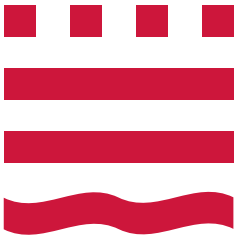


Technische Hochschule Brandenburg
University of Applied Sciences

BACHELORARBEIT

Serverless / Serverlose Architekturen für Konventionelle Webanwendungen

Vorgelegt von: Dragoljub Milasinovic
Matrikelnummer: 20140076

am: XX. Monat XXXX

zum
Erlangen des akademischen Grades

BACHELOR OF SCIENCE
(B.Sc.)

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

Inhaltsverzeichnis

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

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

1.1 Motivation

Auf dem Weg zur technologischen Umsetzung einer neuen Idee liegen unbekannte Schwierigkeiten bei der Entscheidungen über deren Umsetzung hinsichtlich auf den Architekturentwurf, die IT Infrastruktur, die Drittanbieter von Software, der Auswahl der Infrastruktur usw. Schwierigkeiten die von spezialisierten Kompetenzen, Fertigkeiten und „Know-How“ bedürfen. Gehören jedoch nicht immer zum Problem des Domäns der Anwendung.

Für dieses Problem wurde Function as a Service, kürz FaaS4, als Lösung unter der Rubrik „ Serverless4.4.1“ von den Hauptanbieter von „Cloud“ Technologien vorgestellt.

Der Begriff Serverless weist darauf hin, dass die Verwaltung der darunter liegende Serverinfrastruktur von Cloudanbieter übernommen wird. Jedoch begränzt FaaS dessen Bedeutung durch die Definition des Programmiermodells, nämlich eine Funktion, oder auch „ Nano-Microservice“.

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 geachtet ob und wie solche Technologien die Umsetzung erleichtern. Die Entwurfsmuster und die Kernfunktionalität werden mit ausschließlich Serverless Technologien am Beispiel von Kompetenz Matrix (KOMA), kurz KOMA6.2, mit AWS umgesetzt.

1.2 Ziel

Das Ziel ist ein Minimal Viable Product (Mivip), kürz Mivip, in Form von einer Single Page Application (SPA), kürz SPA6.4.7, mit ausschließlich Serverless Technologien vor zur Verfügung zu stellen.

Nach der Umsetzung werden die Erfahrungen und Ergebnisse ausgewertet, um dem Leser bei dem Entscheidungsprozess bei der Umsetzung einer Webanwendung besser zu Informieren.

Die Webanwendung soll möglichst für zukünftige Änderungen flexibel sein.

1.3 Aufbau der Arbeit

Zuerst wird den Leser in die Klassischen Service Modellen2 auffrischt. Zunächst werden die technische Anforderungen und die dazugehörige Beispiele des Serverless Ansatzes erläutert. Das nächste Kapitel überblickt die aktuelle Serverless Angebote der größten Cloud Anbietern. Anschließend trifft der Leser den Kern der Arbeit bei der Analyse und Darstellung von Serverless Architekturen5 fokussiert auf Amazon Web Services (AWS), kurz AWS. Darin werden die Serverless4.4.1 Architekturen eingeführt, das Programmiermodell vorgestellt und die Entscheidungsprinzipien4.4.2.3 erläutert. Der praktische Teil mit der Umsetzung und Bewertung von der oben genannte Serverless Webanwendung KOMA6.2. Am Ende wird es darüber diskutiert, welche Trade-offs entstehen und die Zukünftperspektiven von Serveless Technologien.

2 Klassische Service-Modellen

2.1 Cloud Eigenschaften

Als Software Architekt, Entwickler oder Projektmanager es ist wichtig, die spezifische Eigenschaften von Cloudangebote zu verstehen. Je nach Anforderungen und Art der technologischen Umsetzung lässt sich richtige Auswahl von ein Meer von Cloud Dienste schwer zu treffen.

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

Im allgemein heben sich folgende Eigenschaften bei der Betrachtung von Cloud Angebote.

- Probably not in a list. .... but explained narrativelly
- on-demand self-service
- broad network access
- measured service (pay-per-use)
- resource pooling and rapid elasticity

Daher können die in der Cloud betreibende Anwendungen folgende Eigenschaften haben:

- Isolated state
- Distribution
- Elasticity
- Automated management
- Loose coupling

2.2 IaaS

Infrastructure as a Service (IaaS), kurz IaaS, kann als ein Service beschrieben werden, dass Abstraktionen für Hardware, Servers und Netzwerk Komponenten bereitstellt. Der Serviceanbieter besitzt the Ausrüstung und ist für die Behausung, die Inbetriebnahme und die Wartung verantwortlich [You15]. Der Benutzer bezahlt nicht für das Hardware, dessen Lagerung und den Zugang auf ihn, sondern für die Nutzung des gesamtes Servicemodel z.B.: Zahlung nach benutzte Stunden, Ressourcen usw.

