



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 ...

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Zielsetzung . . . . .	2
1.3	Themenabgrenzung . . . . .	3
1.4	Struktur der Arbeit . . . . .	3
<b>2</b>	<b>Grundlagen</b>	<b>4</b>
2.1	Grundlagen der Virtualisierung . . . . .	4
2.1.1	Virtuelle Maschine . . . . .	5
2.1.2	Gastbetriebssystem . . . . .	5
2.1.3	Hypervisor . . . . .	6
2.2	Konfigurationsmanagement-System . . . . .	8
<b>3</b>	<b>Anforderungsanalyse</b>	<b>10</b>
3.1	Zielsetzung . . . . .	10
3.2	Stakeholder . . . . .	11
3.3	Funktionale Anforderungen . . . . .	11
3.4	Use-Cases . . . . .	13
3.4.1	Business-Use-Case . . . . .	13
3.4.2	System-Use-Case . . . . .	15
3.5	Nichtfunktionale Anforderungen . . . . .	18
3.6	Randbedingungen . . . . .	27
3.6.1	Technische Randbedingungen . . . . .	28
3.7	Zusammenfassung . . . . .	28
<b>4</b>	<b>Evaluation</b>	<b>30</b>
4.1	Vergleichbare Anwendungen . . . . .	30
4.1.1	OpenNebula . . . . .	30
4.1.2	OpenStack . . . . .	31
4.1.3	Fazit . . . . .	31
4.2	Virtualisierungsprodukte . . . . .	32
4.2.1	VMware Player (Plus) . . . . .	32
4.2.2	Microsoft Hyper-V . . . . .	33
4.2.3	Oracle VM VirtualBox . . . . .	33
4.2.4	Zusammenfassung . . . . .	33
4.2.5	Fazit . . . . .	34

4.3	Konfigurationsmanagement-Systeme . . . . .	35
4.3.1	Ansible . . . . .	35
4.3.2	Saltstack . . . . .	36
4.3.3	Puppet . . . . .	36
4.3.4	Zusammenfassung . . . . .	36
4.3.5	Fazit . . . . .	37
4.4	Vagrant . . . . .	37
<b>5</b>	<b>Softwareentwurf</b>	<b>39</b>
5.1	Kontextabgrenzung . . . . .	40
5.2	Verteilungssicht . . . . .	42
5.3	Systemarchitektur . . . . .	43
5.3.1	Client-Server-Modell . . . . .	43
5.3.2	Model-View-Controller Entwurfsmuster . . . . .	45
5.4	Kommunikation . . . . .	47
5.5	Server . . . . .	49
5.5.1	Bausteinsicht . . . . .	49
5.5.2	Laufzeitsicht . . . . .	60
5.5.3	Datenbank . . . . .	63
5.6	Client . . . . .	67
5.7	Zusammenfassung . . . . .	67
<b>6</b>	<b>Realisierung</b>	<b>69</b>
6.1	Hardware . . . . .	69
6.2	Fremdsoftware . . . . .	70
6.2.1	Webserver . . . . .	70
6.2.2	Phusion Passenger . . . . .	70
6.2.3	VirtualBox . . . . .	71
6.2.4	Ansible . . . . .	71
6.2.5	Vagrant . . . . .	71
6.2.6	Bundler . . . . .	71
6.2.7	Datenbank . . . . .	71
6.3	Implementierung . . . . .	72
6.3.1	Komponenten Umsetzung . . . . .	73
6.3.2	Datenbank . . . . .	75
6.3.3	Funktionen . . . . .	77
<b>7</b>	<b>Schluss</b>	<b>79</b>
7.1	Zusammenfassung und Fazit . . . . .	79
7.2	Ausblick . . . . .	79

<b>Anhang</b>	<b>86</b>
3    Mockups . . . . .	86
3.1    Aufbauprozess . . . . .	86
3.2    Optionen . . . . .	86
3.3    Einstellungen . . . . .	86

# 1 Einleitung

Aufgabe: Vom Problem zur Fragestellung

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

dieser Satz stammt von dem römischen Philosophen Seneca (ca. 49 n. Chr.) und besitzt noch heute in der Informatik seine Daseinsberechtigung. Die Informatik hat durch Automatisierung Prozessabläufe vereinfacht, durch effizientere Ressourcennutzung Zeit eingespart und den Arbeitsalltag beschleunigt. Gerade durch das Erforschen von effizienterer Ressourcennutzung und der Suche nach Zeitersparnis, ist in der Informatik die Virtualisierung entstanden. Sie hat in vielen Firmen die Serverlandschaften deutlich verkleinert, Clients durch virtuelle Desktops ersetzt und gestattet es Entwicklern und Testern ihren Arbeitsbereich auf produktionsnahen virtuellen Klonen auszuführen. Gerade der letzt genannte Aspekt, ist die Grundlage für diese Arbeit. Um genauer zu sein, wird der Fokus auf den automatisierten Aufbau von Entwicklungsumgebungen gelegt, inklusive der Vervollständigung durch ausgewählte Softwarekomponenten. Es wird also versucht, dem Entwickler Zeit zu sparen, damit er seine Zeit effektiver nutzen kann. Gewünschter Nebeneffekt ist die Unabhängigkeit von IT-Support Instanzen. So wird der Anwender in die Lage versetzt autonom zu handeln und Bearbeitungsschritte zu beschleunigen.

## 1.1 Motivation

Das Ziel dieser Arbeit ist die Entwicklung einer Applikation, zur automatisierten Erstellung von Linux basierten ad hoc Umgebungen. Der Anwender soll in die Lage versetzt werden, ohne Grundlegende Kenntnisse über Anwendungsstruktur und Betriebssystem, mit minimalem Aufwand, die benötigte virtuelle Maschine zu erstellen. Großer zeitlicher Aufwand für die Erstellung und Organisation wird entsprechend vermieden und es Anwendern ermöglicht sich auf Kerntätigkeiten zu fokussieren. Somit muss keine Zeit mehr in Aufbau, Installation und Problembehebung investieren werden.

Der normalerweise große zeitliche Aufwand für die Erstellung von virtuellen Maschinen soll möglichst minimiert werden und es Anwendern sowohl in Unternehmen als auch bei

Projektarbeiten erleichtern, sich auf die vorhandenen Usecase zu fokussieren. Somit muss keine Zeit mehr in Aufbau, Installation und Problembehebung investieren werden.

die durch vereinfachte Handhabung und einer minimalen Einarbeitungszeit den Anwender in die Lage versetzt schnellst möglich eine ad hoc Entwicklungsumgebung zu erstellen. Dies soll

Das Ziel dieser Arbeit ist es, eine Software zu entwickeln, die durch vereinfachte Handhabung und minimaler Einarbeitungszeit, es dem Benutzer ermöglicht eine ad-hoc Umgebung zu erstellen, ohne dass ein bürokratischer Aufwand erforderlich ist und ohne Grundwissen über die darunterliegende Anwendungsstruktur. Der normalerweise große zeitliche Aufwand für die Erstellung von virtuellen Maschinen soll möglichst minimiert werden und es Anwendern sowohl in Unternehmen als auch bei Projektarbeiten erleichtern, sich auf die vorhandenen Usecase zu fokussieren. Somit muss keine Zeit mehr in Aufbau, Installation und Problembehebung investieren werden.

## 1.2 Zielsetzung

Das Ziel der vorliegenden Arbeit ist es, durch inkrementelles und interaktives Vorgehen eine Applikation zu modellieren, die den Anwender der Applikation beim Aufbau von virtuellen Umgebungen unterstützt. Je nach Wunsch des Anwenders, wird nicht nur der Aufbau einer Umgebung vereinfacht, sondern auch die direkte Installation von Programmen veranlasst. Eine Weboberfläche soll die entsprechenden Optionen zur Verfügung stellen und den Anwender durch seine ausgewählte Funktion leiten. Große Konfigurationen oder komplizierte Einstellungen sollen dem Anwender abgenommen werden. Sie geschehen im Hintergrund. Damit auch ein Sichern oder ein Zurückspielen von vorhandenen virtuellen Maschinen möglich wird, sollen Im- und Exportfunktionen dies unterstützen. Die Realisierung soll auf einem zentralen Knotenpunkt stattfinden, um es mehreren Anwendern zu ermöglichen, ihre notwendigen Maschinen zu erstellen und zu verwalten. Kernaufgaben sollen Open-Source Anwendungen übernehmen, die in ihrem Einsatzbereich etabliert sind. Ebenfalls im Fokus steht die Leichtigkeit der Konfiguration der auszuwählenden Open-Source Anwendung. Bei der Erstellung der einzelnen Softwarekomponenten sind stets die Prinzipien von hoher Kohäsion und loser Kopplung zu beachten. Darunter wird der Grad der Abhängigkeit zwischen mehreren Hard- und Softwarekomponenten verstanden, der Änderungen an einzelnen Komponenten erleichtert. Um auch die Anwendungsoberfläche unkompliziert zu halten, soll der Anwender mit ein paar Klicks zu seinem Ziel geführt werden. Durch das Vermeiden sowohl von unnötigen verschachtelten Menüführungen, als auch von einer Vielfalt an Optionen und Konfigurationen,



soll der Anwender in der Applikation einen Helfer für seine Tätigkeit finden.

### 1.3 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. Sie werden 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 vereinfachen das Zusammenspiel der einzelnen Anwendungen.

### 1.4 Struktur der Arbeit

Diese Arbeit wird in Kapitel 2 Grundlagen schaffen, um ein besseres Verständnis über die Virtualisierung zu erhalten

## 2 Grundlagen

Um ein besseres Verständnis für die Begrifflichkeiten dieser Arbeit zu schaffen, werden in diesem Kapitel grundlegende Themen aufgegriffen und erklärt. Im Vordergrund steht dabei die Virtualisierung in Abschnitt 2.1, da sie den Kern der Applikation bilden wird. Der Abschnitt beinhaltet zudem die Beschreibung des Begriffs 'virtuelle Maschine', der im Laufe dieser Arbeit immer wieder aufgegriffen wird. Danach wird in Abschnitt 2.2 ausgeführt, welchen Anwendungszweck Konfigurationsmanagement-Systeme erfüllen und ihre Funktionsweise betrachtet.

### 2.1 Grundlagen der Virtualisierung

Für den Begriff Virtualisierung existiert keine allgemeingültige Definition. In der Regel wird damit der parallele Einsatz mehrerer Betriebssysteme beschrieben. Detaillierter bedeutet das, dass Hardware-Ressourcen zu einer logischen Schicht zusammengefasst werden und dadurch deren Auslastung optimiert wird. Die Realisierung einer logischen Schicht wird durch einen Hypervisor übernommen (siehe Abschnitt 2.1.3), der zwischen dem Host und dem Gastsystem (Abschnitt 2.1.2) liegt. So kann die logische Schicht bei Aufforderung auf die Ressourcen des Hosts zugreifen und automatisch dem Gastsystem zur Verfügung stellen. Das Prinzip dahinter ist die Verknüpfung von Servern, Speichern und Netzen zu einem virtuellen Gesamtsystem. Daraus können sich darüber liegende Anwendungen direkt und bedarfsgerecht ihre Ressourcen beziehen.

Grundsätzlich wird unterschieden zwischen

1. **Virtualisierung von Hardware**

die sich mit der Verwaltung von Hardware-Ressourcen beschäftigt und

2. **Virtualisierung von Software**

die sich mit der Verwaltung von Software-Ressourcen, wie z.B. Anwendungen und Betriebssystemen beschäftigt.

[Bengel (2008)]

### 2.1.1 Virtuelle Maschine

Eine virtuelle Maschine ist nach Robert P. Goldberg

*'a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor's instructions execute directly on the host processor in native mode'* [Siegert und Baumgarten (2006)]

Wörtlich übersetzt handelt es sich also bei einer virtuellen Maschine, um ein Hardware-/Software-Duplikat eines real existierenden Computersystems, in der eine statistisch dominante Untermenge an virtuellen Prozessoren und Befehlen im Benutzer-Modus auf dem Host-Prozessor ausgeführt werden. Dieses Duplikat kann als Software-Container betrachtet werden, der aus einem vollständigen Satz an emulierten Hardwareressourcen, dem Gastbetriebssystem und entsprechenden Software-Anwendungen besteht. Durch die Containerstruktur wird eine Kapselung hervorgerufen, die es ermöglicht, mehrere virtuelle Maschinen komplett unabhängig auf einem Hostsystem zu installieren und zu betreiben. Ist eine virtuelle Maschine fehlerhaft oder nicht mehr erreichbar, betrifft dies nicht automatisch die restlichen parallel laufenden Maschinen und stellt damit einen der charakteristischen Vorteile von virtuellen Maschinen dar: die Verfügbarkeit. Backup-Systeme oder mehrere Instanzen einer Applikation können so unabhängig von einander auf einem Host ausgeführt werden, ohne sich gegenseitig zu beeinflussen.

Virtuelle Maschinen können ohne größeren Aufwand verschoben, kopiert und auf unterschiedliche Hostserver transferiert werden, um eine optimierte Hardware-Ressourcen-Auslastung zu erhalten. Diese Eigenschaften können von Administratoren genutzt werden um z.B. effizienter Backups zu erstellen, Disaster-Recovery zu betreiben, Umgebungen für Tests/Deployments bereits zu stellen und grundlegende Aufgaben der Systemadministration zu erleichtern, da das Wiederherstellen aus Speicherpunkten oder gespeicherten Abzügen, innerhalb von Minuten zu realisieren ist.

### 2.1.2 Gastbetriebssystem

Der Unterschied zwischen einem üblichen und einem Gastbetriebssystem besteht darin, dass ein übliches Betriebssystem im privilegierten Prozessor-Modus ausgeführt wird, während dies dem Gastbetriebssystem verweigert wird. Der privilegierte Modus (auch Kernel-Mode genannt) befähigt das Betriebssystem die absolute Kontrolle über die vorhandenen Ressourcen zu gewinnen und alle Funktionen des Prozessors nutzen zu können. Anwendungen die im Betriebssystem ausgeführt werden, dürfen nur im Benutzer-Modus arbeiten, der restriktiv auf die Anwendungen wirkt. Ein direkter Zugriff auf Ressourcen wird nur sehr selten und unter

genau kontrollierten Bedingungen gestattet. So wird verhindert, dass laufende Anwendungen durch fehlerhafte Programmierung das System zum Absturz bringen können. (Reuther (2013)) Eine virtuelle Maschine (siehe 2.1.1) läuft als normaler Benutzer-Prozess im Benutzer-Modus. Für das auf der virtuellen Maschine installierte Betriebssystem, das Gastbetriebssystem, hat dies zur Folge, dass es nicht den privilegierten Prozessor-Modus nutzen kann, wie es ein nicht virtualisiertes Betriebssystem könnte. Da weder die Anwendungen noch das entsprechende Gastbetriebssystem Kenntnis darüber haben, dass sie in einer virtuellen Maschine laufen, müssen ausgeführte Instruktionen, die den privilegierten Prozessor-Modus erfordern, entsprechend anders gehandhabt werden. Dies ist unter anderem die Aufgabe des Hypervisors (siehe 2.1.3).

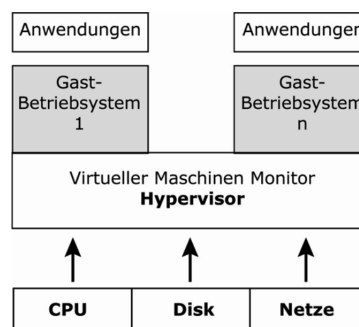


Abbildung 2.1: Betriebssystemvirtualisierung [Siegert und Baumgarten (2006)]

### 2.1.3 Hypervisor

Die Benennung des Hypervisors kann von Hersteller zu Hersteller variieren. Bei Microsoft wird er z.B. Hyper-V genannt, während der Hypervisor bei VMware unter der Bezeichnung ESXi läuft. Der Hypervisor, oder in der Literatur auch VMM (Virtual Machine Monitor) genannt, ist die sogenannte abstrahierende Schicht zwischen der tatsächlich vorhandenen Hardware und den ggf. mehrfach existierenden Betriebssystemen. Zu sehen in Abbildung 2.1. Seine primäre Aufgabe ist die Verwaltung der Host-Ressourcen und deren Zuteilung bei Anfragen der Gastsysteme. Lösen Instruktionen oder Anfragen eines Gastbetriebssystems eine CPU-Exception aus, weil diese im Benutzer-Modus ausgeführt werden, fängt der Hypervisor diese auf und emuliert die Ausführung der Instruktionen (trap and emulate). Die Ressourcen des Hostsystems werden durch den Hypervisor so verwaltet, dass diese bedarfsgerecht zur Verfügung stehen, egal ob ein oder mehrere Gastsysteme laufen. Zu seinen Aufgaben zählen unter anderem E/A-Zugriffe (insbesondere Hardwarezugriffe), Speichermanagement, Prozesswechsel und System-Aufrufe.

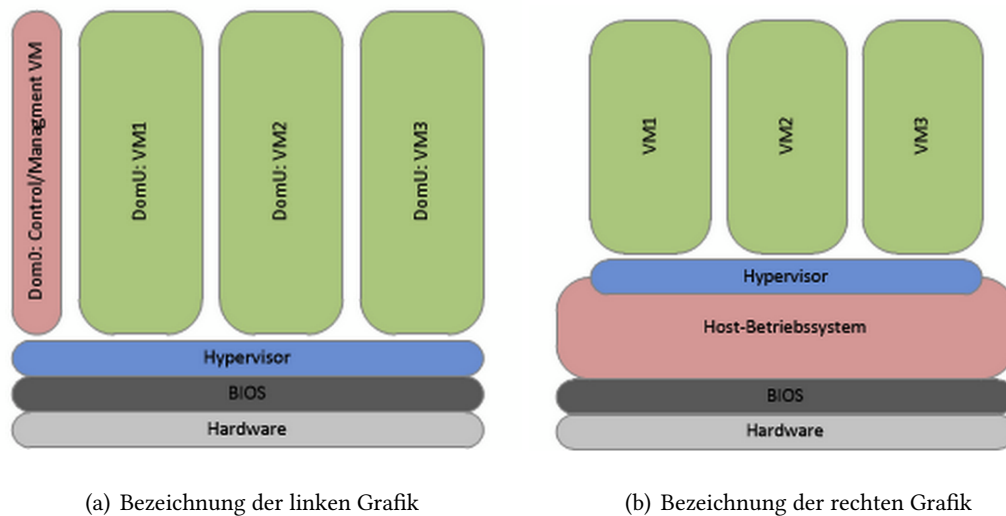


Abbildung 2.2: Hypervisor Typ 1 und 2

Der Hypervisor kann in zwei verschiedene Typen kategorisiert werden:

### Typ 1 Hypervisor

arbeitet direkt auf der Hardware und benötigt kein Betriebssystem, welches zwischen ihm und der Hardware betrieben wird. Alle darüber liegenden virtuellen Maschinen laufen in sogenannten Domains. Weder der Hypervisor noch die anderen Domains sind für die jeweilige Domain sichtbar. Die Verwaltung der laufenden Domains wird durch eine privilegierte Domain geregelt, die in der sogenannten 'Dom0' läuft. Dadurch erhält die privilegierte Domain die Möglichkeit andere Domains zu starten, zu stoppen und zu verwalten.

Der Hypervisor Typ-1 verfügt selbst über die nötigen Gerätetreiber, um den virtuellen Maschinen sowohl CPU, Speicher als auch I/O zur Verfügung zu stellen. Durch die wegfallende Schicht, nämlich dem nicht benötigten Betriebssystem, gewinnt der Hypervisor Typ 1 an Performance und reduziert den Ressourcenverbrauch. Siehe Abbildung [2.1.3.a].

### Typ 2 Hypervisor

lässt durch seine Bezeichnung als 'Hosted' bereits erahnen, dass der Unterschied zu Typ 1 darin besteht, dass er auf einem Hostsystem aufsetzt. Es muss also eine Schicht implementiert sein, die zwischen dem Hypervisor und der Hardware liegt. Siehe Abbildung [2.1.3.b].

