



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Lepel

**Automatisierte Erstellung und Provisionierung von ad hoc
Linuxumgebungen -
Prototyp einer zentralisierten Weboberfläche zur vereinfachten
Umsetzung individuell erstellter Systeme**

Jan Lepel

**Automatisierte Erstellung und Provisionierung von ad hoc
Linuxumgebungen -
Prototyp einer zentralisierten Weboberfläche zur vereinfachten
Umsetzung individuell erstellter Systeme**

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Ulrike Steffens
Zweitgutachter: MSc Informatik Oliver Neumann

Eingereicht am: 1. Januar 2015

Jan Lepel

Thema der Arbeit

Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen -
Prototyp einer zentralisierten Weboberfläche zur vereinfachten Umsetzung individuell erstellter
Systeme

Stichworte

Ad hoc Umgebung, automatisierter Umgebungsaufbau und Provisionierung

Kurzzusammenfassung

Dieses Dokument ...

Jan Lepel

Title of the paper

TODO

Keywords

Keywords, Keywords1

Abstract

This document ...

Listings

Inhaltsverzeichnis

1	Einleitung	1
1.1	Problemstellung	1
1.2	Zielsetzung	2
1.3	Motivation	2
1.4	Themenabgrenzung	2
1.5	Struktur der Arbeit	3
2	Die Software	4
2.1	Idee und Anforderung	4
2.2	Übersicht der Komponenten	4
2.2.1	Webkomponente	4
2.2.2	Vagrant	4
2.2.3	Ansible	4
2.3	Struktur und Zusammenspiel	5
2.4	Konfiguration	5
2.5	Datenbank	5
2.6	Virtualisierung	5
2.7	Provisionierung	5
2.8	Kommunikation der einzelnen Komponenten	5
2.9	Export Funktionen	5
2.9.1	Export der Konfiguration	5
2.9.2	Export zu Git	5
2.10	Import einer Maschine	5
2.11	Import der Konfiguration	5
2.12	Sharing einer Maschine	5
3	Grundlagen	6
3.1	Vagrant	7
3.1.1	Konfiguration	7
3.1.2	Vergleich zu Docker	7
3.2	Ansible	9
3.2.1	Vergleich zu Salt	9
3.2.2	Vergleich zu Puppet	9
3.3	Passenger	10
3.4	Sidekiq	11

4	Schluss	12
4.1	Zusammenfassung	12
4.2	Fazit	12

1 Einleitung

“Es ist nicht zu wenig Zeit, die wir haben, sondern es ist zu viel Zeit, die wir nicht nutzen” - Lucius Annaeus Seneca, [Seneca \(2005\)](#)

Seneca formulierte 49 n. Chr. das Gefühl welches jeder kennt. Die Zeit die er hat, nicht richtig zu nutzen. Technische Neuerungen helfen uns unsere Zeit besser zu planen, mehr Zeit in andere Aktivitäten zu stecken und unsere Prioritäten zu überdenken. Diese Arbeit beschäftigt sich mit dem Teil-Aspekt der Informatik, der virtualisierung von Servern im Entwicklungsumfeld.

1.1 Problemstellung

Server-Virtualisierung ist in der heutigen Zeit keineswegs mehr eine Seltenheit. Sie ist eher Standard in den meisten Unternehmen.

Virtualisierung hat in vielen Bereichen den physischen Server abgelöst, denn es müssen keine aufwändigen Planung im Vorfeld getroffen werden, der Einkauf muss nicht mehr involviert werden, das Budget braucht nicht kalkuliert werden und die Frage, was in ein paar Jahren mit der Hardware passiert, wird obsolet.

Im idealfall heisst das für Unternehmen: Weniger Server sind gleichbedeutend mit weniger Stellfläche, mit weniger Verkabelung oder Racks. Somit ist die Konsolidierung der ehemals großen Server-Zentren für viele Unternehmen eine direkte Konsequenz. Wodurch eine Kostenreduzierung der gesamten Infrastruktur entsteht.

Nicht nur der Finanzielle Aspekt spricht oft für die Virtualisierung, sondern auch die leichte Automatisierung, die Erhöhung der Verfügbarkeit und

Das Verschieben von kompletten Applikationen von einem physischen Ort zu einem anderen,.... Verbesserung der Verfügbarkeit und Business Continuity. Dazu gehören Live Migration, Storage Migration, Fehlertoleranz, Hochverfügbarkeit und Ressourcen-Management. Virtuelle Maschinen können damit leicht verschoben und vor ungewünschten Auszeiten geschützt werden.

