragend für Cloud Functions, da der Applikation-Server diesen Service nicht länger verwalten muss und auch die Uptime des Services drastisch verringert wird.

Mit einem Aufruf der Web-GUI (In diesem Falle zum Beispiel ein einfacher NGINX Webserver) und der Eingabe einer Kundennummer wird die FaaS per REST Aufruf gestartet und konvertiert die 12-stellige Kundennummer in das gewünschte Format. Nach erfüllter Aufgabe antwortet die Function dem Webserver mit dem Ergebnis und schaltet sich ab. Wie auf dem Diagramm zu sehen ist, sind die Architekturen zwischen Google und Microsoft gleich. Für AWS würde in diesem Fall allerdings ein weiteres Produkt benötigt, um HTML-Calls korrekt zu verarbeiten:



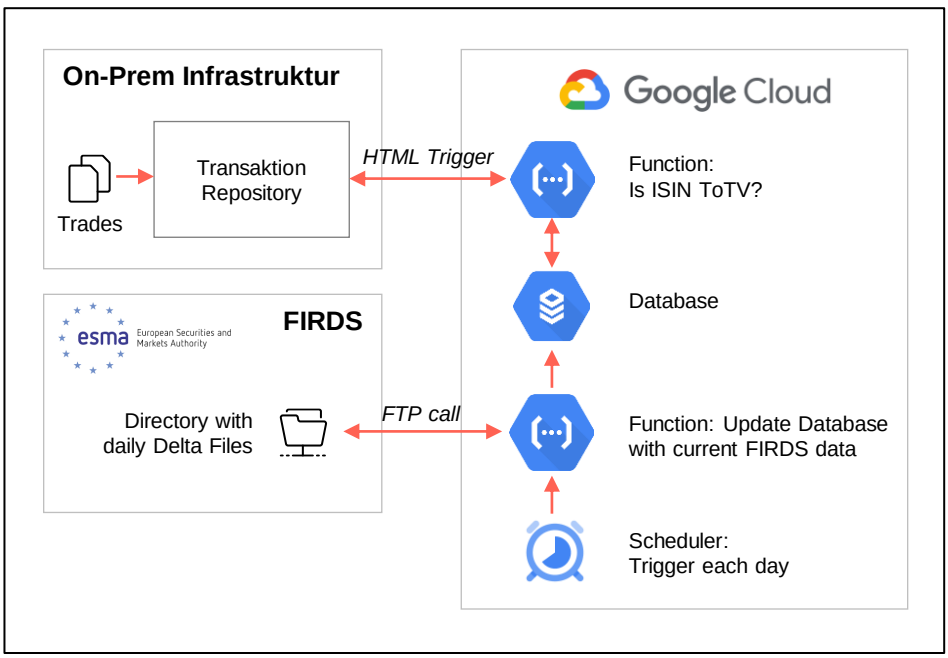




Use Case 2: Bestimmung der Meldepflichtigkeit unter MiFIR

Im Rahmen der Überprüfung auf die Reporting-Pflicht eines Instruments unter MiFIR ist der Marker für das Handeln auf Handelsplätzen (ToTV = Trading on Trading Venue) ein ausschlaggebendes Kriterium. Die Informationen hierzu muss sich die meldende Bank aus den FIRDS Datenbeständen selbst zusammensuchen und gegen Ihre Daten halten. Hierzu wird oft der gesamte Bestand der FIRDS sowie Delta-Verarbeitungen gespeichert und innerhalb des Melde-Moduls verarbeitet. Dies führt zu langen Laufzeiten und einer insgesamt Performance.

Auch hier bietet sich eine Umsetzung als Microservice mit Functions an. In einer möglichen Zielarchitektur werden nun Trades wie üblich in einem Transaktions-Repository angelandet. Zur Bestimmung der Meldepflichtigkeit wird nun auf Basis der ISIN eine Funktion abgerufen, die diese Prüfung gegen eine Cloud-Datenbank durchführt. Eine zweite Function verarbeitet davon unabhängig die von FIRDS propagierten Änderungen sowohl aus dem Trading Venues Schema wie auch aus dem ISIN / Venue Mapping Schema. In diesem Case ist die Architektur bei allen CSPs ähnlich. Im Beispiel ist der Aufbau auf der Google Cloud Plattform dargestellt.



