hochschule mannheim

Fakultät für Informatik
Hochschule Mannheim

Studienarbeit

Virtualisierte Arbeitsumgebung für den Test verteilter Systeme

Microservice-Architektur mit Docker, Kafka, Clojure & Elm

Jan-Philipp Willem

im Sommersemster 2017

Zusammenfassung

Ein verteiltes Softwaresystem zu entwickeln und ebenso dessen Auswirkungen zu verstehen, ist kein Leichtes für einen durchschnittlichen Informatik-Studenten. Oft ist es der Fall, dass es schon anspruchsvoll genug ist, alleine die Grundlagen umzusetzen, welche es bei einer Aufgabe anzuwenden vermag. Um diese Hürde etwas zu erleichtern, soll eine Umgebung geschaffen werden, welche es ermöglicht, eine verteilte Aufgabe unabhängig des eigenen Computers zu testen. Eine Programmieraufgabe soll von einem Dozenten als ein Experiment definiert werden können, welches aus einer gegebene Anzahl an Instanzen eines ebenso definierten Servers-Systems besteht. Weiterhin soll das Experiment von einem Studenten durchgeführt werden können, welches das Hochladen der eigenen Lösung und das anschließende Analysieren der darin resultierenden Ausgaben der voneinander getrennten Rechner-Instanzen darstellt.

Abstract

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Problemstellung

Diese Arbeit unterteilt sich in vier Kapitel. Das erste Kapitel soll eine Einleitung in die Problematik geben, welche erforscht werden soll. Als nächstes folgen Grundlagen, welche die Techniken und Technologien des VAR-Tool's erklären. Im dritten Kapitel wird der Planungsprozess und mit dessen Hilfe der praktische Teil der Studienarbeit erläutert. Weiterhin folgt ein Fazit der geleisteten Vorgänge und ein Ausblick, wie das Projekt weitergeführt werden könnte.

1.1 Bisheriger Ablauf

Der typische Ablauf bei Programmieraufgaben in der Vorlesung Verteilte Architekturen (VAR) ist folgendermaßen: Ein Student soll verteilte Aufgaben programmieren, welche es nötig machen, zunächst eine passende lokale Entwicklungsumgebung zu installieren. Dazu wird die Nutzung von schwergewichtigen Integrated Development Environments (IDE's) wie Eclipse oder Netbeans vorgeschlagen. Grund dafür sind die schon vorhandenen IDE-Integrationen von diversen Serven und Technologien, welche eingesetzt werden sollen. Dennoch führt schon dieser Prozess bei einigen Studenten zu Problemen.

Gelingt es die Integrationen einzurichten, so kann die Funktionsweise der eigenen Programme getestet werden. Allerdings führt diese Herangehensweise zu keiner echten Verteilung, da alle Services auf der gleichen Maschine ausgeführt werden. Das Ergebnis kann sich bei einer Trennung der Services auf Netzwerkebene erheblich unterscheiden oder ist unter Umständen gar nicht lauffähig.

[TODO: Beispiel einer Aufgabe z.B. RMI-Chat]

1.2 Anforderungen

Ziel soll es sein, dass ein Dozent Experimente definieren kann, welche eine gegebene Anzahl an Instanzen besitzen. Eine jeweilige Instanz wird durch die Angabe eines Namens, dessen geöffneten Netzwerk-Ports und eine Argumentenliste beschrieben. Die Konfiguration der Systeme soll mit Hilfe eines Dockerfiles realisiert werden. Die Instanzen sollen sich gegenseitig erreichen können, jedoch soll der Zugriff auf Instanzen anderer Nutzer und dem Internet unterbunden werden.

Weiterhin sollen von einem Studenten gewisse Aufgaben innerhalb dieser Experimente gelöst werden. Dies geschieht durch einzelnes Hochladen der Programmpakete (Jar, War,..) pro Instanz und der Angabe einer Main-Class und einer Argumentenliste. Anschließend kann eine Instanz gestartet werden und dessen Ausgaben beobachtet werden.

Der Grund für das getrennte Hochladen ist darin begründet, dass die echte Verteilung der Instanzen so von den Studenten eventuell besser verstanden werden kann.

Kapitel 2

Grundlagen

Dieses Kapitel möchte einige Grundlagen definieren, welche im Kapitel 3 vorausgesetzt werden. Es handelt von der Definition des funktionalen Programmier-Paradigma, welches in den Sprachen Clojure und Elm zum Einsatz kommt. Weiterhin werden die Webtechnologien Websockets und Single Page Applications (SPA's) erklärt, welche das Frontend des VAR-Tool's nutzt. Die Architektur der ganzen Applikation wird maßgebend durch die Trennung in unabhängige Microservices beeinflusst, welche mit Docker realisiert sind. Als Integrationspunkt fungiert Apache Kafka, welches ebenso in diesen Kapitel erklärt wird.

