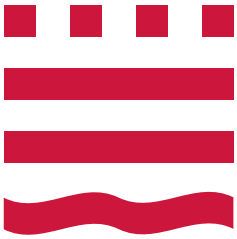


Technische Hochschule Brandenburg
University of Applied Sciences

BACHELORARBEIT

Serverless Architekturen für Konventionelle Webanwendungen

Vorgelegt von: Dragoljub Milasinovic
Matrikelnummer: 20140076

am: 23. September 2017

zum
Erlangen des akademischen Grades

BACHELOR OF SCIENCE
(B.Sc.)

Erstbetreuer: Prof. Dr.-Ing. Schafföner
Zweitbetreuer: Jonas Brüstel, M.Sc.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Vorgehen . . . 2
- 1.2 Ziel . . . 2
- 1.3 Aufbau der Arbeit . . . 3

2 Klassische Service-Modelle **5**
- 2.1 Cloud Eigenschaften . . . 5
- 2.2 IaaS . . . 6
- 2.3 PaaS . . . 7
- 2.4 SaaS . . . 8

3 FaaS **9**

4 Serverless-Angebote und Architekturen **11**
- 4.1 Serverless . . . 13
  - 4.1.1 Pipes and Filters, Compute as a Glue . . . 14
  - 4.1.2 Legacy Api Proxy . . . 14
  - 4.1.3 Compute as a Backend . . . 15
  - 4.1.4 Graph Query . . . 15
  - 4.1.5 Real time processing . . . 15
- 4.2 Patterns Serverless . . . 16
  - 4.2.1 Priority Queue . . . 16
  - 4.2.2 Fan Out . . . 16
  - 4.2.3 Federated Identity . . . 17

5 Fokus auf AWS, Vorstellung der AWS-Serverless-Angebote **19**
- 5.1 AWS Serverless Angebote . . . 19

6 KOMA, eine Beispielanwendung **23**
- 6.1 Anforderungen Analyse . . . 24
  - 6.1.1 Komponenten Übersicht . . . 24
- 6.2 Umsetzung . . . 25
  - 6.2.1 Datenhaltung Analyse und Auswahl . . . 25
    - 6.2.1.1 Semantic Web . . . 26
    - 6.2.1.2 Ontologie . . . 26

6.2.1.3 RDF 28
6.2.1.4 Sparql 28
6.3 Entwurf 29
6.3.1 Datenhaltung Synchronisierung und Replizieren 29
6.3.2 Datenverarbeitung 30
6.3.3 Abtrennung des Monoliths 30
6.3.4 RESTful API 30
6.3.5 Single Page Application 31

7 Bewertung **33**

8 Ausblick **37**

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die von mir eingereichte Masterarbeit selbstständig verfasst, ausschließlich die angegebenen Hilfsmittel benutzt und sowohl wörtliche, als auch sinngemäße entlehnte Stellen als solche kenntlich gemacht habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Brandenburg an der Havel, 21. September 2017

Dragoljub Milasinovic

Abstrakt

Pomodoro @Deutch @English

1 Einleitung

Idee-Ausführung-Markt

Ideen entstehen, verändern sich, werden im Laufe der Zeit vergessen, manchmal begeistern sie. Ihnen Form und Inhalt zu geben, also sie umzusetzen, ist die Voraussetzung, um nachzuvollziehen ob die ursprüngliche Idee wirklich ausgebaut und verstanden worden ist.

Die Technik kann als Medium für den Ausdruck solcher Ideen eingesetzt werden. Diese kann so komplex werden, dass sie eine Barriere in Form eines Wissensmonopols darstellt, die hinderlich für die Umsetzung neuer Ideen ist.

Die Faktoren am Anfang einer technologischen Umsetzung einer Idee sind:

- Prof of Concept
- Time-To-Market
- Cost of Human Resources, Skill-shortage
- Technical technological details
- Profitability

Ein Zeichen für die Existenz dieser Komplexität im Rahmen des Cloudcomputings ist die Entstehung neuer Technologien für die Herstellung von Technologie.

Je mehr Anforderungen auf einem System z.B. Webanwendung, desto komplexer wird es. Je mehr Softwarekomponente, desto mehr Verwaltungsaufwand mittels Load Balancing, Messaging usw. Je mehr Veränderlichkeit, desto schwieriger ihre Integration und Skalierung. [HW04]

Die Cloud Anbieter versuchen diesen Verwaltungsaufwand, Skalierung und Integrationschwierigkeiten mittels einem neuen Architekturstil Serverless.

Um die Umsetzungsvorgänge einer Webanwendung möglichst simpel zu halten, werden in der vorliegende Arbeit die Serverless Architekturen für konventionelle Webanwendungen untersucht.

1.1 Vorgehen

Auf dem Weg zur technologischen Umsetzung einer neuen Idee liegen unbekannte Schwierigkeiten bei den Entscheidungen über ihrer Umsetzung. Problematisch können sich der Architekturentwurf, die IT Infrastruktur, die Drittanbieter von Software, die Auswahl der Infrastruktur usw. gestalten. Hinzu kommen Schwierigkeiten, die spezialisierte Kompetenzen, Fertigkeiten und „Know-How“ erfordern. Diese gehören jedoch nicht immer zum Problem der Domain der Anwendung.

Der Begriff Serverless weist darauf hin, dass die Verwaltung der zugrunde liegenden Serverinfrastruktur der Anwendung von Cloudanbietern übernommen wird.

Für dieses Problem wurde Function as a Service (FaaS) 3, als Lösung unter der Rubrik „ Serverless“ 4.1 von den Hauptanbietern von „Cloud“ Technologien vorgestellt.

FaaS definiert das Programmiermodel, eine Funktion oder auch „Nano-Microservice“genannt, um den serverless Architekturstil zu adoptieren.

Im Rahmen des Cloud-Computing handelt es sich in dieser Arbeit um eine Untersuchung der Serverless Architekturen am Beispiel einer konventionellen Webanwendung. Dabei wird besonders beachtet, ob und wie solche Technologien die Umsetzung erleichtern. Die Entwurfsmuster und die Kernfunktionalität werden ausschließlich mit Serverless Technologien am Beispiel der Kompetenz Matrix (KOMA) 6, mit AWS umgesetzt.

1.2 Ziel

Das Ziel ist ein Minimal Viable Product (Mivip),in Form von einer Single Page Application (SPA) 6.3.5, ausschließlich mit Serverless Technologien zur Verfügung zu stellen.

Nach der Umsetzung werden die Erfahrungen und Ergebnisse ausgewertet, um bei zukünftigen Entscheidungsprozessen bei der Umsetzung einer Webanwendung zu unterstützen.

Die Webanwendung soll möglichst flexibel für zukünftige Änderungen sein.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit beschäftigt sich mit dem Serverless Ausschnitt der Cloud Dienste.

Zuerst wird der Leser in die klassischen Servicemodelle 2 eingeführt. Zunächst werden die technischen Anforderungen und die dazugehörigen Beispiele des Serverless Ansatzes erläutert. Das nächste Kapitel überblickt die aktuellen Serverless Angebote der größten Cloud Anbieter. Den Kern der Arbeit bildet die Analyse und Darstellung von Serverless Architekturen 5 und sie fokussiert sich auf Amazon Web Services (AWS). In diesem Abschnitt 5 werden die Serverless4.1 Architekturen und das Programmiermodell vorgestellt, sowie die Entscheidungsprinzipien4.1 erläutert. Der praktische Teil beschäftigt sich mit der Umsetzung und Bewertung von der oben genannten Serverless Webanwendung KOMA 6.

Am Ende erfolgt eine Diskussion darüber, welche Trade-offs entstehen und welche Zukunftsperspektiven Serveless Technologien bieten.

2 Klassische Service-Modelle

Cloud Computing beschreibt die Bereitstellung von IT-Infrastruktur und IT-Leistungen wie beispielsweise Speicherplatz, Rechenleistung oder Anwendungssoftware als Service über das Internet. [Cha14]

2.1 Cloud Eigenschaften

Als Softwarearchitekt, Entwickler oder Projektmanager es ist wichtig, die spezifischen Eigenschaften von Cloud Angeboten zu verstehen. Aus einem Meer von Cloud Diensten ist die richtige Auswahl je nach Anforderungen und Art der technologischen Umsetzung schwer zu treffen.

Im Allgemeinen teilen Cloud Angebote laut Chandrasekaran (2014) folgende Eigenschaften:

- On-Demand Self-Service - Nutzer können die IT-Kapazitäten, die sie benötigen, selbständig ordern und einrichten. Der Anbieter muss in den Prozess nicht eingebunden werden.
- Broad Network Access bezeichnet den standardbasierten Netzzugriff von verschiedenen Endgeräten (z.B. Smartphones, Tablets, Laptops, PCs) aus.
- Measured Service bietet eine automatische Kontrolle und Optimierung der genutzten Ressourcen durch Metering, wodurch Transparenz für Anbieter und Nutzer sichergestellt wird. Somit bezahlen Kunden nur die Dienstleistungen, die sie auch tatsächlich in Anspruch nehmen.
- Resource Pooling - 3 – Ressourcen des Anbieters (z.B. Speicher oder Bandbreite) werden gebündelt, multimandantenfähig bereitgestellt und nach Bedarf zugewiesen.

