Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

**Jan Lepel**

## Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

*Fakultät Technik und Informatik*
*Studiendepartment Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

Jan Lepel

# Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

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 Weboberfläche zur vereinfachten Inbetriebnahme individuell erstellter Entwicklungsumgebungen

**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

> "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. ein Gefühl das 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

Virtualisierung hat in vielen Bereichen den physischen Server abgelöst, denn der Finanzielle Aspekt ist für Unternehmen nicht unerheblich. Im Idealfall heißt der Umstieg auf virtuelle Landschaften gleich weniger Server, was gleichbedeutend mit weniger Stellfläche ist. Somit auch mit weniger Racks und weniger Verkabelung.
Aufwändige Vorplanung von Serverzentren entfällt, die Kostenplanung der unterschiedlichen Hardware wird minimiert und die Frage, was in ein paar Jahren mit der Hardware passieren soll, wird obsolet.
Gerade im Entwicklungsbereich ist es meist sinnvoller virtuelle Umgebungen zu realisieren, als reale Maschinen aufzubauen. Entwickler haben so die Möglichkeit bei Bedarf sich Abzüge der Produktionsumgebung zu erstellen oder Fehlerszenarien nachzustellen.
Meist ist dazu die Involvierung des Betriebs-Teams oder des IT-Support notwendig, die nach Priorität ihrer Auftragslage, eine gewissen Vorlaufzeit benötigen, um die gewünschte Maschine aufzubauen.
In dem Fall, dass die Firmengröße es nicht erlaubt, eine eigene Support-Abteilung zu haben, muss die Zeit des jeweiligen Mitarbeiters herhalten, um das Wissen über die jeweilige Virtualisierungslösung aufzubauen, die gewünschte Maschine zu erstellen und die Installationen der nötigen Programme zu realisieren. Der Rückschluss daraus ist, geringere Produktivität in den Kerntätigkeiten des Mitarbeiters.

Auch wenn die Softwarebranche eine Vielfalt an Möglichkeiten bereitstellt, sind diese entweder in ihrer Struktur überdimensioniert, um sie in der Anwendung schnell zu erlernen, oder komplex in ihrer Konfiguration in Bezug auf Automatisierungen und/oder Provisionierungen.

## 1.2 Zielsetzung

Ziel der vorliegenden Arbeit ist es, ein Softwareprodukt zu erarbeiten, welches aktuelle Virtualisierungs-, sowie Provisionierungslösungen verwendet, um mit deren Hilfe den Aufbau von temporären (ad hoc) Umgebungen im virtuellen Umfeld zu vereinfachen.
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, sowie Entwickler anzusprechen.
Es ist angestrebt, bei Ende dieser Arbeit eine zentralisierte und leicht lokal zu implementierende Anwendung zur Verfügung zu stellen. Sie soll es dem Benutzer ermöglichen, sich selbstständig und mit geringem Zeitaufwand eine virtuelle Maschine mit gewünschter Software zu erstellen, ohne große Einarbeitung in Benutzung und Konfiguration.
Somit wird die administrative Instanz des z.B. Unternehmen entlasten und befähigt den Benutzer sich auf seine Kerntätigkeiten zu konzentrieren.

## 1.3 Motivation

Die Motivation dieser Ausarbeitung besteht darin, eine Software zu entwickeln, die durch vereinfachte Handhabung und minimaler Einarbeitungszeit, es dem Benutzer ermöglich 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 Anwendern in Unternehmen und Projekten erleichtert wird, sich auf die vorhandenen Usecase zu fokussieren und keine Zeit in Aufbau, Installation und Problembehebung 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 modifiziert, sondern für eine vereinfachte Benutzung durch eigene Implementierungen kombiniert und mit einem Benutzerinterface versehen, welches die Abläufe visualisiert 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 Anforderungen

An dieser Stellen sollen die Anforderungen an die Software detaillierter konzipiert und aufgelistet werden.
Ein direkter Vergleich mit bestehenden Lösungen wird veranschaulicht und das Konzept der zu entwickelnden Software genauer betrachtet.