Diese Schicht wird durch ein Betriebssystem realisiert, das dem Hypervisor den Zugang

zur Hardware durch die eigenen Hardwaretreiber ermöglicht. Ist das Betriebssystem mit der Hardware kompatibel, ist transitiv gesehen, der Hypervisor ebenfalls mit installiert und ausführbar. Dies vereinfacht die Installation gegenüber dem Hypervisor Typ 1.

Aus Implementierungssicht gibt es für beide Hypervisoren Vor- und Nachteile. Für einige Bereiche ist die Anforderung eines Betriebssystems nur von Vorteil. Vor allem, wenn es um Hardware- und Treiber-Kompatibilität, Konfigurationsflexibilität und vertraute Management-Tools geht.

Auf der anderen Seite kann genau das zum Nachteil gereichen. Es entsteht nicht nur ein höherer Management-Aufwand, um das Betriebssystem zu konfigurieren und zu verwalten, auch die Performance und der Sicherheitsaspekt leiden unter dieser zusätzlichen Schicht für das Betriebssystem.

## 2.2 Konfigurationsmanagement-System

Die Hauptaufgabe eines Konfigurationsmanagement-Systems ist es, eine zuvor definierte Zustandsbeschreibung eines Zielrechners umzusetzen. Dies kann die Installation von Softwarepaketen bedeuten, das Starten oder Beenden von Diensten oder Konfigurationen erstellen/anpassen/entfernen zu lassen. Im Allgemeinen wird dieser Vorbereitungsprozess auch 'provisioning' (zu Deutsch: Provisionieren) genannt und stattet den Zielrechner mit allen Voraussetzungen für seine Aufgaben aus. In der Regel verwenden Konfigurationsmanagement-Systeme eigene Agenten auf den Zielrechnern, über die die Kommunikation abgewickelt und die Zustandsbeschreibung realisiert wird. Neuere Anwendungen wie 'Ansible' (Kapitel 4.3.1) kommen ohne Agenten aus. Die Kommunikation wird im Falle von Ansible über eine SSH-Schnittstelle realisiert. Pull-basierte Tools, wie beispielsweise 'Puppet' (Kapitel 4.3.3), fragen in regelmäßigen Abständen ein zentrales Konfigurations-Repository ab, in dem die jeweils aktuelle Zustandsbeschreibung der Maschine gespeichert ist. Sie sorgen dafür, dass die Änderungen auf dem Ziel-Client ausgeführt werden. Es spielt keine Rolle, ob das Ziel eine virtuelle Maschine oder eine standard Hardware-Maschine ist. Konfigurationsmanagement-Systeme sind in der Regel dazu fähig, ganze Gruppen an Rechnern parallel zu bearbeiten und die entsprechenden Zustandsbeschreibungen umzusetzen. In dem bereits genannten Beispiel 'Ansible', können mehrere Rechner simpel in Inventory-Dateien (siehe Listing 2.1) als Gruppen zusammengefasst werden. Jede Gruppe erhält einzeln die entsprechend vorgesehenen Zustandsbeschreibungen.

Listing 2.1: Beispiel Inventory-Datei

```
1 [atomic]
2 192.168.100.100
```

```
3 192.168.100.101  
4 [webserver]  
5 192.168.1.110  
6 192.168.1.111
```

Die Integration und Anwendung von Konfigurationsmanagement-Systemen beschleunigt also nicht nur den Aufbau eines oder mehrerer Zielrechner, sondern hilft auch bei der Organisation der Softwareverteilung.

## 3 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.

### 3.1 Zielsetzung

’Keine Systementwicklung sollte ohne wenigstens eine halbe Seite schriftliche Ziele angegangen werden. Dabei ist es wichtig, quantifizierbare Angaben aufzuzählen, denn Ziele sind Anforderungen auf einer abstrakten Ebene.’ **Rupp und die SOPHISTen (2014)**

Die zu erstellende Applikation (in den folgenden Kapiteln auch als VM-Builder bezeichnet) soll den Anwender in der Umsetzung und Konfiguration von virtuellen Entwicklungsumgebungen unterstützen. Angestrebte Funktionalitäten, wie der Aufbau einer Entwicklungsumgebung inklusive der automatisierten Installation von Programmen, oder der Austausch von bereits erstellten Entwicklungsumgebungen zwischen beteiligten Benutzern, soll den Anwender in seiner Tätigkeit unterstützen. Dabei spielen das User-Interface und der Funktionsumfang der VM-Builder eine entscheidende Rolle. Während die Strukturierung des User-Interfaces hilft, sich mit geringem Zeitaufwand in die Applikation einzuarbeiten, vereinfacht ein übersichtliches Konfigurationsspektrum die Erstellung der gewünschten virtuellen Umgebung. Die Konfiguration einer virtuellen Maschine muss auch für unerfahrene Nutzer möglich sein und keine speziellen Kenntnisse voraussetzen. Alle virtuellen Maschinen, die zu einem Zeitpunkt aktiv sind, sollten in getrennten Instanzen laufen und voneinander unterscheidbar sein. Die Unterscheidbarkeit soll Funktionen wie den Export oder den entfernten Zugriff auf die Maschine unterstützen. Sie soll dem Anwender die Möglichkeit schaffen, die gewünschte virtuelle Maschine zu beeinflussen, indem er die Umgebung abschalten oder zerstören kann. Des Weiteren soll der VM-Builder Unterstützung bieten, voneinander abhängige Applikationen zu konfigurieren und automatisiert zu installieren.



## 3.2 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 3.1: Stakeholder

## 3.3 Funktionale Anforderungen

### Anforderungen - Anwender

- FA 1. Die Applikation muss über den Browser ausführbar sein, ohne zusätzliche lokale Installationen auf Anwenderseite.
- FA 2. Die Applikation muss dem Anwender die Möglichkeit bieten, eine virtuelle Maschine zu erstellen.
- FA 3. Möchte der Anwender auf der virtuellen Maschine Software installiert haben, sollte die Applikation ihm Softwarekomponenten zur Auswahl vorschlagen.
- FA 4. Falls der Anwender keine zusätzliche Software auf der virtuellen Maschine haben möchte, muss die Applikation entsprechend damit umgehen können.
- FA 5. Möchte der Anwender zusätzlich Dateien auf die zu erstellende virtuelle Maschine übertragen, ermöglicht die Applikation eigene Dateien auszuwählen.

- FA 6. Möchte der Anwender eine virtuelle Maschine nach dem Abbild einer bereits im System vorhandenen virtuellen Maschine erstellen, bietet die Applikation Optionen dafür an.
- FA 7. Die Applikation sollte durch den Erstellungsprozess die Konfiguration der virtuellen Maschinen automatisch speichern können.
- FA 8. Wenn der Anwender eine virtuelle Maschine erstellen möchte, muss die Applikation bei wichtigen Konfigurationsschritten, für den Benutzer sichtbare Statusmeldungen anzeigen.
- FA 9. Treten Fehler bei der Erstellung einer virtuellen Maschine auf, muss das System eine Fehlermeldung ausgeben.
- FA 10. Die Applikation sollte dem Anwender die Option bieten, virtuelle Maschinen zu exportieren.
- FA 11. Ist der Export durchgeführt worden, muss die Applikation dies mit einer Meldung auf dem Bildschirm bestätigen.
- FA 12. Die Anwendung muss fähig sein, Exporte wieder importieren zu können. Falls während des Imports Datenfehler auftreten, muss die Anwendung den jeweiligen Fehler (Fehlerbeschreibung) auf dem Bildschirm ausgeben.
- FA 13. Ist der Import erfolgreich durchgeführt worden, soll die Applikation eine entsprechende Meldung anzeigen.
- FA 14. Die Applikation soll fähig sein, anderen Anwendern bereits erstellte virtuelle Maschinen über das Internet und das lokale Netzwerk zur Verfügung zu stellen.
- FA 15. Möchte der Anwender eine bereits erstellte und laufende virtuelle Maschine beenden, muss die Applikation dafür eine entsprechende Option bieten.
- FA 16. Falls der Anwender eine bereits erstellte virtuelle Maschine löschen möchte, muss die Applikation ihm dafür ein Hilfsmittel bereitstellen.

#### **Anforderungen - Administrator**

- FA 17. Falls während des Betriebes der Anwendung Änderungen an der Konfiguration durchgeführt werden müssen, soll die Applikation dies über eine Konfigurationsseite anbieten.
- FA 18. Zu den Konfigurationsmöglichkeiten soll das Ändern des Namens, sowie des Speicherorts von Logdateien gehören.

- FA 19. Falls in den Einstellungen ein Fehler während der Bearbeitung auftritt, muss die Anwendung eine Fehlermeldung auf dem Bildschirm ausgeben.
- FA 20. Die Anwendung sollte dem Administrator die Option bieten, sich den Inhalt von Logdateien anzeigen zu lassen.
- FA 21. Für das Hinzufügen, Ändern und Löschen von Softwarekomponenten, muss die Applikation eine Oberfläche bereitstellen.
- FA 22. Die Applikation muss eine Oberfläche anbieten, in der Softwarepakete konfiguriert werden können. Es muss dort möglich sein, ein neues Softwarepaket anzulegen, Relationen zu anderer Software herzustellen und ggf. Dateien auszuwählen und dies als Paket zu speichern.
- FA 23. Das Erstellen, Ändern und Löschen von Softwarepaketen muss als Option in der Applikation angeboten werden.

Definition Funktionale Anforderungen nach **Rupp und die SOPHISTen (2014)**

## 3.4 Use-Cases

Use-Cases (Anwendungsfälle) helfen, die fachlichen Anforderungen eines Systems zu bezeichnen, indem dort Interaktionen zwischen System und Benutzer dargestellt werden und das System grob in seine Hauptfunktionen gegliedert wird. Der Business-Use-Case spiegelt dabei ein Gesamtbild aller Funktionalitäten der Applikation wieder und grenzt diese voneinander ab, während die darauf folgenden System-Use-Cases helfen eine erste Skizze der zu entwickelnden Hauptfunktionalitäten zu erstellen, die sich wie folgt aus den funktionalen Anforderungen in Kapitel 3.3 ergeben haben:

1. Erstellung einer virtuellen Maschine
2. Export einer vorhandenen Maschine
3. Import einer zuvor erstellten Maschine
4. Eine virtuelle Maschine für andere Anwender zugreifbar machen (teilen)
5. Software-Abhängigkeiten konfigurieren

### 3.4.1 Business-Use-Case

**TODO:** Provisioning klein; Ändern, ANwendungseinstellung bearbeiten

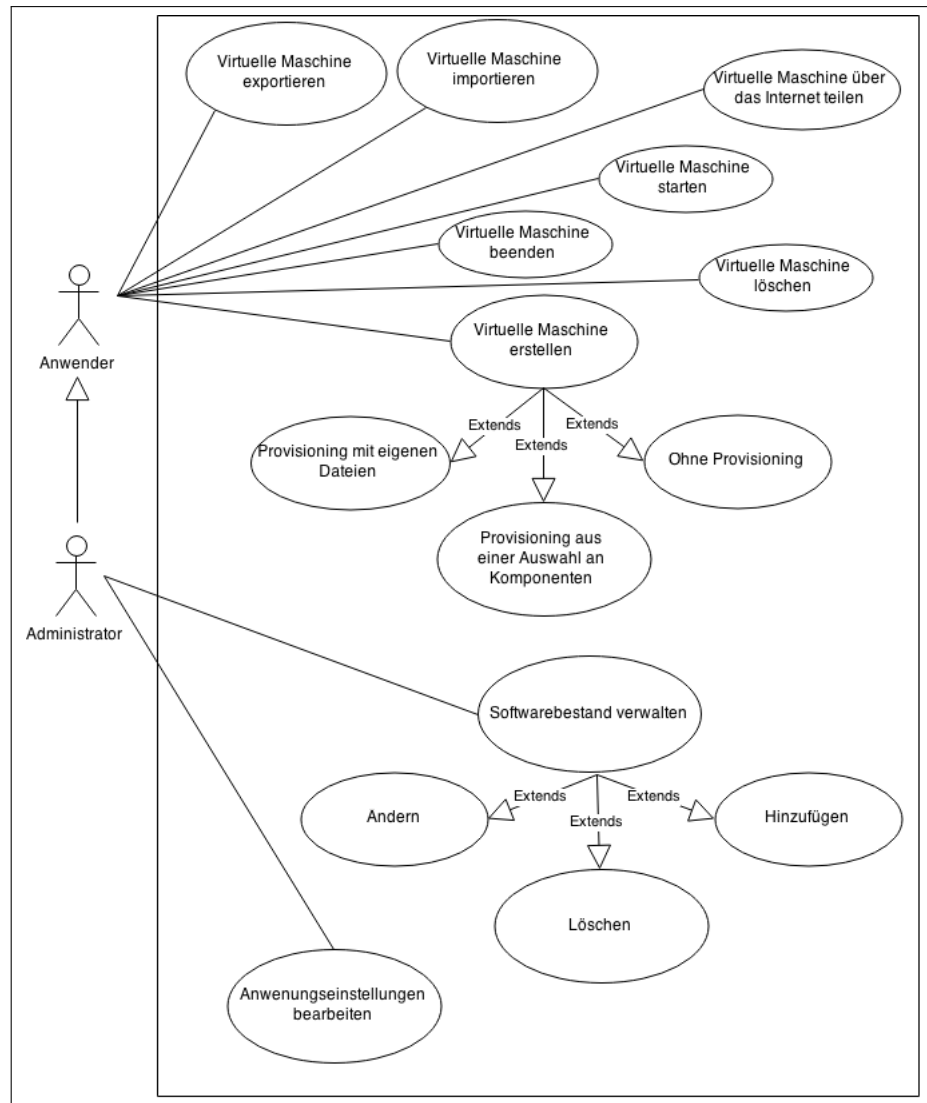


Abbildung 3.2: Business-Use-Case Übersicht

### 3.4.2 System-Use-Case

#### Use-Case 1 - Virtuelle Maschine erstellen

<b>Bezeichnung</b>	Virtuelle Maschine erstellen
<b>Ziel im Kontext</b>	Erstellung einer virtuellen Maschine.
<b>Akteur</b>	Anwender
<b>Auslöser</b>	Der Anwender möchte eine virtuelle Maschine erstellen.
<b>Vorbedingung</b>	Die Anwendung ist installiert und lauffähig. Der Anwender kann auf die Benutzeroberfläche zugreifen.
<b>Nachbedingung</b>	Der Anwender kann auf eine lauffähige virtuelle Maschine zugreifen, die seiner Konfiguration entspricht.
<b>Anforderungen</b>	FA 2, FA 3, FA 4, FA 5 , FA 7, FA 8, FA 9
<b>Erfolgsszenario</b>	

1. Der Anwender startet den VM-Builder über den Browser.
2. Der Anwender wählt aus, dass er eine neue virtuelle Maschine erstellen möchte.
3. Der Anwender wird gebeten Konfigurationsparameter für die zu erstellende Maschine einzugeben.
4. Die Applikation schlägt dem Anwender vor, Software mit auf die gewünschte Maschine zu installieren.
5. Der Anwender kann optional eigene Dateien mit übertragen lassen.
6. Nach der entsprechenden Auswahl zeigt die Applikation den aktuellen Aufbaustatus der virtuellen Maschine.
7. Die Applikation zeigt dem Anwender die Zugriffsmöglichkeiten für die Maschine. an

#### Use-Case 2 - Virtuelle Maschine exportieren

<b>Bezeichnung</b>	Virtuelle Maschine exportieren
<b>Ziel im Kontext</b>	Export aller notwendigen Konfigurationsdateien, um eine Maschine mit der gleichen Konfiguration erneut erstellen zu können.
<b>Akteur</b>	Anwender
<b>Auslöser</b>	Der Anwender möchte eine virtuelle Maschine exportieren.
<b>Vorbedingung</b>	Die zu exportierende virtuelle Maschine existiert bereits.
<b>Nachbedingung</b>	Der Anwender erhält eine gepackte Datei, in der alle nötigen Daten enthalten sind, die für einen erneuten Import nötig wären.
<b>Anforderungen</b>	FA 11, FA 12
<b>Erfolgsszenario</b>	

1. Der Anwender startet den VM-Builder über den Browser.
2. Der Anwender wählt bei der gewünschten Maschine die Exportfunktion aus.
3. Die Applikation beginnt mit dem Download der benötigten Dateien.

#### Use-Case 3 - Virtuelle Maschine importieren

<b>Bezeichnung</b>	Virtuelle Maschine importieren
<b>Ziel im Kontext</b>	Erstellung einer Maschine aus einem Import.
<b>Akteur</b>	Anwender
<b>Auslöser</b>	Der Anwender möchte eine virtuelle Maschine importieren.
<b>Vorbedingung</b>	Die Anwendung ist installiert und lauffähig. Der Anwender kann auf die Benutzeroberfläche zugreifen.
<b>Nachbedingung</b>	Die Maschinenkonfiguration konnte importiert werden und eine virtuelle Maschine wurde erstellt.
<b>Anforderungen</b>	FA 12, FA 13
<b>Erfolgsszenario</b>	

1. Der Anwender startet den VM-Builder über den Browser.
2. Der Anwender wählt die Importfunktion aus und kann die gewünschte(n) Datei(en) hochladen.
3. Die Applikation zeigt dem Anwender, dass der Import erfolgreich war.

4. Der Anwender kann nun entscheiden, ob er die virtuelle Maschine erstellen lassen möchte.

#### Use-Case 4 - Virtuelle Maschine teilen

<b>Bezeichnung</b>	Virtuelle Maschine teilen
<b>Ziel im Kontext</b>	Eine erstellte Maschine über das Internet oder das lokale Netzwerk für andere Anwender zugreifbar machen.
<b>Akteur</b>	Anwender
<b>Auslöser</b>	Der Anwender möchte eine virtuelle Maschine für andere Anwender zugreifbar machen.
<b>Vorbedingung</b>	Die Anwendung ist installiert und lauffähig Die zu teilende Maschine ist erstellt.
<b>Nachbedingung</b>	Die virtuelle Maschine ist von intern und/oder extern zugreifbar.
<b>Anforderungen</b>	FA 14
<b>Erfolgsszenario</b>	

1. Der Anwender startet den VM-Builder über den Browser.
2. Der Anwender wählt die gewünschte virtuelle Maschine aus und aktiviert die Teil-Option.
3. Die Applikation zeigt dem Anwender die Zugriffsmöglichkeiten auf die virtuelle Maschine.

#### Use-Case 5 - Softwarepakete konfigurieren

<b>Bezeichnung</b>	Softwarepakete konfigurieren
<b>Ziel im Kontext</b>	Softwarepakete konfigurieren, die andere Software und ggf. das Kopieren/Entpacken von Dateien beinhaltet.
<b>Akteur</b>	Administrator
<b>Auslöser</b>	Eine Installation einer Anwendung, die mehrere Softwarekomponenten benötigt.
<b>Vorbedingung</b>	Wissen über die zu installierenden Softwarekomponenten.
<b>Nachbedingung</b>	Ein Softwarepaket, dass die Konfigurationen des Anwenders beinhaltet und in der Applikation anwendbar ist.
<b>Anforderungen</b>	FA 21, FA 22, FA 23
<b>Erfolgsszenario</b>	

1. Der Administrator startet den VM-Builder über den Browser.
2. Der Administrator begibt sich in das Administrationsmenü.
3. Der Administrator kann dort ein neues Softwarepaket erstellen.
4. Der Administrator gibt die entsprechenden Optionen wie z.B. Name des Softwarepakets ein.
5. Der Administrator wählt die Komponenten aus, die mit installiert werden sollen.
6. Der Administrator wählt Dateien aus, die auf den Zielrechner kopiert und/oder entpackt werden sollen.
7. Der Administrator speichert das Paket und kann es bei der Installation einer virtuellen Maschine auswählen.

### 3.5 Nichtfunktionale Anforderungen

Die Literatur gibt keine einheitliche Definition von nichtfunktionalen Anforderungen vor, aber wie (Burge und Brown (2002)) es ausdrückte, können durch nichtfunktionale Anforderungen neue Lösungsmöglichkeiten vorgegeben werden oder schlicht die Menge an potentiellen Designentwürfen, im Bezug auf die Funktionalitäten, reduziert werden. Im Wesentlichen gibt es eine begrenzte Auswahl an Definitionen und Perspektiven bei nichtfunktionalen Anforderungen, die im Folgenden nach Rupp und die SOPHISTen (2014) zusammengefasst werden.