- Rapid Elasticity - 2.1 – Kapazitäten sind schnell und dynamisch verfügbar und können je nach Bedarf skaliert werden. [Cha14]

Daher ergeben sich für die, in der Cloud betriebenen Anwendungen, folgende Eigenschaften:

- Isolated state - Der Zustand wird in kleinen Einheiten der Anwendung isoliert, so dass sie besser skaliert. Eine zustandslose IT Ressource kann ohne Synchronisierungen aggregiert oder gelöscht werden. Dieser Zustand bezieht sich nicht nur auf die Verwaltung der Interaktionen eines Clients, sondern auch auf dessen Datenverarbeitung.
- Distribution - Anwendungen müssen so in Komponenten zerlegt werden, dass sich ihre Ressourcen weltweit auf-/verteilen.
- Elasticity - Die Anpassung sowohl auf die Anzahl als auch auf die Leistungsfähigkeit der zu benutzenden IT Ressourcen kann im Sinne einer Addition oder Subtraktion erfolgen. Im ersten Fall nimmt auf der Ebene der horizontalen Skalierung (scale out) die Anzahl der Server zu. Während bei der vertikalen Skalierung (scale up) die Leistungsfähigkeit der Ressourcen der Server steigt.
- Automated management - Die konstanten Aufgaben zur Verwaltung von Elasticity sollen automatisiert werden, um eine Cloudanwendung fehlerresitent auf Ressourcenebene zu implementieren.
- Loose coupling - Die Minimierung von Abhängigkeiten einer Anwendung von IT Ressourcen vereinfacht die Bereitstellung, die Fehlerkontrolle und Wiederverwendung von Komponenten. 4 [Cha14]

2.2 IaaS

Infrastructure as a Service (IaaS) kann als ein Service beschrieben werden, der Abstraktionen für Hardware, Server und Netzwerkkomponenten bereitstellt. Der Serviceanbieter besitzt die Ausrüstung und ist für die Unterbringung, die Inbetriebnahme und die Wartung der Server verantwortlich [You15]. Der Benutzer bezahlt nicht für die Hardware, deren Lagerung und den Zugriff auf sie, sondern für die Nutzung des gesamten Servicemodells z.B.: Zahlung nach benutzten Stunden, Ressourcen usw.

Die Aufgaben für Systeme mit Softwareelementen wie Load Balancing, Transaktionen, Clustering, Caching, Messaging, und Datenredundanz werden komplexer. Diese Elemente fordern an, dass Server verwaltet, gewartet, patched und gesichert werden brauchen. In einer nicht-trivialen Systemumgebung sind solche Aufgaben zeitintensiv und ebenso aufwändig fertigzustellen, wie effizient zu betreiben. Infrastruktur und Hardware sind zwar nötige Komponenten für jegliche IT-Systeme, aber gleichzeitig stellen sie nur das Medium für deren Anwendung dar - sei es Geschäftslogik, oder ein darauf bauender Dienst.

2.3 PaaS

Platform as a Service (PaaS), kann als ein Service beschrieben werden, der eine Rechenplattform liefert, z.B. ein Betriebssystem, eine Ausführungsumgebung für Programmiersprachen (siehe ElasticBeanstalk), eine Datenbank oder einen Webserver. Dieser Dienst übernimmt je nach benutzerdefinierter Konfiguration sowohl die Wartung der Datenbank, des Webservers und der Versionen des Laufzeitquellcodes, als auch deren Skalierbarkeit. [You15].

Inkonsistenzen in der Infrastruktur oder den Umgebungen können durch den hohen Aufwand der Serververwaltung entstehen. Sie werden durch standardisierte und automatisierte Angebote von PaaS umgangen. Deren effiziente Benutzung is abhängig davon, wie gezielt der Quellcode auf die Features der Plattform abgestimmt ist. Dies ergibt auf einer Seite weniger Wartung, aber andererseits erfordern die importierten Anwendungen (z.B. für ein Standalone Server) eine Anpassung an die Plattform.

Die Containerisierung ist eine Isolierung der Anwendung von ihrer Umgebung. Die Konfiguration des Containers, sowie dessen Deployment ist nicht trivial und erfordert daher spezialisiertes Wissen über Containerisierung. Für das Monitoring werden bestimmte Tools wie Boot2Docker [Goo17a] oder cAdvisor [Goo17b] benötigt. Jedoch bietet die Containerisierung eine ausgezeichnete Lösung für Anwendungen mit starker Kopplung zu anderen Softwarekomponenten. [You15]

2.4 SaaS

Software as a Service (SaaS) kann als ein Service beschrieben werden, der OS-Images mit konfigurierbaren Diensten wie Datenbanken, Webanwendungen usw. bereitstellt. SaaS gestaltet sich benutzerfreundlich, da die Konfiguration und das Deployment dieser Softwaredienste nicht erlernt werden müssen, um sie in eine größere Anwendung einzubinden. Anfallende Gebühren berechnen sich nach der Nutzungsdauer. Viele traditionelle Software bietet seine SaaS nicht an. Dies Impliziert laut Chandrasekaran, K. dass dieses Serviceliefermodell sich für Anwendungen nicht gut eignet wegen folgenden Punkten:

- Geringe Latenz kann durch die Entfernung der gespeicherten Daten für Echtzeitanwendungen nicht gewährleistet werden.
- Die Datensicherheit kann nicht sichergestellt werden, da die mitbeteiligten Drittanbieter bei SaaS die Service Level Agreement (SLA) von Kunden nicht immer erfüllen.
- Anforderungen bestimmter Software verlangen eine Zentralisierung und eine Lokalisierung vor Ort, anders als bei SaaS.

3 FaaS

Function as a Service (FaaS) kann als ein Rechenservice beschrieben werden, der nach Anfrage isoliert, unabhängig und granular ausgeführt wird. Komplexe Probleme wie horizontale und vertikale Skalierbarkeit, Fehlertoleranz und Elascitity werden von Kunden nur noch nach Bedarf konfiguriert und von dem Anbieter verwaltet. Die Besonderheit von FaaS ist die „Unit of Deployment“ und die Skalierung in Form einer Funktion. [You15]

Events innerhalb eines verteilten Systems müssen verwaltet werden. Technologien wie Virtualisierte oder Containerized Server erzeugen neue Serverinstanzen zur Verarbeitung von einer Kette von variablen Events, die danach gelöscht wird. [Kin16] Die entstehende Problematik ergab sich durch eine starke Zunahme an Elementen, die einen hohen Verwaltungsaufwand forderten. Was zurück auf die oben genannten Aufgaben 2.2 führte.

Polling ist der Ausdruck für eine zyklische Abfrage über einen Status z.B. von Ports oder Locks über Ressourcen. Die Verwendung von Systemressourcen ist ineffizient im Vergleich zu Alternativansätzen wie z.B in dem Push- oder Pull- Kommunikationsmodell. [Kin16]

Funktionale Programmierung ist ein Programmierparadigma, in dem Funktionen nicht nur definiert und angewendet werden können, sondern auch wie Daten miteinander verknüpft, als Parameter verwendet und als Funktionsergebnisse auftreten können. Zustand und mutable Daten werden vermieden damit Seiteneffekten nicht entstehen und die Komposition flexibler wird. Das stellt einen Vorteil für die Skalierung eines Softwarekomponenten dar. [Ray13]

Die Implementierung einer solchen Funktion geschieht durch die Auswahl der Programmiersprache und der von dem Cloud-Anbieter vorgegebenen Funktionsfassade.

Diese wird vom Cloud-Anbieter aufgerufen, stellt aber keine zusätzlichen Bibliotheken bereit, daher ist es nötig, dass die auszuführende Datei alle Abhängigkeiten enthält.

Der Fokus bei FaaS liegt auf der Quellcodeentwicklung und nicht auf dem Provisioning von Servern, der Installation von Software, dem Deployment von Containern oder auf konkreten Details der Infrastruktur.

Für die Betrachtung, ob FaaS eine Lösung für eine konkrete Problemstellung ist, folgt eine Auflistung von Kriterien:

Es ist nicht empfehlenswert FaaS zu benutzen, wenn:

- der Entwickler Rootzugriffsrechte auf alle Ressourcen eines Servers benötigt.
- die Priorisierung von Betriebssystemattributen wie CPU, GPU, Networking oder Speichergeschwindigkeit angefordert ist.
- Sicherheit relevant ist. Unautorisierte Zugriffe können mit FaaS nur auf Systemebene erkannt werden.
- dauerhafte Prozesse angefordert sind.

Es ist empfehlenswert FaaS zu benutzen, wenn:

- Aufgaben als Reaktion auf Events erledigt werden.
- ein Scriptbehälter für z.B Cron Aufgaben benötigt wird. Die Zugriffsrechte sind beschränkter, Fehler sind einfacher zu erkennen und an einer Stelle aggregiert (siehe CloudWatch), des weiteren können Deployments einfacher angestoßen werden.
- die Skalierung des Servers bei einer ressourcenintensiven Verarbeitung vermieden werden soll.4.1.1
- Services vorgegeben sind, die selten benutzt werden.
- die Verwaltung von API-Server umgangen werden soll.

