fhwedel
UNIVERSITY OF APPLIED SCIENCES

Department of Computer Science

Bachelor Thesis

Vergleich eines Usecases mit Serverless Technologie gegenüber Spring Boot Technologie

am Beispiel von Instant Payments

Eingereicht am:

25. August 2021

Eingereicht von:
Silas Hoffmann
Traberweg 52
22159 Hamburg
Tel.: (040) 643 94 73
E-mail: inf103088@stud.fh-wedel.de

Referent:
Prof. Dr. Dennis Säring
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
Phone: (041 03) 80 48-43
E-mail: dennis.saering@fh-wedel.de

Betreut von:
Kai Roßdeutscher
DPS Engineering GmbH
Eiffestraße 78
20537 Hamburg
Phone: (040) 25 15 41-44
E-mail: kai.rossdeutscher@dps.de

Inhaltsverzeichnis

Abbildungsverzeichnis

List of Listings

1 Einleitung

1.1 Überblick

- Was gibt es bisher, nur kurz anreissen
- wird im Grundlagen Kapitel tiefgehender behandelt...

Google, Spotify oder Netflix: Immer mehr Unternehmen setzen heutzutage auf Virtualisierungsplattformen. Aber warum sind Technologien wie zum Beispiel Docker als Containerisierungsplattform, oder Kubernetes als Orchestrator, so beliebt? In der folgenden Thesis möchte ich hierauf eingehen. Ich werden einen Überblick über die verfügbaren Technologien geben und auf diverse Eigenarten eingehen (was heißt es serverless zu deployen etc.).

1.2 Motivation

- Problemstellung kurz beschreiben
- DPS möchte untersuchen ...
- portierung der anwendung in die cloud (amazon) Problem ist: spring boot anwendung benötigt viel startup-zeit, heute ist die maximale skalierung in produktion vorbestimmt und muss immer vorgehalten werden, alle server sind 24x7x365 aktive und werden eigentlich nur in 1-2h pro tag in den spitzenzeiten ausgelastet... also nicht nur ineffiziente nutzung von hardware und energie sondern auch von kapital. -> hyperscaler geschäftsmodell verspricht abhilfe...
- hier ruhig auch einen Satz zur neuen Sau, die jetzt durchs dorf getrieben wird „new green economy“.. ressourcen sparen um eisbären und gretha thunberg zu retten... energieffizienz... etc bla bla bla sofort-ready vollautomatische Skalierung etc..das sind dann die gganzen Versprechungen der hyperscaler zu der cloud... kannst du hier alle auflisten. Kunde möchte vorbereitet sein, um ggf. Auflagen der Regulierer zu „green“ erfüllen zu können.
- Welche Erkenntnisse sollen gewonnen werden?
- Kriterienkatalog nennen

Mit den neuen Möglichkeiten die diese neuartigen Technologien mit sich bringen, möchte sich die DPS Gmbh in Zukunft ebenfalls in diese Richtung orientieren, denn die Ansprüche an eine moderne IT-Infrastruktur ändern sich. Man möchte effizienter arbeiten. Nicht nur bezogen auf verbrauchte Energie, sondern zum Beispiel auch auf nötiges Fachpersonal zur Wartung der Artifacts in Produktion. Denn diese Technologien ermöglichen ein Abstraktionslevel, welches es dem Anwendungsentwickler ermöglicht direkt Arbeitspakete für das Deployment zu generieren. Die Grenze zwischen der Operation- und der Entwicklungsabteilung verschwimmen immer mehr und es kommt zu weniger Tradeoffs zwischen mehreren Teams. Auch ermöglicht diese Abstraktion, falls gewünscht, ein komplettes Auslagern der Infrastruktur, wie es zum Beispiel mit Cloud Technologien der Fall ist.

Diese Abstraktion ermöglicht den Entwicklern sich mehr auf das Schreiben der Businesslogik zu konzentieren und können durch das Nutzen von Standartisierten Schnittstellen ein vollautomatisiertes Deployment gewährleisten. Diese Prinzip wird je nach Abstraktionstiefe auch als *Infrastructure as Code* (IaC) beziehungsweise *Infrastructure as a Service* (IaaS) bezeichnet. Gerade IaC erlaubt den Entwicklern ihre Programmierkenntnisse auf die Konfiguration von Infrastruktur zu übertragen. Auch die DPS plant in geraumer Zukunft auf Cloud Technologie zu setzen, langfristig ist hierbei eine Portierung in die Cloud von Amazon (AWS) geplant.