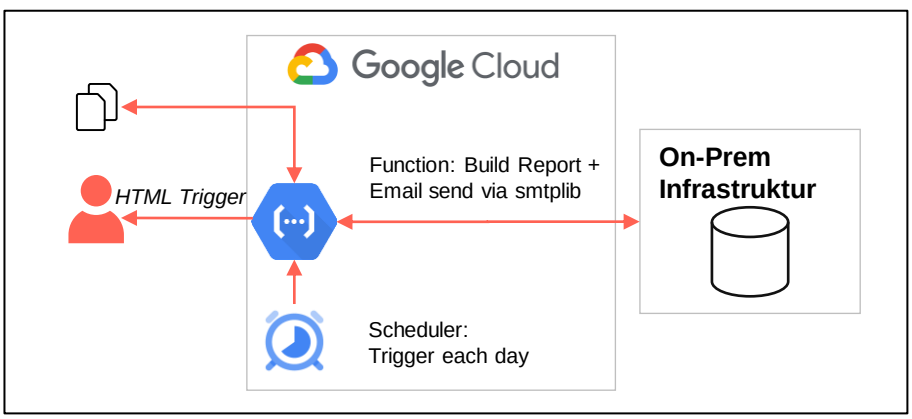
Use Case 3: Report-Generierung aus On-Prem Datenbank

Sowohl im Regelbetrieb wie auch für verschiedene Auditworkflows werden in Reports der Datenbestände oder eines bestimmten Teils der Datenbestände benötigt. Diese Reports oder Teilauszüge der Daten können auf zwei Wegen generiert werden:

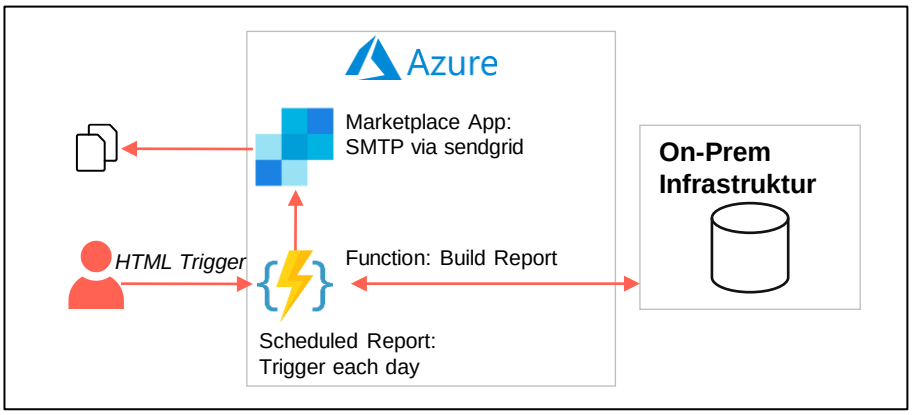
1. Ad-hoc, zum Beispiel durch manuelle Trigger aus einer GUI
2. Automatisiert durch maschinelle Trigger, beispielsweise eine bestimmte Uhrzeit, zu welcher der Report täglich erstellt werden soll oder wenn ein vorhergehender Prozess abgeschlossen ist.

In aktuellen Systemlandschaften finden sich Report-Services meist auf Applikations-Servern. Für diesen Use Case eignen sich jedoch FaaS-Dienste auch besonders gut aufgrund der seltenen und vergleichsmäßig einfachen Ausführung. Die Generierung der Reports wird in diesem Fall von einer HTTP-getriggerten Function übernommen, welche nach Aufruf Daten aus der on-Prem Datenbank holt und den Report erstellt.