[Kin16]

4 Serverless-Angebote und Architekturen

Die Softwarearchitekturen helfen uns zu kommunizieren, welchen Zweck unsere Software erreichen möchte. Ihre **Entwurfsmuster** bieten generische Lösungen für wiederkehrende Probleme bei der Softwareentwicklung.

Auf konventionelle Webanwendungen wird hier Bezug genommen, als ein System, das über Presentation-, Data-, and Logik-Tiers verfügt. Jedes Tier kann mehrere Logik-Layers enthalten, die für unterschiedliche Funktionalitäten der Domains verantwortlich sind. Logging ist ein Beispiel für Cross-Cutting Concern, das Layers überspannt. Die Komplexität der Anwendung wächst zusammen mit der Beschichtung. Eine Überprüfung der Architektur ist sinnvoll, wenn eine erfolgreiche Codeänderung von einer Anderen abhängt.

Service Oriented Architecture (SOA) unterlegt der Annahme, dass ein System aus mehreren kleinen, austauschbaren, wiederverwendbaren und entkoppelten Diensten besteht. Für die Entwicklung und Integration solcher Systeme bietet SOA eine Menge von Entwurfsprinzipien und Standards. Entwickler stellen autonome Services her, die durch Nachrichtenübergabe kommunizieren und oft einen Schema haben oder eine Schnittstelle, die definiert wie die Nachrichten erzeugt werden. [Cha14]

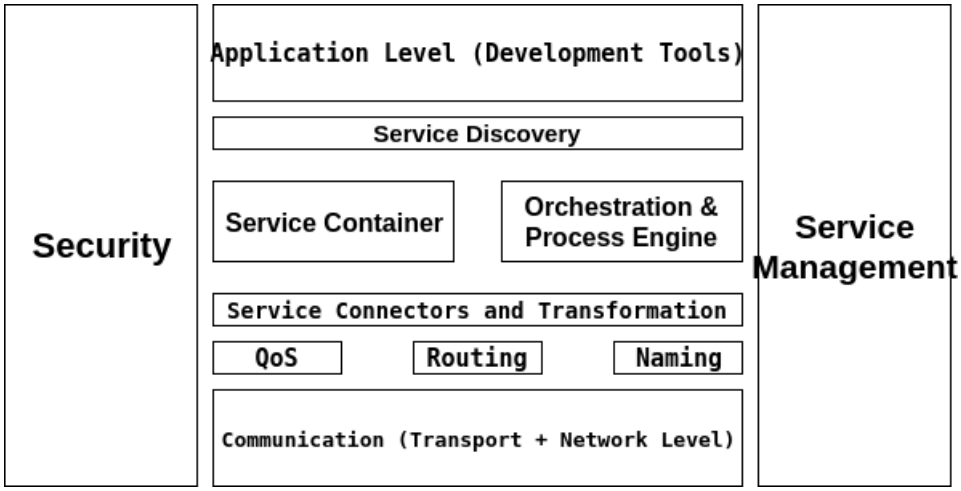


Abbildung 4.1: SOA Architekturreferenz [Sta07]

SOA ermöglicht gegenseitigen Datenaustausch zwischen Programmen von unterschiedlichen Anbietern, ohne dass zusätzliche Änderungen an den Services vorzunehmen sind. Die Bestandteile eines solchen Architekturstils sind sowohl Standardschnittstellen als auch voneinander unabhängige Services [Sta07].

Der Fokus in der Cloud liegt daher auf Service und Servicekomposition [Cha14].

Microservices und Serverless versuchen die Komplexität der SOA (Abbildung 4.1) anzusprechen. Beide Ansätze führen Separation of Concerns, häufige Deployments und Heterogene Domain Specific Language (DSL) mit sich. [H+17]

Auf einer Seite können Microservices ihren Zustand und Daten speichern und mit Hilfe von Frameworks implementiert werden. Auf der anderen Seite sind Serverless zustandlos, ihre Datenspeicherung ist zeitlich begrenzt und sie unterstützen Frameworks nicht direkt. ConnectWise [Con17a], Netflix [Net17] und UNLESS [UNL17] sind Beispiele für Unternehmen, die auch von Serverless Architekturen profitieren.

Event Driven Architecture (EDA) ist eine Softwarearchitektur, in der das Zusammenspiel der Komponenten durch Ereignisse gesteuert wird. Die Ereignisorientierung besitzt das Potenzial, dass die Architekturen von Anwendungen agiler, reaktionsschneller und echtzeitfähig werden. Laut Ralf Bruns und Jürgen Dunkel ist EDA für komplexe Fachlogik, große Datenvolumina, geringe Latenzzeit, Skalierbarkeit und Agilität geeignet. [RB]
Die Serverless Technologien können durch Benachrichtigungen gestartet werden. Dieser EDA-Stil verstärkt die Entkopplung auf einer temporären Ebene zwischen Producer

und Consumer. Weiterhin ermöglicht ein Kommunikationskanal zur Benachrichtigung eine asynchrone Verarbeitung, ohne dass das System auf Grund von Fehlern abstürzt. [HW04]

Zusammenfassend sind folgende Vorteile ersichtlich:

- Die Wiederverwendung von Services in unterschiedlichen Anwendungen senkt die Entwicklungskosten und den Time-To-Market.
- Durch die Standarisierung der Services kann ein System mit einer Rekonfiguration und ohne Weiterentwicklung schnell auf die geschäftliche oder externe Bedürfnisse angepasst werden. Somit wird ein agiles Arbeiten möglich.
- Das Monitoring hilft Fehler zu erkennen und die Leistung zu messen.
- Aggregate von bereitgestellten Services können komplexere und domainübergreifende Aufgaben ausführen.

[Cha14]

Im späteren Kapitel wird REST als Teil des EDA-Architekturstils vorgestellt.

4.1 Serverless

Serverless kann als ein Ansatz beschrieben werden, dar die Verwendung von einem Rechen-Service, Dienste von Drittanbieter, von Application Interface (API)s, und die Anwendung von Architekturmustern(wie ein Front-End, dass direkt mit Services kommunizieren anhand eines „Delegation-Tokens“) fördert. FaaS ist nur ein Aspekt dessen.

Prinzipien von Serverless Architeturen: [Sba17] Wie in der Sektion von SOA4 erwähnt wurde, das gezwungene Separation of Concerns bring das Single Responsability Principle (SRP) mit sich. Funktionen werden dadurch mehr Überprüfbar.

Deren Vernetzung erlaubt komplexe Systeme zu entwerfen, die dank der Stateless Natur der Funktionen schnell zu skalieren sind. Diese Vernetzung kann durch einen Push basierte Event-driven Pipe-line. Um die Komplexität des Systems zu reduzieren und die längerfristige Wartbarkeit zu verbessern, wird der Controller und/oder Router im Client und Third Party Dienste hinzugefügt.

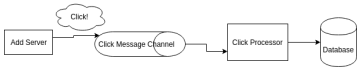


Abbildung 4.2: Classic Add Server

In einem beispielhaften konventionellen AdServer, nach einem Click auf eine Werbung wird eine Nachricht über ein Kanal an einen Clickprozessor, derer Laufzeit eine Anwendung ist, geschickt.

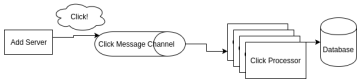
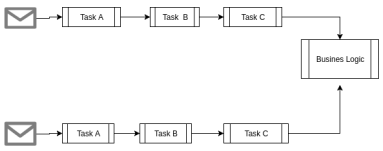


Abbildung 4.3: Serverless Add Server

Im Serverless wird dieser Clickprozessor als Funktion pro Nachricht instantiiert derer Laufzeitumgebung und Messagebroker der Cloudanbieter liefert. [Fow17]

Für den Entwurf von Serverless Architekturen wird eine Reihe von Mustern von unterschiedlichen Autoren vorgeschlagen.

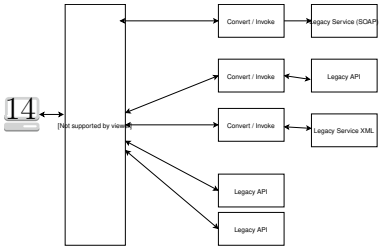
4.1.1 Pipes and Filters, Compute as a Glue



Eine Anwendung kann Aufgaben von unterschiedliche Komplexität und Vielfalt. Deren Implementierung in einen monolithischen Modul erschwert das Refactoring, die Optimierung oder die Wiederverwendung.

Zerteile man die Aufgaben in Diskreten Elementen (oder Filter) nach dem SRP und kombiniere sie in eine Pipeline. Dies hilft redundanter Quellcode zu vermeiden, ihn zu löschen, ersetzen oder zu integrieren in zusätzlichen Komponenten sobald die Aufgabenanforderungen sich ändern [HSB+14]. Ein weiter Vorteil entsteht wenn ein Element zu langsam ist, können mehrere Instanzen erzeugt werden und daher ein Flaschenhals vermeiden.