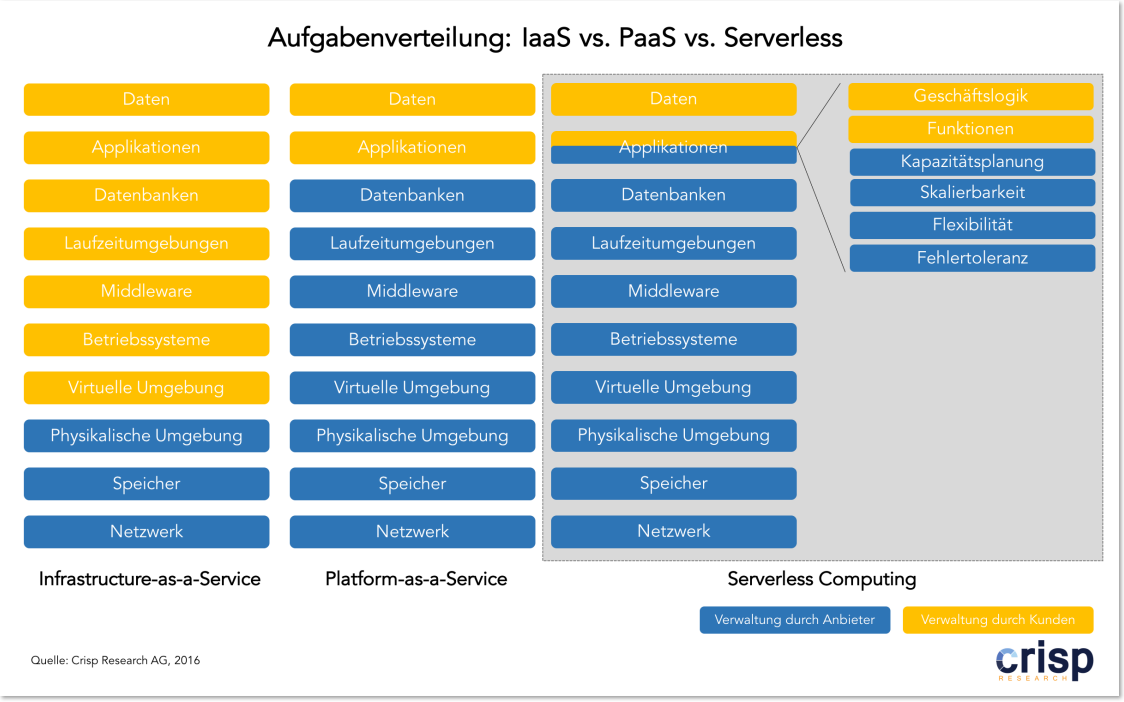
2.3 PaaS

Platform as a Service (PaaS), kurz PaaS, kann als ein Service beschrieben werden, dass eine Rechenplatform liefert, z.B. ein Betriebssystem, eine Ausführungsumgebung für Programmiersprachen(siehe ElasticBeanstalk), eine Datenbank oder ein Webserver. Dieser Dienst übernimmt sowohl die Wartung der Datenbank, Webserver und die Versionen des Laufzeitquellcodes als auch dessen Skalierbarkeit die nun nur Konfiguriert werden bracht [You15].

2.4 SaaS

Software as a Service (SaaS), kurz SaaS, kann als ein Service beschrieben werden, dass OS-Images mit konfigurierbaren Diensten wie Datenbanke, Webanwendungen usw. Der Nutzer muss die Konfiguration und Deployment nicht lernen, um die Dienste in einen größeren „Stack“ einzubinden. Die Gebühren sind generell nach benutzte Stunde.

Das „Virtualisation“ einer ganzen Umgebung oder Stack zerteilt sich in eine Sammlung von kleinen spezialisierten Aufgaben, die durch Drittanbieter implementiert wurden. Dieser Fakt steigerte die wirtschaftlichen Kosten und machte die Skalierungsmöglichkeiten komplexer [You15].


Aufgabenverteilung: IaaS vs. PaaS vs. Serverless
Daten
Applikationen
Datenbanken
Laufzeitumgebungen
Middleware
Betriebssysteme
Virtuelle Umgebung
Physikalische Umgebung
Speicher
Netzwerk
Infrastructure-as-a-Service
Platform-as-a-Service
Serverless Computing
Geschäftslogik
Funktionen
Kapazitätsplanung
Skalierbarkeit
Flexibilität
Fehlertoleranz
Verwaltung durch Anbieter
Verwaltung durch Kunden
Quelle: Crisp Research AG, 2016
crisp

3 Architektur Orient

3.1 SOA

Service Orientierte Architektur SOA unterlegt die Annahme dass, ein System aus mehreren kleinen, austauschbaren, wiederverwendbaren und entkoppelten Diensten bestehen kann. Stellt eine Menge von Entwurfsprinzipien und Standards für dessen Entwicklung und Integration [Cha14].

Cloud Computing ist ein Serviceliefermodel in dem Services und Ressourcen von Benutzern Internetweit, wie z.B. bei öffentliche Dienste nach Anfrage, verbraucht werden.

SOA ermöglicht gegenseitiges Datenaustausch zwischen Programmen von unterschiedlichen Anbieter ohne zusätzliche Änderungen an die Services vornehmen zu brauchen. Diese Services sollen unabhängig sein und „standard“ Schnittstellen haben.

