



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**

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	1
1.1	Problemstellung	1
1.2	Zielsetzung	2
1.3	Motivation	2
1.4	Themenabgrenzung	2
1.5	Struktur der Arbeit	3
2	Anforderungsanalyse	4
2.1	Anforderungen	4
2.1.1	Nichtfunktionale Anforderungen	4
2.1.2	Stakeholder	6
2.1.3	Funktionale Anforderungen	6
2.2	Anwendungsfälle	10
3	Entwurf/Design	12
3.1	Systemkontext	12
3.2	Technische Randbedingungen	12
4	Umsetzung/Realisierung	13
4.1	Frontend	13
4.2	Backend	13
4.3	Ruby	13
4.4	Externe Komponenten	13
4.4.1	Sinatra	14
4.4.2	Vagrant	14
4.4.3	Ansible	14
4.5	Struktur und Zusammenspiel	14
4.6	Konfiguration	14
4.7	Datenbank	14
4.8	Virtualisierung	15
4.9	Provisionierung	15
4.10	Kommunikation der einzelnen Komponenten	15
4.11	Export Funktionen	15
4.11.1	Clonen einer Maschine	15
4.11.2	Export zu Git	15
4.12	Sharing einer Maschine	16

5	Grundlagen	17
5.1	Ruby	17
5.2	Sinatra	17
5.3	Vagrant	18
5.3.1	Konfiguration	18
5.3.2	Vergleich zu Docker	18
5.4	Ansible	20
5.4.1	Vergleich zu Salt	20
5.4.2	Vergleich zu Puppet	20
5.5	Passenger	21
5.6	Sidekiq	22
6	Schluss	23
6.1	Zusammenfassung	23
6.2	Fazit	23

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 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 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 Anforderungsanalyse

Die Anforderungsanalyse hilft Systemeigenschaften und Systemanforderungen der einzelnen beteiligten Gruppen, auch als Stakeholder bezeichnet, zu erfassen, zu analysieren und ggf. eine Diskussionsgrundlage zu schaffen. Das resultierende Ergebnis, kann dann wiederum als Grundstein für ein zukünftiges Lastenheft genutzt werden.

Im Folgenden werden die Anforderungen an die Software detaillierter konzipiert, aufgelistet und beschrieben. Zudem grundlegende Fragen bezüglich der Kernaufgaben, Qualitätsanforderungen, beteiligte Gruppen und benachbarter Systeme geklärt.

2.1 Anforderungen

Anforderungen beschreiben die Kernaufgabe des Systems und hinterfragen die Qualitätsanforderungen, die das System erfüllen muss. Daher werden Anforderungen in der Regel zwischen funktionale Anforderungen und nichtfunktionale Anforderungen unterschieden, wobei die Abgrenzung zwischen den beiden Anforderungsarten nicht immer eindeutig bestimmt werden kann.

2.1.1 Nichtfunktionale Anforderungen

Nichtfunktionale Anforderungen können als Qualitätskriterium an System und Software angesehen werden, wobei die Qualität der Software/des Systems immer auf die folgenden Qualitätsmerkmale bezogen wird.

1. **Funktionalität** Die geforderten Funktionen sind in der Software umgesetzt worden und erfüllen die definierten Aufgaben und Ziele.
2. **Zuverlässigkeit** Die Software kann unter bestimmten Bedingungen und über einen bestimmten Zeitraum ein definiertes Leistungsniveau aufrechterhalten.
3. **Benutzbarkeit** Analyse der Benutzung und des Aufwandes. Der Benutzer steht hier im Fokus.

4. **Effizienz** Das Verhältnis zwischen der zu erbringenden Leistung der Software und der dafür zugrundeliegenden Betriebsmittel.
5. **Änderbarkeit** Der Aufwand, der betrieben werden muss um Änderungen an der Software vorzunehmen. Änderungen sind definiert als Korrekturen, Verbesserungen oder Anpassungen an Veränderungen der Umgebung, der Anforderungen oder der funktionalen Spezifikationen.
6. **Übertragbarkeit** Der Aufwand, der zum Übertragen oder Implementieren in eine neue Umgebung benötigt wird. Als Umgebung ist sowohl Hardware- als auch Softwareumgebung gemeint.

(Qualitäten nach ISO/IEC 9126 (DIN 66272))

Das Bilden von Qualitätsszenarien hilft bei der Erfassung von Qualitäten und deren Sortierung zu entsprechenden Qualitätsmerkmalen. Zusätzlich können neue Anforderungen aufgedeckt werden und leichter Anwendungsfälle abgeleitet werden.