4.1.2 Legacy Api Proxy



Wenn eine API veraltet oder schwer zu benutzen ist, kann eine extra (RESTful) API in Vordergrund

erstellt werden, die mit Prozesse die Daten Transponieren und Aufstellen (marshall) für die angeforderte Formate. Dies ist besonders nützlich wenn die legacy Services selten benutzt werden und erleichtert die Integration mit anderen Architekturansätze.

4.1.3 Compute as a Backend

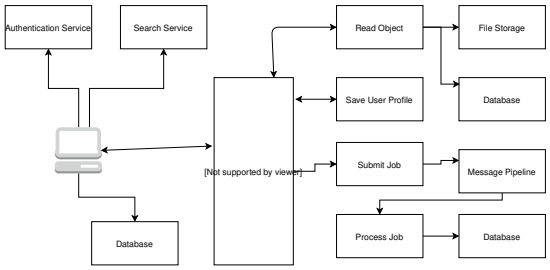


Abbildung 4.4: Compute as a Backend

Obwohl der Client direkt mit Services kommunizieren kann, vertraute Informationen müssen geschützt werden, indem sie das Backend verarbeitet [Sba17]. Diese Aufgaben können hinter einer REST Schnittstelle als koordiniert werden. Wie es in Abschnitt 4.1 erwähnt wurde, die Einbettung von Dienste von Drittanbieter und Thick-FrontEnds minimiert den Fußabdrück des eigenes Backends.

4.1.4 Graph Query

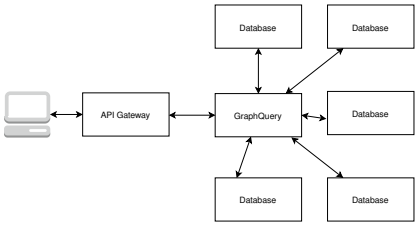


Abbildung 4.5: Graph Query

Wenn mit ein Request mehrere Datenbanken abgefragt werden, können multiple Round-Trip, wenn die Schnittstelle in einer Anfrage zu wenig Queryparameter zur Verfügung stellt, und Overfetching, wenn die Query nicht präzise genug formuliert werden kann, als Nachteile bei eine REST Schnittstelle entstehen. Dagegen kann der Client die Parameter für die Abfrage oder Query spezifizieren und Sie hinter eine API zusammenbauen und ausführen.

4.1.5 Real time processing

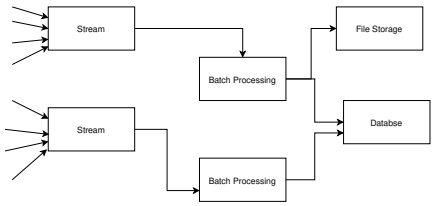


Abbildung 4.6: Real Time Processing

Die Verarbeitung von Streams in Echtzeit kann durch einen Buffer der nach Konfiguration die weiter an den Worker die Daten leitet. Der Vorteil entsteht wenn das Stream, der Worker und die Speicherung nach Anfrage Skaliert und nach Fehler die Verarbeitung der Daten neu versucht werden.

4.2 Patterns Serverless

4.2.1 Priority Queue

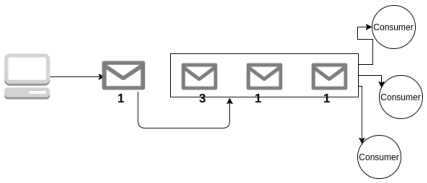


Abbildung 4.7: Priority Queue

Anwendungen können spezifische Aufgaben delegieren, wie z.B integrieren mit anderen Anwendungen und Services. Am Beispiel einer FIFO Queue können die Nachrichten nach Priorität automatisch sortiert und asynchron verarbeitet werden. Für Systemen ohne integrierte Priorisierung können mehrere Queues für unterschiedliche Prioritäten benutzt werden und die Anzahl von Consumerprozessen entsprechend angepasst. Im letzteren Fall wird es dabei die Starvation von Nachrichten mit wenige Priorität vermieden.

Competing Consumers Aus der Perspektive dieses Musters stellt das Priority Queue4.2.1 ein Consumer Service dar. Die Anzahl von Requests kann sprengen und das System überbelasten. Daher kann das Consumer Service als Moderator von Anfragen an die Consumers arbeiten.

4.2.2 Fan Out

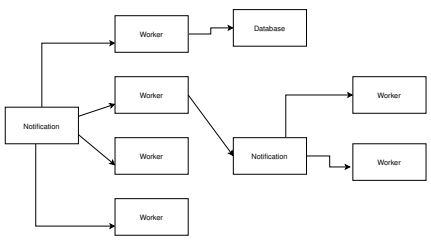


Abbildung 4.8: Fan Out

Bei dem eintritt eines Events können eins oder mehreren Subscribers durch die gepushte Benachrichtigung getriggert werden. Damit wird ein Kommunikationskanal4.2.1 zur Verwaltung von Nachrichten wiederverwendet und

extra Geschäftslogik z.B. Command Pattern für die gleiche Funktionalität [Sba17].

4.2.3 Federated Identity

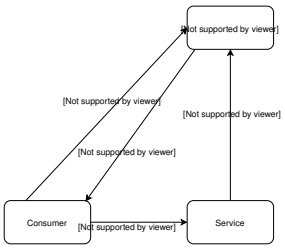


Abbildung 4.9: Federated Identity

Benutzer arbeiten mit multiple Anwendungen von unterschiedlichen Organisationen. Um die gleiche Zugangsdaten für Benutzter wiederverwenden zu können, wird dessen Identität bei einem Drittanbieter festgelegt und mit einem Token an die Anwendung weitergeleitet. Der Authentisierungscode kann daher von dem Anwendungscode abgetrennt werden.

5 Fokus auf AWS, Vorstellung der AWS-Serverless-Angebote

AWS ist ein Cloudandbieter

5.1 AWS Serverless Angebote

Lambda ist ein Rechen-Service, dass aus Quellcode und dessen Abhängigkeiten besteht. Die horizontale Skalierung erfolgt automatisch, elastisch und vom Cloud Anbieter verwaltet. Die vertikale skaliert nach Konfiguration. Lambda wird als Einheit für die Skalierung und Deployment benutzt und AWS unterstützt Javascript, Python, C# und Java. Der letzten Programmiersprache betrifft das Konzept „Warm-Up“ besonders, da die Java Virtual Machine (JVM) in derer Laufzeitumgebung auch hochfahren muss. AWS unterstützt das Pull- und Push- Kommunikationsmodell: Bei Pull überprüft die Lambda Laufzeitumgebung in regelmäßigen zeitlichen Abständen ob Events z.B. bei Kinesis eingetreten worden sind und ruft die entsprechende Lambda Funktion mit den Event-Datennutzlast auf; bei Push ruft die Eventquelle z.B S3 basiert auf deren Konfiguraion (event source mapping) die entprechende Lambda Funktion auf.

API Gateway ist eine Fassade, um Operationen wie Kunden per Email benachrichtigen, Identitätüberprüfung usw. sicher auszuführen. Weil die Skalabilität von API Gateway und von Lambda automatisch ist, sind die Bereitstellung und Wartung von EC2 Instanzen und die Konfiguration deren Load Balancer nicht mehr nötig.

Simple Notification Service (SNS) erweitert den schon gut etablierten Beobachtermuster in dem ein Kanal für „Events“ hinzufügt. Wird als „Publish-Subscribe Channel [HW04]“ Muster gennant und entspricht den oben „Fan Out“ 4.2.2 Architekturmuster. AWS kann durch Redundanz Regionen hinaus mindestens eine Lieferung der Nachricht gewährleisten.

Simple Storage Service (S3) ist ein Speicher-Dienst, dass durch SNS5.1 Events, wenn ein Objekt erzeugt oder gelöscht wurde, an SNS5.1, Simple Queue Service (SQS)5.1 oder Lambda schicken kann. „Buckets“ sind Wurzelverzeichnisse und Objekt ist eine Kombination von Daten, Metadaten und ein innerhalb des Buckets eindeutiges Key.

DynamoDb ist eine NoSQL Datenbank. Deren Tabellen bestehen aus Items (Zeilen) und deren Attributen (Spalten). Die Datenbank hat unendliche Datenkapazität und bedient jegliche Menge von Traffic. Mit der automatischien Skalierung mindert die Performance nicht. Lambda Funktionen können bei einen Update getriggert.

Simple Queue Service (SQS) ist eine message queue. Erlaubt die Interaktion von mehreren Publishers und Consumers in eine SQS und verwaltet automatisch das Lebenszyklus der Nachrichten. Lässt die Timeouts oder individuelle Verzögerungen zu kontrollieren.

Kinesis Streams ist ein Service für Real-time Prozessieren von Streams von Daten. Wird für Logging, Datein Einnahme, Metriken, Analytics und Reporting benutzt. Ein Kinesis Stream ist eine sortierte Folge von Datensätzen die auf „Shards“ verteilt. Diese definieren den Throughtput-Kapazität von einem Stream und können nach bedarf vergrößert werden.

Relational Database Service (RDS) hilft bei dem Setup und Wartung von mySQL, MariaDB, Oracle, MS-SQL, PostreSQL und Amazon Aurora mit automatische provisioning, backup, patching, recovery, repair, and

failure detection. RDS kann auf eigene Events mit Events SNS5.1 benachrichtigen.