Im Cloud wird es daher in Services und Servicekomposition zentriert.

Microservices und Serverless versuchen die Komplexität der SOA anzusprechen. Beide Ansätze zwingen auf Separation of Concerns, häufige Deployments und Heterogene Domain Specific Language (DSL), kürz DSL.

Auf einer Seite Microservices können ihren Zustand und Daten halten und werden mithilfe von „Frameworks“implementiert. Auf der anderen Seite Serverless sind Zustandlos. ConnectWise [Con17], Netflix [Net17] und UNLESS [UNL17] sind beispiele von Unternehmen die von Serverless Architekturen profitieren.

3.2 EDA

Die Serverless Technologien können durch Benachrichtigungen gestartet werden. Diese Event Driven Architecture (EDA) Still, verstärkt die Entkopplung temporäre-weise zwischen Producer/Anfrage und Consumer und ermöglicht eine Asynchrone Verarbeitung, ohne dass Fehler das System zum Absturz bringen.

Es lassen sich folgende Vorteile ersichtlichen:

- Wiederverwendung von Services in unterschiedlichen Anwendungen sinkt die Entwicklungskosten und Time-To-Market.
- Durch die Standarisierung der Services, ein System kann mit eine Rekonfiguration und ohne Wieder-Entwicklung schnell anpassbar auf die Geschäftliche/Externe Bedürfnisse. Agilität.
- Monitoring hilft Fehler zu erkennen und die Leistung zu messen.
- Aggregate von Services können Komplexere und Domän übergreifende Aufgaben ausführen.

Im späteren Kapitel tritt REST als teil der EDA Architekturstil.

4 FaaS

nichtfunktionalen/technischen Anforderungen zur Entwicklung des Serverless-Ansatzes

FaaS kann als ein Rechen-Service beschrieben werden, dass nach Anfrage isoliert, unabhängig und granular ausgeführt wird. Dessen „Unit of Deployment“ ist eine Funktion.

Komplexe Probleme wie horizontale und vertikale Skalierbarkeit, Fehlertoleranz, Flexibilität werden von Kunden und Benutzer nur noch nach bedarf Konfiguriert und von Anbieter verwaltet.

4.1 Technische Anforderungen

4.2 Ja/Neins

Die Unterschiede zwischen den [Kin16]

Wann Nicht :

Betriebsystemabhängigkeiten erkennen: Traditionellen Computing wie EC2, wo die Entwickler root zugrifsrechte auf alle Ressources hat, in Lambda ist diser Opaque.

OS Attribute Konfigurationen: Priorisieren CPU, GPU, Networking, or Disk Speicher und Geschwindigkeit. In Lambda skalieren proportional.

Sicherheit: Lambda nicht sichtbar host-based intrusion detection systems cannot be installed, system-level access logs

Dauerhafte Prozesse. Lambda kann bis 300s. Kosten pro Monat 1Gb Lambda = 37 EC2 = 9Dollar. Wenn warten auf Requests, oder auf Callbacks geht nicht.

Wann Ja EventDriven Tasks: Lambda solves the polling problem by creating an on-demand response to particular events.

Cron. Events: EC2 als behälter von Scripts, kostet auch ohne sie auszuführen. Fehlerhafte Scripts nicht einfach zu erkennen. Skallierung auch von Fehler. Permissions zu offen. Mit Lambda: the permissions can be much more narrowly applied, failures are much more easily noticed, deployments can be easily triggered, logs are aggregated in one place, and the underlying server management is handled by AWS.

Heavy Processing: Um autoskalling zu vermeiden wegen z.B.Bildverarbeitung. So entkoppelt sich der Empfänger und der Verarbeiter und können mehr Requests angenommen werden.

Serverless API Gateway vermeidet api servers

Selten verwendete Services. z.B <= 5 porZent average CPU t2.micro < > 3x10hoch6 = 9dollar

4.3 wie es sich in die Informatik einbettet

HTTP, Funktionale Programmierung, Fassaden, ... ?? Funktionale programmierung abstrahiert. auf Infrastruktur. ???

4.3.1 Use Cases

next [Sba17] Application Backend: z.b Internet of Things IoT: push to S3, push queue to SQS and invoke Lambda.

Data Procesisng and manipulation: pipeline of :collation and aggregation of data; image resizing; and format conversion.

Real time processing and analitics: Ingestion of Data -> Kinesis Streams; if Batch size -> Process, Save, Discard -> Lambda

Legacy Api Proxy: Extra RESTful GateWay with lambda on top legacy api. Easier Usage.

Scheduled Services

Bots and Skills

next [Kin16] Shuttdown untagged EC2 instances.