## 2.1 Funktionale Anforderungen

Funktionale Anforderungen beschreiben Dienste oder Funktionalitäten, die ein System bereitstellen soll.

FA 1.0 **Berechtigungssystem**
Die zu entwickelnde Software soll zwei Berechtigungsstufen beinhalten, um Benutzer und Administratoren von einander zu trennen.

FA 1.1 Rolle: Administrator
Die Administrator-Rolle, soll Zugriff auf konfigurative Möglichkeiten bekommen. Beispiele hierfür wären: Speicherpfad der Logdatei ändern; neue Softwarekomponenten in die Auswahlliste(FA 2.0) übernehmen.

FA 1.2 Rolle: User
Benutzer in der User-Rolle sind nicht berechtigt Konfigurationen an der Software vorzunehmen. Ihnen steht der Administrative Punkt nicht zur Verfügung

FA 2.0 **Ausswahlliste Softwarekomponenten**
Um dem Benutzer die Möglichkeit zu bieten, auf die gewünschte Maschine Softwarekomponenten zu provisionieren, soll das System eine entsprechende Auswahlliste beinhalten

FA 2.1 Die Auswahlliste kann durch Benutzer mit Administrationsberechtigungen geändert werden. Benutzer mit Berechtigungen der Stufe **User**, dürfen die Liste nicht

manipulieren.

FA 2.2 Folgende Funktionen sollen dem Administrator zur Verfügung gestellt werden:

i. Löschen - Löscht den kompletten Datensatz aus der Liste und somit aus der Datenbank.

ii. Ändern - Beschreibung, sowie Linux-Befehlssatz kann geändert werden.

iii. Hinzufügen - Name der Software, Beschreibung und Linux-Befehlssatz können eingegeben werden und zur Datenbank hinzugefügt werden.

FA 3.0 **Hochladen eigener YAML-Dateien**
Es soll möglich sein, statt die Auswahlliste(FA 2.0) zu verwenden, eine YAML-Datei hochzuladen, die die gewünschte Software enthält.
Dies geschieht über einen Menüpunkt in der Benutzeroberfläche.

FA 3.1 Die YAML-Datei muss den Vorgaben von Ansible genügen.

FA 3.2 Administratoren und User dürfen YAML-Dateien verwenden.

FA 3.3 Fehler sind in einer Logdatei zu protokolieren.

FA 4.0 **Fehler**
Falsche Benutzung oder Systemfehler sollen Software angezeigt werden und entsprechend in einer Logdatei protokoliert werden.

FA 4.1 Falsche Benutzung, darf nicht zum Abbruch des kompletten Vorgangs führen.

## 2.2 Nichtfunktionale Anforderungen

Beschränkungen von Funktionalitäten und Diensten werden in nichtfunktionalen Anforderungen definiert.
Diese beziehen sich entweder auf Teile des Systems oder auf das gesamte System und werden unterschieden in Qualitätsanforderungen an die Diensterbringung, Rahmenbedingungen, die das System einschränken und unterspezifische funktionale Anforderungen, die noch nicht detailliert spezifiziert wurden.

NFA 1.0 **Verschlüsselung**
Die Kommunikation zwischen Frontend und Backend wird nicht verschlüsselt und läuft über HTTP.

NFA 2.0 **Fehlermeldungen**
Die Anwendung hat für alle Fehlerfälle eine entsprechende Fehlermeldung.

NFA 3.0 **Zuverlässigkeit**
Die Anwendung hat zu jedem Zeitpunkt stabil zu laufen und nicht abzustürzen.

NFA 4.0 **Austauschbarkeit**
Komponenten sollen leicht austauschbar sein.

NFA 5.0 **Sprache**
Die Benutzerführung erfolgt in Deutsch

NFA 6.0 **Nachvollziehbarkeit**
Das System muss den Benutzer über den aktuellen Konfigurationsstand informieren

## 2.3 Anwendungsfälle

Anwendungsfälle helfen fachlichen Anforderungen eines Systems darzustellen, indem dort Interaktionen zwischen System und Benutzer dargestellt werden.