Qualitätsszenarien

- NFA 1 Ein Benutzer ohne Vorkenntnisse muss bei der erstmaligen Verwendung des Systems innerhalb von 10 Minuten in der Lage sein, die gewünschte Funktionalität zu lokalisieren und zu verwenden.
- NFA 2 Die Anzahl der gleichzeitig laufenden virtuellen Umgebungen, ist maximal bei 10.
- NFA 3 Jeder auftretende Fehler ist eindeutig identifizierbar und nachvollziehbar.
- NFA 4 Die Architektur soll so erweiterbar sein, dass es zukünftig möglich ist, mehrere Maschinen gleichzeitig aufbauen zu können.
- NFA 5 Das Importieren von Konfigurationsdateien beträgt <2 Minuten.
- NFA 6 Die Validierung einer YAML-Datei beträgt <1 Minute.
- NFA 7 Der bei Import verwendete Validierungsalgorithmus muss unter 0.5 Personentagen austauschbar sein.
- NFA 8 Der Export von Konfigurationen beträgt <1 Minute.
- NFA 9 Änderungen an der Softwarekomponentenliste muss innerhalb von 2 Sekunden in der GUI sichtbar sein.

NFA 10 Das Betriebssystem soll auf eine höhere Version aktualisiert werden. Dies kann ohne Änderungen des Quellcodes der Software vorgenommen werden.

NFA 11 Soll ein anderer Provisionierer verwendet werden, liegt der Aufwand des Austausches bei unter einem Personentag.

NFA 12 Die Implementierung weiterer Grundfunktionalitäten soll möglichst einfach erfolgen.

NFA 13 Softwareupdates der benutzen Softwarekomponenten müssen mit Rücksprache des Entwicklers erfolgen.

NFA 14 Die Kommunikation zwischen Frontend und Backend muss nicht verschlüsselt ablaufen.

NFA 15 Die Anwendung sollte zu 99.0 Prozent der Zeit lauffähig sein.

2.1.2 Stakeholder

‘Stakeholder in Softwareentwicklungsprojekten sind alle Personen (oder Gruppen von Personen) die ein Interesse an einem Softwarevorhaben oder dessen Ergebnis haben.’ **Zörner (2012)**

Rolle	Anwender
Beschreibung	Ein Anwender ist ein Benutzer des Systems, ohne administrative Einflüsse auf die Applikation.
Ziel	Gute Benutzbarkeit, schnell erlernbar, komfortable Steuerung, leichter Aufbau der gewünschten Umgebung

Rolle	Administrator
Beschreibung	Der Administrator kann die Applikation wie der Anwender nutzen. Er hat erweiterte Möglichkeiten, im Bezug auf die Konfiguration des Systems.
Ziel	leicht ersichtliche Konfigurationsmöglichkeiten, schnelles auffinden von auftretenden Fehlern, gut protokollierte Fehler

Abbildung 2.1: Stakeholder

2.1.3 Funktionale Anforderungen

Die funktionalen Anforderungen decken die Frage nach den Kernaufgaben des Systems ab. Beschreiben somit Dienste oder Funktionalitäten, die ein System bereitstellen soll. Im folgenden

sind die Anforderungen Stakeholdern zugeordnet, die vorher definiert werden.

Anforderungen Anwender

FA 1 Starten der Applikation

Ist die Applikation Lokal oder Serverseitig installiert, benötigt der Anwender keine zusätzliche Installation, um die Software im Browser auszuführen.

FA 2 Aufbau ohne Provisionierung

Der Anwender hat die Möglichkeit eine Umgebung aufzubauen, ohne vorinstallierte Software.

FA 3 Aufbau mit Provisionierung inkl. Auswahlliste

Es ist dem Anwender möglich, eine Umgebung inkl. Provisionierung zu erstellen, in dem er über eine Auswahlliste Softwarekomponenten auswählen kann.

FA 4 Aufbau mit Provisionierung inkl. eigener Dateien

Durch das Hochladen eigener Dateien, ist der Anwender in der Lage, Umgebungen aufzubauen inkl. seiner eigens ausgesuchten Dateien/Programme. Dies ist in Kombination mit FA 3 2.1.3 möglich

FA 5 Import von Konfigurationen

Durch den Import von Konfigurationen wird es dem Benutzer ermöglicht, selbst- oder von Kollegen erstellte Umgebungen wieder aufzubauen.