#### 1. Technologische Anforderungen

Die detailliertere Beschreibung von Lösungsvorgaben oder der Umgebung, in der das System betrieben werden soll, können und sollen den Lösungsraum, für die Realisierung des Systems beschränken.

#### 2. Qualitätsanforderungen

Qualitätsanforderungen lassen sich in detailliertere Unterpunkte unterteilen. Dies kann nach zwei Standards erfolgen: ISO 25000 und ISO 9126, mittlerweile ist jedoch der ISO 9126 Standard in ISO 25000 aufgegangen. Beide Standards sollen sicherstellen, dass über Qualitätsanforderungen die Qualität des Systems und des Entwicklungsprozesses festgelegt wird.

#### 3. Anforderungen an die Benutzeroberfläche

Die Anforderungen, die festlegen, wie sich die Anwendung dem Benutzer darstellt, werden unter 'Anforderungen an die Benutzeroberfläche' gebündelt.

#### 4. Anforderungen an die sonstigen Lieferbestandteile

Alle Produkte, die zu dem System oder der Anwendung geliefert werden müssen, wie z.B. ein Handbuch oder Quellcode, werden unter 'Anforderungen an die sonstigen Lieferbestandteile' beschrieben.

#### 5. Anforderungen an durchzuführende Tätigkeiten

Die 'Anforderungen an durchzuführende Tätigkeiten' beeinflussen Tätigkeiten, wie Wartung oder Support, die der Entwicklung nachgelagert sind.

#### 6. Rechtliche-vertragliche Anforderungen

'Rechtliche-vertragliche Anforderungen' beschreiben die Regelungen, die zwischen Auftraggeber und Auftragnehmer vor der Entwicklung des System oder deren Anwendung, festgelegt werden müssen.

Im Folgenden werden die nichtfunktionalen Anforderungen hinsichtlich der Punkte **1** 'Technologische Anforderungen' und **3** 'Anforderungen an die Benutzeroberfläche' betrachtet, da diese Zielführend für die Entwicklung der angestrebten Applikation sind.

#### **Technologische Anforderungen**

1. Das für den Betrieb der Anwendung zugrunde liegende Betriebssystem muss Ubuntu 12.04 oder höher sein.
2. Die Anzahl der gleichzeitig laufenden virtuellen Umgebungen, liegt maximal bei 10.
3. Die Kommunikation zwischen Frontend und Backend muss nicht verschlüsselt ablaufen.
4. Softwareupdates der benutzten Softwarekomponenten müssen mit Rücksprache des Entwicklers erfolgen.

#### **Qualitätsanforderungen**

1. Die Installation und Inbetriebnahme der Anwendung sollte über einen automatischen Installationsprozess erfolgen.
2. Die Anwendung sollte zu 99.0 Prozent der Zeit lauffähig sein.
3. Jeder auftretende Fehler ist eindeutig identifizierbar und nachvollziehbar.
4. Änderungen am vorgeschlagenen Softwarebestand müssen innerhalb von <10 Sekunden in der Anwendungsoberfläche sichtbar sein.
5. Falls das Betriebssystem auf eine höhere Version aktualisiert werden soll, muss dies ohne Änderungen des Quellcodes der Anwendung vorgenommen werden können.
6. Soll ein anderer Provisionierer verwendet werden, muss der Aufwand des Austausches bei unter fünf Personentagen liegen.
7. Wird angedacht weitere Grundfunktionalitäten zu implementieren, soll dies möglichst einfach erfolgen.
8. Das Importieren von virtuellen Maschinen sollte <10 Minuten betragen.
9. Der Export einer virtuellen Maschine sollte <5 Minuten betragen.

#### **Anforderungen an die Benutzeroberfläche**

1. Ein Benutzer ohne Vorkenntnisse muss bei der erstmaligen Verwendung des VM-Builders innerhalb von maximal 10 Minuten in der Lage sein, die gewünschte Funktionalität zu lokalisieren und zu verwenden.

2. Der VM-Builder sollte einen Oberflächendialog für die Verwaltung der virtuellen Maschinen bereitstellen. Die folgenden Bezeichnungen, die Art der abgebildeten Elemente, deren Größe und Anordnung sind keine Umsetzungsanforderung. (Siehe Abbildung 3.3). Primär steht eine Übersicht der erstellten virtuellen Maschinen im Vordergrund, inklusive derer verfügbaren Optionen und Statusinformationen.

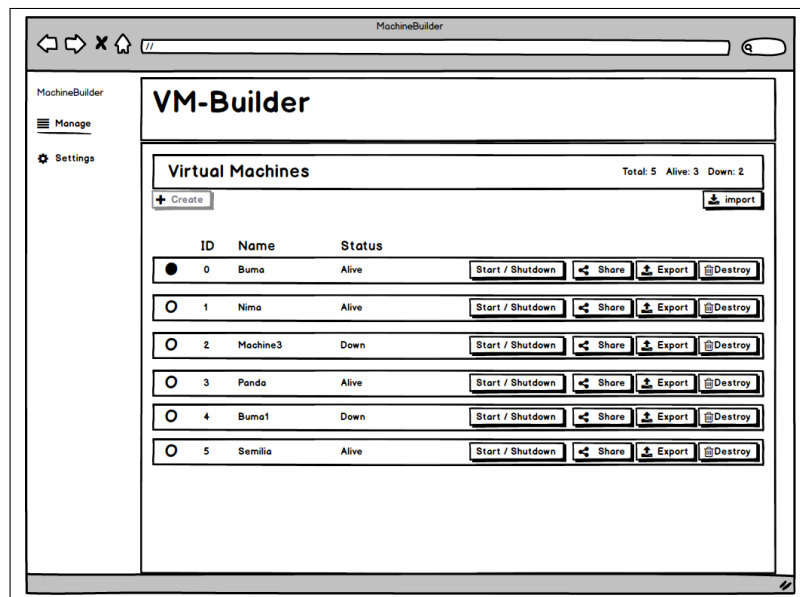
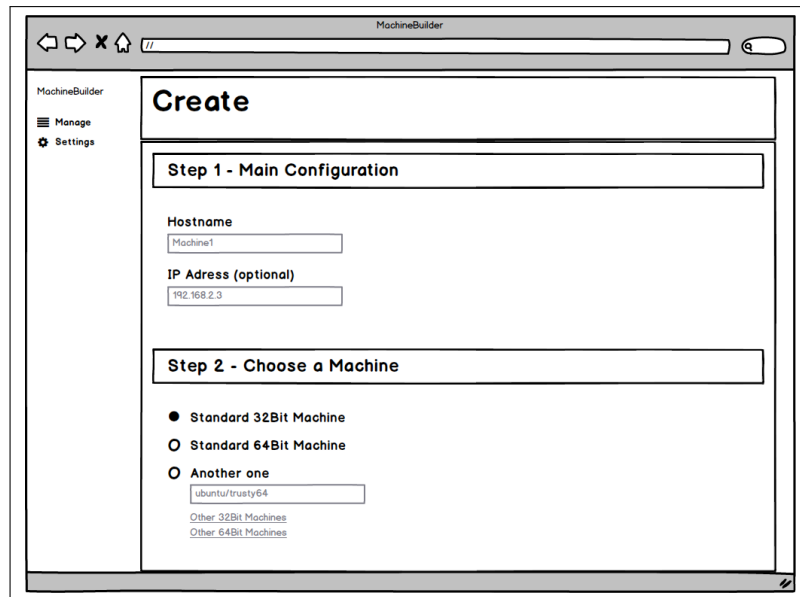


Abbildung 3.3: Entwurf der Verwaltungsoberfläche

3. Der VM-Builder sollte den Oberflächendialog 'Virtuelle Maschine erstellen' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung 3.4 und 3.5). Keine umzusetzenden Anforderungen sind die genaue Größe und die Anordnung der einzelnen Elemente. Abbildung 3.4 zeigt die Menüführung des geleiteten Aufbaus einer virtuellen Maschine mit Eingabemöglichkeiten. Abbildung 3.5 zeigt den nachfolgenden Auswahlprozess an Softwarekomponenten. Zu beachten ist, dass die Eingabe der optionalen- und Pflichtparameter gegeben ist, sowie die Auswahlmöglichkeit an Softwarekomponenten.



The image shows a web browser window titled "MachineBuilder". On the left is a sidebar with "MachineBuilder", "Manage", and "Settings" (selected). The main content area is titled "Create" and contains two steps:

**Step 1 - Main Configuration**

Hostname:

IP Address (optional):

**Step 2 - Choose a Machine**

☒ Standard 32Bit Machine

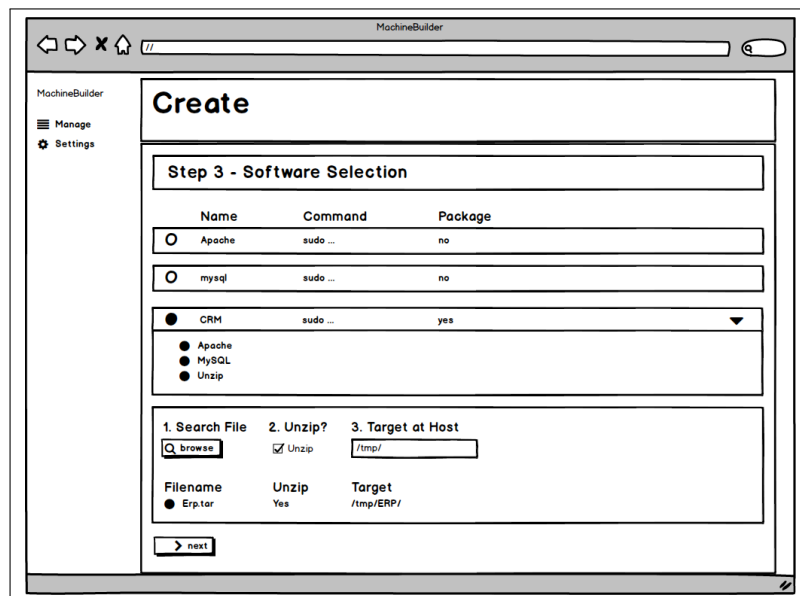
☐ Standard 64Bit Machine

☐ Another one

[Other 32Bit Machines](#)

[Other 64Bit Machines](#)

Abbildung 3.4: Entwurf der Aufbauoberfläche - Eingabe Parameter



The image shows the same web browser window, now at "Step 3 - Software Selection".

**Step 3 - Software Selection**

Name	Command	Package
<input type="radio"/> Apache	sudo ...	no
<input type="radio"/> mysql	sudo ...	no
<input checked="" type="radio"/> CRM	sudo ...	yes

Below the table are three sections:

1. Search File:

2. Unzip?: ☒ Unzip

3. Target at Host:

Filename: ☐ Erp.tar

Unzip: Yes

Target: /tmp/ERP/

Abbildung 3.5: Entwurf der Aufbauoberfläche - Auswahl der Softwarekomponenten

4. Der VM-Builder sollte den Oberflächendialog 'Sharing' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung 3.6). Keine umzusetzenden Anforderungen sind die genaue Größe und die Anordnung der einzelnen Elemente. Um die Oberfläche übersichtlich zu halten, soll der Anwender nur nötige Informationen erhalten, die er zum Weitergeben an Dritte benötigt. Dies wären z.B. Konnektivitätsmöglichkeiten und Informationen zur geteilten virtuellen Maschine.

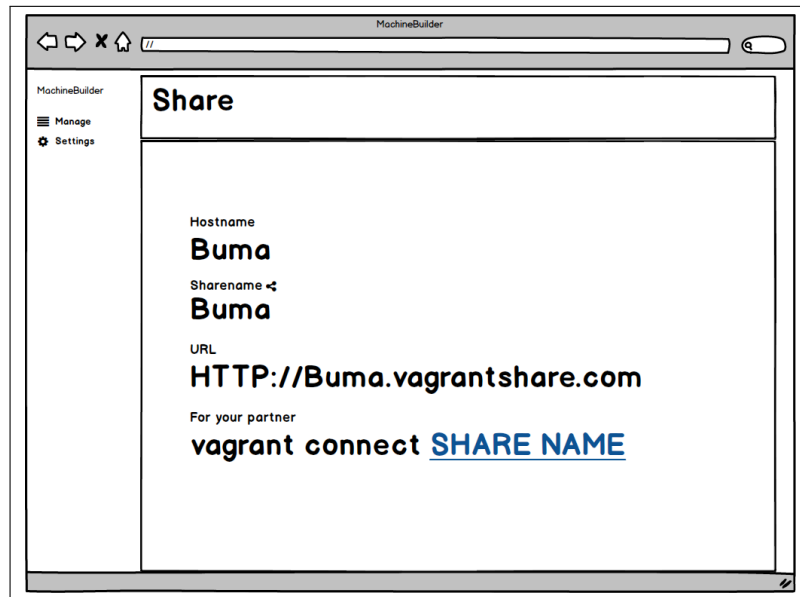


Abbildung 3.6: Entwurf der Sharing-Oberfläche

5. Der VM-Builder sollte den Oberflächendialog 'Export' besitzen. Durch den Export sollen, von einer zuvor ausgewählten virtuellen Maschine, alle nötigen Konfigurationsdateien exportiert werden können. Exemplarisch in Abbildung 3.7 zu sehen.

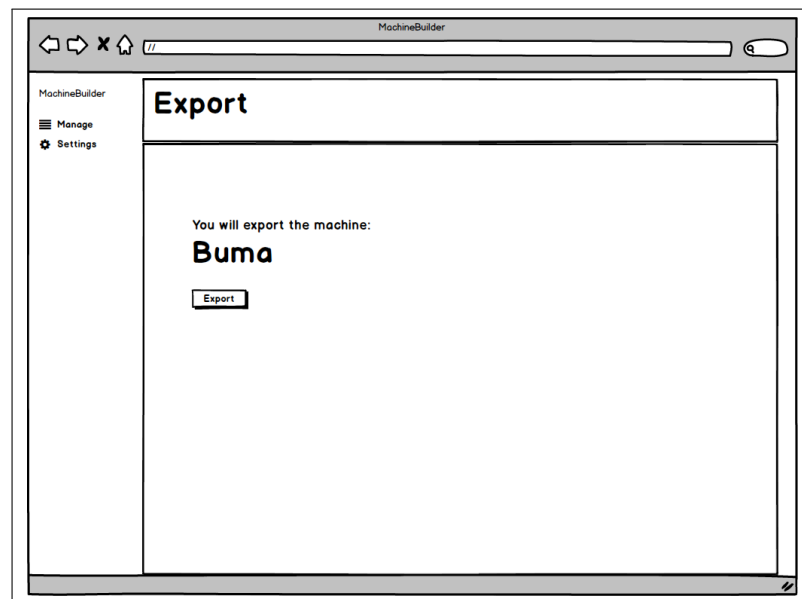


Abbildung 3.7: Entwurf der Export-Oberfläche

6. Der VM-Builder sollte den Oberflächendialog 'Import' besitzen, die den Anwender befähigt zuvor exportierte Dateien wieder zu importieren und daraus eine virtuelle Maschine zu erstellen. So sollen Schaltflächen zum Auswählen der benötigten Dateien angeboten werden. Eine exemplarische Sicht der Oberfläche ist unter [Abbildung 3.8](#) zu finden.

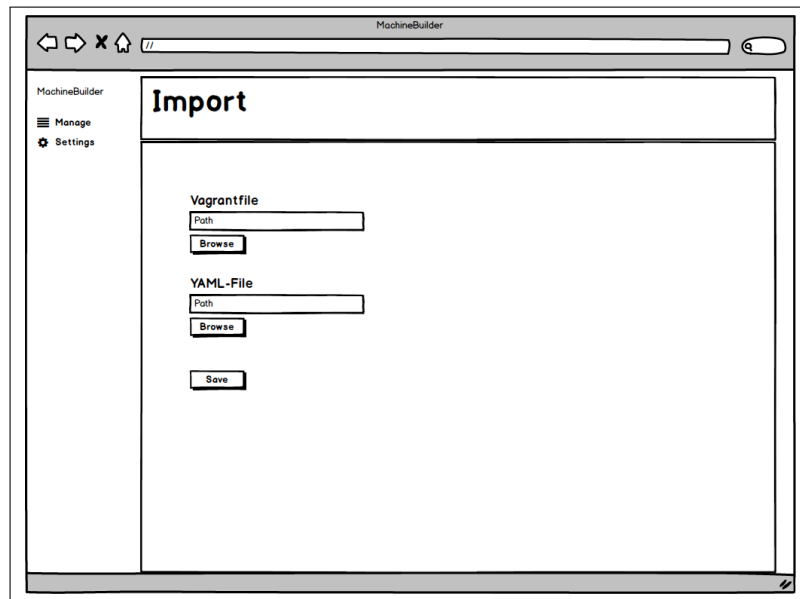


Abbildung 3.8: Entwurf der Import-Oberfläche

7. Der VM-Builder sollte den Oberflächendialog 'Einstellung' besitzen. Die Einstellungen sollen dem Anwender eine generelle Übersicht über Applikations-Konfigurationen geben. Zudem soll dort der Inhalt von Log-Dateien einsehbar sein und Softwarekomponenten / Softwarepakete hinzugefügt werden können. Abschnitt 3.3 zeigt die einzelnen Konfigurationsseiten des Einstellungsmenüs als Vorschlag zur Umsetzung.