Ein Fokus, welchen ich mir in dieser Thesis im Detail anschauen möchte betrifft das Startup-Verhalten einer Anwendung in genau solch einer Virtualisierten Umgebung. Denn unter den genannten Effizienzgesichtspunkten möchte man heutzutage auch nötige Resourcen vorallem auf Anfrage verwenden und nicht mehr rund um die Uhr laufen lassen, selbst wenn dies zu einem gegebenen Zeitpunkt eigentlich gar nicht nötig wäre. Gerade im Banking Bereich gibt es Zeiträume in denen ein relativ geringer Geldfluss festzustellen ist, während es zu anderen Zeitpunkten zu regelrechten Burst kommen kann, wenn zum Beispiel zu Feiertagen relativ viel Geld den Besitzer wechselt. Um dieses Prinzip der Resourcennutzung auf Anfrage etwas anschaulicher zu gestalten, werde ich eine vereinfachte Kopie einer realen Anwendung vom Unternehmen nachbauen und hinsichtlich der Startzeiten von Containern untersuchen. Der Prototyp beinhaltet mehrere verschiedene Komponenten zur Abarbeitung der Logik, um im Nachhinein auf die Performanz hinsichtlich der genannten Effiziensgesichtspunkte zu untersuchen und zwischen den verwendeten Technologien zu vergleichen.

2 Zielsetzung

2.1 Problemstellung

- Vergleich der Startup-Zeit einer containerbasierten Cloudanwendung bei steigender Last. Es soll auf Unterschiede zwischen Spring und Node eingegangen werden.
- insbesondere auf Durchsatz der Instant Payments eingehen
- da anwendung in spring boot ist nicht cloud-fähig weil zu lange für startup, dadurch riskieren wir timeouts also rejects – das ist die besonderheit der anwendung. . . - ..neuer container muss sofort verfügbar sein! Eine alternative wird gesucht: serverless vespricht minimale startup zeiten!....

2.2 Lösungsweg

- Prinzip / Ablauf erklären
- Grundlegende Struktur des Prototypen erklären
- vergleich der startupzeiten zwischen serverless und aktueller springboot variante anhand eines jeweils protypen und fiktiven workflows und fiktiven lastszenarios in einer cloud umgebung und messung des startup verhaltens bei plötzlich auftetenen spitzen oder so. . . .

[?]

3 Ist-Analyse

3.1 JBoss (Microprofile)

- aktuelle Architektur beschreiben
- Hinweis darauf geben, dass Prototyp in der Thesis vereinfacht mit Spring dargestellt wird
- jetzt ja fiktiv SpringBoot, hier rein technischer Ist-Stand, Systemarchitektur....

Die aktuelle Verarbeitung von Payments innerhalb der Anwendung läuft in Produktion auf vier Instanzen des kommerziellen Applikation-Servers „*JBoss*“. Im Development wird hierbei eine Open-Source Variante namens „*Wildfly*“ verwendet. In diesen Application Servern werden entsprechende .war Dateien deployed welche den ausführbaren Code der Anwendung beinhalten. Der Application Server bietet dem Server-Teil der Client-Server-Anwendung eine Laufzeitumgebung, in der dieser ausgeführt werden kann. JBoss bietet nun standartisierte Schnittstellen nach dem jeweiligen Java Enterprise Standard um zum Beispiel die Kommunikation mit der Außenwelt zu ermöglichen oder um der Anwendung eine Persistenzschicht zur Verfügung zu stellen. Außerdem laufen die Instanzen im *Microprofile* Modus, der es ermöglicht, bestimmte Konfigurationsparameter auszulagern. So können Applikationen auf unterschiedlichen Systemen deployed werden ohne komplett neu gebaut zu werden ([**?**]). Um eine gleichmäßige Aufteilung der Last zu gewährleisten, teilt ein so genannter „*Load Balancer*“ die eingehenden Nachrichten den entsprechenden Instanzen zu. Jede der vorhandenen Instanzen besitzt eine minimale sowie maximale Anzahl an parallel ausführbaren Prozessen. Diese Angaben werden auch „*max. / min. Poolsize*“ genannt. Eine minimale Poolsize muss gegeben sein, um sicherzustellen, dass eine gewisse Grundlast falls nötig sofort bearbeitet werden kann, daher darf diese minimale Anzahl auch nicht Null betragen. Die maximale Poolsize stellt sicher, dass es zu keiner Überlastung des Systems kommt. Wenn eine Instanz bereits mit der maximale Anzahl an Prozessen arbeitet, wird dies dem Loadbalancer signalisiert und dieser teilt der entsprechenden Komponente in diesem Zeitraum keine weiteren Nachrichten mehr zu. Um zu gewährleisten, dass die Nachrichten nicht verloren gehen, werden sie in eine „*Request Queue*“ geschrieben, welche lediglich dazu gedacht ist den Overhead abzuspeichern. Wie die Daten im Detail verarbeitet werden, ist für die weitere Betrachtung irrelevant und wird daher nicht weiter erläutert.