FA 6 Export von Konfigurationen

Exportieren der erstellten Konfiguration, soll den Austausch und das Speichern von Konfigurationen für den Benutzer erleichtern.

Der Benutzer kann sich den Speicherort aussuchen.

FA 7 Rückmeldung während Konfiguration 1/2

Die Anwendungen informiert den Benutzer bei jedem Konfigurationsschritt über Erfolg oder Misserfolg.

FA 7.1 Rückmeldung während Konfiguration 2/2

Ist eine Maschine ohne Fehler aufgebaut, wird dies dem Benutzer mitgeteilt. Ausserdem erscheint der Erstellungsordner, sowie Verbindungsmöglichkeiten zu der Maschine.

Anforderungen Administrator

FA 1 **Administratorenkonto**

Das anlegen eines Administratorenkonto bevollmächtigt den entsprechenden Benutzer zu Konfigurationsänderungen bezüglich der Anwendung.

FA 2 **Logdateien - Speicherort ändern**

Der Administrator kann den Speicherort von Logdateien ändern.

FA 3 **Logdateien - Name ändern**

Der Administrator kann den Namen von Logdateien ändern.

FA 4 **Logdateien - Anzeigen**

Der Administrator kann den Inhalt von Logdateien in der Applikation anzeigen lassen.

FA 5 **Protokollierung 1/2**

Auftretende Fehler während der Benutzer die Umgebung konfiguriert, werden in einer separaten Logdatei protokolliert.

FA 5.1 **Protokollierung 2/2**

Erfolgreiche Konfigurationsschritte werden in einer separaten Logdatei protokolliert.

FA 6 **Export/Import - Dateiformat YAML**

Erlaubte Export-, sowie Importdateien sind ausschliesslich YAML-Dateien.

FA 7 **Anbindung an Git**

Export und Import sind optional mit einem Git Konto verknüpfbar, um die Kommunikation zwischen gewünschten Personen zu vereinfachen.

FA 8 **Validierung von Importen**

Dateien, die importiert werden sollen, werden beim Import validiert.

FA 9 **Softwarekomponentenliste**

Der Administrator kann die angebotene Softwarekomponentenliste bearbeiten, um diese aktuell zu halten.

FA 9.1 **Hinzufügen von Softwarekomponenten**

Der Administrator kann neue Software zur Softwareliste hinzufügen.

FA 9.2 **Löschen von Softwarekomponenten**

Der Administrator kann Software aus der Softwareliste entfernen

FA 9.3 Ändern von Softwarekomponenten

Der Administrator kann Software aus der Softwareliste abändern. Dazu gehört 'Name', 'Beschreibung' und 'Befehl'.

FA 9.4 Export der Softwarekomponentenliste

Der Administrator kann die Softwarekomponentenliste als XLS exportieren.