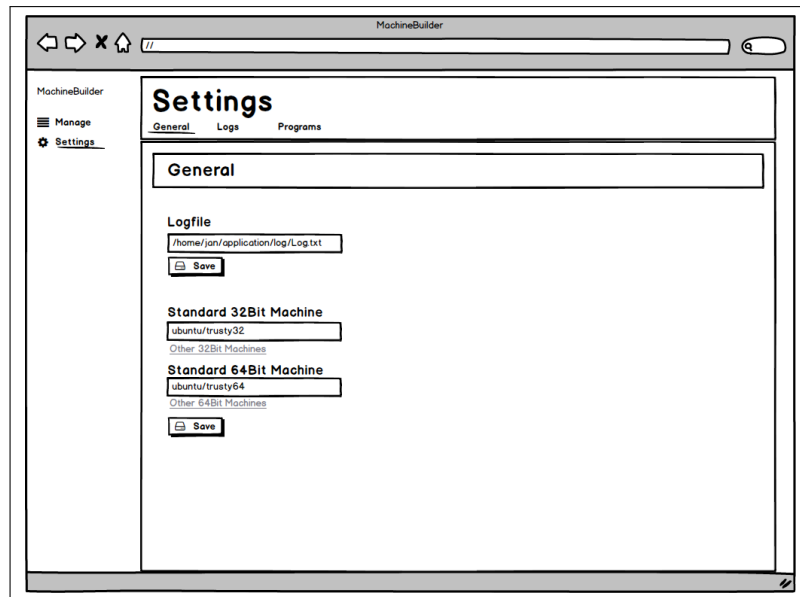


Abbildung 3.9: Entwurf der Einstellungen - Generelle Konfiguration

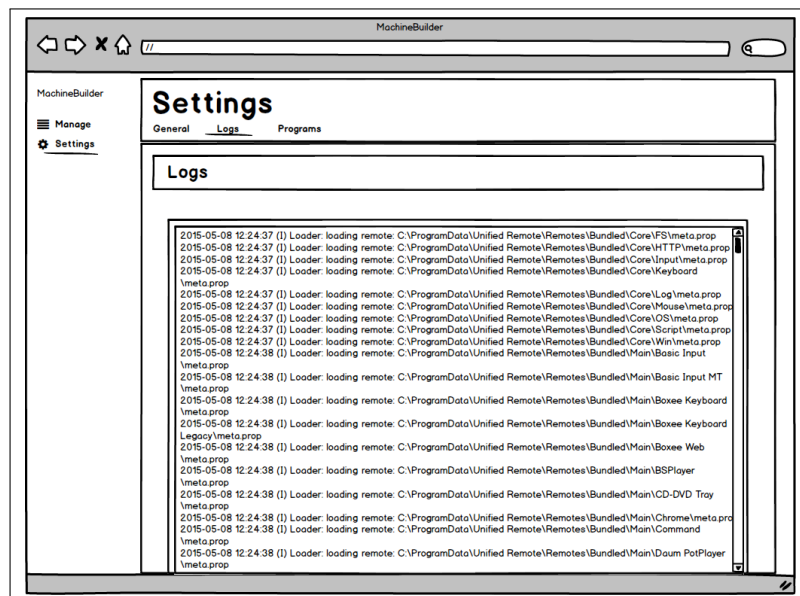


Abbildung 3.10: Entwurf der Einstellungen - Logdatei



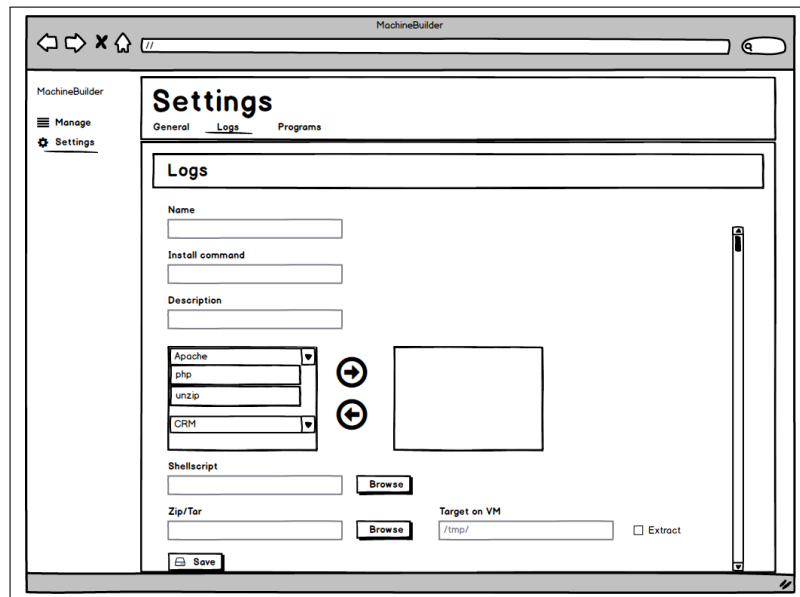


Abbildung 3.11: Entwurf der Einstellungen - Softwarekomponenten hinzufügen

## 3.6 Randbedingungen

Um anwendungs- und problembezogene Entwurfsentscheidungen treffen zu können, werden Faktoren analysiert, die die Architekturen der Anwendung beeinflussen oder einschränken können. Dies geschieht über die im Vorfeld formulierten Anforderungen. Laut **Starke (2014)** werden diese Einflussfaktoren und Randbedingungen in folgende Kategorien unterteilt:

- Organisatorische und politische Faktoren: Manche solcher Faktoren wirken auf ein bestimmtes System ein, während andere eine Vielzahl an Projekten eines Unternehmens beeinflussen können. (**Rechtin und Maier, 2000**) charakterisiert diese Faktoren als 'facts of life'.
- Technische Faktoren: Durch sie wird das technische Umfeld geprägt und entsprechend eingeschränkt. Sie betreffen nicht nur die Ablaufumgebung des zu entwickelnden Systems, sondern umfassen auch die technischen Vorgaben für die Entwicklung, einzusetzender Software und vorhandener IT-Systeme.

Da weder die organisatorischen, noch die politischen Faktoren auf dieses Projekt Einfluss haben, werden diese nicht weiter betrachtet.

#### 3.6.1 Technische Randbedingungen

Randbedingung	Erklärung
Server Hardware	Die Leistung des Servers sollte entsprechend der Anforderungen genügen. Es muss möglich sein, mehrere virtuelle Maschinen gleichzeitig laufen zu lassen, ohne dass es die einzelnen Maschinen beeinflusst.
Server Betriebssystem	Die Lösung sollte auf einem Ubuntu Server 64Bit installiert und betrieben werden.
Implementierung in Ruby	Implementierung der Anwendung erfolgt in Ruby mit dem Framework Sinatra. Da keine separate Frontend-Kommunikation benötigt wird, bietet sich Ruby als Backend-Sprache an. Entwickelt wird in der Version 1.9, welche als stabil gilt.
Fremdsoftware	Fremdsoftware die hinzugezogen wird, sollte idealerweise frei verfügbar und kostenlos sein. Es sollte darauf geachtet werden, dass die Software etabliert ist und der Fokus auf eine stabile Version gelegt wird.
Web-Frontend	Die Kompatibilität zu allen gängigen Browsern wird nicht weiter betrachtet. Optimiert wird ausschließlich für Mozilla Firefox.

Abbildung 3.12: Technische Randbedingungen

### 3.7 Zusammenfassung

In der Anforderungsanalyse wurde in Abschnitt 3.1 ein Überblick über die Ziele des VM-Builders geschaffen. Danach wurden die Stakeholder inklusive ihrer Beschreibung und Ziele definiert, wodurch die Analyse der funktionalen Anforderungen in Abschnitt 3.3 strukturiert wurde. Das Ergebnis der funktionalen Anforderung stellte die Kernpunkte der Applikation heraus, die sich wie folgt ergaben.

1. Die Erstellung einer virtuellen Maschine
2. Der Export von Maschinen
3. Der aus dem Export resultierende Import
4. Die Möglichkeit virtuelle Maschinen mit anderen zu teilen (Sharing)
5. Die Konfiguration von Softwarepaketen, um Abhängigkeiten zu anderen Softwarekomponenten zu definieren.

Diese Punkte wurden in Abschnitt 3.4 als Use-Caseszenarien detaillierter in den Kontext des VM-Builders gestellt. Die nichtfunktionalen Anforderungen (siehe Abschnitt 3.5) bestimmen die Qualitätsmerkmale der Applikation und ergaben die ersten Anforderungen an die Oberfläche (siehe Abschnitt 3). Zuletzt wurden die Randbedingungen betrachtet, um Einfluss- und einschränkende Faktoren zu finden, die sich auf die Architektur der Applikation beziehen könnten. Die insgesamt geschaffenen Anforderungen werden im folgenden Kapitel 4. ('Evaluation') als Grundlage für die Recherche nach Produkten verwendet, die für den VM-Builder nutzbar sein könnten. Die resultierenden Ergebnisse aus der Anforderungsanalyse und der Evaluation schaffen so wiederum die Basis für den Entwurf in Kapitel 5.

## 4 Evaluation

Um herauszufinden, ob und welche Softwareprodukte der Anwendung hinzugefügt werden können, kann eine Evaluation des Marktes hilfreich sein. Es ist zu eruieren, ob gewünschte Funktionalitäten unter Verwendung der technischen Randbedingungen aus Kapitel 3.6.1 - Unterpunkt 'Femdssoftware', von externen, etablierten, sowie kostenlosen Softwarekomponenten unterstützt oder sogar durchgeführt werden können. Da es bereits Anwendungen gibt, die sich mit der Verwaltung und dem Aufbau von virtuellen Maschinen befassen, werden im ersten Schritt zwei Produkte vorgestellt, die als Open-Source erhältlich sind.

### 4.1 Vergleichbare Anwendungen

Der Markt an Verwaltungs- und Administrationssoftware ist über die Jahre immer größer geworden. Besonders nach dem auftretenden Trend, alles über die Cloud steuern und administrieren zu können. Das wiederum hat diverse Vorteile für Administratoren, da ihnen teils lästige und zeitaufwendige Konfigurationen erspart werden. Die Open-Source Community hat ebenfalls ihren Beitrag dazu geleistet und diverse Lösungen hervorgebracht, wovon zwei im Folgenden genauer betrachtet werden.

#### 4.1.1 OpenNebula

OpenNebula ist eine Open-Source-Cloud Lösung, die sich selbst als ein 'Open-Source-Industriestandard zur Virtualisierung von Rechenzentren' bezeichnet (Hock (2012)) und hochverfügbare, skalierbare Virtualisierungsplattformen zur Verfügung stellt. Als 'Infrastructure as a Service' (kurz IaaS) kann OpenNebula die zentrale Administration und Überwachung von virtuellen Systemen übernehmen. Kommt es zu Ausfällen von Komponenten, kann OpenNebula sich um den Neustart auf einem anderen Hostsystem kümmern. Zudem sind dynamische Erweiterungen der Hardwareressourcen möglich. Die Kernkompetenzen liegen dabei auf schon vorhandenen Systemen im Bereich Virtualisierung, Speicherung und Vernetzung. In erster Linie kann OpenNebula als Werkzeug zur Integration vorhandener Systeme gesehen werden, da es Möglichkeiten zur Wartung und Skalierung der gegebenen Infrastruktur zur Verfügung

stellt.

Auch wenn OpenNebula sich selbst in das Enterprise Segment kategorisiert (Llorente (2014)), ist es jedoch auch für kleinere Organisationen praktikabel. Gerade durch die Installation von wenigen Komponenten und seine Steuerung über nur ein Frontend, macht die Steuerung und Konfiguration komfortabel.

### 4.1.2 OpenStack

Der Ursprung von OpenStack liegt in einer Kooperation zwischen Rackspace (ein US-Webhoster) und der NASA, die versuchten eine Lösung zu finden, um Rechenkapazität an einer zentralen Stelle zu bündeln und einzelnen Abteilungen benötigte Ressourcen nach Bedarf anzubieten. Nachdem die NASA das Projekt abgegeben hat und sich diverse IT-Unternehmen um die Weiterentwicklung kümmerten, ist die Lösung ein fester Bestandteil des Portfolios aller großen Linux-Anbieter geworden. Die Struktur von OpenStack besteht aus einer Vielzahl an Komponenten, die alle unterschiedlich weit entwickelt sind. Während 'Keystone', welches oft als Hauptkomponente von OpenStack bezeichnet wird, als stabil und gut gewartet gilt, liegen neuere Komponenten noch teilweise hinter ihren Erwartungen. Ansonsten bietet OpenStack Administratoren ein großes Spektrum an Werkzeugen, um ihre Cloud-Computing Umgebung zu verwalten. Durch seine vielfältigen Einsatzmöglichkeiten und die große Auswahl an Optionen, ist OpenStack ein guter 'Allrounder', mit Defiziten in der Einarbeitungszeit. (Loschwitz und Sysseleven (2015))

### 4.1.3 Fazit

Eines der wichtigen Merkmale von OpenNebula ist der Schwerpunkt auf Rechenzentrums-virtualisierung inklusive bestehender Infrastruktur wodurch dies mehr im Enterprise-Cloud-Computing angesiedelt wird, OpenStack dagegen ein Vertreter des Public-Cloud-Computing ist. Für OpenNebula spricht, dass sich eigene VM-Images per Knopfdruck auf Virtualisierungsservern und dem dazugehörigen Storage instantiieren lassen und Zuweisungen wie IP, Load-Balancing und Speicherplatz von der eigenen Verwaltungssoftware übernommen wird. Zudem kommt eine übersichtliche Steuerung, die Fähigkeit mehrere hundert Server / virtuelle Maschinen verwalten zu können und die Eigenschaft verhältnismäßig einfach in eine bestehende Infrastruktur integriert zu werden. OpenStack versucht für alle alles anzubieten und kommt daher mit einer Vielzahl an unterschiedlichen Komponenten daher, die schnell unübersichtlich werden können. Das Komponentensystem bringt aber auch den Vorteil mit sich, nur Komponenten zu verwenden, die benötigt werden. So wird im Grunde mehr Flexibilität

und die Chance auf Kostenreduzierung erreicht.

Für die Verwaltung von virtuellen Maschinen und Infrastrukturen sind beide Applikationen mehr als geeignet. Allerdings fehlt bei beiden die Möglichkeit direkt Software mit aufzuspielen. Zudem sind beide Anwendungen für den anvisierten Verwendungszweck zu überdimensional in ihren Funktionalitäten.

### 4.2 Virtualisierungsprodukte

Virtualisierungsprodukte oder entsprechende Tools helfen virtuelle Maschinen für temporäre Zwecke aufzubauen, wie beispielsweise um alternative Betriebssysteme zu testen, Softwareprodukte auszuprobieren oder Entwicklungsumgebungen zu erstellen. Die wohl derzeit (Stand Mitte 2015) bekanntesten und etabliertesten Virtualisierungslösungen werden von drei verschiedenen Herstellern repräsentiert und zum Teil als Freeware angeboten.

1. **VMware Player / Plus**

Preiswerte Virtualisierungslösung für Unternehmen. Source-Code.

2. **Microsoft Hyper-V**

Microsofts Virtualisierungslösung, die nur in Windows 8 oder höher lauffähig ist.

3. **Oracle VM VirtualBox**

Gratis Open-Source-Tool von Oracle mit großem Funktionsumfang und frei verfügbar.

#### 4.2.1 VMware Player (Plus)

Eines der bekanntesten Unternehmen für Virtualisierungslösungen ist VMware. VMware bietet für den Privatanwender den VMware Player an, der das kostenlose Pendant zur professionellen Lösung VMware Workstation / VMware Fusion darstellt. Mittlerweile ist der VMware Player für Unternehmen unter dem Namen VMware Player Plus, zum Preis von ca. 90 Euro zu erwerben. Der VMware Player (Plus) unterstützt ca. 200 Gastbetriebssysteme, ganz gleich ob 32Bit oder 64Bit, und lässt sich auf Windows und verschiedenen Linux-Distributionen installieren. Virtuelle Maschinen lassen sich nur als VMDK-Format (Virtual Machine Disk) speichern, was somit die Wahl der Dateiformate enorm einschränkt. Features wie eine virtuelle Maschine in einen vorherigen Systemzustand zurückzusetzen fehlt gänzlich, genau so wie das Duplizieren einer Maschine. Allerdings werden Export- und Import Funktionen, sowie das erstellen von Snapshots vom VMware Player unterstützt.

### 4.2.2 Microsoft Hyper-V

Microsoft hat in Windows 8 (und höher) Hyper-V integriert, das sich über Windows-Funktionen einfach nachinstalliert lässt. Damit dies möglich ist, müssen zwei Faktoren gegeben sein. Windows 8 muss als Professional-Version in der 64-Bit Version vorliegen. Der Vorteil gegenüber der Konkurrenz besteht in der Fähigkeit mehrere virtuelle Maschinen prallel betreiben zu können, ohne signifikanten Performanceverlust. Dies erreicht Hyper-V dank SLAT, einer Technik zur dynamischen RAM Verwaltung.

### 4.2.3 Oracle VM VirtualBox

Die im April 2005 entstandene Virtualisierungssoftware von Oracle (erst InnoTek Systemberatung GmbH) eignet sich für Windows, Linux, Mac OS X, FreeBSD und Solaris als Hostsystem. An Gastsystemen wird eine Vielzahl von x64- und x86-Betriebssystemen unterstützt, so wie diverse Linux Distributionen, Windows ab Version 3.11, Mac OS X, IBM OS/2 und FreeBSD. VirtualBox lässt dem Anwender viele Freiheiten im Speichern seiner virtuellen Umgebung. Etwa vier gängige Formate werden angeboten und ermöglichen einen leichteren Austausch unter Anwendern. Ein weiterer Vorteil besteht darin, dass VirtualBox vollständig kostenlos und OpenSource ist. Der Quellcode steht jedem Interessierten zur Verfügung. Im Funktionsumfang enthalten sind das Importieren und Exportieren von virtuellen Maschinen, das Erstellen von Snapshots und das Klonen von virtuellen Maschinen.

### 4.2.4 Zusammenfassung

Die wichtigsten Faktoren bezüglich der zu entwickelnden Anwendung zusammengefasst:

	<b>VMware Player (Plus)</b>	<b>Microsoft Hyper-V</b>	<b>Oracle VM VirtualBox</b>
<b>Host-Betriebssystem</b>	Windows, diverse Linux Distributionen	Windows 8 Pro 64 Bit	Windows, Linux, MacOS X
<b>Gastbetriebssysteme</b>	Mehr als 200 Gast-Betriebssysteme		Diverse Linux Distributionen, Windows, FreeBSD, Mac OS X, IBM OS/2
<b>64-Bit-Gast-Betriebssystem</b>	Ja	Ja	Ja
<b>Dateiformate für virtuelle Disks</b>	VMDK	VHDX	VMDK, VHD, Parallels Hard Disk, OVF
<b>Snapshots</b>	Nein	Nein	Ja
<b>Klonen</b>	Nein	Nein	Ja
<b>Export von virtuellen Maschinen</b>	Ja	Ja	Ja
<b>Preis</b>	90 Euro für Unternehmen	Kostenlos	Kostenlos

#### 4.2.5 Fazit

Auch wenn VMware zu den bekannteren Herstellern gehört und der VMware Player (Plus) eine Vielzahl an Funktionen bietet, macht die Unterscheidung zwischen der Unternehmensvariante und der für Privatanwender die zukünftige Benutzung kompliziert. Sollte die zu erstellende Anwendung in einem nicht privaten Umfeld betrieben werden, so muss auf den VMware-Player Plus zugegriffen werden, wodurch ggf. Änderungen an der Implementierung vorzunehmen sind und die entsprechenden Lizenzkosten zu beachten wären. Außerdem schränkt nicht nur die geringe Auswahl an Dateiformaten zum Export von Maschinen die Funktionsweise ein, sondern auch das Nichtvorhandensein von Snapshot- und Klon Funktionen. Microsofts Lösung stellt sich von den Grundanforderungen her als nicht nutzbar heraus, da Hyper-V Windows als Grundlage benötigt, dieses jedoch nicht den gestellten technischen Randbedingungen (Kapitel 3.6 - Server Betriebssystem) an die zu erstellende Software genügt. Oracles VirtualBox hingegen vereint viele für die geforderten Funktionen unterstützende und hilfreiche Aspekte. Die Vielzahl an unterstützenden Dateiformaten kann für die geforderte Exportfunktion interessant



sein. Da VirtualBox für Privatanwender und Firmenkunden kostenlos ist, fällt der Lizenzierungsaspekt weg. VirtualBox lässt sich auf diversen Linux-Umgebungen installieren und somit ebenfalls mit den technischen Randbedingungen aus Kapitel 3.6.1 in Einklang bringen. Ggf. könnte der Punkt, dass VirtualBox Open-Source ist, für eine spätere Weiterentwicklungen interessant werden. VirtualBox wäre durch seinen Funktionsumfang in der Lage, wichtige Funktionen in der angestrebten Applikation zu übernehmen. Dies hätte zudem den Vorteil, dass die Funktionsbestandteile nicht aufwändig implementiert werden müssen.

### 4.3 Konfigurationsmanagement-Systeme

Wie in Kapitel 2.2 beschrieben, helfen Konfigurationsmanagement-Systeme bei der Umsetzung von sogenannten Zustandsbeschreibungen eines Hostes. Kurz gefasst, Konfigurationsmanagement-Systeme können gewünschte Software auf Zielrechner(n) installieren, Dienste und Applikationen starten/beenden, Konfigurationen ändern/erstellen/löschen und wenn gewünscht, dies alles gleichzeitig auf einem oder mehreren Zielrechnern. Durch die Anwendung von Konfigurationsmanagement-Systemen, kann die aus der Anforderungsanalyse herausgestellte Anforderung nach automatisierter Installation von Software übernommen werden. Auch hier gibt es bekannte und häufig verwendete Applikationen, die im Folgenden betrachtet werden.

#### 4.3.1 Ansible

Die Hauptdesignidee bei dem in Python geschriebenen Ansible ist es, Konfigurationen so leicht wie möglich durchführen zu können. Es werden weder aufwändige Deployment-Scripts benötigt, noch komplizierte Syntaxen verwendet. Ansible wird nur auf der Maschine installiert, die die Infrastruktur verwaltet und kann von dort auf die gewünschten Maschinen zugreifen. Die Clients benötigen weder eine lokale Ansible Installation noch andere spezielle Softwarekomponenten. (Hall (2013)) Die Kommunikation zwischen dem Host, auf dem Ansible installiert ist und den Clients wird über SSH ausgeführt. Für Linux-Distributionen, auf denen SSH für den Root Benutzer gesperrt sind, kann Ansible 'sudo' Befehle emulieren, um die gewünschte Zustandsbeschreibung durchzuführen. Windows wird in der aktuellen Version 1.9.1 nicht unterstützt. Zustandsbeschreibungen werden in YAML Syntax ausgeführt und in Playbooks geschrieben. Playbooks haben eine simple YAML Struktur und können somit schon vorgefertigt als Template gespeichert und wieder verwendet werden (ScriptRock (2014)).

