



**Hochschule  
Bonn-Rhein-Sieg**  
*University of Applied Sciences*

**Fachbereich Informatik**  
*Department of Computer Science*

# Exposé

im Bachelor-Studiengang Computer Science

## Virtualisierung von Testumgebungen zur parallelen Testausführung

von

**Per Bernhardt**

Betreuer: Prof. Dr. Rudolf Berrendorf  
Zweitbetreuer: Prof. Dr. Andreas P. Priesnitz  
Eingereicht am: 28. Oktober 2014

## **1 Inhalt der Arbeit**

### **1.1 Problemstellung**

Diese Bachelorarbeit entsteht in Zusammenarbeit mit der Pixelhouse GmbH. Diese betreibt seit ca. 10 Jahren die Webseite Chefkoch.de, Europas größtes Kochportal. Neben der Vermarktung der Webseite und der Betreuung der Community wird von den Mitarbeitern vor allem auch die softwaretechnische Entwicklung der Plattform und das Hosting durchgeführt.

Die Webseite Chefkoch.de basiert auf einer seit Beginn des Projektes stetig wachsenden PHP-Anwendung. Große Teile der Anwendung sind dabei ohne fundierte Kenntnisse in der Softwareentwicklung entstanden. Dies merkt man vor allem an einer fehlenden Systemarchitektur und einer unübersichtlichen und sehr redundanten Struktur der Daten.

Auch die auf Linux/Ubuntu-Systemen basierende Infrastruktur der Chefkoch-Anwendung ist mit den Jahren sehr komplex geworden ist, vor allem auch, um die hohe Last von mehreren Millionen Besuchern täglich bedienen zu können. So werden Loadbalancer, mehrere Application-Server und diverse Persistenz-Dienste wie SQL-Datenbanken und Key-Value-Stores eingesetzt.

Der Wartungsaufwand und auch die benötigte Zeit für Neuentwicklungen sind entsprechend hoch. Die Herausforderung für das aktuelle Entwicklungsteam besteht darin, neben dem Alltagsgeschäft und der Umsetzung neuer Produktideen auch grundlegende Modernisierungen am System vorzunehmen.

Hierbei ist es Vorgabe des Managements, bestehende und neue Funktionen über automatisierte Tests abzusichern. Gerade bei bestehenden Funktionen kommen hierbei oft nur Systemtests in Frage, da die zugrundeliegende Software wenig modular aufgebaut ist und so kaum Unit- oder Integrationstests erlaubt.

Diese Systemtests werden mit Hilfe des Testtools Behat implementiert, das die Webdriver-Schnittstelle dazu verwendet, echte Webbrowsern über die Seite laufen zu lassen, um die benötigten Funktionen abzutesten.

Das größte Problem mit dieser Art von Systemtest ist, dass sie vergleichsweise langsam sind und bereits heute mehrere Stunden laufen. Dies bedeutet, dass man nach fertiger Implementierung einer neuen Funktion oder der Modernisierung einer bestehenden Komponente sehr lange auf das Testergebnis warten muss und so Zeit vergeht, bis sich die Änderung in den Hauptentwicklungszweig integrieren oder sogar in Produktion nehmen lässt.

Eine mögliche Lösung hierfür wäre es, mehrere Testumgebungen anzubieten, um das Ausführen der automatischen Systemtests parallelisieren zu können oder auch einfach manuelle Abnahmen durch das Produktmanagement zu unterstützen.

Stand heute ist es allerdings sehr aufwändig, neue Testumgebungen aufzusetzen. Der entsprechende Prozess kann mitunter mehrere Tage dauern, da Softwareentwicklung und Betrieb hier wenig effizient zusammenarbeiten und viele Teilschritte manuell erfolgen.

Die Testumgebungen sind außerdem meist wenig isoliert und laufen z.B. auf den gleichen Datenbankservern oder hinter dem gleichen Loadbalancer. So lassen sich Änderungen an diesen Infrastruktur-Komponenten heute mitunter gar nicht testen.