Simples Schaubil einfueger

3.2 Probleme

Quelle einfügen

- Probleme mit aktuellem System (Stichwort Deployment, Wartbarkeit)
- Prof. hat extra darauf hingewiesen, dass es nicht nur um die Vorteile der Cloud gehen soll
- „Erwartete Probleme in einer Cloud Umgebung“ à aktuell „..starre Hardware und Software-Skalierung...“ unerwartete lastspitzen könnten zu problemen führen, wenn sie die erwartete und verfügbare obergrenze an kapazität übersteigt und wie oben gesagt ineffiziente nutzung von kapital....nochmal mit anderen worten aus

- 1.2 à die auflistung der probleme hier, müssen dann in der zusammenfassung wieder auftauchen und abgehakt werden! Die wollen wir ja auch lösen...

Mit der monolithischen Struktur des aktuellen Systems kommt es im Laufe der Zeit zu unterschiedlichen Problemen. Über die Zeit werden Komponenten immer weiter verstrickt, sodass es in einem solchen System auch nach Personalwechsel nicht einfacher wird den Überblick über die gesamt Applikation zu behalten. Dies führt zu fehleranfälligerem Code, welcher sich wiederum negativ auf das Kapital einer Firma niederschlägt. Diese Fehler führen in einem monolithischen System zu einem großräumigen Ausfall der gesamten Applikation, da es keine klare Abgrenzung der einzelnen Komponenten gibt.

Außerdem laufen die Application Server auf Servern, die (wenn überaupt) eine sehr dünne Abstraktionsschicht bieten („*starrer Hardware*“). Es ist nicht möglich diese schnell und flexibel zu ersetzen, falls es zu Fehlern in der Produktionsphase kommen sollte. Application Server bringen einen großen Konfigurationsaufwand mit sich, der sich schlecht automatisieren lässt. Man braucht geschultes Fachpersonal um ein solches System zum Laufen zu bekommen.

Eine Skalierung ist mit monolithischen Strukturen nur dadurch möglich, den ausführbare Code auf zusätzlichen Servern zu deployen, dies wird auch *horizontale Skalierung genannt*. Jede dieser Kopien nutzt die gleiche Resourcenanzahl, was es zu einem ineffizienten Design macht, da sie sich nicht dynamisch der gegebenen Last anpassen. Tatsächlich ist ein es jedoch ein viel größeres Problem, wenn ein bereits aufgeteiltes System an seine Kapazitätsgrenze stößt, denn wie bereits erwähnt, besitzten Java-Anwendungen eine relativ lange Initialisierungsphase. Um den heutigen Anforderungen der dynamischen Skalierung gerecht zu werden, werden Resourcen auf Abruf gebraucht. Die Systeme sollen so schnell wie möglich verfügbar sein.

4 Vorgehensmodell

Nachdem im letzten Kapitel der aktuelle Aufbau der Payment-Anwendungen, sowie die durch die monolithische Struktur hervorgebrachten Probleme, erläutert wurden, werde ich im Folgenden beschreiben wie ich vorgehen werde um den Problemfokus der Startzeit zu untersuchen.