### 4.3.2 Saltstack

Saltstack oder kurz 'Salt' ist wie Ansible in Python entwickelt worden. Zur Kommunikation mit den gewünschten Clients wird ein Master benötigt und Agenten, oder auch Minions genannt, die auf den Zielclients installiert werden müssen. Die eigentliche Kommunikation funktioniert über eine ZeroMQ messaging lib in der Transportschicht, wodurch die Verständigung zwischen Master und Minions vereinfacht wird. Dadurch ist die Kommunikation zwar schnell, jedoch nicht so sicher wie die SSH Kommunikation von Ansible. ZeroMq bietet nativ keine Verschlüsselung an und transportiert Nachrichten über IPC, TCP, TIPC und Multicast. Der größte Vorteil von Salt ist die Skalierbarkeit. Diese macht es möglich mehrere Ebenen an Mastern zu erstellen, um so eine bessere Lastverteilung zu erhalten. Salt benutzt für seine Konfigurationsdateien zwar ebenfalls YAML, allerdings müssen in der Konsole ausgeführte Befehle in Python oder PyDSL geschrieben werden. [ScriptRock \(2014\)](#)

### 4.3.3 Puppet

Der größte Vertreter auf dem Markt im Bereich Konfigurationsmanagement-Systeme ist wohl Puppet von Puppet Labs. Puppet hat nicht nur eine ausgereifte Monitoring-Oberfläche und läuft auf allen gängigen Betriebssystemen, sondern ist darüber hinaus Open-Source und bietet einen professionellen Support. Entwickelt wurde Puppet in Ruby. Entsprechend ist das Command-Line Interface an Ruby angelehnt, was natürlich den Nachteil mitsicht bringt, dass neben den Puppet Befehlen auch noch Ruby gelernt werden muss. Wie bei Salt muss es einen Master geben, auf dem der Puppet-Daemon (Puppetmaster) installiert ist. Der Puppetmaster hält die Zustandsbeschreibung für die jeweiligen Clients und verteilt diese auf Anfrage via REST-API. Die Clients selbst benötigen ebenfalls einen Agenten (Puppet-Agent), um die Zustandsbeschreibungen zu erfragen. Dieser vergleicht die Zustandsbeschreibung mit der aktuellen Konfiguration des Clients und nimmt entsprechende Änderungen vor ([Rhett \(2015\)](#)).

### 4.3.4 Zusammenfassung

Die wichtigsten Faktoren bezüglich der zu entwickelnde Anwendung zusammengefasst.

	Vagrant	Saltstack	Puppet
<b>Host-Betriebssystem</b>	Diverse Linux Distributionen	Diverse Linux Distributionen	Linux Distributionen, Windows
<b>Client-Betriebssysteme</b>	Diverse Linux Distributionen, Windows	Diverse Linux Distributionen, Windows	Linux Distributionen, Windows
<b>Lokale Client Installation nötig</b>	Nein	Ja	Ja
<b>Command-line Interface Sprache</b>	Python	Python	Ruby
<b>Zustandsbeschreibung</b>	YAML	YAML	Puppet Code
<b>Push oder Pull</b>	Push	Push	Push und Pull
<b>Open-Source</b>	Ja	Ja	Ja
<b>Preis</b>	Kostenlos	Kostenlos	Kostenlos

#### 4.3.5 Fazit

Der größte Unterschied in den verglichenen Konfigurationsmanagement-Systemen besteht im Kontext, in dem das jeweilige System eingesetzt werden soll. Alle drei Anwendungen erfüllen die Grundbedingung auf Linux lauffähig zu sein und diverse Linux-Distributionen als Client bespielen zu können, wobei Puppet sich hervorragend im Umfeld größerer Systemlandschaften eignet, d.h. wenn es darum geht, mehrere Clients gleichzeitig zu verwalten und zu konfigurieren. Wie Puppet benötigt Saltstack für die Clients extra Software, um die gewünschte Zustandsbeschreibung durchzuführen, was einen gewissen Zeitaufwand und eine Fehlerquelle mehr bedeutet. Ansible ist zwar der unbekanntere Vertreter unter den Konfigurationsmanagement-Systemen, kann aber leicht eingerichtet werden und benötigt keine weitere Ergänzungen auf der Clientseite. Dadurch wird auf den Zielmaschinen, ausschließlich das Gewünschte installiert, der gesamte Installationsprozess zudem beschleunigt und der erforderliche Zeitaufwand minimiert.

### 4.4 Vagrant

Ein weiteres Produkt, das die zu erstellende Applikation unterstützen könnte, ist Vagrant. Dabei handelt es sich um ein Softwareprojekt, welches 2010 von Mitchell Hashimoto und

John Bender 2010 ins Leben gerufen wurde. Vagrant ist ein Entwicklungswerkzeug, das als Wrapper zwischen Virtualisierungssoftware wie VirtualBox und Konfigurationsmanagement-System fungiert. Das command-line Interface und die einfache Konfigurationssprache helfen, virtuelle Maschinen schnell zu konfigurieren und zu verwalten. Die Konfiguration einer Maschine geschieht über das 'Vagrantfile', in dem Parameter wie IP-Adresse konfiguriert und Konfigurationsmanagement-Systeme hinzugeschaltet werden können. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, bedeutet das für den Anwender, ein problemloses Teilen des Vagrantfiles mit anderen Kollegen über Versionskontrollen (z.B. Git oder Subversion).

In puncto Konfigurationsmanagement-Systeme, wird dem Benutzer die Freiheit gegeben, auf eine Vielzahl bekannter Konfigurationsmanagement-Systeme zurückzugreifen, unter anderem auch Ansible, Salt und Puppet.

Um die Virtualisierung vorzunehmen, greift Vagrant standardmäßig auf VirtualBox zurück. Allerdings werden auch Amazon-Web-Services und VMware-Fusion unterstützt und Teamarbeit ermöglicht durch eine 'sharing'-Option, die mit Version 1.5 implementiert wurde. Das Teilen einer Maschine ermöglicht es Teams, an gemeinsamen und entfernten Standorten, auf die gleiche Maschine zuzugreifen (Peacock (2013)). Gerade die zusätzlichen Funktionen von Vagrant und das Vereinen von Konfigurationsmanagement und Virtualisierer in einem Produkt würde den Entwicklungsprozess der Applikation unterstützen helfen. Funktionen, wie das Teilen einer Maschine, könnten direkt übernommen und in die Applikation eingefügt werden.

## 5 Softwareentwurf

Nach Balzert (2011) ist der Softwareentwurf die Entwicklung einer software-technischen Lösung im Sinne einer Softwarearchitektur, auf Basis der gegebenen Anforderungen an ein Softwareprodukt. Die Kunst bei einem Softwareentwurf besteht darin, eine Softwarearchitektur zu entwerfen, die die zuvor erarbeiteten funktionalen- (Kapitel 3.3) und nichtfunktionalen Anforderungen (Kapitel 3.5) betrachtet, einschließlich der Berücksichtigung von Einflussfaktoren, wie definierte Randbedingungen (Kapitel 3.6). Der Softwareentwurf ist als Richtlinie zu sehen, der bei der Umsetzung der geforderten Software unterstützt. Die zu erstellende Softwarearchitektur hingegen beschreibt Architekturbausteine, deren Interaktionen und Beziehungen untereinander sowie ggf. deren Verteilung auf physischer Ebene. Dabei ist die Spezifizierung der entsprechenden Schnittstellen der einzelnen Architekturbausteine mit zu beachten. Zur Visualisierung können verschiedene Abstufungen von Sichten herangezogen werden, die Kontextabgrenzung, Bausteinsicht, Laufzeitsicht und die Verteilungssicht.

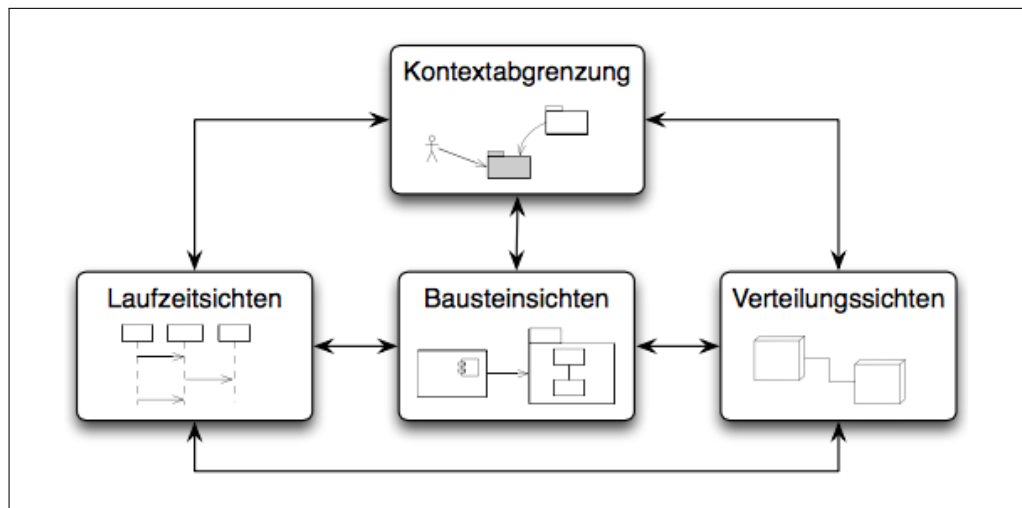


Abbildung 5.1: Vier Arten von Sichten (Starke (2014))

## 5.1 Kontextabgrenzung

Die Kontextabgrenzung beschreibt die Einbettung des Systems in seine Umgebung, sowie die wesentlichen Teile der umgebenden Infrastruktur. Die ermittelten Anforderungen aus Kapitel 3.3 und 3.5 haben ergeben, dass die Hauptfunktionalitäten aus Erstellen, Exportieren, Importieren, Teilen und dem Provisioning von virtuellen Maschinen bestehen. Um dies weiter zu bündeln, können Teile der Hauptfunktionalitäten bestimmter Produkte übernommen werden, die in Kapitel 4. betrachtet wurden. VirtualBox kann das Erstellen, Exportieren und den Import von virtuellen Maschinen übernehmen. Das Konfigurationsmanagement-System Ansible ist darauf ausgelegt, mit bekannten Virtualisierungslösungen zusammen arbeiten zu können und übernimmt somit die gewünschte Anforderung nach automatisierter Softwareinstallation. Um die beiden Anwendungen zu vereinen, kann Vagrant als Wrapper eingesetzt werden, der zusätzliche Funktionen, wie das Teilen (Sharen) einer Maschin, mitbringt. Logik und Oberfläche werden durch den VM-Builder bereitgestellt, der in den folgenden Abschnitten näher konzipiert wird.

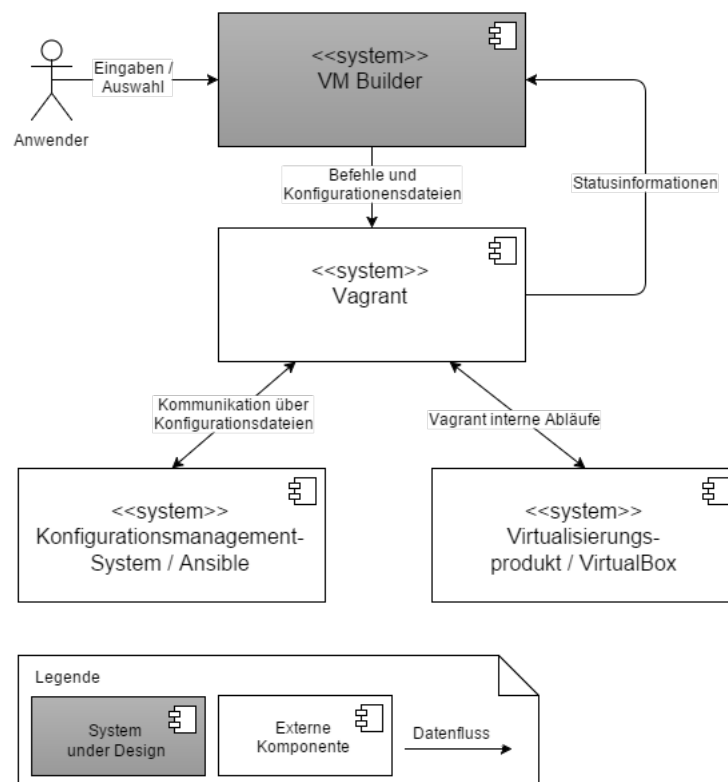


Abbildung 5.2: Kontextsicht

#### 5.1.0.1 Kurzbeschreibung der externen Schnittstellen

<b>Eingaben / Auswahl</b>	Der Anwender tätigt Eingaben und wählt unter bereitgestellten Optionen aus. Diese werden direkt von der Applikation verarbeitet
<b>Befehle und Konfigurationen</b>	Die Applikation erstellt nötige Konfigurationsdateien für Vagrant und Ansible und leitet Befehle für die Weiterverarbeitung an Vagrant weiter
<b>Kommunikation über Konfigurationsdateien</b>	Vagrant ruft über die erstellten Konfigurationsdateien den Konfigurationsmanager Ansible auf, um die virtuelle Maschine in den beschriebenen Zustand zu überführen
<b>Vagrant interne Abläufe</b>	VirtualBox erstellt die virtuelle Maschine und gibt Statusmeldungen an Vagrant weiter. Dies sind interne Abläufe, die zwischen Vagrant und VirtualBox ablaufen und nicht vom Entwicklungsprozess beeinflusst werden können

Die Applikation selbst, inklusive der oben genannten Produkte, läuft auf einem Server, der zentralisiert positioniert ist und entsprechend angesprochen werden kann. Dem Anwender wird von der Applikation eine Weboberfläche angeboten, die es ermöglicht Eingaben zu tätigen und Optionen auszuwählen, um eine virtuelle Maschine zu erstellen oder zu verwalten. Um die Applikation auf dem Server zu betreiben, muss eine Internetverbindung bestehen, die es Vagrant ermöglicht, das gewünschte Abbild des Betriebssystems herunterzuladen und die virtuelle Maschine mit anderen Anwendern zu teilen. Die vom Anwender gestellten Anfragen an den VM-Builder, werden in Konfigurationsdateien übersetzt, die speziell für Vagrant und Ansible passend erstellt werden. Diese Konfigurationsdateien dienen nicht nur zur Erstellung der gewünschten virtuellen Maschine, sondern auch zur Kommunikation zwischen Vagrant und Ansible. Vagrant entnimmt den Konfigurationen das gewünschte Image und leitet den entsprechenden Download des Betriebssystems ein. Allerdings nur, falls das Image des Betriebssystems nicht schon auf dem Server vorliegt.

Durch die Hilfe von VirtualBox und ggf Ansible, wird die zu erwartende virtuelle Maschine komplettiert.

## 5.2 Verteilungssicht

Um die Beschreibung aus 5.1.0.1 der Kontextabgrenzung visuell zu unterstützen, wird in diesem Fall die Verteilungssicht herangezogen. In Starke (2011) wird die Verteilungssicht als '[...] die Verteilung von Systembestandteilen auf Hard- und Softwareumgebungen' beschrieben, die '[...] klärt, welche Teile des Systems auf welchen Rechnern, an welchen geographischen Standorten oder in welchen Umgebungen ablaufen können, wenn es in einer konkreten technischen Infrastruktur installiert wird'.

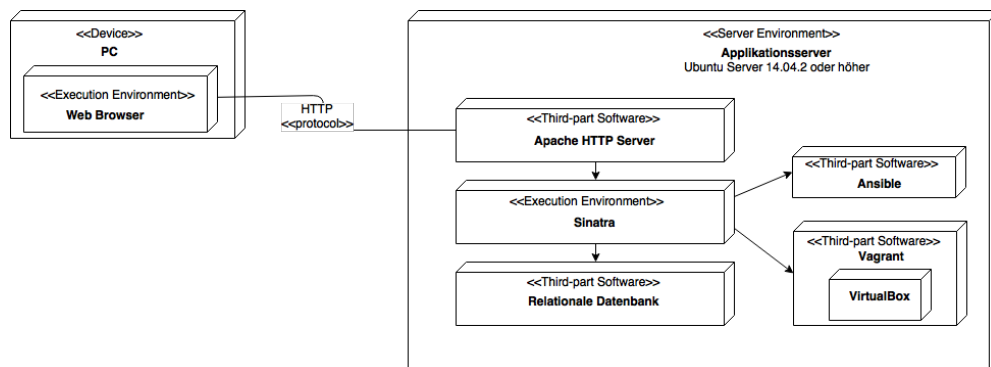


Abbildung 5.3: Verteilungssicht des VM-Builders

Die genaue Platzierung der Softwarekomponenten aus Abschnitt 5.1.0.1 verhilft zu einer besseren Umsetzung der kompletten Softwarestruktur und verdeutlicht die Beziehung zwischen den einzelnen Komponenten. In Kapitel 3. unter den Abschnitten 3.5 und 3.6.1 wurden technologische Anforderungen und entsprechende Randbedingungen beschrieben und definiert. Die dortige Beschreibung besagt, dass die zu verwendene Software frei verfügbar und kostenlos sein sollte. Aus diesem Grund, wird bei dem Betrieb des Servers auf Ubuntu zurückgegriffen, wobei hier von aktuellen Version (Stand Juni 2015) ausgegangen wird. Als Webserver kann Apache eingesetzt werden, der mittels dem zustandslosen HTTP Protokoll mit dem Client kommuniziert. Die benötigte relationale Datenbank kann z.B durch MySQL umgesetzt werden. Sie wird für das Speichern von Konfigurationen einzelner virtueller Maschinen genutzt, deren Konfigurationen und zum persistieren der angebotenen Softwarekomponenten. Apache und MySQL sind lizensfreie und kostenlose Produkte, die etabliert und leistungsfähig sind. In dem Execution Framwork wird der VM-Builder ausgeführt, der wiederum an die Drittanbieter Ansible und Vagrant angebunden ist. Wie bereits in Kapitel 4. aufgeführt, sind beide Produkte ebenfalls kostenlos und frei verfügbar. Da Hardwarespezifikationen in dieser Phase noch rein spekulativ sind, wird dieser Aspekt in der Umsetzung weiter betrachtet.



## 5.3 Systemarchitektur

Nachdem die Umsysteme, deren Verteilung und der technische Aufbau des VM-Builders konzipiert ist, steht die Struktur der Anwendung im Fokus. Dafür gibt es etablierte Architektur- und Entwurfsmuster, auf die zurückgegriffen werden kann. Diese helfen bei der Verteilung der Verantwortlichkeiten und bilden eine Vorlage für die Systemstrukturen. Da sie sich in einigen Punkten überschneiden und sogar ergänzen, entsteht bei der Verwendung mehrerer Architekturmuster keine Widersprüche.

### 5.3.1 Client-Server-Modell

Um eine möglichst gute strukturelle Verteilung der Aufgaben und Dienstleistungen des VM-Builders auf physikalischer Ebene zu erhalten, wird auf das Client-Server-Modell zurückgegriffen. Das Client-Server-Modell verdeutlicht die Aufgabenverteilung innerhalb einer Applikation und die Zentralisierung von Prozessorenleistung und gemeinsamer Dienste (Schäfer (2009)). Die klare Unterscheidung zwischen den Client- und Servertätigkeiten kann durch einen geschichteten Architekturstil erreicht werden, der die Aufgaben in folgende Schichten unterteilt:

1. Benutzerschnittstelle (User interface)  
Die Benutzerschnittstelle enthält alles Erforderlich, um direkt mit dem Anwender zu interagieren
2. Verarbeitungsebene (Application)  
Die Verarbeitungsebene enthält die Anwendung / Kernfunktionalität
3. Datenebene (Database)  
Daten werden unabhängig von der Applikation persistent gespeichert

Im Zusammenhang mit dem Client-Server-Modell steht das n-Tier Model oder auch Schichtenmodell. Eine Schicht ist entweder ein physikalischer Rechner oder mindestens ein Systemprozess, der eine gewisse Aufgabe übernimmt. Die einfachste Anordnung dieser Schichten besteht darin, sie auf zwei Computer zu verteilen (2-Tier Model), den Client und den Server. Dabei können die Schichten wie in Abbildung 5.4 zwischen Client und Server verteilt sein (Tanenbaum und van Steen (2007)).