### **1.2 Fragestellung**

Virtualisierungstechniken im Allgemeinen erlauben die einfache Replikation von Systemumgebungen, sind also ein möglicher Kandidat, um solche Problemstellungen auf technischer Ebene zu lösen. Dabei gibt es unterschiedliche Ansätze der Virtualisierung, die sich in ihrer Funktionalität, aber auch in ihrer Leistung und ihrem Verwaltungsaufwand unterscheiden. Hier wäre also die Frage, ob durch Anwendung solcher Virtualisierungstechniken

die Gesamttestzeit für dieses Problemszenario reduziert werden kann. Ebenso sind damit verbundene Fragen, ob es (prinzipielle) Unterschiede in Eignung, Aufwand und anfallender Testzeit zwischen den verschiedenen Techniken in Hinsicht auf die Problemstellung gibt. Andere Lösungen, wie z.B. Änderungen an der Organisationsstruktur, dem Entwicklungsprozess, dem eigentlichen Programmcode oder der Infrastruktur selbst sind nicht Teil der Fragestellung.

### **1.3 Zielsetzung**

Ziel dieser Bachelorarbeit ist es zu untersuchen, ob man mit Hilfe von Virtualisierungstechniken eine Lösung für das oben beschriebene Problem finden kann. Mit dieser Lösung soll nicht nur die Anwendung sondern auch deren Infrastruktur in einem konkreten Zustand nachgehalten und effizient aufgesetzt werden können, um so einfach und beliebig oft Testumgebungen anbieten zu können. Neben der konzeptionellen Arbeit soll auch eine prototypische Umsetzung und nachfolgende Bewertung erfolgen.

### **1.4 Theoriebezug**

Der hohe Zeitaufwand für das Aufsetzen neuer Testumgebungen und das Durchführen der Systemtests widersprechen schon lange etablierten Softwareentwicklungsmethodiken wie Continuous Integration und Continuous Delivery. Diese zielen darauf ab, kurze Feedbackschleifen für Entwickler und Produktmanager zu ermöglichen, in dem sich neue Versionen der Software kurzfristig mit denen anderer Entwickler integriert lassen und ebenso kurzfristig in Produktion genommen werden können, um so auch Feedback von echten Benutzern zu erhalten.

Die Tatsache, dass sich bestimmte Infrastrukturkomponenten gar nicht testen lassen und dass die Zusammenarbeit zwischen Entwicklern und Betrieb hier wenig effizient erfolgt, werden von der Wissenschaft heute bereits mit der DevOps Bewegung beantwortet.

Wie konkrete Virtualisierungsansätze, z.B. die Verwendung von Containern, hier effizient Abhilfe schaffen kann, ist dabei allerdings weniger umfangreich beleuchtet.

### **1.5 Vorgehensweise**

Diese Bachelorarbeit soll einen Überblick über die für diese Problemstellung relevanten Virtualisierungsansätze geben und diese in Hinsicht auf die Problemstellung bewerten. Basierend auf zwei ausgesuchten Ansätzen soll jeweils ein Konzept zur Realisierung für die konkrete Problemstellung angegeben und prototypisch umgesetzt werden. Nachfolgend sollen die Implementierungen evaluiert werden.

### **1.6 Evaluationsstrategie**

Im Rahmen der Arbeit sollen Bewertungskriterien definiert und die gefundenen Lösungen dahin gehend evaluiert werden. Ein wichtiges Kriterium ist auf jeden Fall die Reduzierung der Gesamtausführungsdauer der Tests aber z.B. auch die einfache Verwendbarkeit der Lösung. Andere Kriterien wie z.B. monetäre Kosten oder die Virtualisierbarkeit anderer Umgebungen (Windows, MacOS) fallen hingegen weniger ins Gewicht.