2.2 Anwendungsfälle

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

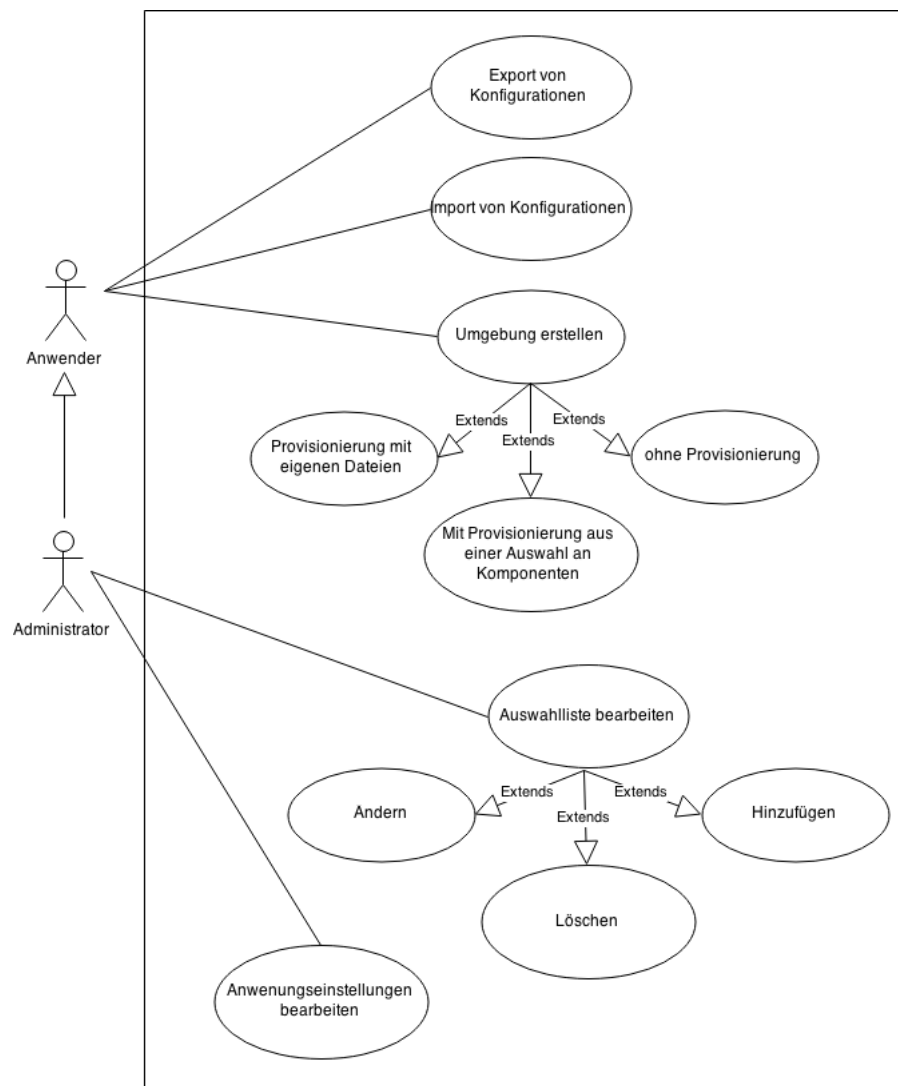


Abbildung 2.2: Bildunterschrift

Anwendungsfall AF 1

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 Benutzeroberfläche zugreifen.
Nachbedingung	Die Anwendung stellt dem Benutzer einen zugreifbare und lauffähige virtuelle Maschine zur Verfügung.
Erfolgsszenario	<ol style="list-style-type: none">1. Der Benutzer startet die Anwendung über den Browser.2. Er wählt keine Software aus der vorhandenen Auswahlliste aus und klickt auf den Knopf "weiter".3. Er bekommt die Bestätigung, dass er keine Software ausgewählt hat. Dies muss er bestätigen.4. Die Anwendung zeigt die einzelnen Schritte bezüglich des Aufbaus der virtuellen Maschine.5. Die Anwendung zeigt dem Benutzer die Zugriffsmöglichkeiten auf die erstellte Maschine.
Fehlerfälle	<ol style="list-style-type: none">1. Der Benutzer wählt eine Softwarekomponente aus der Liste und erstellt eine Maschine inklusive Provisionierung der gewählten Software.

Abbildung 2.3: Bildunterschrift

Anforderungen: [12312312312312312312]

3 Entwurf/Design

3.1 Systemkontext

3.2 Technische Randbedingungen

4 Umsetzung/Realisierung

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

4.1 Frontend

4.2 Backend

4.3 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.

4.4 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

4.4.1 Sinatra

Um die

4.4.2 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 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.

4.4.3 Ansible

Ansible.....

4.5 Struktur und Zusammenspiel

4.6 Konfiguration

4.7 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. ...

4.8 Virtualisierung

4.9 Provisionierung

4.10 Kommunikation der einzelnen Komponenten

4.11 Export Funktionen

4.11.1 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.

4.11.2 Export zu Git

Die Voraussetzung 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.

4.12 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...

5 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.

5.1 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 Framework 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.

5.2 Sinatra

5.3 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.

5.3.1 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.

5.3.2 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.

5.4 Ansible

5.4.1 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. ...

5.4.2 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. ...

5.5 Passenger

5.6 Sidekiq

6 Schluss

6.1 Zusammenfassung

6.2 Fazit

See also ?.

Literaturverzeichnis

- [Seneca 2005] SENECA: *Von der Ku"rze des Lebens*. Deutscher Taschenbuch Verlag, 11 2005. – URL <http://amazon.de/o/ASIN/342334251X/>. – ISBN 9783423342513
- [Zörner 2012] ZÖRNER, Stefan: *Softwarearchitekturen dokumentieren und kommunizieren: Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten*. Carl Hanser Verlag GmbH und Co. KG, 5 2012. – URL <http://amazon.de/o/ASIN/3446429247/>. – ISBN 9783446429246

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