In der physischen Welt war es bisher üblich, jeder Applikation einen eigenen Server zuzuweisen. Damit war dafür Sorge getragen, dass die einzelnen Software-Programme sauber voneinander

isoliert waren. Aber das führte auch zu einem Wust von Rechnern, von denen viele noch dazu nicht optimal ausgelastet waren. Und die Kosten für diese Server-Landschaft liefen schnell aus dem Ruder. Nicht so bei Virtualisierung. Inzwischen sind auch die nötigen Funktionen und Tools vorhanden, um VMs und die in ihnen verpackten Anwendungen sauber voneinander zu trennen. CPU, Memory und Storage können exakt ausgelastet werden, die Kosten in einem solchen Modell sinken. ...

1.2 Zielsetzung

Ziel der vorliegenden Arbeit ist es, eine Softwareprodukt zu erarbeiten, welches aktuelle Virtualisierungs-, sowie Provisionierungslösungen verwendet, um mit deren Hilfe den Aufbau von temporären(ad hoc) virtuellen Umgebungen zu vereinfachen und den administrativen Aufwand gering zu halten.

Eine Auswahl von leicht erlernbaren und unkomplizierten Softwarekomponenten, fördern ein weiteres Ziel. Den Administrationsaufwand gering wie möglich zu halten und auch Linux/Unix unerfahrene Administratoren anzusprechen.

Es ist angestrebt, bei Ende dieser Arbeit eine Zentralisierte Anwendung zur Verfügung zu stellen, die es dem Benutzer ermöglicht sich selbstständig und mit geringem Zeitaufwand eine virtuelle Maschine mit gewünschter Software zu erstellen und somit die administrative Instanz des z.B Unternehmen zu entlasten.

1.3 Motivation

Die Motivation dieser Arbeit besteht darin, eine Software zu entwickeln, die durch vereinfachte Handhabung und ohne Einarbeitungszeit, es dem Benutzer ermöglicht eine ad-hoc Umgebung zu erstellen, ohne bürokratischen Aufwand und ohne Grundwissen über die darunterliegende Anwendungsstruktur. Der normalerweise große zeitliche Aufwand soll möglichst minimiert werden und es gerade Anwendern in Unternehmen und Projekten erleichtert werden sich auf die vorhandenen Usecase zu fokussieren und keine Zeit in Aufbau, Installation und Problembewältigung investieren zu müssen.

1.4 Themenabgrenzung

Diese Arbeit greift bekannte und etablierte Softwareprodukte auf und nutzt Diese in einem zusammenhängenden Kontext. Dabei werden die verwendeten Softwareprodukte nicht modifi-

ziert, sondern für eine vereinfachte Benutzung durch eigene Implementierungen kombiniert und mit einem Benutzerinterface versehen, welches die Abläufe visuallisiert und dem Benutzer die Handhabung vereinfacht. Die vorzunehmenden Implementierungen greifen nicht in den Ablauf der jeweiligen Software ein, sondern vereinfacht das Zusammenspiel der einzelnen Anwendungen.

1.5 Struktur der Arbeit

2 Die Software

2.1 Idee und Anforderung

2.2 Übersicht der Komponenten

2.2.1 Webkomponente

2.2.2 Vagrant

Durch die Verwendung von Vagrant wird es möglich durch wenig Aufwand eine schnell verfügbare Umgebung aufzubauen. Diese Arbeit bezieht sich in der Entwicklungsphase ausschliesslich auf eine Ubuntu 32Bit Version, die Online von Vagrant bereitgestellt wird. So wird eine stabile Testbasis erzeugt, die vom Hersteller geprüft wurde. Da Vagrant auf den gängigen Systemen OS X, Windows und diversen Linux Distributionen installiert werden kann, es hervorragend mit diversen Provisionierern zusammenarbeitet und VM-Tools wie VirtualBox, VMWare und AWS unterstützt, wird es zu einem guten Allrounder. Gerade die Provisionierung von Software ,auf die zu startende virtuelle Maschine, macht Vagrant für das vorliegende Projekt attraktiv. Zwar dauert der Aufbau im Gegensatz zu Docker nicht Sekunden sondern Minuten, aber die Provisionierung ist mit einer der entscheidende Punkte für die angestrebte Software. Die leicht zu handhabende Konfiguration und die eingängigen Begrifflichkeiten sind weiterer Punkte, die positiv für die Benutzung in diesem Projekt sprechen.