Anwendungsfall AF 1 Anforderungen: [12312312312312312]

| | |
|---|---|
| **Bezeichnung** | Automatische Erstellung einer virtuellen Maschine exklusive Provisionierung |
| **Ziel im Kontext** | Ein Benutzer erstellt eine Umgebung |
| **Akteur** | Benutzer |
| **Auslöser** | Benutzer benötigt eine virtuelle Maschine ohne zusätzliche Software |
| **Vorbedingung** | Die Anwendung ist installiert und lauffähig. Der Benutzer kann auf die Benutzer-oberfläche zugreifen. |
| **Nachbedingung** | Die Anwendung stellt dem Benutzer einen zugreifbare und lauffähige virtuelle Maschine zur verfügung. |
| **Erfolgsszenario** | 1. Der Benutzer startet die Anwendung über den Browser.<br>2. Er wählt keine Software aus der vorhandenen Auswahlliste aus und klick auf den Knopf "weiter".<br>3. Er bekommt die Bestätigung, dass er keine Software ausgewählt hat. Dies muss er bestätigen.<br>4. Die Anwendung zeigt die einzelnen Schritte bezüglich des Aufbaus der virtuellen Maschine.<br>5. Die Anwendung zeigt dem Benutzer die Zugriffsmöglichkeiten auf die erstellte Maschine. |
| **Fehlerfälle** | 1. Der Benutzer wählt eine Softwarekomponente aus der Liste und erstellt eine Maschine inklusive Provisionierung der gewählten Software. |

Abbildung 2.1: Anwendungsfall AF 1

# 3 Die Software

Dieses Kaptitel beschreibt die Entwicklung der Software, Designentscheidungen und die verwendeten externen Komponenten.

## 3.1 Frontend

## 3.2 Backend

## 3.3 Ruby

Für die Entwicklung der Software wurde Ruby in der Version 1.9 im Zusammenhang mit dem Micro-Framework Sinatra verwendet. Durch die leicht zu erlernenden Syntax und die einfache Wartung des Quellcodes, ist Ruby eine einsteigerfreundliche Programmiersprache, im Bereich der Webentwicklung.
Durch die Installation von Bundler, ein Dependency Manager für Ruby, werden alle benötigten Abhängigkeiten bei der Installation der hier thematisierten Software, heruntergeladen und installiert. Somit entsteht ein zentraler Punkt, der sich um die Abhängigkeiten von Ruby kümmert und es ermöglicht leicht und schnell änderungen vorzunehmen.

## 3.4 Externe Komponenten

Die folgende Aufzählung gibt einen überblick über die verwendeten Softwarekomponenten. Im nachfolgenden wird detaillierter auf die einzelnen Bauteile eingegangen und deren Funktionsweise ausführlicher erklärt.

- Sinatra
- Passenger
- Apache
- SQLite

- Vagrant
- VirtualBox
- Ansible

### 3.4.1 Sinatra

Um die

### 3.4.2 Vagrant

Vagrant ermöglicht es 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 Distrubutionen 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.

### 3.4.3 Ansible

Ansible.....

## 3.5 Struktur und Zusammenspiel

## 3.6 Konfiguration

## 3.7 Datenbank

Im Backend der Applikation befindet sich eine relationale Datenbank.
Die dazu verwendete ORM-Schicht namens Datamapper unterstützt MySQL, PostgreSQL und

SQLite.
Für den Prototypen der Anwendung wurde eine SQLite Datenbank benutzt um die Portierung auf andere Systeme zu erleichtern und den Installations-, sowie administrativen Aufwand gering zu halten.
Allerdings ist SQLite weniger für Mehrbenutzerzugriffe ausgelegt, sondern für portable- und einbenutzer Anwendungen. Daher sollte die Datenbank in der Endfassung auf MySQL geändert werden.
Durch die verwendete ORM-Schicht, ist ein Austausch der Datenbank im Aufwand minimal.
Die Datenbank beinhaltet die Softwarekomponenten, die zur Provisionierung der virtuellen Maschine eingesetzt werden können. ...