4.1 Anforderungen an Daten zur Messung des Startup-Verhaltens von Containern

- Problem reduzieren
- Messkriterien festlegen
- Quellen finden
- hier nur theoretisch schwafeln über einen kriterienkatalog und die möglichen „Messwerte“, die man ggf. braucht -> in Kap. 5 dann den Kriterienkatalog konkret aufstellen

4.2 Anforderungen an Prototypen

- einmal als spring boot und einer weiteren variante mit einer cloud-native technologie -> hier wurde vorgegeben mit serverless zu arbeiten.

4.2.1 Festlegung fiktiver Workflow

Da der Fokus auf der Untersuchung der Startupzeit der Komponenten liegt, wird lediglich eine minimale beispielhafte Implementierung erfolgen, welche die Arbeitsschritte der eigentlichen Applikation in vereinfacht darstellen soll. Allerdings werden im System konkrete Nachrichten im XML Format vermittelt, welche einer XSD-Spezifikation folgen wie sie im realen Umfeld ebenfalls genutzt wird.

Auszug aus XM einbin- den???

Sobald eine neue Nachricht eingetroffen ist, soll drei Arbeitsschritte ausgeführt werden:

1. Es soll geprüft werden, ob das eingegangene XML der XSD-Spezifikation folgt oder nicht. Wenn dies nicht der Fall sein sollte, wird die Nachricht zwar derartig acknowledged, dass sie zwar aus der Eingangsqueue im Message Broker enfernt wird, allerdings bei der Verarbeitung ignoriert wird.
2. Falls es sich um valides XML handelt, wird ein Feld aus dem XML-Inhalt ausgelesen.
3. in einem letzten Schritt wird dieses Element in eine Datenbank geschrieben damit auch eine Persistenz-Operation in die Verarbeitungszeit einfließt.

4.2.2 Serverless

- 4.2.2 spring boot 4.2.3 serverless à so bekommt der prof schon beim draufgucken auf das inhaltsverzeichnis die story mit....

4.3 Anfoderungen an Containerisierungsplattform

4.4 Anforderungen an Lasttest

4.5 Anforderungen Visualiserung und Monitoring zur Unterstützung der Auswertung

5 Problemlösung

5.1 Bestimmung von Daten zur Messung des Startup-Verhaltens von Containern

5.1.1 Kriterienkatalog

- Vorallem Skalierbarkeit und Performance
- Transaktionen - Durchsatz wichtig
- Deployment egal, da bereits im Container deployed wird
- Generell sagen, warum einige Aspekte egal sind
- Wartbarkeit nicht so wichtig
- Ressourcennutzung
- Störungsfälle (Chaos Monkey)
- Non Functionals - Kai hat PDF geschickt

5.2 Implementierung Prototyp

5.2.1 Node.js

5.2.2 Spring Boot

5.3 Implementierung mittels Containerisierungsplattform

5.3.1 Container Lifecycle

- Auf verschiedene Schichten eingehen
- Auf Ergebnisse beziehen

5.3.2 Docker Swarm

- Prototypen im Detail erlaeutern

```
cat .env

# qbn: queue bound (level) n
# cbn: container bound (level) n

QB0=15
QB1=30
QB2=100
CB0=1
CB1=5
CB2=10
CB3=30
```

QL3 $QB2 < \overline{MC}$	UP *abs(CB0 – CB3)*	UP *abs(CB1 – CB3)*	UP *abs(CB2 – CB3)*	OK _
QL2 $QB1 < \overline{MC} \leq QB2$	UP *abs(CB0 – CB2)*	UP *abs(CB1 – CB2)*	OK _	DOWN *abs(CB2 – CB3)*
QL1 $QB0 < \overline{MC} \leq QB1$	UP *abs(CB0 – CB1)*	OK _	DOWN *abs(CB1 – CB2)*	DOWN *abs(CB1 – CB3)*
QL1 $QB0 < \overline{MC} \leq QB1$	UP *abs(CB0 – CB1)*	OK _	DOWN *abs(CB1 – CB2)*	DOWN *abs(CB1 – CB3)*
QL0 $QB0 == \overline{MC}$	OK _	DOWN *abs(CB0 – CB1)*	DOWN *abs(CB0 – CB2)*	DOWN *abs(CB0 – CB3)*
	CL0 $CB0 == \overline{MC}$	CL1 $CB0 < \overline{MC} \leq CB1$	CL1 $CB1 < \overline{MC} \leq CB2$	CL1 $CB2 < \overline{MC} \leq CB3$