Simple Email Service (SES) behandelt die Absendung und die Empfang-Operationen wie Spamfilterung, Virus Scann und Ablehnung von nicht vertraute Quellen. Events können weiterhin an S3, Lambda oder SNS.

6 KOMA, eine Beispielanwendung

Dieser Beispielanwendung implementiert manche der oben dargestellten Entwurfsmuster.

Kapitel 6 Die Beschleunigung der Veränderungen in der Heutigen Gesellschaft und der technologischen Landschaft prägt sich sowohl in Bildung als auch in Beruf in so fern aus, dass die heutige Rahmenlehrpläne nicht mehr fachlich sondern nach Kompetenzentwicklung orientiert sind, um Kompetenzprofile für Lerner zu gestalten [EQF17]. Es existiert bereits ein anerkanntes Europäisches Rahmen für Kompetenzbildung: European Qualifications Framework (EQF). Am Beispiel von Sachsen-Anhalt [SA17] wird diese Kompetenzorientierung auf die spezifische Bedürfnisse der Schule beschrieben und die Unterrichsstunden entsprechend gestaltet.

Die Umsetzung der Anwendung ermögliche auf einer Seite die von einem Individuum oder Schüler erworbene und zu erwerbenden Kompetenzen und deren Niveau nachvollziehen. Und auf der Anderen die entsprechende Bildungs-, Unterrichts- und Stundenplanung unterstützen. Der Kompetenzstand einer Person ist mit dem EQF vergleichbar, und daher International anerkennbar.

Wenn diese Anwendung in Bildungsinstitutionen eingesetzt wird, dienen die Rahmenlehrpläne als Leitpfad für die Belegung der Kompetenzen und KOMA für die Organisation der einzelnen Fachrichtungen oder Lehrveranstaltungen.

Der Kern solcher Organisation ist die Zuweisung von Aktivitäten auf vordefinierten Kompetenzen. Aktivitäten lassen sich einzeln oder in einer Sequenz anordnen. Sequenzen werden in Lehrveranstaltungen zusammengestellt. So können Aktivitäten, Sequenzen und Kompetenzen als Gestaltungsmittel für Lehrveranstaltungen genutzt werden.

Bekannte Beispiele TODO

6.1 Anforderungen Analyse

Die Auflistung 6.1 stellt ein für diese Arbeit angepasster Ausschnitt der Anforderungen für KOMA dar.

- Mit dem EQF vergleichbare Kompetenzeindefinitionen
- Zukünftige Erweiterungen berücksichtigen
- Von Browser abrufbar
- Private Datenspeicherung u.d Login
- Ertrag von großen Nutzlastschwankungen

Das ER-Modell von KOMA6 ergibt sich aus der Beschreibung in Kapitel 6 und der Anforderungsanalyse 6.1

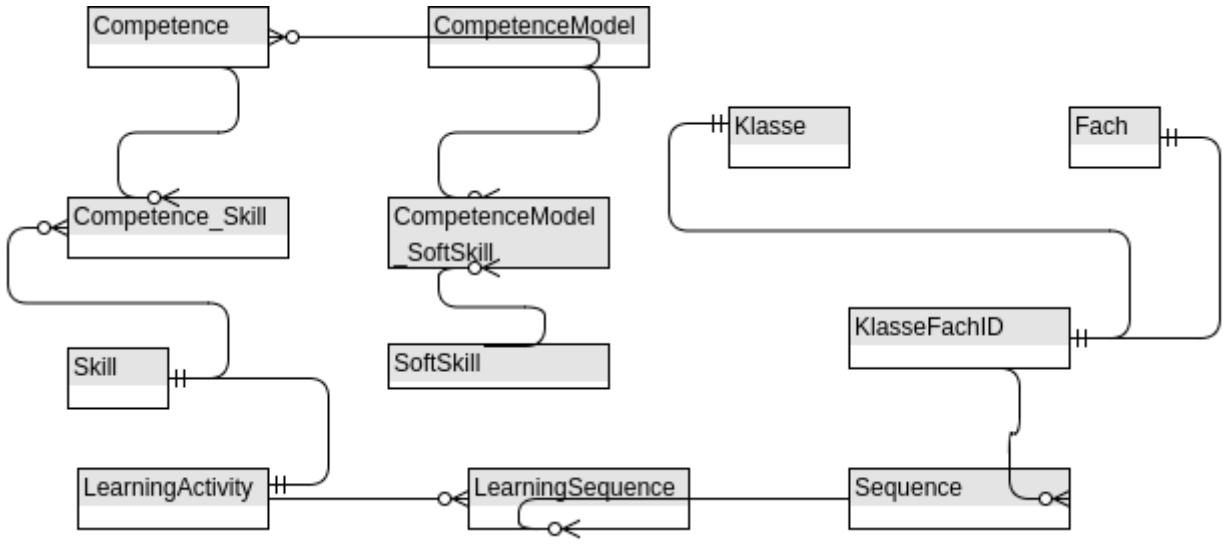
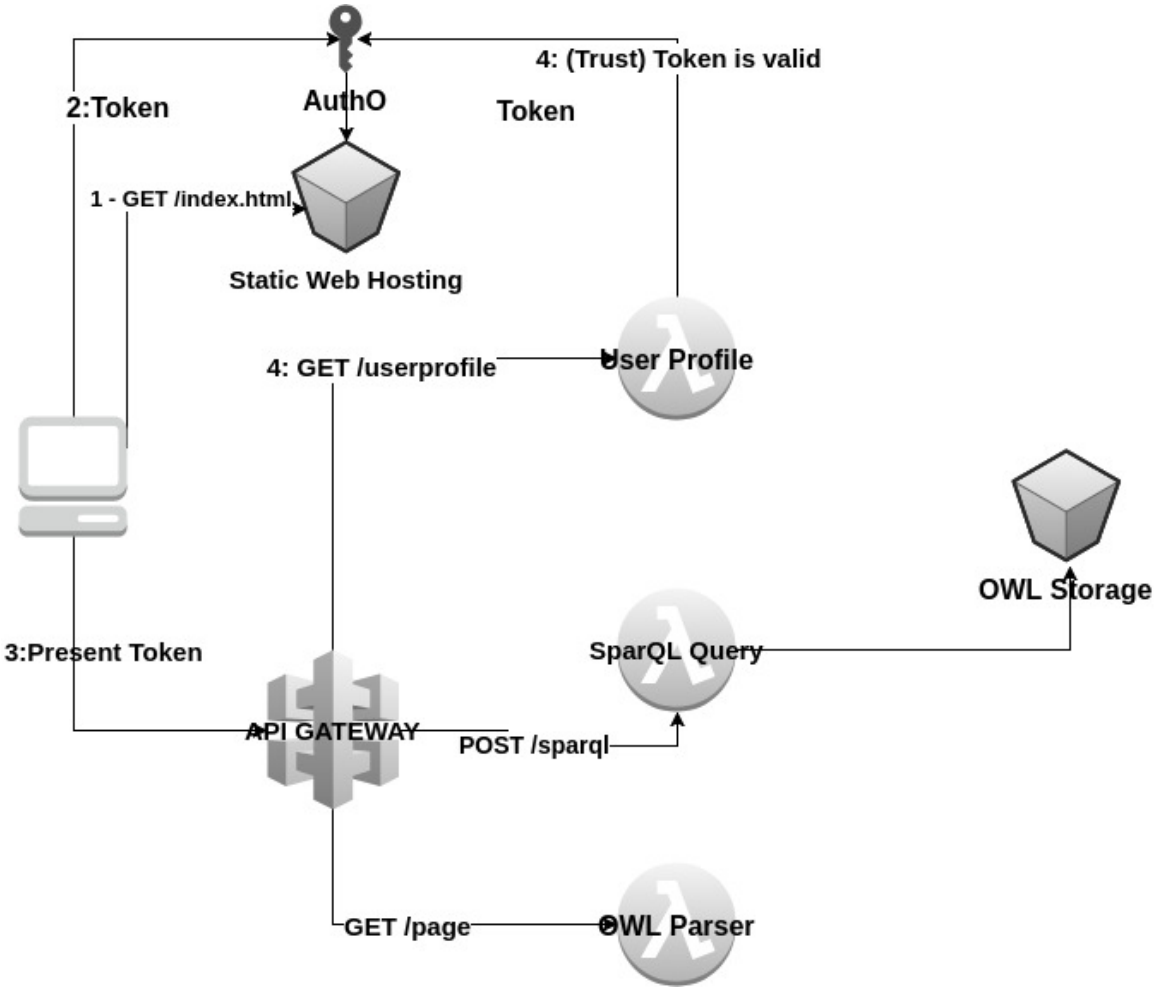


Abbildung 6.1: Entity-Relationship Modell für KOMA

6.1.1 Komponenten Übersicht

Um einen Anhaltspunkt zu verleihen, werden hier die Softwarekomponenten beschrieben.



6.2 Umsetzung

Die grundlegende Vorgehensweise bei der Umsetzung dieser Webanwendung wird zuerst die Datenhaltung Analyse und derer Auswahl. Als Zweites wird es zwischen der Entwurf mit den oben vorgestellten Serverless Architekturmustern und Technologien, und deren Implementierung iteriert, um schnellen Feedback zu erlangen.