Code Deploy

Process inbount mail: attachment to S3 + link to it spam filter

Detect expiring certificates

4.4 kurzer Überblick über Serverless-Angebote/Architekturen

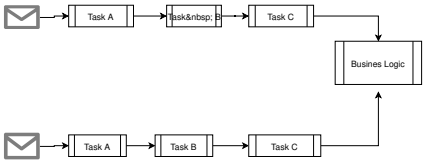
Die Architekturentwurfsmuster helfen uns zu kommunizieren welches Zweck unsere Software erreichen möchte und bieten generische Lösungen für wiederkehrende Probleme bei der Softwareentwicklung. Wenn eine erfolgreiche Codeänderung von andere Änderung abhängt, soll die Architektur überprüft werden. Für Konventionelle Webanwendung wird hier damit gehalten, als ein System das über Presentation, Data, and Application/Logik Tiers/Stufe verfügt. Jede Stufe kann mehreren Logik-Layers/Lagen enthalten, die für unterschiedlichen Funktionalitäten des Domäns verantwortlich sind. Logging wäre ein beispiel für Cross-Cutting Concern der Layers hinaus ausspannt. Die Komplexität wächst mit der Beschichtung zusammen.

4.4.1 Serverless

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

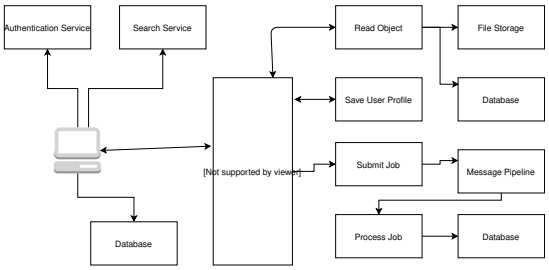
4.4.1.1 pipes and filters

compute as a glue pipelines built to carry out workflows <-> Async Messag Primer [HSB+14]

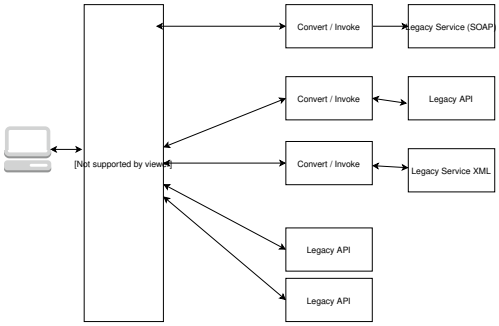


4.4.1.2 compute as a backend

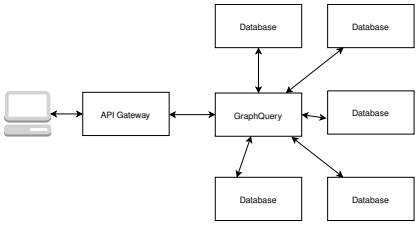
next [Sba17] Compute as a back end for web and mobile



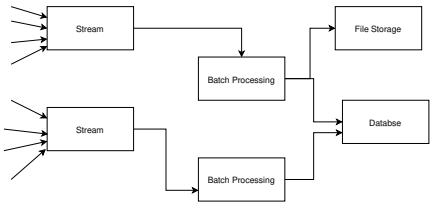
4.4.1.3 legacy api proxy



4.4.1.4 Graph Query



4.4.1.5 Real time processing



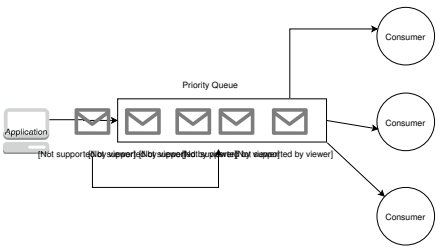
4.4.2 Patterns Serverless

Competing consumers als pattern für die implementaiton vom Kommunikation kanal für events die lambdas triggern.

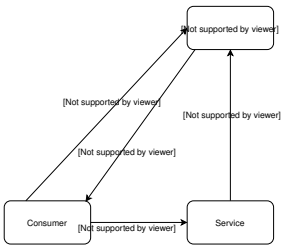
Pipes and Filters [HSB+14] <-> lambda event driven Health endpoint monitoring <-> EC2 certificate Leader election -> TRansaction mngmt S3

Queue Based Load Leveling -> Not appliable for Lambda Computing Model Retry -> Runtime Reconfiguration <-> Api Gateway Zero Downtime on Redploy Scheduler Agent supervisor -> Not needed if Design to Failure. Sharding -> Divide DataStore to Scale better -> Dynamo db + S3 Static Content Hosting -> Solved by S3 Throttling -> Scallability of Serverless Solved vs Transaction Vallet key <-> Auht0

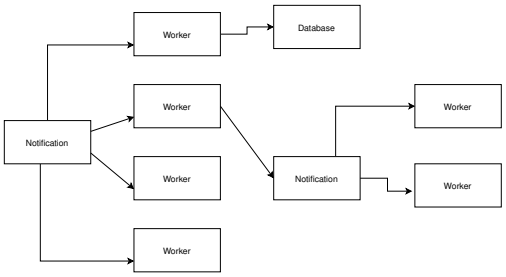
4.4.2.1 Priority Queue



4.4.2.2 Federated Identity



4.4.2.3 Fan Out [Sba17]



Prinzipien von Serverless Architeturen: [Sba17] Wie in der Sektion von SOA3.1 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-Based 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.