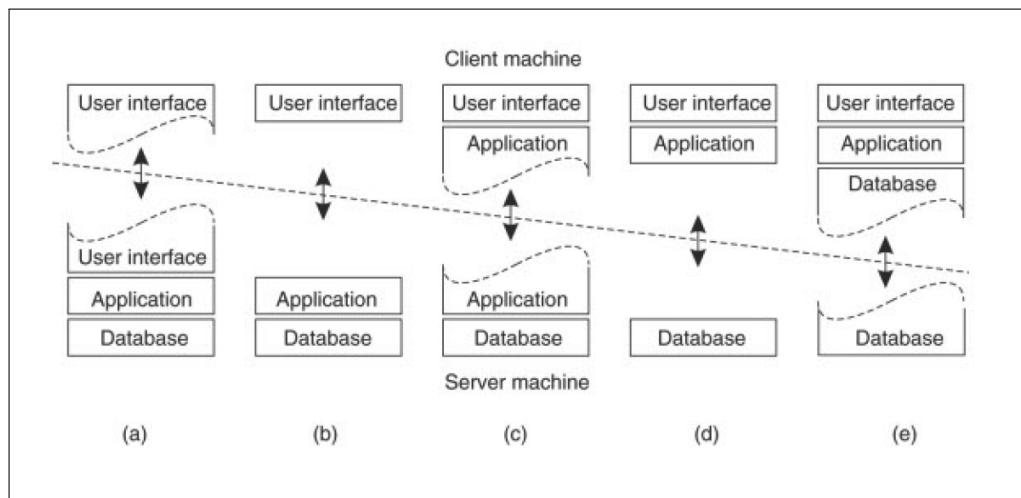


Abbildung 5.4: Client-Server-Anordnungen (Tanenbaum und van Steen (2007))

Die in Abbildung 5.4(a) bis (c) dargestellten Varianten gehören zu der Kategorie Thin-Clients, die für den zu entwickelnden 'VM-Builder' im Fokus stehen. Die Varianten (d) und (e) sind sogenannte Fat-Clients.

Der Vorteil von Thin Clients ist, dass weniger bis keine Client-Software auf die Seite des Anwenders gebracht werden muss. Denn Software auf Clientseite ist nicht nur schwerer zu administrieren, sondern auch anfälliger für Fehler (Tanenbaum und van Steen (2007)).

Die hier angestrebte Client-Server-Anordnung ist die in Abbildung 5.4(a) gezeigte Variante. Der Client soll so schlank wie möglich gehalten werden, um dem Anwender einen schnellen Seitenaufbau zu ermöglichen und lokale Installationen zu verhindern. So liegt die Kontrolle der Darstellung auf Applikations-Seite. Da in der Anforderungsanalyse Kapitel 3.6.1 festgelegt wurde, dass mit Ruby inkl. des Frameworks Sinatra gearbeitet werden soll, wird die Logik auf der Server-Seite implementiert und benötigt keinen Anteil auf Client-Seite. Entsprechend sind die Verarbeitungs- (Application) und die Datenebene (Database) auf der Server-Seite angesiedelt. Auf Verarbeitungsebene wird der Webserver mit den Applikationskomponenten des VM-Builders realisiert. Die Datenebene spiegelt die zu verwendende Datenbank wieder, die für die Persistierung von Daten zuständig sein wird. Beim Client-Server-Aufbau mit Variante (b) wird davon ausgegangen, dass die gesamte Darstellung auf Client-Seite platziert wird, wodurch eine Separierung vom Client zur restlichen Anwendung vollzogen wird. Die Anwendung von Variante (c) findet Gebrauch in Applikationen, in denen zusätzlich Logikverarbeitung auf Anwender-Seite geschehen soll, was ebenfalls gegen die Planung des VM-Builders sprechen würde. So ist die Wahl von Variante (a) als Thin-Client am zutreffendsten.

### 5.3.2 Model-View-Controller Entwurfsmuster

Für eine strukturierte Umsetzung der grafischen Komponente des VM-Builders, wird auf das bekannte architektonische Model-View-Controller Entwurfsmuster zurückgegriffen. Das Model-View-Controller (MVC) Entwurfsmuster findet beim Entwurf grafischer Benutzungsoberfläche anwendung, d.h. bei der Mensch-Maschine-Interaktionen. Dazu wird das System, das Interaktionen anbietet und ausführt, in drei Verantwortlichkeiten strukturiert:

1. **Model**

kapselt alle fachlichen Daten und enthält den Anwendungskern

2. **View**

bereitet Informationen für den Anwender grafisch auf

3. **Controller**

nehmen Benutzereingaben/Events an, die entsprechend an das passende Model oder die View weitergeleitet werden.

Wie das Client-Server-Model, besteht auch das Model-View-Controller Entwurfsmuster aus drei Schichten. Anstatt diese drei Schichten auf mehrere physikalische oder virtuelle Systeme zu verteilen, werden sie auf Applikationsebene abgebildet. Das MVC-Entwurfsmuster kann zwar auf jeder der Schichten des Client-Server-Models implementiert werden, hat aber im Gegensatz zu dem Client-Server-Model entsprechend nichts mit der Verteilung des Systems zu tun.

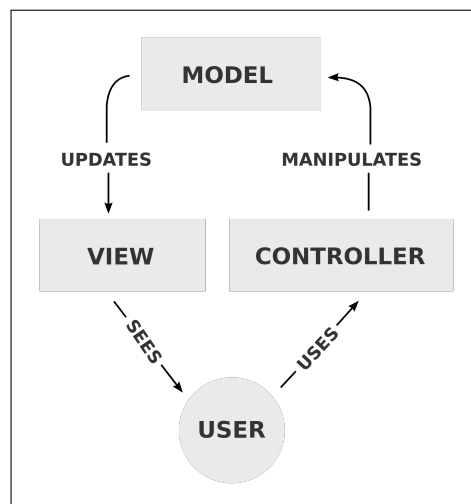


Abbildung 5.5: Model-View-Controller (Wikipedia (2015))

Abbildung 5.5 verdeutlicht die Kommunikation zwischen den Bestandteilen des Mode-View-Controller Entwurfsmusters.

1. Alle Eingaben/Änderungen des **User** werden von der Benutzeroberfläche an den Controller weitergegeben
2. Der **Controller** gibt Zustandsänderungen an das Model weiter
3. Das **Model** verarbeitet die erhaltenden Daten, in dem diese z.B an den persistenten Speicher weitergeleitet werden oder Berechnungen stattfinden
4. Die resultierenden Ergebnisse / Änderungen werden dann über die View sichtbar gemacht

Das MVC-Entwurfsmuster entkoppelt das User-Interface von der Verarbeitungsebene. Es geht der Forderung nach, die View (Repräsentation) und das Model (Fachlichkeiten) zu trennen, da die Änderungshäufigkeit beider Ebenen, unterschiedlich ausfallen können. Durchschnittlich ändern sich z.B. Windows-Oberflächen etwa alle zwei Jahre, Fachlichkeit aber sehr viel langsamer in der Größenordnung von ca 10 bis 15 Jahren. Das MVC-Entwurfsmuster erlaubt also als eine Modernisierungsmaßnahme den entsprechenden Austausch der Oberfläche, ohne die Fachlichkeiten ändern zu müssen (Masak (2009)). Da das geforderte Framework Sinatra diese Konzept nativ umsetzen kann, lässt sich das Entwurfsmuster entsprechend gut anwenden. In Sinatra werden die Views separiert von den sogenannten Rounten (Controllern) implementiert, wodurch die 'V' und 'C' Eigenschaften erfüllt werden. Um die Modell-Anforderungen zu erfüllen, kann z.B. ein Datenbank-Framework, wie Active-Record angewendet werden.

Abbildung 5.6 zeigt die Integration des MVC-Entwurfsmuster in das Schichtenmodell.

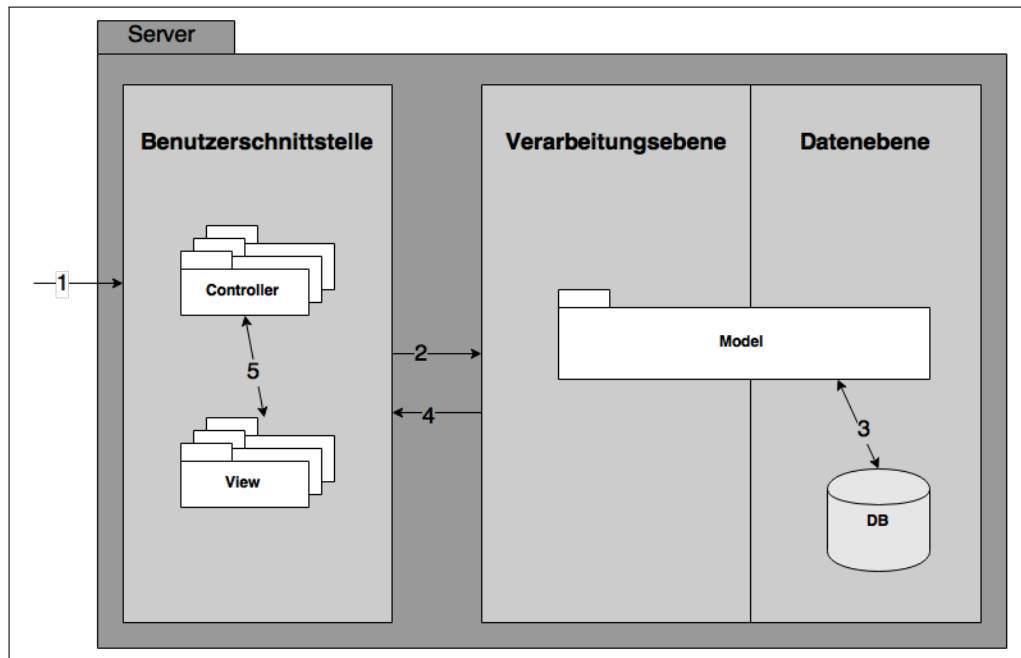


Abbildung 5.6: Model-View-Controller im 3-Schichten-Modell

Da der Client nur für die Darstellung zuständig ist, wird wie in Abbildung 5.6 zu sehen, das MVC-Entwurfsmuster nur auf Server-Seite implementiert. Entsprechend werden die Controller und die Views im serverseitigen Teil der Benutzerschnittstelle angesiedelt, während die Modelle sich über die Verarbeitungsebene und Datenebene erstrecken, um den Zugriff auf die Datenhaltung zu gewährleisten.

## 5.4 Kommunikation

Die Kommunikation zwischen Client und Server wird über das zustandslose HTTP Protokoll realisiert. Möchte ein Client (der theoretisch ein Web-Browser, eine Web-Anwendung, ein Dienst, usw. sein kann) mit einem Webserver kommunizieren, erstellt der Client eine HTTP-Nachricht. Diese Nachricht ist in 'plain-text' geschrieben und Zeilen orientiert. Dementsprechend ist eine Nachricht leicht zu erstellen und auf Serverseite entsprechen leicht auszuwerten. Ist ein Server mit der Anfragebearbeitung fertig, sendet er in der Regel den Status (war die Client-Anforderung erfolgreich, ist ein Fehler aufgetreten, etc.), Inhalte und andere Daten zurück an den Client. Nachrichten enthalten sogenannte HTTP-Verben, die den Typ der Anfrage definieren und wie der Server die Anfrage zu verstehen hat (Harris und Haase (2011)).

### Standard HTTP-Verben

GET

### Definition

Eine GET-Anforderung wird verwendet, um einen Server zu bitten, die Darstellung einer Ressource zurückzuliefern

POST

Eine POST-Anforderung wird verwendet, um Daten an einen Webserver zu übermitteln

PUT

PUT wird verwendet, um auf einem Server eine Ressource zu erstellen oder zu aktualisieren

DELETE

DELETE wird verwendet, um eine Ressource auf dem Server zu löschen

In dem zu verwendenden Framework Sinatra wird das Vokabular der HTTP-Verben benutzt, um Routen zu definieren. Um eine Route in Sinatra zu deklarieren, wird das HTTP-Verb in Verbindung mit der URL benötigt. Das gewünschte Verhalten wird im Anschluss definiert und beim Aufruf der Route ausgeführt. Siehe Abbildung 5.1.

```
1 get '/' do
2   .. zeige etwas ..
3 end
4
5 post '/' do
6   .. erstelle etwas ..
7 end
8
9 put '/' do
10  .. update etwas ..
11 end
12
13 delete '/' do
14   .. entferne etwas ..
15 end
```

```
16
17 options '/' do
18   .. zeige, was wir können ..
19 end
20
21 link '/' do
22   .. verbinde etwas ..
23 end
24
25 unlink '/' do
26   .. trenne etwas ..
27 end
```

Listing 5.1: Routen in Sinatra ([Sinatra \(2015\)](#))

## 5.5 Server

Wie in Kapitel 5.3 beschrieben, wird der Server nach den dort geplanten Entwurfsmuster konstruiert. Damit ein vollständiges Bild des Server-Konstrukts entsteht, werden im Folgenden die noch offenen Sichten aus der Abbildung 5.1 entworfen.

### 5.5.1 Bausteinsicht

Die Bausteinsicht 'zeigt die statische Struktur des Systems, seinen Aufbau aus Softwarebausteinen, sowie deren Beziehungen und Schnittstellen untereinander' [Starke \(2011\)](#). Ausgehend vom Systemkontext, wird das System hierarchisch zergliedert. Somit können Ebenen in unterschiedlichem Detailgrad entstehen. Ebene-0 stellt das System als Blackbox dar, die der Kontextsicht aus Abbildung 5.2 entspricht. In den höheren Ebenen, wird der Detailgrad erhöht und die dargestellte Blackbox zur Whitebox. Whiteboxen geben detailliertere Einblicke in ihre Struktur und Schnittstellen, die allerdings wiederum als Blackboxen dargestellt werden. Abbildung 5.7 ist eine Ebene-1 Darstellung der Serverapplikation in Form des MVC-Entwurfsmusters, in der der VM-Builder mit seinen einzelnen Komponenten betrachtet wird, die als Blackboxen dargestellt sind.

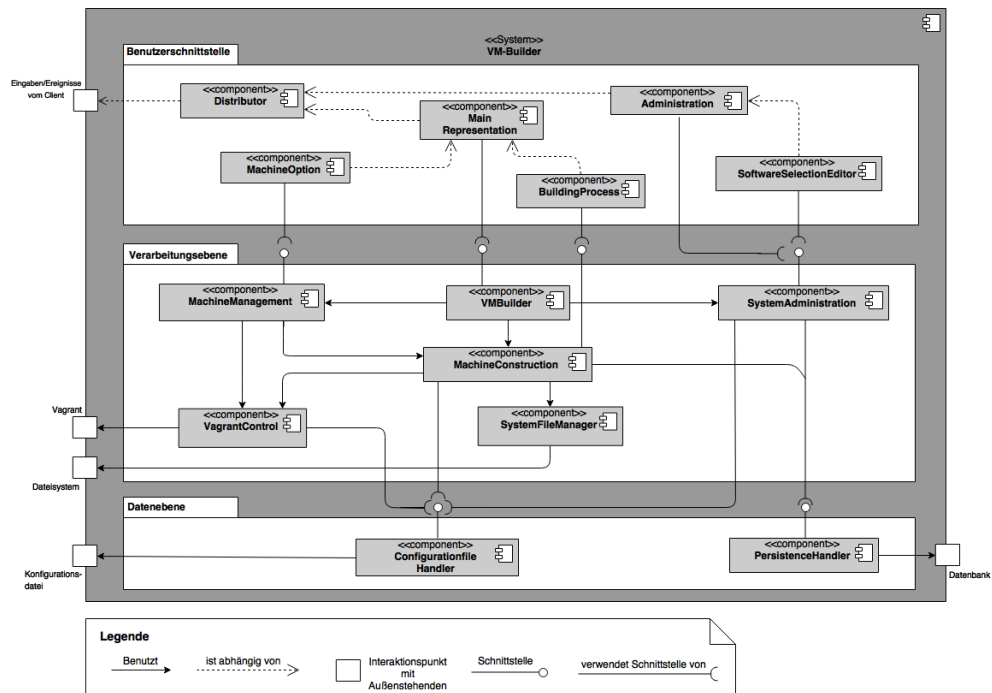


Abbildung 5.7: Bausteinsicht Level 1

Die Komponenten sind in ihre Zuständigkeiten gegliedert und den entsprechenden Schichten zugeordnet. Die Schnittstellen werden durch Interaktionspunkte zwischen den Schichten dargestellt. Jede Komponente enthält, der jeweiligen Schicht entsprechend, Controller, Views und/oder Models, die in dem folgenden Abschnitt detaillierter betrachtet werden.

### 5.5.1.1 Benutzerschnittstelle



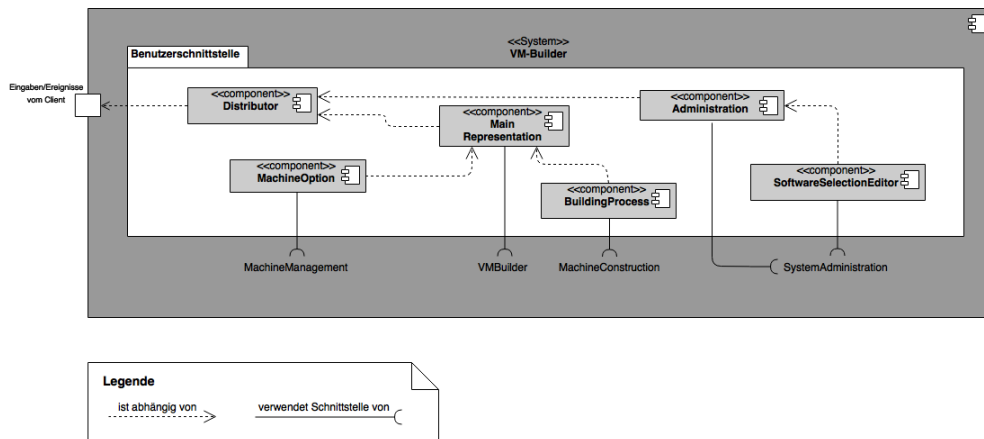


Abbildung 5.8: Benutzerschnittstelle

Der für die Kommunikation zuständige Anwendungsteil, ist die Benutzerschnittstelle (Abbildung 5.8). Die dort enthaltenen Komponenten sind für das Annehmen der User-Interaktionen zuständig und für die Repräsentation der gewünschten Inhalte.

### Distributor - Komponente

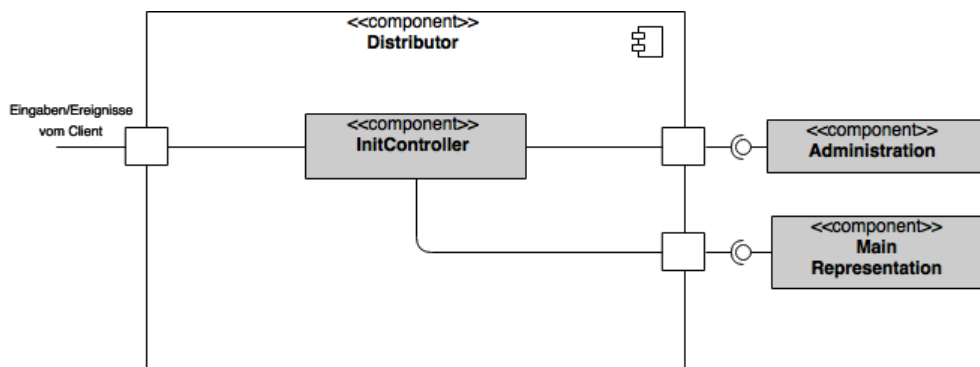


Abbildung 5.9: Komponentensicht Distributor

Die primäre Zuständigkeit der **Distributor** - Komponente liegt im Empfang der eingehenden Kommunikation und der Weiterleitung an den entsprechenden Controller. Da die Administration des VMBuilders eine gänzlich andere Hauptfunktion ist, als der Aufbau inklusive der Verwaltung von virtuellen Maschinen, entsteht durch den Einsatz dieser Komponente eine klare hierarchische Unterteilung der Hauptfunktionalitäten. Zudem erleichtert der Aufbau eine Erweiterung von neuen primären Funktionen der Applikation.