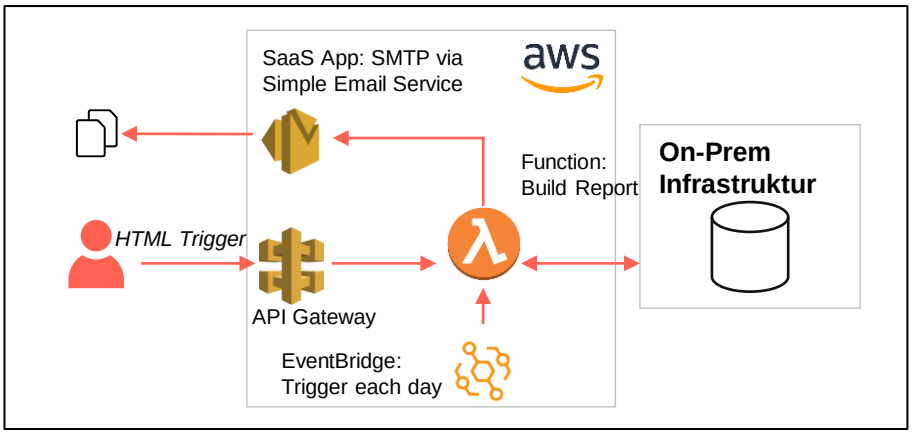
Spannend wird in diesem Fall, wie unterschiedlich man dieses Set-up bauen kann – teilweise aufgrund der unterschiedlichen Funktionalitäten der einzelnen CSPs und teilweise auch durch unterschiedliche Design-Entscheidungen. Im Google Cloud Beispiel übernimmt die Function das Request-Handling, die Report-Generierung und das Versenden der E-Mails über die Python-native SMTP-Funktion smtplib. Um über eine festgelegte Uhrzeit getriggert zu werden, benötigt die GCP Function allerdings den Cloud Scheduler.



Bei einer Lösung in Microsoft Azure funktioniert die Function ähnlich wie bei einer Lösung in GCP, allerdings ist hier kein extra Modul zur zeitlichen Aktivierung nötig. Azure Functions besitzt die Möglichkeit, zu einer festgelegten Uhrzeit gestartet zu werden. In der unten skizzierten Lösung wurde darüber hinaus für das Versenden der E-Mails die Marketplace App „sendgrid“ gewählt. Apps wie diese können Vorteile gegenüber nativen Implementierungen haben, erschweren eine Portabilität aber unter Umständen.



In diesem Beispiel zeigt sich auch die Philosophie von AWS im Vergleich zu den anderen Anbietern deutlicher: In der unten skizzierten Lösung können für dieselbe Funktionalität vier unterschiedlichen Services mit API Gateway, Lambda, EventBridge als Orchestrator und Simple Email Service (SES) eingesetzt werden, wo Google nur mit zweien agiert und Microsoft sogar nur mit einer einzigen Function den kompletten Use Case abbilden kann. Hieran erkennt man gut, dass Portabilität auf die architektonischen Abwägungen in größerem Maße einzahlen sollte.



Wie maximiere ich die Kompatibilität meiner Funktion für alle CSPs?

Damit die Portabilität von Functions zwischen verschiedenen CSPs möglichst einfach ist und eine entsprechende Nutzung auch den Anforderungen gemäß BAIT genügt, gilt es sowohl übergreifende Best Practices soweit möglich zu nutzen wie auch harte Lock-Ins durch agnostische Referenzarchitekturen zu vermeiden.

Best Practices nutzen

Neben der Orientierung an Referenzarchitekturen gibt es insbesondere auf dem Code-Level eine Menge an Best Practices, um die Kompatibilität zu maximieren.

1. Standardisierte Frameworks wie die von Serverless.com und der Cloud Native Computing Foundation bieten einen guten Ausgangspunkt, um sich Portabilität wirklich strukturiert anzunähern. Voraussetzung ist natürlich, dass man sich der Struktur anpasst und die Komplexität eines generalisierenden Frameworks in Kauf nimmt. Dafür ist der Support aber sehr weit verbreitet: Die Unterstützung z. B. für das Modell von Serverless.com wird von allen wichtigen Cloud Anbietern unterstützt und beinhalten unter anderem die gesamte Konfiguration der Events und der Function in einer einzigen Datei (serverless.yml). Zusätzlich bietet das Framework open-source Anweisungsbefehle, wodurch dieselben Befehle für die Bereitstellung der Function über alle Provider hinweg verwendet werden können.