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. Im Serverless wird dieser Clickprozessor als Funktion pro Nachricht instantiiert derer Laufzeitumgebung und Messagebroker der Cloudanbieter liefert.@Cite Flower's Blog

Foo bar ist `f = 2 + 2` wasd asdf

4.4.3 Technologien

Microsoft Azure Funktions AuthO : AuthO Firebase : Google Stack Driver Logging : Google Cloud Machine Learning Engine : Google Cloud DataFlow : Google -> Stream batch pipelines Big Query : Google

Befehlmuster wird bei Fusekiserver6.4.1 als erteiler der Httpanfrage indem die SparQL Abfrage weiterleiten kann. Daher wird ein Pfad haus/hunde und haus/katze zur gleichen Funktion führen. Dieser kann aber offline gehen, also mehreren priorisierten MessagePattern als Queue vor eine oder mehreren Lambdas zu setzen absichert die

Stabilität des Systems und entkoppelt Komponenten @RoundRobin?? BSRN. Die Verkettung von Funktionen mittels „Pipes“erlaubt die mehrfache Filterung von Daten.

4.4.4 Guidance

Caching -> Lambda Reads Same source and Process Compute partitioning -> Lambda per use. solved Data Consistency -> Embrace Eventual Consistency in Distributed DBs Data partitioning <-> Sharding Instrumentation and Telemetry Guidance -> Errors Handling Service Metering -> understand future use of services

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

Per Type: The Company

5.1 AWS Serverless Angebote

Lambda ist ein Rechen-Service, dass aus Quellcode und dessen Abhängigkeiten besteht. Skaliert horizontal (scale out) und Vertikal (scale up), wird als Einheit für die Skalierung und Deployment benutzt und AWS unterstützt Javascript, Python, C# und Java. Der letzte Programmiersprache betrifft das Konzept „Lambda Warm-Up“ besonders, da die Java Virtual Machine (JVM) in derer Laufzeitumgebung auch hochfahren muss.

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“ 4.4.2.3 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.

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.

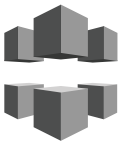
Simple Email Service (SES) behandelt die Absendung und die Empfang-Operationen wie Spamfilterung, Virus Scann und Ablehnung von nicht vertraute Quellen. Events können weiterhin an S3, Lambda oder SNS.

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.

DynamoDb ist eine NoSQL Datenbank. Deren Tabellen bestehen aus Items (Zeilen) und deren Attributen (Spalten). Lambda Funktionen können bei einen Update getriggert.

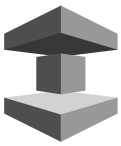
CloudSearch

CloudFront (CDN)



DNS management (Route 53)

Caching (ElastiCache)



Elastic Transcoder

Kinesis Streams

Cognito

6 Beispiel(!)-Fall KOMA

6.1 Anforderungen Analyse

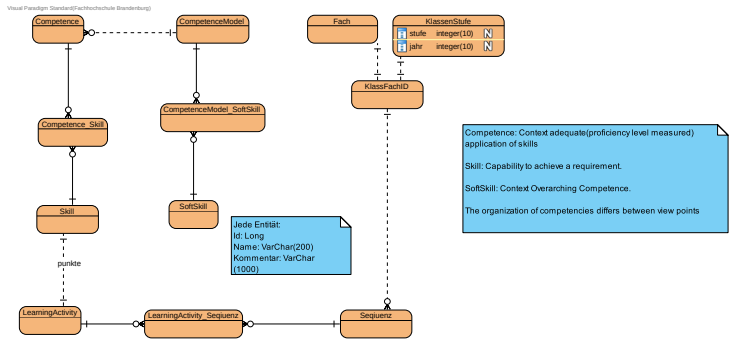
Zu den Anforderungen

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

6.2 KOMA

Beispiel Anwendung Abschnitt 6.2 Die Beschleunigung der Veränderungen in der Heutigen Gesellschaft und dem technologischen Horizont 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 gestallten. 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 Untertischstunden entsprechend gestaltet.

Die Umsetzung der Anwendung soll 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 benutzt. Das Modell verfügt von eine Figur um die erledigte Aktivitäten auszuwerten, nämlich Evaluation.

