

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

1	Quellen	1
2	Einleitung	1
2.1	Motivation	1
2.2	Ziel	1
2.3	Aufbau der Arbeit	2
3	Grundlagen	3
3.1	KOMA	3
3.2	Datenhaltung Analyse und Auswahl	3
3.3	Functionen	4
3.4	Patterns	4
4	Ergebnis und Auswertung	5
5	Zusammenfassung und Ausblick	7

1 Quellen

OpsWorks AWS :: Deployment Strategy Cloud Design patterns :: Profi Patterns Man
Trade Offs Arch :: Auswertung + Design guide <- self-adaptive arch?? AWS Sol. Arch
:: Best Practices AWS vs Patterns general Amazon Web Services in Action :: Best
Practices Arch Impl Cloud Design-Patterns for AWS :: Patterns list Serverless Arch
AWS :: Main :: lambda: compute as a back end

2 Einleitung

Welche Faktoren interessieren beim Anfang der technologischen Umsetzung einer Idee?
Time-To-Market Kost of Human Resources:: Skill shortage Proof of Concept

2.1 Motivation

Auf dem Weg zur technologischen Umsetzung einer neuen Idee liegen unbekannte Schwierigkeiten mit den Entscheidungen über die Architektur der Anwendung/Projekt/Umsetzung?, der 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 Domänen der Anwendung.

Auf diesen Problem wurde „FaaS“ als Lösung unter der Rubrik „Serverless“ von den Hauptanbietern von „Cloud“-Technologien vorgestellt.

2.2 Ziel

Untersuchung von architektonischen Entwurfsmustern mit ausschließlich Serverless Technologien am Beispiel von KOMA mit AWS.

Start : MVP Minimal Viable Product Start::Chars: Arch + Domänen Flexibilität - Schnelle Arch Änderungen

Umsetzung der Kernfunktionalität einer Beispielanwendung mit ausschließlich „Serverlosen“-Architekturen. wenn Zeit: Identifizieren von unverzichtbaren generischen Funktionen für Serverless Anwendungen.



2.3 Aufbau der Arbeit

3 Grundlagen

Stand der Technik: Vorgehensweise bei Traditionelle Webanwendungen: Software Architektur: "What's important". Frühe, un-/schwer- veränderbare Entscheidungen. Web Services are processes that expose their interfaces to the Web so that users can invoke them. Facilitate service discovery and meaning encoded in schemas

Design: Lambda Orchestrator -> Pool of Lambdas to use

3.1 KOMA

Beispiel Anwendung

„KOMA“ ist ein Akronym für Kompetenz-Matrix. Die Umsetzung der Anwendung soll die von einem Individuum erworbene und zu erwerbenden Fertigkeiten, Kompetenzen und deren Niveau nachvollziehen. <- ?

Das Modell von KOMA basiert auf dem Grundmodell von „European Qualifications Framework Semantics“ und dem deutschen Qualifikationsrahmen.

Ein Nutzungs-Fall aka. Use-Case: -Als Professor, will ich eine Auflistung der erworbenen Fertigkeiten einer Klassenstufe abrufen können.

-Als Professor, will ich das Kompetenzniveau einer Kompetenz und derer Fertigkeiten abrufen können.

3.2 Anforderungen Analyse

Von Browser abrufbar.

Private Datenspeicherung.

Zukünftige Erweiterungen berücksichtigen.

Konsistenz Prüfung mit Unterschiedlichen Konnotationen zwischen dargestellten Konzepten.

3.3 Datenhaltung Analyse und Auswahl

→ Tabelle Vergleich:: Relationale : NoSQL : Ontology Konsistenz Erweiterbarkeit Migration Bedeutung

Da die Konnotationen von beispielsweise Schlüsselkompetenzen von Kontext zu Kontext unterschiedlich sind, bietet sich eine Ontologische Datenspeicherung an. Extensible, Migration ok. Consistency ko.

Um die Konsistenz der Semantic des RDF Grundmodells zu bewahren wird ein Relationale schema für die Ontologie benutzt.

3.4 Funktionen

Einloggen: OAuth Google gibt token, der wird in Lambda überprüft, Session in oauth.com verwaltet Query: SparQL ?x, ?y, ?z WHERE ... Datenspeicherung Architektur: DynamoDB: speichert :individual als Schlüssel und seine relative URL S3: speichert die .owl Dateien.

Lambda Funktion: Maps zwischen S3 und DynamoDB.

3.5 Patterns

Valet Key

Static Content Hosting ok

Sharding ok

Compute Resource Consolidation

Command and Query Responsibility Segregation CQRS <- readS3UpdateDynamo.js

4 Ergebnis und Auswertung

cons: kurzlebige konfigurationen herausfinden ?? tracking? viel Konfiguration, kaum Konvention -> .json 4 everything local testing braucht event-simulation.json

Entwurfsmuster von Frameworks wie JEE durch eine Annotation generisch eingesetzt sind neu zu implementieren in der Cloud

5 Zusammenfassung und Ausblick

Beispielhaftes Bild Abbildung 5.1

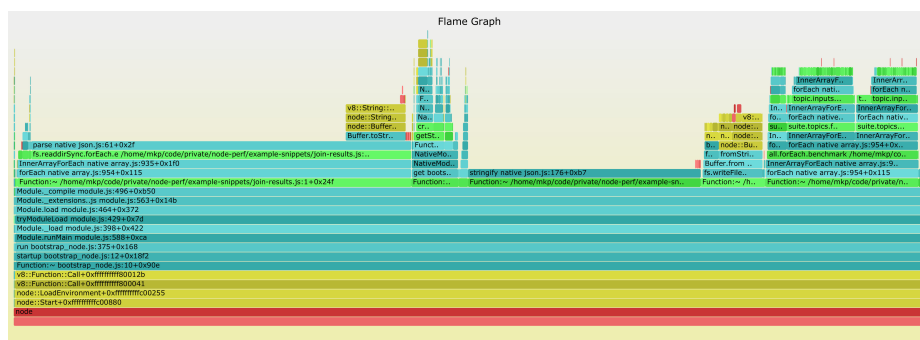


Abbildung 5.1: Beispiel Flame-Graph eines Node.js Skripts

Beispielhaftes Code-Snippet siehe Listing 5.1.

Listing 5.1: Aufnahme der „real“-Zeit

```
1 START=$(date +%s.%N)
2 node ${JS_FILE}
3 END=$(date +%s.%N)
4 DIFF=$(echo "$END - $START" | bc)
```

Hier kommt eine Bibliography-Referenz: [BME⁺07]

Listings

5.1 Aufnahme der „real“-Zeit	7
--	---

Abbildungsverzeichnis

5.1	Beispiel Flame-Graph eines Node.js Skripts	7
-----	--	---

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

- [BME⁺07] G. Booch, R. Maksimchuk, M. Engle, J. Conallen, K. Houston, and B.Y.P. D. *Object-Oriented Analysis and Design with Applications*. Pearson Education, 2007.

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, XX. Monat 2017

Vorname Nachname