2. Die Nutzung von einfachen und portablen Eventtypen zur Kommunikation und Verbindung von Services ist darüber hinaus ein wichtiger Punkt. So sollten Interaktionsmuster möglichst agnostisch replizierbar sein und im Zweifel lieber vereinfacht aufgesetzt werden.

Harte Lock-Ins vermeiden

Insbesondere unter den Faktoren Programmiersprache, Funktionalität, Performance und Skalierung sollten CSP-exklusive Angebote und Maximalnutzungen möglichst vermieden werden:

1. In Go zu entwickeln mag spannend sein und Vorteile bieten, schränkt die Portabilität zu Microsoft Azure zum Beispiel aber sofort ein. Auch insbesondere bei Custom Runtimes sollte lieber auf die eigenen Präferenzen zugunsten der verbreiteten Sprachen Java, Ruby und node.js verzichtet werden.

2. Wrapper-Funktionalitäten mit mehr als 15 Minuten Laufzeit sind nur begrenzt portabel. Insbesondere bei Kaskaden von Functions z. B. in Microservices kann man hier schnell in Lock-Ins Hineinstolpern, auch aufgrund der umliegenden Ökosysteme. Ein gutes Beispiel hier ist das Logic Apps Umfeld bei Microsoft, welches zwar viele praktische Funktionalitäten bietet, welche aber architektonisch kaum portabel sind.

3. Auch zu große Files und Datenmengen sind im Handling ein Portabilitätsrisiko, da die maximal verfügbaren Speicher je nach CSP unterschiedlich sind. Insbesondere bei Use Cases, die Dateien laden sollen, muss hier auch mit genügend Spielraum kalkuliert werden, um auch Spitzen in Bulk Loads abfangen zu können.

4. Wenn die Anzahl der erwarteten parallelen Aufrufe nicht mehr mit allen CSPs abgebildet werden kann, weil sie die 1000 zum Beispiel übersteigt, hat man hier auch sofort Lock-In Effekte. Schon im Design der Architektur sollten diese Last-Faktoren berücksichtigt und umschifft werden.

5. Architekturen, die Funktionen mit Marketplace Apps verbinden, sollten mit Vorsicht genutzt werden und nur nach vorheriger Prüfung zur Verfügbarkeit bei anderen CSPs. Hier ist es leicht, in einen verschärften Lock-In von zwei Vendoren hineinzugeraten, der auf jeden Fall gut abgewogen werden sollte.

Fazit & Bewertung

Abschließend ist zu sagen, dass die regulatorischen Anforderungen durch MaRisk, BAIT oder KWG an die Auslagerung von IT-Diensten in die Cloud insbesondere für FaaS-Implementierungen gut zu bewältigen ist. Für die Nutzung dieses Betriebsmodelles in der Financial Services Industrie ist das ein gutes Signal, denn insbesondere die Ablösung der alten Host-Infrastrukturen durch fortwährende kleine Abspaltungen von Microservices findet in FaaS eine herausragende Unterstützung.

Um eben diese portablen FaaS-Nutzungen zu ermöglichen, ist die Berücksichtigung von und Einschränkung auf einigen Facetten wie den eingesetzten Programmiersprachen, proprietären Funktionen und Marketplace-Applications sowie Architekturen im Hinblick auf deren Leistungs- und Skalierungsansprüche notwendig.

Insbesondere bezüglich der Portabilität der einzelnen Codestrecken sollte aber auf jeden Fall auch klar sein, dass eine Eins-zu-eins-Portierung nicht möglich sein wird. Mit strategischen Vorarbeiten und Überlegungen zur Implementierung von FaaS mit Hilfe von standardisierten Frameworks wie zum Beispiel des Serverless Frameworks und der Verwendung einer serverless.yml können Portabilitätsprobleme jedoch fast vollständig aufgelöst werden, ohne dabei auf die Vorteile einer FaaS zu verzichten.

Interessiert an mehr?
Ihre Ansprechpartner:

Tristan Poetzsch

Manager

tristan.poetzsch@nexgenbc.com

André Rentschler

Senior Consultant

andre.rentschler@nexgenbc.com

NEXGEN

NEXGEN Business Consultants GmbH
Grüneburgweg 101
60323 Frankfurt am Main