Bild des Modells

Bekannte Beispiele

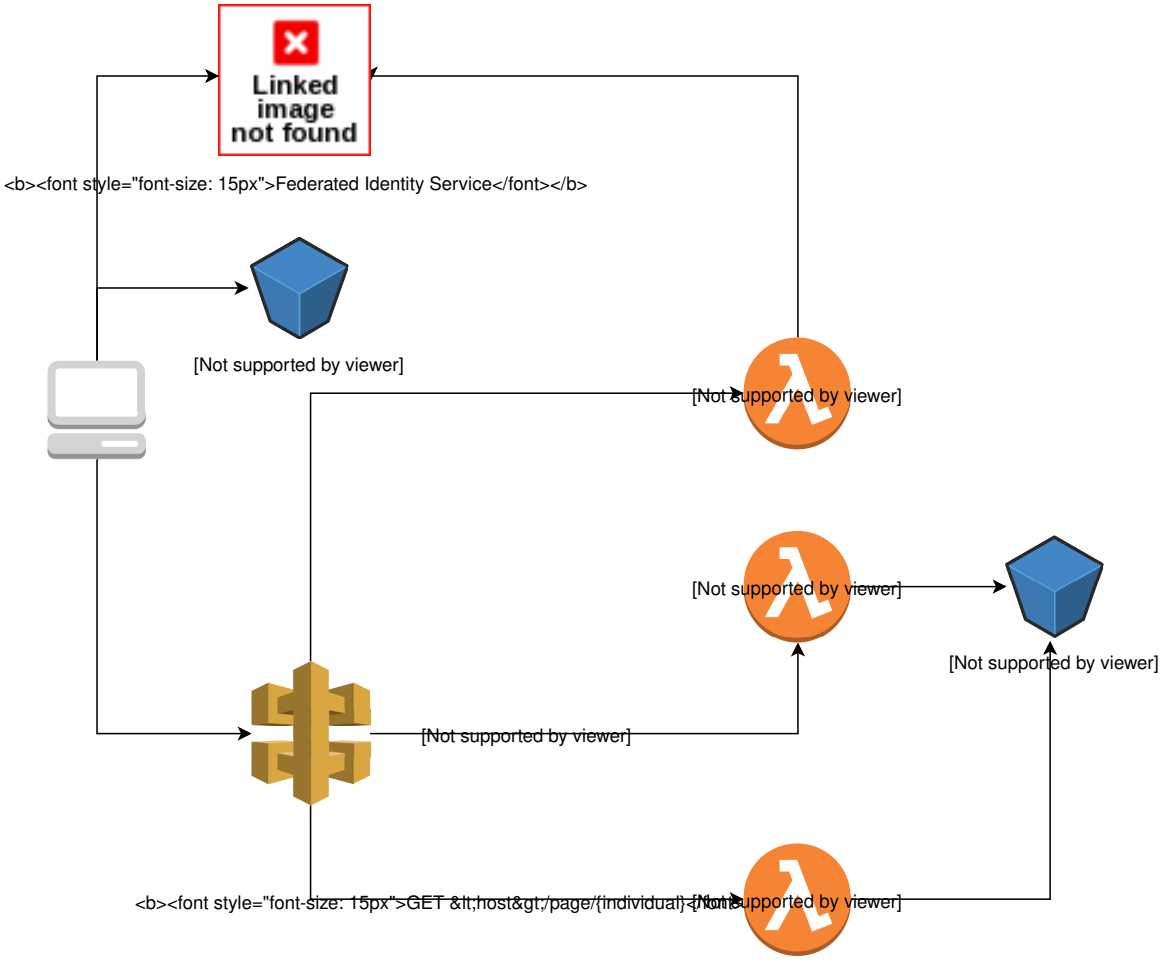
6.3 Entwurf

6.4 Umsetzung

Die grundlegende Vorgehensweise bei der Umsetzung dieses Projekts wird Analyse, Entwurf, Implementierung und Test sein. Der Forschender Charakter dieses Projekts lässt sich nicht Testgetrieben implementieren. Das

6.4.1 Komponenten Übersicht

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



legacy api proxy [Sba17] lambda<convert/invoke> fuseki-server

nice: graphQL= json matcher over multiple DBs

6.4.2 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. Im Folgendem werden die Eigenschaften von relationalen mit ontologischen Schemas verglichen.

Tabelle 6.1: Vergleich relationalem mit ontologischem 6.4.2.2 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 deren Auswahl als Datenhaltungstechnologie.

Es handelt sich daher um eine „Linked Data Driven Web Application“.Dieser Begriff gehört zum „Semantic Web“,der in der Sektion der Ontologie6.4.2.2 weiter erläutert wird.

6.4.2.1 Semantic Web

Das „Semantic Web“ist eine Erweiterung des herkömmlichen Web, in der Informationen mit eindeutigen Bedeutungen versehen werden [GOS09]. set of standards and best practices for sharing data and the semantics of that data over the Web for use by applications [Bob13].

Diese Bedeutungen werden für Maschinen durch Ontologien dargestellt wessen Spezifikation von W3C [W3C17] mit dem Nahme Web Ontology Language (OWL) beschrieben wurde.