6.2.1 Datenhaltung Analyse und Auswahl

Die Gestaltung von Kompetenzmodellen und deren zukünftige Weiterentwicklung hängt stark von spezifischen Bedürfnisse jeweiliger Schulen ab. Die mögliche Erweiterungen oder Anpassungen des Modells stellt die Benutzung des einer Relationale Datenbank für KOMA in Frage. In der folgender Tabelle werden die Eigenschaften von relationalen

mit ontologischen Schemas verglichen.

Tabelle 6.1: Vergleich relationalem mit ontologischem 6.2.1.2 Schema

Eigenschaft	Relational	Ontologisch
Darstellung Welt-Annahme	Existiert nur	Existiert mindestens
Individual	muss Unique	kann >= 1
Info	Ableitung = x	ja
Oritentation	Data	Bedeutung

Der Fokus auf die Erweiterung und Bedeutung des ontologischen Schemas führt zu deren Auswahl als Datendarstellungsformat. Anwendungen die von Vernetzten Datenbanken profitieren und danach entwickelt sind, werden unter dem Begriff „Linked Data Driven Web Application“bezeichnet [Con17b]. Dieser Begriff gehört zum „Semantic Web“ der in der Sektion der Ontologie6.2.1.2 weiter erläutert wird.

6.2.1.1 Semantic Web

Das „Semantic Web“ist eine Erweiterung des herkömmlichen Web, in der Informationen mit eindeutigen Bedeutungen versehen werden [GOS09]. Das World Wide Web Consortium (W3C) spezifiziert eine Menge von Standards und best practices für die Mitteilung von daten und deren semantische Darstellung. [Bob13]. Diese Bedeutungen werden für Maschinen durch Ontologien dargestellt, welche in „.owl “ Dateien gespeichert werden. [Con17b]

6.2.1.2 Ontologie

Konzeptuelle Modellierung

Eine Ontologie ist eine formale Spezifikation über eine Konzeptualisierung [SBF98]. Die Denotation jeweiliger dargestellten Signifikanten lässt sich durch seinen weltweit eindeutigen Präfix identifizieren z.B: `PREFIX owl: <http://www.w3.org/2002/07/owl#>` [Con17b]. Deren Beziehungen können auch zu externen Ontologie-signifikanten verweisen und dadurch ein Consensus über Begrifflichkeiten erreichen.

Während der Umsetzung wurde Protege [**?**] benutzt. Der Entwurf der Ontologie wurde nach Ontology-Engineering-101 durchgeführt:

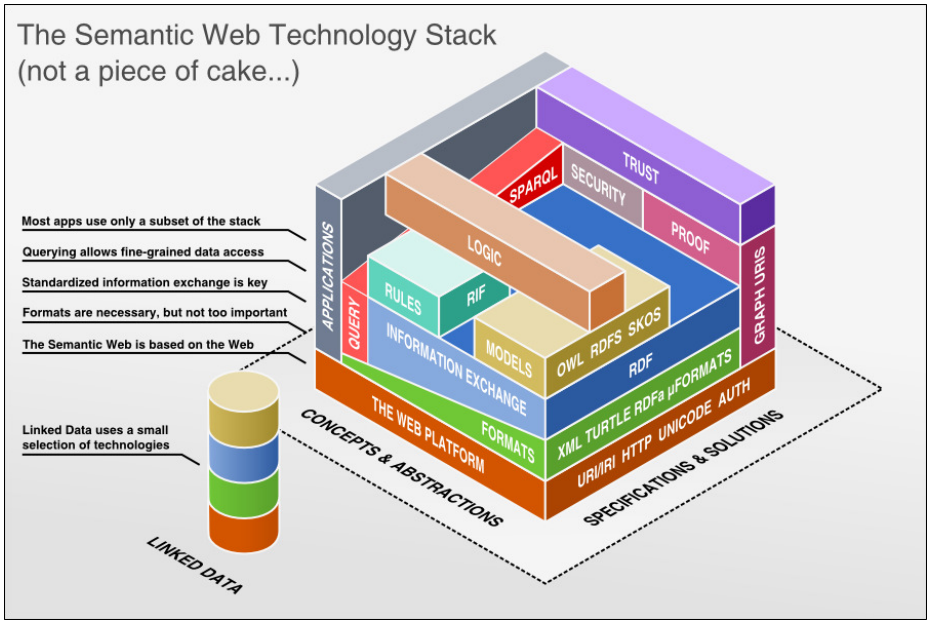
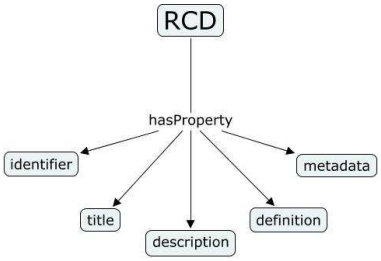


Abbildung 6.2: Überblick von Semantic Web Stack

Um die Neuerfindung des Rades zu vermeiden, die Recherche ergab einen aktuell öffentlichen graphischen ontologischen Entwurf [RMG14] siehe6.3 der in Moodle mit eine Relationalen Datenbasis und PHP umgesetzt. Nach dessen Ontologien wurde mittels Watson [Wat17] und LOD [LOD17] nichts öffentlich gefunden.

Bei der Analyse lässt der Entwurf und dessen Dokumentation freie Interpretation über Begriffe und deren Zweck, beispiele davon sind „isComposedOf“, „subsumes“. Ein Standard zur graphischen Darstellung ist zur Zeit?? noch nicht anerkannt. Obwohl Graphische Benutzeroberfläche @Cite research-gate graphol

Anderseits wurde das EQF für Ontologien beschrieben, aber nicht öffentlich umgesetzt. Es bietet dabei eine europäisch anerkannte Definition von Kompetenz, nämlich RCD [DCAB17]



Daher folgt eine beispielhafte Erklärung der auf unseren Anwendungsfall angepasste und ergänzende Interpretation der dargestellten Terminologie des Entwurfs und der RCD.

Erklärung der Terminologie Die zwei Leitmotive sind auf eine Seite Kompetenzanforderungen: sie legen fest, über welche Kompetenzen ein Schüler, eine

Schülerin verfügen muss, wenn wichtige Ziele der Schule als erreicht gelten sollen. Systematisch geordnet werden diese Anforderungengen in Kompetenzmodellen, die Aspekte, Abstufungen und Entwicklungsverläufe von Kompetenzen darstellen [Kli03]. Und auf der Anderen nach Kompetenz als die bei Individuen verfügbaren oder durch sie erlernbaren kognitiven Fähigkeiten und Fertigkeiten, um bestimmte Probleme zu lösen, sowie die damit verbundenen motivationalen, volitionalen und sozialen Bereitschaften und Fähigkeiten, um die Problemlösungen in variablen Situationen erfolgreich und verantwortungsvoll nutzen zu können [Wei02].

6.2.1.3 RDF

Die bisher erreichte Analyse des Domainsproblems soll nun anhand von Protégé in einen Resource Description Framework (RDF) format beschrieben werden. Der Menschen lesbarsten RDF Format ist Terse RDF Triple Language (TURTLE). Seine Syntax besteht aus Triples mit „.“beendete Zeilen. Ein Triple stellt ein Fakt dar, un besteht aus einem Subjekt, einem Prätikat und einem Objekt. Diese können sich in TURTLE verschachteln wie das folgende Listing 6.1 zeigt.

Listing 6.1: Darstellung von Triples in TURTLE

```
1 :EvidenceRecord rdf:type owl:Class .
2
3 :actionPerformed rdf:type owl:ObjectProperty ;
4 rdfs:domain :EvidenceSource ;
5 rdfs:range :Action .
```

6.2.1.4 Sparql

Um aus Ontologien Informationen zu entnehmen, wird die Abfragesprache Protocol And RDF Query Language (Sparql) verwendet. Diese ist ähnlich zu SQL. Mit dem Programm „Protétegé“können SPARQL Abfragen lokal ausgeführt werden.

Die einfachste Abfrage in Sparql wählt alle Triples vom Datenmodell.

```
1 SELECT * WHERE { ?s ?p ?o .}
```

Am Beispiel von KOMA, konkatenieren die Ergebnisse mit Zwei Abfragetriples dank eine Hilfsvariable. Die folgende Abfrage ließe sich „Wähle alle Properties des Graphes

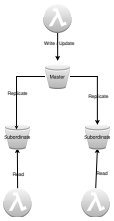
und wähle alle dessen Subjekten mit Alice als Objekt“formulieren.

```
1 PREFIX owl: <http://www.w3.org/2002/07/owl#>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
4 PREFIX koma: <https://s3-us-west-2.amazonaws.com/ontology.thb.de/koma-complex.owl#>
5 SELECT ?x WHERE {
6   ?y rdf:type owl:ObjectProperty .
7   ?x ?y koma:Alice .
8 }
```

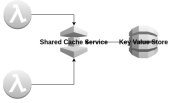
So dass auch die Benutzer von KOMA solche Abfragen stellen können wird ein „Sparql-endpoint“mit Hilfe von Apache Jena ARQ, ein Sparql-Engine, zur Verfügung gestellt. Dieser Sparql-Endpoint entspricht der Repositoryschicht der Anwendung und wird nach Anfrage von der Ontologie in S3 mittels Sparql JSON Objekte zurückliefern. Eine Lambdafunktion arbeitet als Schnittstelle **??** zwischen die ARQ Bibliothek, den Client und die darunterliegende Infrastruktur.