2.1 Funktionale Programmierung

Die funktionale Programmierung gehört zu den Deklarativen Programmierparadigmen und stellt neben der imperativen und objektorientierten Programmierung eines der am Häufigsten genutzten Paradigmen dar.

Anders als bei imperativen Sprachen, unterscheiden sich funktionale Sprachen vor Allem an ihrer Unveränderbarkeit (engl. Immutability) von Variablen und ihrer Datenstrukturen. Es wird dabei angestrebt, bei einer Veränderung des Zustandes einer Variablen nicht den Inhalt ihrer Referenz zu verändern, sondern stattdessen eine Kopie der Daten zurückzuliefern. In der Praxis(TODO: quote) hat sich vielfach herausgestellt, dass sich die häufig verwendete Veränderbarkeit (engl. Mutability) von Zuständen gerade auch in der objektorientierten Programmierung zu häufigen Fehlerquellen führen kann. Es wurde jedoch gezeigt, dass das Zurückgeben von Kopien ebenso sehr performant funktionieren kann (TODO: add quote).

Wie der Name schon vermuten lässt basiert die funktionale Programmierung zu größten Teilen aus Funktionen. Diese sind meist als Module gekapselt, welche nach einer bestimmten Aufgabe gruppiert sind. Anders als bei Objekten möchte man offen mit den an die Funktionen übergebenen Daten umgehen und sie nicht in der Implementierung verstecken (engl. Information-Hiding). Dabei sollte eine Funktion nur diese eine Aufgabe erledigen, welche mithilfe ihres Namens beschrieben wird. Um dieses deterministische Verhalten zu erreichen, werden oft sehr granulare Funktionen erstellt, welche anschließend durch eine Komposition miteinander vereint werden[1]. Jegliche Seiteneffekte sollten ver-

[1] `f(x) = g(x) • h(x) <=> f(x) = h(g(x))`

mieden werden, was man auch als reine (engl. pure) Funktionen bezeichnet. So sollten Funktionen als Daten-Transformationen verstanden werden: Man erhält Daten, verarbeitet diese und gibt sie anschließend transformiert zurück, welches als Eingabe-Verarbeitung-Ausgabe (EVA)-Prinzip bekannt ist.

Da Funktionen selbst auch Typen darstellen, können sie ebenso als Parameter anderer Funktionen dienen, welche man dann auch als Funktionen Höherer Ordnung (engl. Higher Order Funktions) bezeichnet. Mit dieser Voraussetzung können Verhalten aus einer aufzurufenden Funktion heraus gelöst werden, womit das jeweilige Verhalten dynamisch bestimmt werden kann. Dies wird oft auch als Callbacks bezeichnet. Hilfreich sind bei dieser Vorgehensweise auch so genannte Lamdas, welche Funktionen darstellen, die inline definiert werden können und somit anonym beziehungsweise (bzw.) Unbenannt sind.

Grundsätzlich werden funktionale Sprachen nochmals zwischen solchen Sprachen unterschieden, welche neben anderen Paradigmen auch das Funktionale Paradigma unterstützen und solchen, welche ausschließlich auf den Prinzipien der funktionalen Programmierung beruhen. Die sogenannten reinen funktionalen Sprachen verbieten beispielsweise (bspw.) eine jegliche Möglichkeit Seiteneffekte auszuführen und sind sehr oft strikt typisiert. In Multiparadigmensprachen beruhen viele funktionale Prinzipien auf deren Einsatz und der generellen Bedachtheit des jeweiligen Programmierers bzw. des einsetzenden Teams.

2.2 Verwendete Programmiersprachen

Das Projekt verwendet für den Server die Sprache Clojure, wohingegen das Frontend mithilfe von Elm umgesetzt ist.