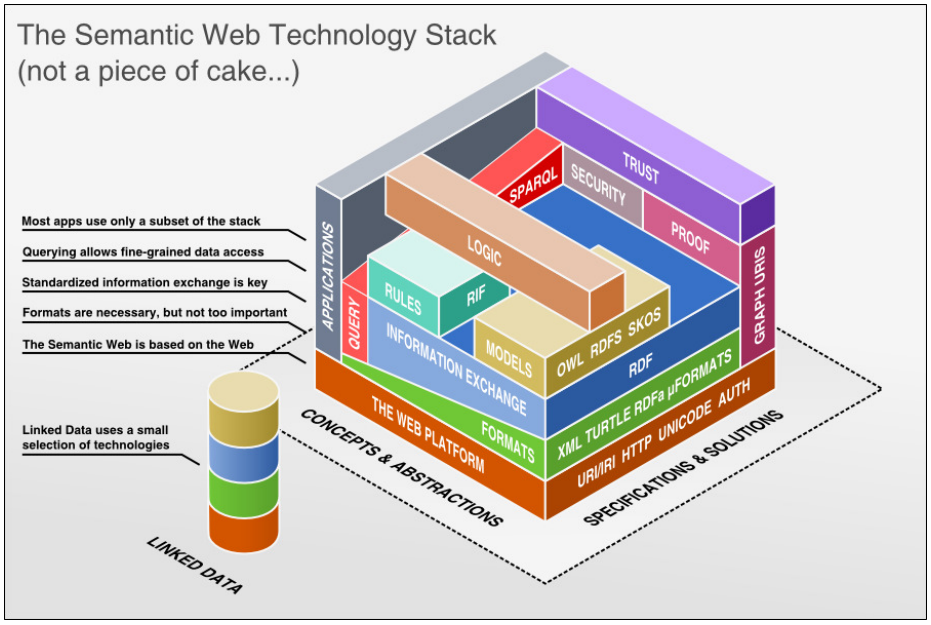


Abbildung 6.1: Überblick von benutzten Technologien

6.4.2.2 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. Deren Beziehungen können auch zu externen Ontologie-signifikanten verweisen und dadurch ein Consensus über Begrifflichkeiten.

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

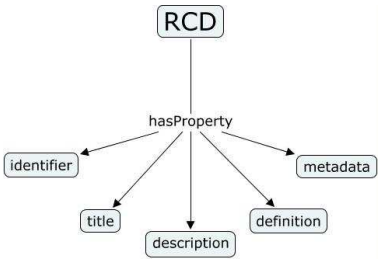
Um die Neuerfindung des Rades zu vermeiden, die Recherche ergab einen aktuell öffentlichen graphischen ontologischen Entwurf [RMG14] siehe6.2 der in Moodle mit eine Relationalen Datenbasis und PHP umgesetzt. Nach dessen Ontologien wurde mittels Watson [Wat17] und LOD [LOD17] nichts öffentlich gefunden.

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].

6.4.2.3 RDF

Die bisher erreichte Analyse des Domänsproblems 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 .
```

6.4.2.4 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 .}
```

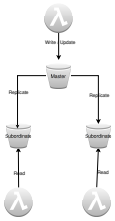
Am Beispiel von KOMA, konkatenieren die Ergebnisse mit Zwei Abfragetriples dank eine Hilfsvariable. Die folgende Abfrage ließe sich „Wähle alle Properties des Graphes und wähle alle dessen Subjekten mit Alice als Objekt“formulieren.

```
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 }
```

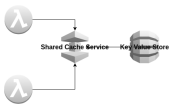
So dass auch die Benutzer von KOMA solche Abfragen stellen können wird ein „Sparql-endpoint“mit Hilfe von Apache Jena ARQ, ein Sparql-Engine, zur Verfügung gestellt. Dieser Sparql-Endpoint entspricht der Repositoryschicht der Anwendung und wird nach Anfrage von der Ontologie in S3 mittels Sparql JSON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

Um den Datenmodell möglichst simpel programmatisch abzufragen wird zunächst dessen Schnittstelle6.4.6 definiert.

6.4.3 Datenhaltung Synchronisierung und Replizieren

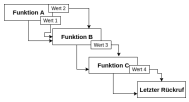


Master - Slave



cache shared

6.4.4 Datenverarbeitung



lambda

6.4.5 Abtrennung des Monoliths

Im Folgendem es wird beispielhaft eine abtrennung einer JEE Anwendung.

JEE.war „And“Test vs SRP Identifizieren der Concerns und ihre Trennung:

Einloggen Login beispiel: user + password + userData -> user + passord -> uuID -> userData refactor and share common code

Autorisierung und Authentifizierung. Einleitung

Auth0 bietet Authentifizierung as a Service an. Der Benutzer erhält einen JSON Web Token JWT und schickt ihn Encoded JSON Web Signature JWS oder JSON Web Encription zur Anwendung mit.

Einloggen: 0Auth Google gibt token, der wird in Lambda überprüft, Session in oauth.com verwaltet

Dynamo DB Funtion: Read + Write + updateS3 zu Split S3, Split R/W

Datenspecherung Architektur: DynamoDB: speichert :individual als Schlussel und seine relative URL Ś3: speichert die .owl Dateien.

Lambda Funktion: Maps zwischen S3 und DynamoDB.

6.4.6 RESTful API

Designing a friendly Hypertext Transfer Protocol (HTTP) API means abstracting the intricate business logic and data your service uses into the four basic CRUD operations (create, read, update, delete).

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

6.4.7 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.