### MainRepresentation - Komponente

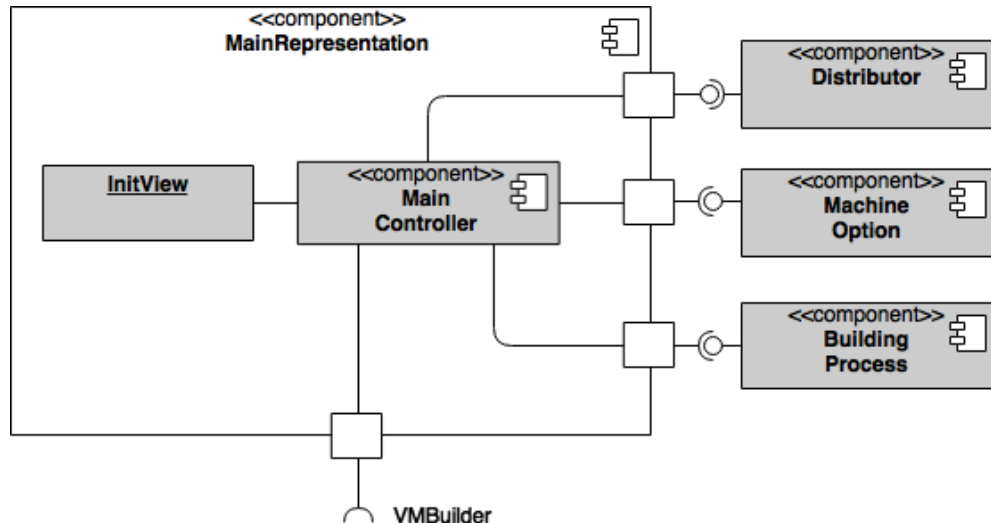


Abbildung 5.10: Komponentensicht VMBuilder

Im Inneren der **MainRepresentation** - Komponente ist der **MainController** platziert, der mit Hilfe der **InitView** eine Übersicht der bestehenden virtuellen Maschinen verschafft. Um die Übersicht zu erhalten, muss die Schnittstelle zum **VMBuilder** der Verarbeitungsebene bestehen. Erst sie stellt die Daten für die Anzeige der vorhandenen Maschinen zur Verfügung. Aus dieser Ansicht kann ausserdem der Aufbauprozess einer neuen Maschine über die **BuildingProcess** - Komponente initialisiert, oder Optionen auf einzelne virtuelle Maschinen aufgerufen werden. Die verfügbaren Optionen werden durch die **MachineOption** - Komponente gesteuert und realisiert.

### BuildingProcess - Komponente

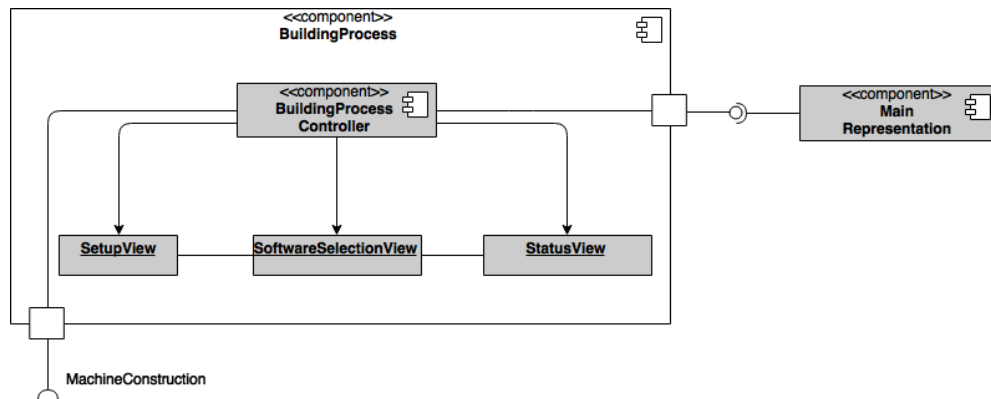


Abbildung 5.11: Komponentensicht BuildingProcess

Der Aufbauprozess einer virtuellen Maschine wird grafisch durch die **BuildingProcess** - Komponente realisiert. Die dort enthaltenen Views leiten den Anwender durch den Aufbauprozess, wobei jede View einem Konfigurationsschritt entspricht:

#### 1. **SetupView**

In der SetupView werden die Eigenschaften wie IP-Adresse und Name der zu erstellenden Maschine festgelegt

#### 2. **SoftwareSelectionView**

Um die virtuelle Maschine in den gewünschten Zustand zu versetzen, bietet die SoftwareSelectionView dem Anwender eine Auswahl, an zu installierender Softwarekomponenten und Paketen.

#### 3. **StatusView**

Die StatusView erstellt eine Übersicht über den aktuellen Aufbauverlauf und präsentiert die Zugangsmöglichkeiten zur virtuellen Maschine

Unterstützt wird der Aufbau durch den Zugriff auf die **MachineConstruction** - Komponente der Verarbeitungsebene. Dieser gibt unter anderem der **SoftwareSelectionView** eine Vorgabe an Auswahloptionen, die der Anwender in Betracht ziehen kann. Da der Aufbau einer virtuellen Maschine erst durch einen bestimmten Aufruf aus der MainRepresentation erfolgen soll, ist der **BuildingProcessController** auch nur über diese Komponente aufrufbar.

### MachineOption - Komponente

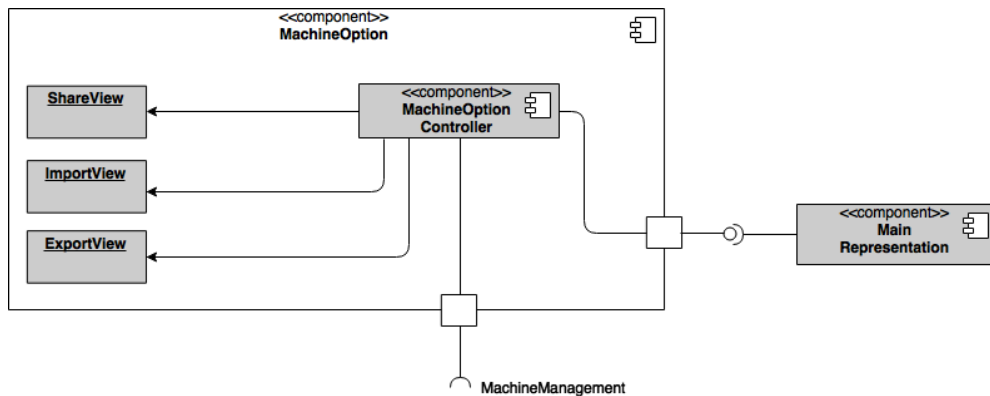


Abbildung 5.12: Komponentensicht MachineOption

In Abbildung 5.12 ist ersichtlich, dass die **MachineOption** - Komponente drei Views bereitstellt, die alle über den Controller indirekt in Beziehung stehen. Sie repräsentieren die Optionen, die ein Anwender auf eine bestehende Maschine ausführen kann. Über die Schnittstelle der **MachineManagement** - Komponente, werden die einzelnen Funktionalitäten bereitgestellt und ausgeführt.

#### Administration - Komponente

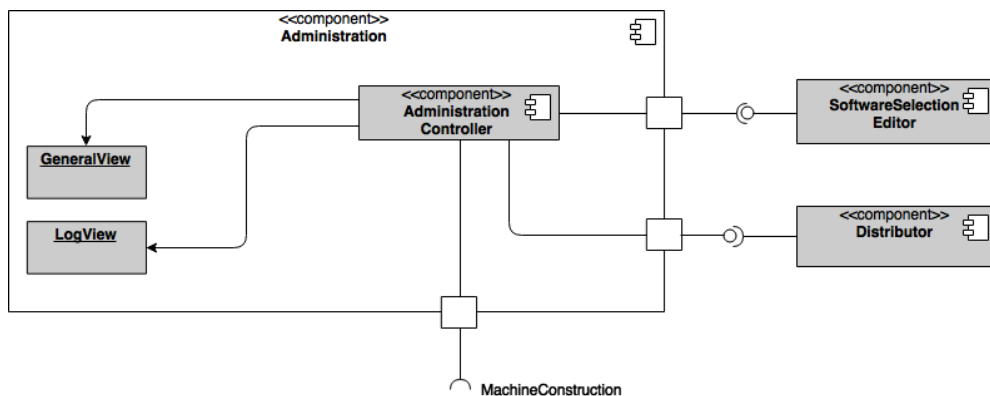


Abbildung 5.13: Komponentensicht Administration

Wie bereits am Anfang im Abschnitt **Distributor** - Komponente (5.5.1.1) erwähnt, gibt es neben der **MainRepresentation** - Komponente auch die **Administration** - Komponente. Sie ist für die Visualisierung der generellen Einstellungen, der Anzeige von Logdateien und dem Softwareeditor zuständig. Der Softwareeditor ist als eigenständige Komponente ange-

schlossen, da dieser spezielle Funktionen bereitstellt. Die Schnittstelle der **Administration** - Komponente ist mit dem SystemAdministration aus der Verarbeitungsebene verbunden, um die voreingestellten Anwendungseigenschaften abzurufen und Neue zu speichern.

### SoftwareSelection - Komponente

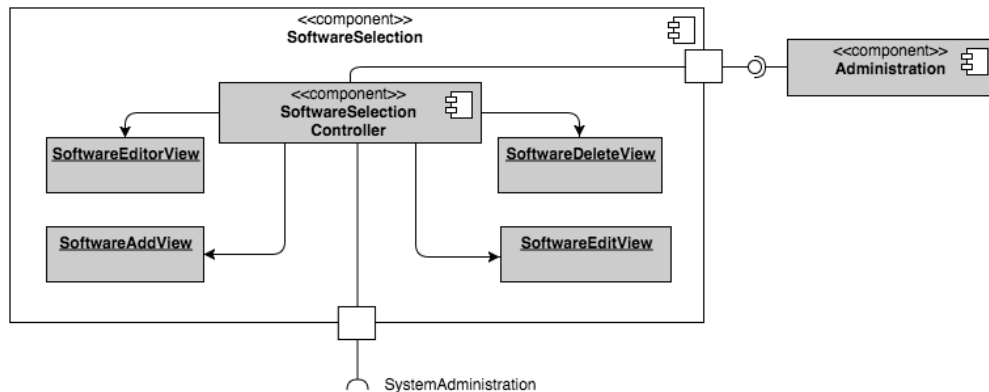


Abbildung 5.14: Komponentensicht SoftwareSelection

Die an der **Administrations** - Komponente angeschlossene **SoftwareSelection** - Komponente unterstützt den Anwender in der Konfiguration einzelner Softwarekomponenten und Pakete. Pakete bestehen aus einzelnen Softwarekomponenten, die in Abhängigkeit gestellt werden. So können beim Aufbau einer virtuellen Maschine, durch die Auswahl eines Paketes, mehrere Softwarekomponenten auf einmal installiert werden. Zudem ermöglicht die **SoftwareSelection** - Komponente neue Softwarekomponenten hinzuzufügen, zu bearbeiten und zu ändern. Entsprechende Views helfen dem Anwender die gewünschten Funktionen durchzuführen.

#### 5.5.1.2 Verarbeitungseben

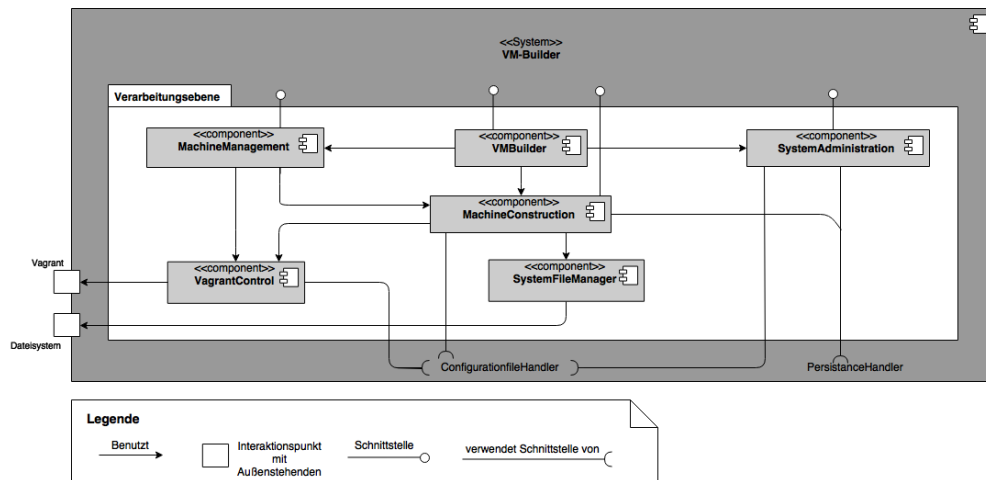


Abbildung 5.15: Ansicht der Verarbeitungsebene

Wie im MVC - Entwurfsmuster vorgesehen, beinhaltet die Verarbeitungsebene die Logik der Anwendung, die durch logisch konstruierte Komponenten repräsentiert wird. Da Komponenten wiederum aus Komponenten bestehen können, werden diese durch eine Whitebox-Darstellung explizit hervorgehoben. Ohne weitere Whitebox-Darstellung kommen die Komponenten aus, die für sich selber stehen.

### SystemFileManager - Komponente

Die Zuständigkeit des **SystemFileManagers** besteht im Zugriff auf das Dateisystems. Durch den **SystemFileManager** werden Standard-Ordner Funktionen des Betriebssystems ermöglicht sowie Kopier-, Duplizierungs- und Erstellungsoperationen. Diese Komponente ist gerade im Bezug auf den Erstellungsprozess essenziell.

### VagrantControl - Komponente

**VagrantControl** vereinigt Befehlsaufrufe für die Steuerung von Vagrant. Zudem extrahiert die Komponente essenzielle Informationen aus den Vagrant - Rückmeldungen, die bei der Ausführung von Vagrant erzeugt werden. Die Informationen werden interpretiert und in Rückgabewerte von Funktionsaufrufen umgewandelt, oder für die Generiert von Fehlermeldungen verwendet. Da Vagrant nur mit einem gültigen Vagrantfile lauffähig ist, übernimmt **VagrantControl** auch das Erstellen der genannten Datei.

## MachineConstruction - Komponente

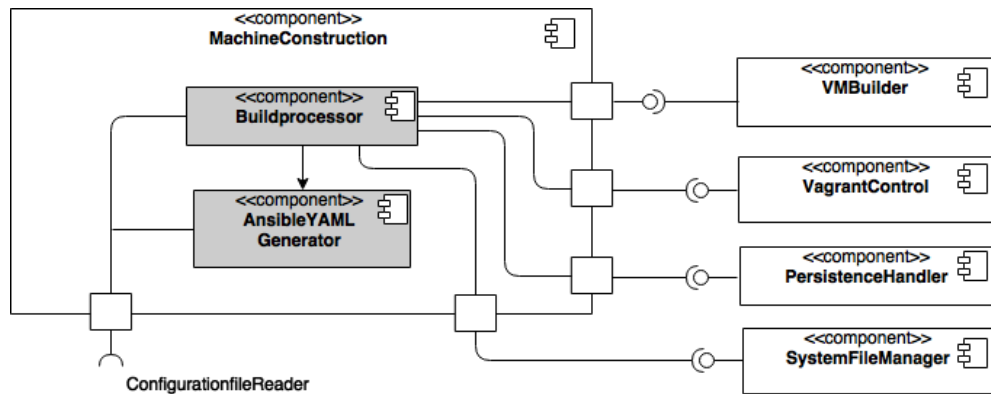


Abbildung 5.16: Komponentenansicht MachineConstruction

Der Aufbau einer virtuellen Maschine wird durch die **MachineConstruction** - Komponente realisiert. Genauer durch das Zusammenspiel zwischen **Buildprocessor** und **AnsibleYAML-Generator**. Eine Zerteilung der beiden Komponenten ist im Bezug auf ihre Zuständigkeiten wichtig. So ist das Auswechseln der Komponenten und die Erweitern von Funktionalitäten einfacher. Der **AnsibleYAMLGenerator** ist die Komponente, die es Ansible ermöglicht zu wissen, welche Software provisioniert werden soll. Während der Aufgabenbereich des **Buildprocessor** sich vom Einsammeln der Eingabeinformationen des Benutzers, über das Erstellen der nötigen Konfigurationsdateien, bis hin zum starten des eigentlichen Aufbauprozesses einer Maschine erstreckt. Soll zukünftig der Aufbauprozess verändert, oder Ansible durch einen anderen Provisionierer ausgewechselt werden, ermöglicht die Aufteilung der Zuständigkeiten den unkomplizierten austausch der jeweiligen Komponente. Die Schnittstellen der **MachineConstruction** - Komponente sind unter anderem mit **VagrantControl**, dem **PersistenceHandler** und dem **SystemFileManager** verbunden. Um den Aufbau korrekt durchführen zu können, benötigt der **Buildprocessor** die Steuerbefehle für Vagrant und den Zugriff auf das Dateisystem um die virtuelle Maschine zu platzieren. Der **PersistenceHandler** liefert für den Aufbau die entsprechende Softwarevorschläge, die der Benutzer auf der virtuellen Maschine installieren kann. Der Provisionierer 'Ansible' arbeitet mit YAML-Datei, die Konfigurationseigenschaften enthalten, die unterstützend beim Aufbau einer virtuellen Maschine wirken können. Die YAML-Dateien sind optional, da sie die gewünschten Softwarekomponenten beinhalten. Wird keine Software benötigt, kann diese Datei weggelassen werden. Der **AnsibleYAMLGenerator** baut aus den Informationen, die er aus dem **BuildProcessor** erhält, die richtige Struktur und den inhaltlichen Kontext der YAML-Datei. Durch den Zugriff auf die **Configurationfi-**

**leHandler** - Komponente, die sich in der Datenebene befinden, erhalten Buildprocessor und AnsibleYAMLGenerator Grundkonfigurationseinstellungen, die zum Aufbau benötigt werden.

### SystemAdministration - Komponente

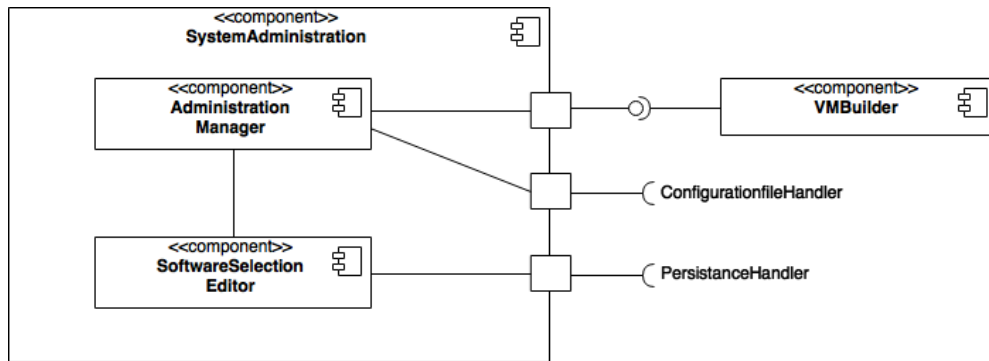


Abbildung 5.17: Komponentenansicht SystemAdministration

Für den administrativen Teil der Applikation ist der **AdministrationManager** zuständig. Der **AdministrationManager** ist die primäre Anlaufstelle für die Konfiguration der Applikation. Die Administration bietet Funktionen zum Auslesen von Logdateien und liefert Grundeinstellungen aus dem **ConfigurationfileHandler**. Der **AdministrationManager** soll zudem einen Softwareeditor für Administratoren bereitstellen, der durch die **SoftwareSelectionEditor** - Komponente realisiert ist. Durch Verwendung des **PersistenceHandler** können neue Softwarebestandteile bearbeitet, gelöscht oder hinzugefügt werden. Zu den weiteren Hauptaufgaben gehört das Kreieren von Softwarepaketen. In denen werden Abhängigkeiten zu anderen Softwarebestandteilen geknüpft und optional Scripte hinzugefügt.