6.3 Entwurf

6.3.1 Datenhaltung Synchronisierung und Replizieren



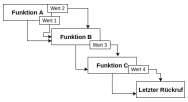
Master - Subordinate Cloud Dienste werden oft in unterschiedlichen Datazentren oder Regionen deployed. Um die Verfügbarkeit, Performance und Konsistenz zu maximiereun und die Datenübertragungkosten zu minimieren, kann eine Master-Datenbasis mit erlaubte Read und Write Operationen und eine Subordinate-Datenbasis mit nur Read Operationen definiert werden. Der Master pusht oder (one-way) synchronisiert die Subordinate-Replikas. Diese Aufteilung favorisiert Read Operationen.



cache shared Caching zielt auf die Verbesserung der Performance und Skalabilität eines Systems, in dem die oft gelesene Daten temporär zwischenspeichert werden. Es gibt zwei Typen von Cache: In-Memory und Shared-Cache. Die Stateless Natur der Lambda kompliziert die Benutzung der ersten Cachevariante. Die Shared-Cache hilft das Problem zu beheben, bei Inkonsistenzen unter verteilten Caches.

6.3.2 Datenverarbeitung

Waterfall Dieser Muster erlaubt eine Menge von Funktionen so zu verketten, dass das Ergebnis der Einer als Eingabeparameter der Nächsten ist. Wenn ein Fehler in eine Funktion entsteht, hält das Waterfall an.



6.3.3 Abtrennung des Monoliths

Im Folgendem es wird beispielhaft eine abtrennung einer JEE Anwendung.

6.3.4 RESTful API

Der Entwurf von einer benutzerfreundliche Hypertext Transfer Protocol (HTTP) API beinhalte die Abstraktion von komplexe Geschäftslogik und Daten, welche das darunterliegende Service benutzt, in den vier Create, Read, Update, Delete Operationen.

Die Komplexität des darunterliegenden Datenmodells erlaubt eine RESTful [H+17] Schnittstelle nur einfache abfragen zu formulieren. Daher zusätzlich ein Endpunkt6.2 für Komplexe Sparql Abfragen, die im Body des HTTP Requests in JSON geschickt wird.

Tabelle 6.2: RESTful API

Methode	URL	Rückgabe
GET	/ontology	Information über KOMA
GET	/ontology/{individual}	RDF von Individual
GET	/page	Auflistung von Entitäten
GET	/page/{individual}	Information über diesen Fakt
POST	/sparql	Abfragenergebnis

AWS API Gateway ermöglicht die Definition, Konfiguration und das Importieren von Schnittstellen. Beispielsweise kann die Abfrage GET https://<host>/page/{individual}

Listing 6.2: API Gateway Request Mapping Template

```
1 GET https://<host>/page/{individual}
2 ...
3 {
4 "individual" : "$input.params('individual')"
5 }
```

Damit wurde die zu erwartende Eingabe für den Sparql-Endpoint definiert. Der Zugang auf die Schnittstelle wird durch CORS konfiguriert um deren Ausnutzung zu vermeiden.

Dieser Endpunkt unterstützt nicht nur GET-Abrufe, sondern auch POST-Anforderungen mit einer Nutzlast. Unter der verfügbaren SparQL endpoints Implementierungen

6.3.5 Single Page Application

Da KOMA ohne Vorkenntnisse gebrauchsfertig sein soll, mit dem Fakt dass Milliarden von Desktop Geräte die Web mit einem Browser erkundigen können, lässt sich die Entscheidung über die Art der Benutzeroberfläche leicht Treffen.

Die Web Anwendung ist für alle Rechenaufgaben verantwortlich die im Browser aus dem Sicherheitssichtpunkt kein Gefahr darstellen, um den Backend oder Servers möglichst wenig auszulasten. Deswegen bietet sich eine Single Page Application an. Die SPA besteht aus ein einziges HTML Dokument. Dies vereinfacht man die Konfiguration der Authentifizierung und unterbricht den Fluss der UI-Darstellung zwischen Seiten.

Ein konfiguriertes Anfangsprojekt/Quickstart kann mithilfe von Initializr [Ini17] oder JHipster [Jhi17]. Für die lokale Entwicklung der Webseite werden anhand von NodeJS und NPM folgende Bibliotheken als Abhängigkeiten verwaltet: Bootstrap als Stylescheet und jQuery als Javascript-Bibliothek. Die Webseite wird Statisch mittels S3 geliefert. Dies geschieht mit einem Befehl:

Listing 6.3: Webseite veröffentlichen

```
1 $ aws --region us-west-2 s3 website --index-document index.html --error-document error.
      html 's3://koma.thb.de'
```

Da der Zugriff auf die Datenspeicherung gesichert werden soll, wird die Login-Funktionalität hinzugefügt.

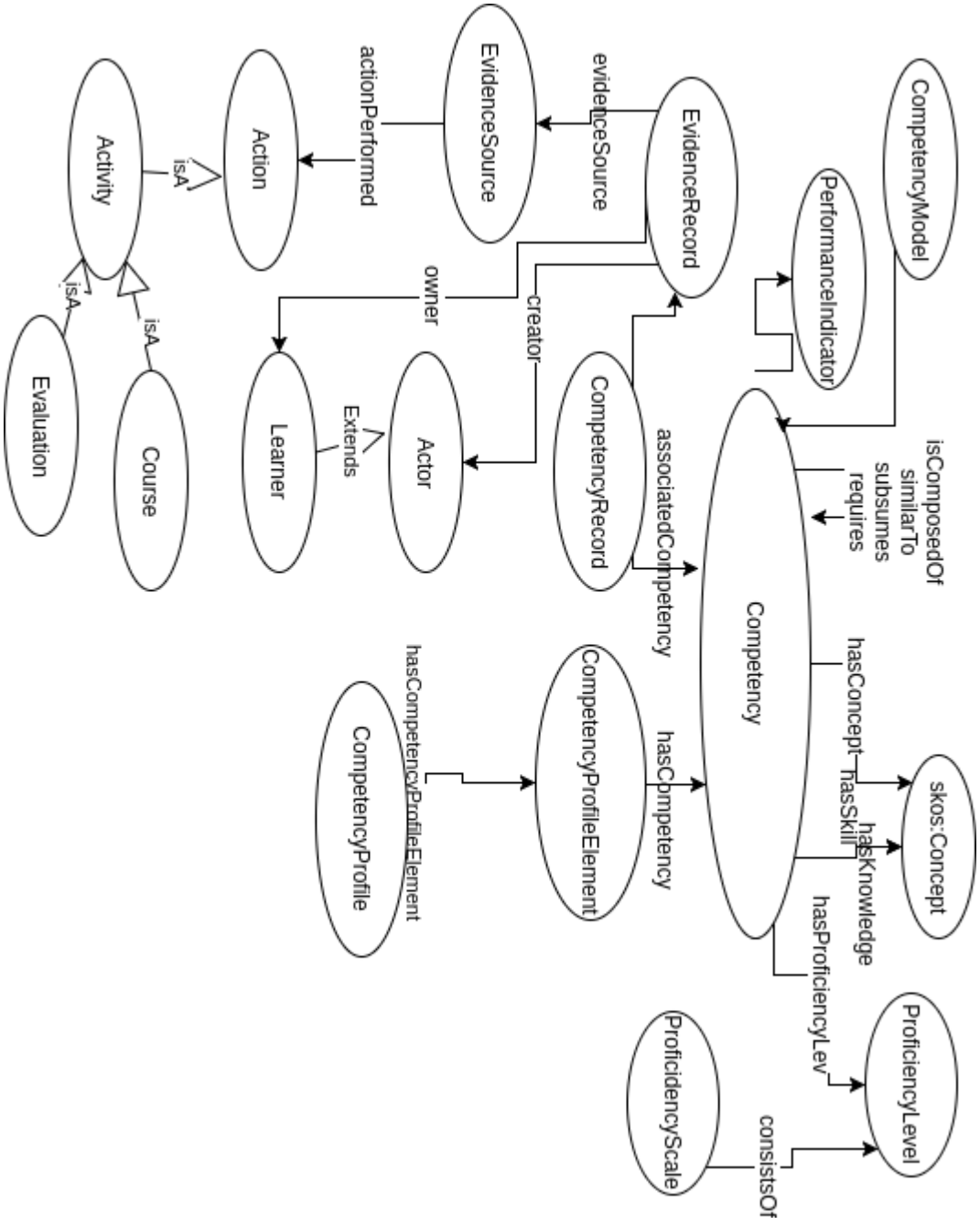


Abbildung 6.3: Kompetenzontologie

7 Bewertung

Zur Skalabilität Skallabilität in Datenhaltung -> Entwerfe für Distribution + vorteile von Lokalität. Read replikas -> evtl. Konsistent. Viele Perspektiven von Daten -> Lebenszyklus von Daten. —- Bounded Context —- Service ist nicht nur Funktion oder nur DB. Entkopplung fordert enkapsulation und Cohesion.