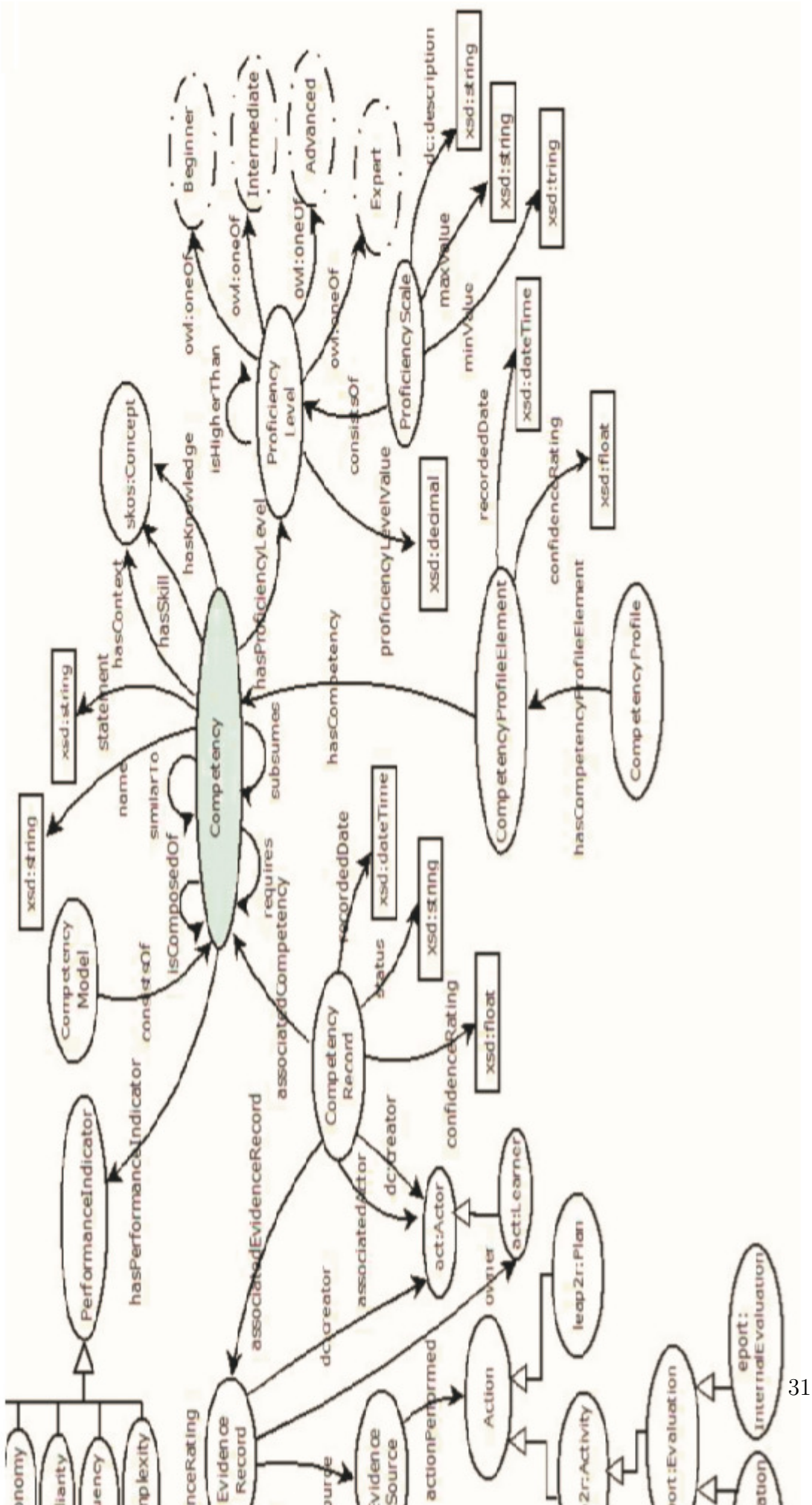
Die Web Anwendung ist für alle Rechenaufgaben verantwortlich die im Browser aus dem Sicherheitssichtpunkt kein Gefahr darstellen, um den Backend oder Servers möglichst wenig auszulasten. Deswegen bietet sich eine Single Page Application an. Die SPA besteht aus ein einziges HTML Dokument. Dies vereinfacht 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.


Competency
Competency Model
Competency Record
Competency Profile
CompetencyProfileElement
Proficiency Level
ProficiencyScale
PerformanceIndicator
Evidence Record
Evidence Source
act:Actor
act:Learner
Action
leap2r:Plan
leap2r:Activity
eport:Evaluation
eport: InternalEvaluation
skos:Concept
xsd:string
xsd:decimal
xsd:float
xsd:dateTime
Beginner
Intermediate
Advanced
Expert
name
statement
hasContext
hasSkill
hasKnowledge
isHigherThan
hasProficiencyLevel
proficiencyLevelValue
hasCompetency
subsumes
similarTo
isComposedOf
requires
recordedDate
status
associatedCompetency
associatedEvidenceRecord
associatedActor
dc:creator
confidenceRating
hasPerformanceIndicator
consistsOf
owl:oneOf
dc:description
maxValue
minValue
recordedDate
confidenceRating
hasCompetencyProfileElement
actionPerformed
owner

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 Entickler 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.

8 Ausblick

RESTful UI RESTful Anfragen für bestimmte UI formate.

Listings

Tabellenverzeichnis

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.

[Con17] ConnectWise. Connectwise, 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.

[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.

[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.

[UNL17] UNLESS, 2017.

[W3C17] W3C. W3c, 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.