## 2 Gliederung

- Inhaltsverzeichnis
- Abbildungsverzeichnis
- Abkürzungsverzeichnis
- Einleitung
  - Problemstellung
  - Motivierende Softwareentwicklungsmethodiken
  - Ziel
  - Vorgehensweise
- Grundlagen und Ansätze der Virtualisierung
  - Bare Metal Hypervisor: KVM
  - Hosted Hypervisor: VirtualBox
  - Container: Docker
  - Alternativen: MirageOS
- Konzeption zweier konkreter Lösungen
  - VirtualBox
  - Docker
- Implementierung
  - Probleme während der Implementierung
  - Besonderheiten
  - Skizzierung der fertigen Lösungen
- Evaluation
  - Methodik der Evaluation
  - Definition der Evaluationskriterien
- Fazit
- Literaturverzeichnis
- Anhang

### 3 Literaturverzeichnis

#### **Bergmann und Priebisch 2013**

BERGMANN, Sebastian ; PRIEBISCH, Stefan: *Softwarequalität in PHP-Projekten*. Hanser, 2013. – ISBN 9783446435391

#### **Boettiger 2014**

BOETTIGER, Carl: *An introduction to Docker for reproducible research, with examples from the R environment*. 2014

#### **Chamberlain und Schommer 2014**

CHAMBERLAIN, Ryan ; SCHOMMER, Jennifer: *Using Docker to support reproducible research*. 2014

#### **Docker 2014**

DOCKER, Inc.: *About Docker*. <https://docs.docker.com/>. Version: 2014. – [Online; Stand 11. Oktober 2014]

#### **Duvall et al. 2007**

DUVALL, Paul M. ; MATYAS, Steve ; GLOVER, Andrew: *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007. – ISBN 9780321336385

#### **Felter et al. 2014**

FELTER, Wes ; FERREIRA, Alexandre ; RAJAMONY, Ram ; RUBIO, Juan: *An Updated Performance Comparison of Virtual Machines and Linux Containers*. IBM, 2014

#### **Fink 2014**

FINK, John: Docker: a Software as a Service, Operating System-Level Virtualization Framework. In: *code4lib Journal* (2014). – ISSN 1940–5758

#### **Humble und Farley 2010**

HUMBLE, Jez ; FARLEY, David: *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*. Addison-Wesley, 2010. – ISBN 0321601912

#### **KVM 2014**

KVM: *Kernel Based Virtual Machine*. <http://www.linux-kvm.org/>. Version: 2014. – [Online; Stand 11. Oktober 2014]

#### **Oracle 2014**

ORACLE: *VirtualBox Documentation*. <https://www.virtualbox.org/wiki/Documentation>. Version: 2014. – [Online; Stand 11. Oktober 2014]

#### **OS 2014**

OS, Mirage: *Mirage OS - Documentation*. <http://www.openmirage.org/docs/>. Version: 2014. – [Online; Stand 11. Oktober 2014]

#### **Scheepers 2014**

SCHEEPERS, Mathijs J.: *Virtualization and Containerization of Application Infrastructure: A Comparison*. 2014

#### **Soltesz et al. 2007**

SOLTESZ, Stephen ; PÖTZ, Herbert ; FIUCZYNSKI, Marc E. ; BAVIER, Andy ; PETERSON, Larry: *Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors*. 2007

**Squires 2014**

SQUIRES, Rob: *Parallel testing with Behat, Docker and Gearman*. <http://robsquires/parallel-testing-with-behat-docker-and-gearman/>. Version: 2014. – [Online; Stand 11. Oktober 2014]

**Strauss 2013**

STRAUSS, David: Containers - Not Virtual Machines - Are the Future Cloud. In: *Linux Journal* (2013)











**Swartout 2012**

SWARTOUT, Paul: *Continuous delivery and DevOps: A Quickstart Guide*. Packt Publishing, 2012. – ISBN 9781849693691

**Wikipedia 2014**

WIKIPEDIA: *Softwaretest* - *Wikipedia*. <http://de.wikipedia.org/wiki/Softwaretest>. Version: 2014. – [Online; Stand 11. Oktober 2014]

#### 4 Zeitplan

		1. Monat				2. Monat				3. Monat			
		1. Woche	2. Woche	3. Woche	4. Woche	1. Woche	2. Woche	3. Woche	4. Woche	1. Woche	2. Woche	3. Woche	4. Woche
Recherche													
Exposé													
Bearbeitungszeit													
Anmeldung													
Ausarbeitung Grundlagen / Virtualisierung													
Ausarbeitung Konzept													
Implementierung und Ausformulierung und Besonderheiten / Skizze													
Ausarbeitung Evaluation													
Ausarbeitung Einleitung und Fazit													
Puffer													
Abgabe												