Event als Bussinesmanager -> Coordinator / Orchestrator -> Lambda Als Finite State Machine oder WorkFlow

Anwendung Latenz Die entstandene Webanwendung befindet sich in US-WEST-2, Oregon, in den USA. Da keine Cache oder CDN Funktionalität weder Implementiert noch konfiguriert ist, ist die Latenz direkt proportional zur Ausführungsdauer der Lambda Funktion. @Benchmark testing curl @Lambda Monitoring

In Zeiten des Cloud computings

Frameworks und FaaS Frameworks helfen aber sind platform abhängig. Entweder JEE und JVM oder PHP. Es kann auf die Layer of Abstraction in FW verzichtet werden. Die Ersetzbarkeit des FaaS entkoppelt die Anwendung und den Entwickler von der darunterliegende Technologie.

DevOps Frameworks Die benötigte Fertigkeiten für die Umsetzung einer Serverless Anwendung werden mithilfe von Deploymentframeworks gemindert. Die Aufnahme von 3.Anbieter ist deswegen notwendig. Es existieren bereits solche Hilfe wie z.B Serverlessframework@Ref

Risiko: Entickler brauchen einen guten Testplan und eine gute DevOps Strategie.<- skills shortage

Transaktionen Transaktionen können nicht parallel ausgeführt werden. Sequenciel aka Messaging Pattern. Zusammenspiel Arch. interfaces prog.modell und FW Arch 1st -> def interfaces and interactions. to program to a inteface

Eventual consistency -> event driven + ontology quality Consistenty -> koma-standalone <- transaction mgm

Vorteile Automatische Skalierung <!– größe und kleine Apps –> und Fehlertoleranz Automatisches Kapazitätsmanagement Flexible Ressourcenverwaltung Schnelle Bereitstellung der Ressourcen Exakte nutzungsabhängige Abrechnung der Ressourcen Konzentration auf den Kern des Source-Codes

Nachteile SLA Service Level Agreement: Latency, Bank:High volume Transactions, Decentralisation of Services = Challenge = Overhead, time, energy <- orchestration of events. Decentralisation vs monotithik != –komplexity Kontrollverlust Erhöhtes Lock-in Risiko

kurzlebige konfigurationen herausfinden ?? tracking? viel Konfiguration, kaum Konveniton -> .json 4 everything local testing braucht event-symulation.json

Zur Entwicklung Die Starke Komponentisierung und Dezentralisierung von Software, die Variabilität von Programmiermodellen, Frameworks, Tools, -Sprachen und dessen Entwicklungsumgebung erhöht die Komplexität des Enwicklungscyklus und hervorhebt die Bedürfnis von Tools zur Automatisierung von Tests, Deployment und Konfiguration. Also ein wohldefiniertes Handlungsplan bei der Softwareentwicklung dass von der nicht zu bearbeitende Details abstrahiert. Die DevOps Kultur spricht solche Probleme an. Neben dem Entwurf der Softwareachitekture muss, um derer Umsetzung Zeitgemäß zu gewährleisten, eine zum Projekt passende DevOps Strategie. Um Vorteil von der neuen Technologien zu nehmen, ist die Recherche nach schon existierenden DevOps Frameworks besonders wichtig. Dessen Integration in der DevOps Strategie diene für eine Agile Entwicklung.

Zum Datenmodel Aus der Anforderungs analyse einer Informations Techonologie Web Anwendung sind die Builder, Texste und dessen Darstellung das ergebniss, dass

ohne Daten inhaltlos wäre. Auf eine Seite Das Relationale Datenschema stellt keine Semantik für sich dar, sondern durch von der Software entstandene Verknüpfung zwischen dem Endergebnis und dem Datenschema. Auf der Anderen Seite die RDF Daten einer Ontologie *is* das Modell.

Zum API Gateway Bei Frameworks wie JEE werden Schnittstellen zwischen Layers und Tiers bereitgestellt und diese am Laufzeit entdeckt aka Service Discovery. Im Fall der API Gateway wird die Kopplung bei derer Konfiguration festgelegt wo derer Rekonfiguration ein neues Deployment ohne Downtime bedeutet. Die Der Quellcode der Dienste bleiben unberührt und kein Load Balancer muss rekonfiguriert werden.

Zum Serverless In dieser Arbeit wurde eine "nach buch"weise die Architektur gestalltet. Die unterschiedliche Interpretationen des Begriffs Serverless kann auch zu Kreativen anzätzen führen@AdamBien JEE Es kann daher auch als Serverless betrachtet wenn neue Quelldataien eine Docker Instance neu Erzeugen oder nur Updaten, dessen LoadBalancing auch als Serverless Quellcode verpakt werden kann.

Zur Annahme dass Quellcode schneller entwickelt werden kann, wenn der Entwickler sich nur damit beschäftigt.

8 Ausblick

RESTful UI RESTful Anfragen für bestimmte UI formate.

Listings

6.1 Darstellung von Triples in TURTLE 28
6.2 API Gateway Request Mapping Template 31
6.3 Webseite veröffentlichen 31

Tabellenverzeichnis

6.1 Vergleich relationalem mit ontologischem 6.2.1.2 Schema 26
6.2 RESTful API 30

Glossar

Abkürzungen

GC Garbage Collection

„Garbage Collection“ bezeichnet die automatische Speicherwaltung zur Minimierung des Speicherbedarfes eines Programmes. Garbage Collection (GC) wird zur Laufzeit durch Identifikation von nicht mehr benötigten Speicherbereichen ausgeführt. Im Vergleich zur manuellen Speicherverwaltung benötigt GC mehr Ressourcen.

Literaturverzeichnis

[Bob13] DuCharme Bob. Learning sparql. sl, 2013.

[Cha14] K Chandrasekaran. *Essentials of cloud computing*. CRC Press, 2014.

[Con17a] ConnectWise. Connectwise, 2017.

[Con17b] World Wide Web Consortium. World wide web consortium, 2017.

[DCAB17] Diego Duran, Gabriel Chanchí, Jose Luis Arciniegas, and Sandra Baldassarri. A semantic recommender system for idtv based on educational competencies. In *Applications and Usability of Interactive TV*. Springer, January 2017.

[EQF17] EQF. Eqf, 2017.

[Fow17] M Fowler. Serverless architectures, 2017.

[Goo17a] Google. Docker, 2017.

[Goo17b] Google. Docker monitoring, 2017.

[GOS09] Nicola Guarino, Daniel Oberle, and Steffen Staab. What is an ontology? In *Handbook on ontologies*, pages 1–17. Springer, 2009.

[H+17] II Hunter et al. Advanced microservices: A hands-on approach to microservice infrastructure and tooling. 2017.

[HSB+14] A. Homer, J. Sharp, L. Brader, M. Narumoto, and T. Swanson. *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*. Patterns & practices. Microsoft Developer Guidance, 2014.

[HW04] Gregor Hohpe and Bobby Woolf. *Enterprise integration patterns: Designing, building, and deploying messaging solutions*. Addison-Wesley Professional, 2004.

[Ini17] Initializr. Initializr, 2017.

[Jhi17] Jhipster. Jhipster, 2017.

[Kin16] W. King. *AWS Lambda: The Complete Guide to Serverless Microservices - Learn Everything You Need to Know about AWS Lambda!* AWS Lambda for Beginners, Serverless Microservices Series. CreateSpace Independent Publishing Platform, 2016.

[Kli03] Eckhard Klieme. ua: Zur entwicklung nationaler bildungsstandards–eine expertise. *Berlin 2003*, 2003.

[LOD17] LOD. Lod, 2017.

[Net17] Netflix, 2017.

[Ray13] Nilanjan Raychaudhuri. *Scala in action*. Manning Publications Co., 2013.

[RB] Jürgen Dunkel Ralf Bruns. *Event-Driven Architecture: Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*.

[RMG14] Kalthoum Rezgui, Hédia Mhiri, and Khaled Ghédira. Extending moodle functionalities with ontology-based competency management. *Procedia Computer Science*, 35:570–579, 2014.

[SA17] Sachsen-Anhalt. Sachsen-anhalt, 2017.

[Sba17] P. Sbarski. *Serverless Architectures on AWS: With Examples Using AWS Lambda*. Manning Publications Company, 2017.

[SBF98] Rudi Studer, V Richard Benjamins, and Dieter Fensel. Knowledge engineering: principles and methods. *Data & knowledge engineering*, 25(1-2):161–197, 1998.

[Sta07] Gernot Starke. *SOA-Expertenwissen: Methoden, Konzepte und Praxis serviceorientierter Architekturen*. dpunkt, 2007.

[UNL17] UNLESS, 2017.

[Wat17] Watson. Watson, 2017.

[Wei02] F.E. Weinert. *Leistungsmessungen in Schulen*. Beltz Pädagogik. Beltz, 2002.

[You15] Marcus Young. *Implementing Cloud Design Patterns for AWS*. Packt Publishing Ltd, 2015.