2.2.3 Ansible

Ansible.....

2.3 Struktur und Zusammenspiel

2.4 Konfiguration

2.5 Datenbank

2.6 Virtualisierung

2.7 Provisionierung

2.8 Kommunikation der einzelnen Komponenten

2.9 Export Funktionen

2.9.1 Clonen einer Maschine

Vagrant enthält nativ einen Befehlssatz um aus einer aktuell laufenden Maschine eine wiederverwendbare Vagrant-Box zu erstellen. Um diese Box später wieder benutzen zu können, muss diese auf ein Laufwerk kopiert werden, welches von dem gewünschten Mitarbeiter zugegriffen werden kann. Durch das kopieren auf dessen lokalen Datenträger und dem erstellen eines angepassten Import-Vagrantfiles, ist es möglich Maschinen an Mitarbeiter weiterzugeben.

<http://www.dev-metal.com/copy-duplicate-vagrant-box/>

2.9.2 Export zu Git

Die Voraussetzung hier für ist ein GitHub Konto, welches mit der Anwendung verknüpft ist. Durch die Benutzung von GitHub kann das zuvor erstellte Vagrantfile automatisiert als Git-Repository hochgeladen werden. Dies ermöglicht es Konfigurationen mit Mitarbeitern auszutauschen, zu archivieren oder zu versionieren.

2.10 Sharing einer Maschine

3 Grundlagen

3.1 Vagrant

Bei Vagrant handelt es sich um ein Softwareprojekt, welches 2010 von Mitchell Hashimoto und John Bender 2010 ins Leben gerufen wurde. Der primäre Gedanke hinter diesem Projekt ist es, gerade Entwicklern und Teams eine schnelle und unkomplizierte Möglichkeit zu bieten, virtuelle Maschinen und Landschaften zu erstellen.

Vagrant ist also ein mächtiges Werkzeug zum virtualisieren, der sonst oft lokalen Entwicklungsumgebungen. Gerade Teamarbeit wird dadurch vereinfacht, denn die gewünschten Maschinen können mit den gleichen Konfigurationen, Komponenten, Infrastrukturen und Bibliotheken erstellt werden.

Um die gewünschte Maschine zu visualisieren, greift Vagrant auf VirtualBox zurück. VirtualBox ist Oracles Freeware Pendant zur kommerziellen VMware Produktlinie. Wenn gewünscht, kann auf VMware Fusion oder Amazon Web Services zurückgegriffen werden.

Die Konfiguration einer Maschine geschieht über das "Vagrantfile"; in dem Parameter wie IP Adresse konfiguriert oder Provisionierer hinzuschaltet. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, bedeutet das für den Anwender, dass er es einfach mit anderen Kollegen über Versionskontrollen (z.B. Git oder Subversion) teilen kann. Bei den Provisionierern wird dem Benutzer die Freiheit gegeben, auf Bekannte wie Chef, Puppet oder Ansible zurückzugreifen.

3.1.1 Konfiguration

Vagrantfile beinhalten die Konfiguration jeder Vagrantmaschine. Sogar jeder Vagrant-"Landschaft". Das in Ruby Syntax geschriebene Konfigurationsfile, wird automatisch über den Befehl "vagrant init" im gewünschten Ordner generiert oder manuell über jeden Editor erstellt werden. Auf Linux-Systemen ist darauf zu achten, dass der gewünschte Ordner über entsprechende Berechtigungen verfügt. Manuelle Erweiterung des Vagrantfiles wird durch die Rubysyntax zusätzlich vereinfacht.

3.1.2 Vergleich zu Docker

Im Zusammenhang mit ad hoc Umgebungen, wird der Fokus in Diskussion auf Docker und Vagrant gelegt. Der Grundgedanke bei beiden Applikationen ist der Gleiche, aber man muss sich bewusst sein, was das Ziel der Anwendung sein soll. Beide Applikationen haben wie alles ihre Vor- und Nachteile, aber in einem sind beide Virtualisierungs-Tools gleich. Die zentrale Steuerung zum Aufbau einer Maschine, geschieht über eine einziges Konfig-File.