5.4 Implementierung Lasttest

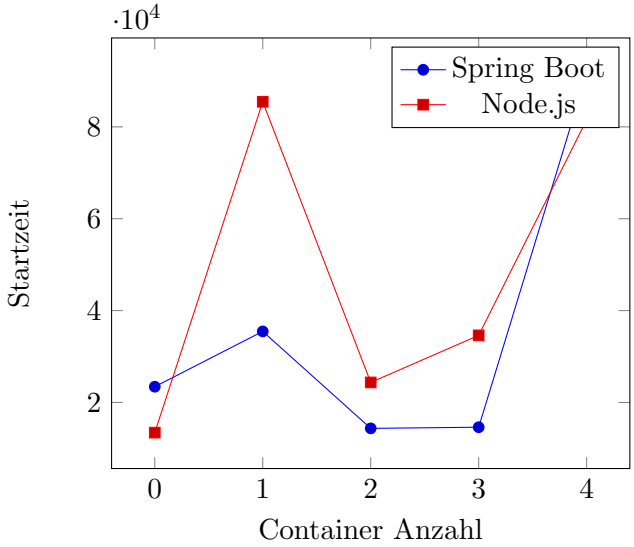
5.4.1 Timeline

5.4.2 Testbedingungen

- Kommt in den Anhang
- hat Prof. zwar als eigenes Kapitel erwaehnt, bin mir aber nicht sicher ob das wirklich noetig ist
- auf welcher Hardware werden Tests durchgefuehrt?
- chaos monkey / Stoerfaelle erlaeutern

Tabelle 5.1: Server Specs

Prozessor	Intel(R) Xeon(R) Gold 6226R CPU @ 2.90GHz
Kerne	6 Prozessoren á 16 Kerne
RAM	16 GB
Storage	150 GB



5.5 Implementierung Visualierung und Monitoring zur Unterstützung der Auswertung

```
1  // standard constructor
2  public Bank(int playerCnt) {
3      this.entries = new Entry[playerCnt];
4      this.bankSize = playerCnt;
5          this.rand = new Random();
6  }
7
8  // testing constructor - no fileIO
9  public Bank(Entry[] entries, Random pseudoRandom) {
10     this.entries = entries;
11     this.rand = pseudoRandom;
12     this.bankSize = entries.length;
13 }
14
15 // testing constructor - with fileIO
16 public Bank(String preallocation, List<Player> players, Random rand) {
17     assert null != preallocation && null != players && null != rand;
18     this.bankSize = players.size();
19     this.entries = new Entry[this.bankSize];
20     if (0 < preallocation.length()) {
21         String[] singleEntries = preallocation.split(SEPERATOR_STRING_REPRESENTATION);
22         int offset = this.bankSize - singleEntries.length;
23         for (int i = singleEntries.length - 1; i >= 0; i--) {
24             this.entries[i + offset] = new Entry(singleEntries[i], players);
```

```
25          }
26      }
27  }
```

```
1  #!/bin/bash
2
3  # Add two numeric value
4  ((sum=25+35))
5
6  #Print the result
7  echo $sum
```

6 Ergebnisanalyse

6.1 Ergebnisse

- Vorstellung der erhaltenen Daten
- Interpretation / Analyse allerdings Teil vom naechsten Kapitel?

6.2 Analyse

- Interpretation / Analyse der Daten
- Begruendung fuer Verhalten suchen

6.3 Diskussion

6.3.1 Begründung Startupzeit

- Warum Node.js schneller ist
- Erlaeutern warum die erhaltenen Ergebnisse in einem real-life Szenario vielleicht nicht aussagekraeftig sein koennten

7

Zusammenfassung

8 Ausblick

9

Literaturverzeichnis

10 Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Ort, Datum Silas Hoffmann