2.2.1 Clojure

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.2.2 Elm

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.3 Eingesetzte Webtechnologien

2.3.1 Websockets

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.3.2 Single Page Applications

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

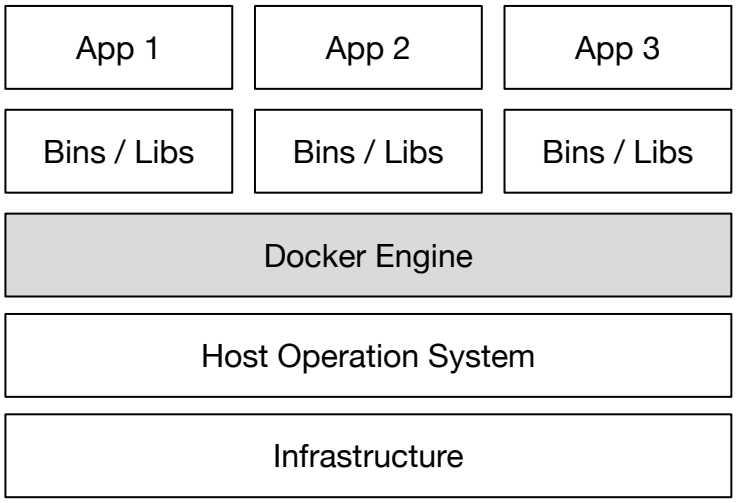

Abbildung 2.1: Schichten einer Virtualisierung mit Docker

2.4 Microservices

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2.5 Container-Virtualisierung mit Docker

Docker[2] ist eine Software zum Deployment von Applikationen innerhalb von Containern. Im Vergleich zu Virtuelle Maschinen (VM's), ähnelt sich die Vorgehensweise, jedoch laufen die Prozesse der Container direkt auf dem Host-Betriebssystem. Trotz dieser Tatsache sind die Prozesse mithilfe von unter Anderem (u.A.) Control-Groups und Kernel Namespaces voneinander isoliert. Weiterhin hat man die Möglichkeit mit gängigen Mandatory-Access-Control-Frameworks wie SE-Linux oder App-Armor die Rechte innerhalb der Container zu beschränken. Als Anforderung besteht keine spezifische Hardware-Infrastruktur wie bei beispielsweise VMWare ESXi. Ebenso ist Docker mittlerweile auf allen gängigen Betriebssystemen lauffähig, wobei Linux-Derivate die beste Unterstützung erhalten. Dies ist damit begründet, dass die Docker-Engine (Abbildung (Abb.) 2.1) auf den einzelnen Betriebssystemen verschieden aussieht.

Ein Container wird mithilfe eines Dockerfiles beschrieben. Darin werden

[2] `https://docs.docker.com/`

deskriptive Instruktionen definiert um Abhängigkeiten zu installieren, Konfigurationen vorzunehmen und andere Build-Steps auszuführen. Ein spezifischer Container wird als Image bezeichnet und setzt sich aus granularen Sub-Images zusammen. Beim Erzeugen von Images eigener Dockerfiles müssen im Vergleich zu VM's keine großen Dateien transferiert werden, da sie aus ihrem Rezept reproduzierbar sind. Es besteht auch die Möglichkeit, von anderen Dockerfiles zu erben und diese über das Netzwerk verteilt bereitzustellen. Falls die über eine Docker-Registry angebotenen Schichten eines Containers noch nicht auf dem eigenen Rechner vorhanden sind, so werden diese heruntergeladen.

Weiterhin gibt es einen entscheidenden Vorteil gegenüber einer traditionellen VM: Ein Dockerfile trägt implizit zur Dokumentation bei, da jede Änderung an einem Container in seinem Rezept ergänzt werden muss, um diese bei einem erneuten Start bzw. erneuten Build zu erhalten.

Wird eine Applikation in einzelnen Services aufgeteilt, so bietet sich das Tool **Docker-Compose**[3] an. Es ermöglicht in einer einzelnen Datei (`docker-compose.yml`) alle Konfigurations-Parameter der einzelnen Containern zu definieren. Das können bspw. Volume-Mounts, Ports, Environment-Variables oder Netzwerke sein, welche sonst bei jedem `docker exec` Command angegeben werden müssten. Alternativ dazu stünden eigene und oft fehlerintensive Shell-Scripts um ähnliches zu erreichen.

Es bietet sich auch an, getrennte Environments für Development, Test und Production als zentrale Dokumentation der Applikation zu nutzen (`docker-compose.[env].yml`).

2.6 Apache Kafka

Zwei[4]

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

[3] `https://docs.docker.com/compose/overview/`
[4] `https://kafka.apache.org/documentation/`

Kapitel 3

VAR-Tool

3.1 UI-Mockup

3.2 Architektur

3.2.1 Geschäftsprozesse

3.2.2 Deployment

3.3 Umsetzung

3.3.1 Backend

3.3.2 Frontend

3.4 Installation

Kapitel 4

Fazit

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Quellenverzeichnis

Abkürzungsverzeichnis

Abb. Abbildung

bspw. beispielsweise

bzw. beziehungsweise

EVA Eingabe-Verarbeitung-Ausgabe

VAR Verteilte Architekturen

VM Virtuelle Maschine

IDE Integrated Development Environment

SPA Single Page Application

u.A. unter Anderem