Docker ist ein Linux-only VE (Virtual Environment)-Tool und arbeitet im Gegensatz zu Vagrant

mit Operating-System-Level Virtualisierung, auch Linux Containern (LxC) genannt. Während Vagrant Hypervisor-basierten virtuellen Maschinen aufbaut. Für die sogenannten Container nutzt Docker spezielle Kernelfunktionalitäten, um einzelne Prozesse in Containern voneinander zu isolieren.

Dadurch wird für den Benutzer der Eindruck erweckt, dass für Prozesse die mit Containern gestartet werden, diese auf ihrem eigenen System laufen würden. Docker wird in drei Teile unterteilt. Die Arbeitsweise ist nicht wie bei einer VM (virtuelle machine) bei der ein virtueller Computer mit einem bestimmten Betriebssystem, Hardware-Emulation sowie Prozessor simuliert wird. Ein VE ist quasi eine leichtgewichtige virtuelle Maschine. In einem Container ausgeführte Prozesse greifen gemeinschaftlich auf den Kernel des Hostsystems zu. Durch die Kernelnamensräume(cnames) werden die ausgeführten Prozesse von einander isoliert. Allerdings ist gerade die Isolation der Prozesse, durch den gemeinsam genutzten Kernel etwas geringer, als bei der Benutzung durch einen Hypervisor.

Durch die zugrunde liegende Architektur von Vagrant, wird Vagrant gerne für immer gleich aufbauende Entwicklungsumgebungen benutzt. Die Vielzahl an unterstützten Betriebssystemen macht es für viele Benutzer attraktiver. Allerdings kooperieren Vagrant und Docker auch hervorragend zusammen. Vagrant verspricht durch die leichtere Handhabung und Konfiguration eher das, was für das vorliegende Projekt entscheidend ist.

ZITAT: Docker wird in diversen Varianten als Lösung für das Setup und die Kapselung lokaler Entwicklungsumgebungen, als temporär verfügbarer Service im Rahmen von Integrationstests und als Laufzeitumgebung auf Produktionsservern eingesetzt. Gerade der temporäre Charakter von Docker-Containern wird für einen anderen Anwendungsfall interessant: als dynamisch erzeugte Buildsysteme mit einer projektspezifischen Buildumgebung.

Informationen: <http://www.scriptrock.com/articles/docker-vs-vagrant> ; <https://entwickler.de/online/docker-basics-system-level-virtualisierung-125514.html>

3.2 Ansible

3.2.1 Vergleich zu Salt

Da Ansible mit Salt am ehesten verwandt ist, wird auf Puppet und Chef im weiteren nicht weiter eingegangen. Puppet und Chef widersprechen dem vorgesehenen Konzept der leichten Konfiguration.

Salt ist wie Ansible in Python entwickelt worden. Da Salt zur Kommunikation mit seinen Clients Agenten (Minions) benötigt, ist dies wiederum schwer zu automatisieren. Zwar kann Vagrant mit Salt zusammenarbeiten, allerdings nicht nativ. Salt wäre eine gute Alternative, wenn nicht nur einen Provisioner haben möchte, sondern auch ein remote execution framework. Salt implementiert eine ZeroMQ messaging lib in der Transportschicht, was die Kommunikation zwischen Master und Minions im Gegensatz zu Chef und Puppet vervielfacht. Dadurch ist die Kommunikation zwar schnell, aber nicht so sicher, wie bei der SSH Kommunikation von Ansible. ZeroMq bietet keine native Verschlüsselung und transportiert Nachrichten über IPC, TCP, TIPC und Multicast.

<http://www.scriptrock.com/articles/ansible-vs-salt>

3.2.2 Vergleich zu Puppet

Einer der wohl bekanntesten Configuration Management Tool ist Puppet. Puppet wird von allen gängigen Betriebssystemen unterstützt. Windows, Unix, Mac OS X und Linux.

Nachteile: Flexibilität und Agilität sind keine Stärken. Zu groß.

3.3 Passenger

3.4 Sidekiq

4 Schluss

4.1 Zusammenfassung

4.2 Fazit

See also ?.

Literaturverzeichnis

- [Seneca 2005] SENECA: *Von der Kunst des Lebens*. Deutscher Taschenbuch Verlag, 11 2005.
– URL <http://amazon.de/o/ASIN/342334251X/>. – ISBN 9783423342513

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 1. Januar 2015 Jan Lepel