## 3.8 Virtualisierung

## 3.9 Provisionierung

## 3.10 Kommunikation der einzelnen Komponenten

## 3.11 Export Funktionen

### 3.11.1 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.

### 3.11.2 Export zu Git

Die Vorraussetzung hierfür 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.

## 3.12 Sharing einer Maschine

Mit der Version 1.5 hat Vagrant die Möglichkeit implementiert, eine erstellte Maschine mit anderen Mitarbeitern zu teilen.
Da der Mitbenutzer nicht im gleichen Netzwerk sein muss, sondern einfach nur an das Internet angeschlossen sein braucht, ist diese Option...

# 4 Grundlagen

Dieses Kapitel gibt einen Einblick in die verwendeten Programme, die für das Verständnis der vorliegenden Arbeit von Vorteil sein können. Es wird zunächst auf die prägnanten Eigenschaften der Produkte eingegangen und wenn möglich ein Vergleich mit Konkurrenzprodukten herangezogen. Zudem wird teilweise die Handhabung und die Konfiguration kurz erläutert.

## 4.1 Ruby

Die Programmiersprache Ruby wurde in etwa Mitte der 90er Jahre entwickelt und ist eine objektorientierte, dynamisch typisierte Sprache. Sie zeichnet sich unter anderem durch ihre einfache und leicht erlernbare Syntax aus und ihrer einfachen Wartung des Quellcodes. Bekannt geworden ist sie durch besonders durch das Framwork Ruby on Rails.
Ruby Unterstützt diverse Programmierparadigmen, wie unter anderem prozedurale und funktionale Programmierung sowie Nebenläufigkeit und wird direkt von einem Interpreter verarbeitet. Außerdem bietet die Sprache eine automatische Garbage Collection, Reguläre Ausdrücke, Exceptions, Blöcke als Parameter für Iteratoren und Methoden, Erweiterung von Klassen zur Laufzeit und Threads.
Da Ruby nicht Typisiert ist, wird alles als ein Objekt angesehen. Auf primitive Datentypen wird gänzlich verzichtet.

## 4.2 Sinatra

## 4.3 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.
Standardmäßig greift Vagrant auf VirtualBox zurück, um die Virtualisierung vorzunehmen. VirtualBox ist Oracles Freeware Pendant zur kommerziellen VMware Workstation/Fusion. Die Installation von Plugins ermöglicht es, statt auf VirtualBox auf VMware Workstation/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 werden können. Bei den Provisionierern wird dem Benutzer die Freiheit gegeben, auf Bekannte wie Chef, Puppet oder Ansible zurückzugreifen.
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.
Abgesehen von dem Austausch der Konfigurationen, wird die Teamarbeit durch eine 'sharing' Option unterstützt, die mit Version 1.5 implementiert wurde. Das Teilen eine Maschine ermöglicht es Teams an gemeinsamen und entfernten Standorten auf die gleiche Maschine zuzugreifen.

### 4.3.1 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.

### 4.3.2 Vergleich zu Docker

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 voneinan-

der 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 voneinander 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.

## 4.4 Ansible

### 4.4.1 Vergleich zu Salt

Da Ansible mit Salt am ehesten verwandt ist, wird auf Puppet kurz eingegangen und auf Chef gänzlich verzichtet. Salt ist wie Ansible in Python entwickelt worden. Da Salt zur Kommunikation mit seinen Clients Agenten (Minions) benötigt, bedeutet dies, einen höheren Installationsaufwand. 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. ...

### 4.4.2 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. ...

## 4.5 Passenger

## 4.6 Sidekiq

# 5 Schluss

## 5.1 Zusammenfassung

## 5.2 Fazit

See also ?.

# Literaturverzeichnis

[Seneca 2005] Seneca: *Von der KÃ¼rze 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