### MachineManagement - Komponente



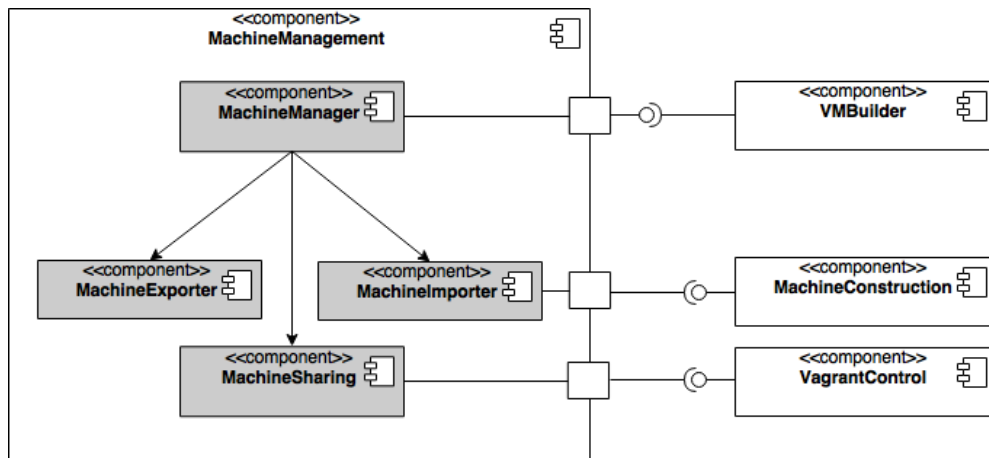


Abbildung 5.18: Komponentenansicht MachineManagement

Um dem Anwender Optionen für die bereits bestehenden virtuellen Maschinen anbieten zu können, gibt es den **MachineManager**. Dieser agiert als Verwalter für Optionen, die auf virtuelle Maschinen ausgeführt werden können. Wie in [Abbildung 5.15](#) werden Optionen wie Export-, Import- und das Sharing von Maschinen angeboten, die durch hinzufügen weiterer Komponenten erweitert werden können. Die **MachineSharing** - Komponente bereitet eine virtuelle Maschine so vor, dass auf sie von überall aus zugegriffen werden kann. Die einzige Beschränkung sind die Richtlinien des Netzwerkes. Zudem ist der **MachineExporter** in der Lage eine virtuelle Maschine zu exportieren, in dem er Konfigurationsdateien packt und dem Anwender zur Verfügung stellt. Diese Dateien können in anderer Virtualisierungsprodukte geladen werden oder durch den **MachineImporter** wieder in die Anwendung importiert werden. Durch die Hinzunahme des **BuildProcessor** kann der Import wieder zu einer virtuellen Maschine aufgebaut werden.

### VMBuilder - Komponente

Die VMBuilder-Komponenten ist der Informationslieferant für die MainRepresentation - Komponente aus der Benutzerschnittstelle. Da die MainRepresentation die meisten Optionen für virtuelle Maschinen anbietet, liegt die Steuerung der Optionen, ebenfalls im Verantwortungsbereich der VMBuilder-Komponente.

#### 5.5.1.3 Datenebene

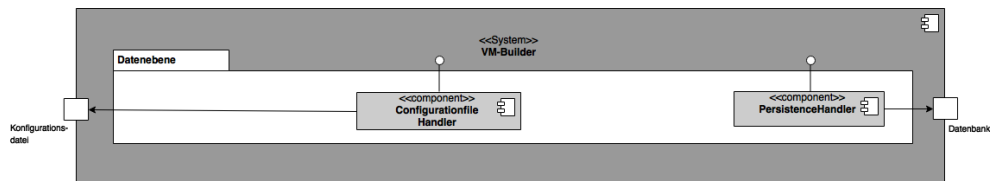


Abbildung 5.19: Ansicht der Datenebene

In der Datenschicht werden Funktionen verankert, die zum direkten Lesen aus dem Datenspeicher verwendet werden. Dies können Funktionen sein, die mittels SQL-Abfragen auf die Datenbank zugreifen oder die lesenden- und/oder schreibenden Zugriff auf Dateien ermöglichen.

### PersistenceHandler - Komponente

Der **PersistenceHandler** ist eine der beiden Komponenten in der Datenebene. Durch die dort definierte Funktionen werden kontrollierte Zugriffe auf die Datenhaltung ermöglicht. Manipulation der Daten soll ausschliesslich durch diese Komponente erfolgen. Der **PersistenceHandler** wird somit zur Schnittstelle Richtung Datenbank.

### ConfigurationfileHandler - Komponente

Die andere Komponente in der Datenebene ist der **ConfigurationfileHandler**. Durch sie werden Grundeinstellungen der VMBuilder - Applikation aus einer Konfigurationsdatei für das System les- und editierbar. Diese enthält Applikationseinstellungen, Einstellungen für Vagrant und Konfigurationen für Ansible. Der **ConfigurationfileHandler** extrahiert diese drei Einstellungstypen heraus und bereitet sie für den entsprechenden Anwendungszweck auf. So kann z.B. der **AnsibleYAMLGenerator** seine benötigten Informationen aus dem **ConfigurationfileHandler** beziehen.

## 5.5.2 Laufzeitsicht

Nach **Zörner (2012)** zeigt die Laufzeitsicht Elemente der Bausteinsicht in Aktion, veranschaulicht dynamische Strukturen und das Verhalten des Systems. Um einen interessante Aspekte des VM-Builders herauszufiltern, wird dieser in die Laufzeitsicht übernommen. Die dort verwendeten Funktionsaufrufe sind eine Abstraktion der späteren Implementierung, da im Entwurf Programmiersprachen unabhängig gearbeitet, sowie auf Implementierungsdetails verzichtet wird.

### 5.5.2.1 Aufbau einer virtuellen Maschine

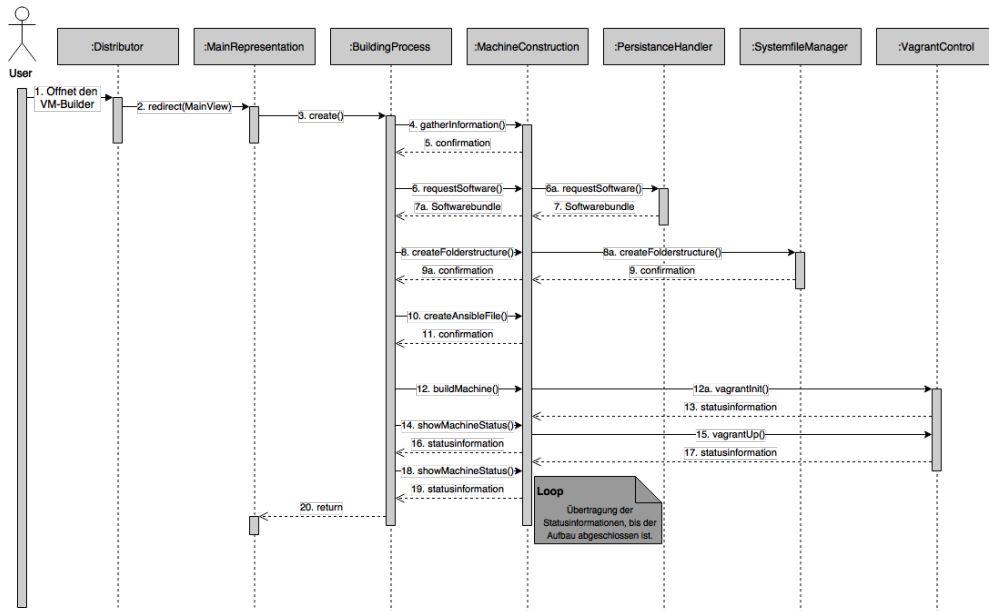


Abbildung 5.20: Laufzeitsicht eines VM-Aufbaus

Der Aufbau einer virtuellen Maschine ist eines der prägnanten Leistungsmerkmale, der hier zu entwerfenden Software. Das Sequenzdiagramm aus Abbildung 5.20 verdeutlicht den Aufbau einer virtuellen Maschine, in dem die Kommunikation zwischen den beteiligten Komponenten etwas genauer aufgezeigt wird.

#### Ausgangssituation:

Der Anwender hat Zugriff auf die Applikation und möchte sich eine virtuelle Maschine erstellen.

1. Der Anwender öffnet im Webbrowser dem VM-Builder...
2. ...und wird durch den Distributor auf die Hauptseite geleitet.
3. Dort hat der Anwender die Möglichkeit eine neue Maschine zu erstellen, in dem er dort den Create-Button betätigt. Daraufhin wird der erste Teil des Aufbaus im Buildingprocess angestoßen.

4. Nachdem der Anwender Eingaben bezüglich der Grundinformation eingegeben hat, werden diese Informationen in der MachineConstruction gespeichert und ...
5. ... eine Rückmeldung über den Erfolg an die Ansicht gegeben, damit der nächste Schritt des Installationsprozesses aufgerufen werden kann.
6. Dazu benötigt die dort verwendete Ansicht alle Softwarekomponenten, die auf einer Maschine installiert werden können. Diese Informationen erhält die Sicht über einen Request an die MachineConstruction.
- 6a. Damit die angeforderten Daten an die anfragende Ansicht übermittelt werden könne, holt sich die MachineConstruction über den PersistenceHandler die Daten aus dem Datenspeicher.
7. Die Daten werden vom PersistenceHandler in Form eines Bundles an die MachineConstruction zurückgegeben,
- 7a. die wiederum der Darstellung zur Verfügung gestellt werden, damit der Anwender seine Auswahl treffen kann.
8. Nach der Auswahl der Softwarekomponenten, weist die Ansicht die MachineConstruction an, die Ordnerstruktur für die virtuelle Maschine zu erstellen.
- 8a. Um die Ordnerstruktur erstellen zu können, wird auf den SystemFileManager zurückgegriffen, der Dateisystemfunktionen bereitstellt.
9. Sind die Dateistrukturen angelegt worden, wird dies durch eine Rückmeldung des SystemfileManagers quittiert.
- 9a. Die nach Erhalt den nächsten Bearbeitungsschritt in der Oberfläche einleitet.
10. MachineConstruction konstruiert durch die Daten aus Schritt 4. die Konfigurationsdateien und ...
11. ... bestätigt dies.
12. Nach der Bestätigung aus Schritt 11. wird der eigentliche Aufbauprozess initialisiert.
- 12a. Danach wird VagrantControl angesprochen um den initialen Prozess in Vagrant anzustossen.

13. - 19 Durch automatisierten Aufbau von Vagrant, werden Statusinformationen erzeugt, die von VagrantControl verarbeitet und an die MachineConstruction weitergeleitet werden. Ein Asynchroner Prozess erfragt während des Aufbauprozesses immer wieder den Status, um den aktuellen Stand in der Anwendung anzeigen zu können.
20. Nach dem erfolgreichem Aufbau, wechselt die Anwendung wieder auf die Hauptseite.

### 5.5.3 Datenbank

Wie in der Verteilungssicht (Abbildung 5.3) beschrieben, wird der Server eine relationale Datenbank bereit halten, deren Zuständigkeitsbereich im Speichern von virtuellen Maschinen, deren Konfigurationen und der Verwaltung von Softwarebestandteilen liegt. Optionen bezüglich des Verhaltes von Dateien werden zusätzlich in einer separaten Tabelle abgelegt. Für den Entwurf der Datenbank, werden zwei Notationsformen verwendet. Abschnitt 5.5.3.1 veranschaulicht in der Notation des Entity-Relationship-Model den ersten konzeptionellen Entwurf des Datenmodells, während Abschnitt 5.5.3.2 die tabellarische Umsetzung des Konzeptes klärt.

#### 5.5.3.1 Entity-Relationship-Model

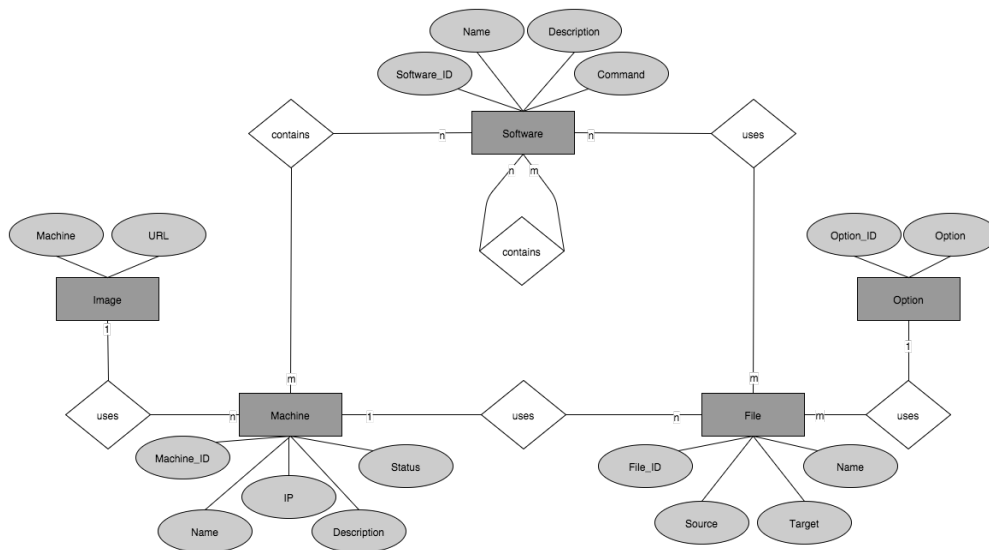


Abbildung 5.21: Entity-Relationship-Model

Die in der Abbildung 5.21 veranschaulichte Konzipierung, zeigt das semantische Konstrukt der angestrebte Datenbank, wobei die folgende Beschreibung die Bedeutung und die Abhängigkeiten des Modells erklärt. Die Tabelle **Machine** soll zukünftig jede Maschine beinhalten,

die durch den Anwender aufgebaut wurde, bis hin zu dem Zeitpunkt ihrer Löschung. Die eindeutige Identifizierung jeder Maschine wird durch eine **ID** gelöst, die bei jedem Speichern einer Maschine fortlaufend hochgezählt wird. Die Tabelle enthält zusätzlich zu der **ID** der Maschine, den **Name**, eine optionale Beschreibung (engl. **Description**), den aktuellen **Status** (Online/Offline) und die **IP-Adresse** der Maschine.

Um den Anwender von Beginn an eine Auswahl an VM-Images anbieten zu können, werden in der Tabelle **Image** die vorgefertigten VM-Images abgelegt. Die Tabelle besteht aus den Attributen **Name**, das den Namen des Images beinhaltet und einer **URL**, die auf das Image im Web verweist.

Die Tabelle **File** speichert Informationen zu Dateien und stellt sie in eine Relation zur virtuellen Maschine oder Softwarepaketen. Nicht nur der **Name** der Datei wird persistiert, sondern auch das Quellverzeichnis (engl. **Source**) und das Zielverzeichnis (engl. **Target**), das auf das Zielverzeichnis der virtuellen Maschine zeigt. Für die eindeutige Identifikation, bekommt jede Datei eine **ID** zugewiesen, die fortlaufend inkrementiert wird. Damit der VM-Builder in der Lage ist, Dateien wiederzuverwenden, werden in der Tabelle **Option** Eigenschaften wie das Kopieren oder Entpacken von Dateien hinterlegt. Jeder Datei, die im VM-Builder gespeichert wird, kann solch eine Eigenschaft zugewiesen werden, um eine immer wieder identisches Verhalten zu erhalten. Die Tabelle selbst besteht aus einer **ID**, durch die jede Option eindeutig identifiziert werden kann und dem Attribut **Option**, welches die Option beschreibt.

Damit dem Anwender eine Softwareauswahl angeboten werden kann, wird die Tabelle **Software** benötigt. Jeder Eintrag besteht aus einer automatisch generierten, fortlaufenden **SoftwareID**, dem **Namen** der Software, einer optionalen Beschreibung (engl. **Description**) und der zwingend notwendigen Befehlszeile (engl. **Command**), die den Linux-Befehl enthält, mit dem die Software installiert wird.

Die Abbildung 5.21 zeigt zusätzlich eine rekursive Eigenschaft der Tabelle **Software**. Die Rekursion entsteht durch die gewünschte Funktion der Softwarepaketerstellung. Da jedes Softwarepaket aus mehreren Softwarekomponenten bestehen kann, ein Softwarepaket aber auch in der Liste der Softwarekomponenten geführt wird, entsteht so die rekursive Eigenschaft. Die genaue Umsetzung von rekursiven Tabellen, wird im folgenden Abschnitt genauer erklärt.

### 5.5.3.2 Relationales Datenbank Modell

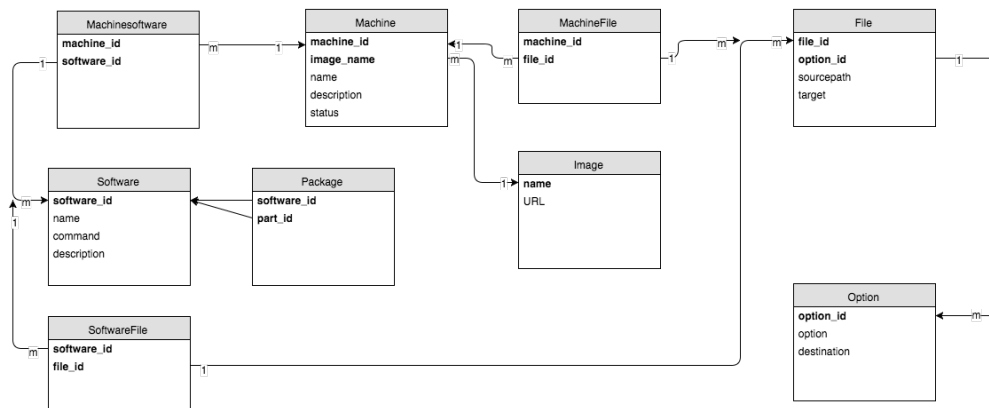


Abbildung 5.22: RelationalDatabaseModel

Die zweite Darstellung konkretisiert den Datenbankentwurf durch eine detaillierte Ansicht der Tabellenkonstrukte und erleichtert die Implementierung der einzelnen Tabellenstrukturen. Dafür werden im Weiteren Beziehungen zwischen Tabellen aufgelöst und ggf. neue Tabellen hinzugefügt. Zudem werden Primär- und Fremdschlüssel Beziehungen verdeutlicht, um die Darstellung nah an die Implementierung zu bringen.

Ein Primärschlüssel wird immer aus einer Menge von Schlüsselkandidaten bestimmt, deren Definition in [Pernul \(2003\)](#) lautet:

’Ist die identifizierte Attributmenge "minimal", d.h. ein Oberschlüssel, aus dem keine Attribute gestrichen werden können, ohne dass die Schlüsseleigenschaften verloren gehen, so handelt es sich um einen Schlüsselkandidaten K.’

Ein Fremdschlüssel wird dadurch ausgezeichnet, dass er entweder ein Primärschlüssel oder ein Schlüsselkandidat aus einer anderen Tabelle ist. Da das RDM (Relationales Datenbank Modell) nah an die reale Umsetzung angelehnt ist, erfordert es die Auflösung der Beziehungstypen aus dem vorherigen Abschnitt [5.5.3.1](#).

Die erste betrachtete Regel bezieht sich bei der Umwandlung auf N:M Beziehungen. Nach [Jarosch \(2002\)](#) müssen N:M Beziehungen in 1:N und N:1 Typen umgeformt werden, wodurch eine Zwischentabelle (Koppeltabelle) entsteht. Angewendet auf [Abbildung 5.21](#) bedeutet das, dass neue Tabellen entstehen müssen, die jeweils als Relation zwischen den folgenden Tabellen entstehen:

1. **Software und Machine**
2. **Software und File**

### 3. **Machine** und **File**

### 4. **Software** und **Software**

Die neu entstandenen Zwischentabellen nutzen die Primärschlüssel der Tabellen, die vorher N:M verknüpft waren, als Fremdschlüssel und wandeln ihn in ihren eigenen Primärschlüssel um. So werden beispielsweise die Tabellen **Machine** und **Software** mit der Zwischentabelle **MachineSoftware** verknüpft und die Umformung in eine 1:N und N:1 Beziehung durchgeführt. Dies gilt auch für die Auflösung der Beziehungen zwischen den Tabellen **Software** und **File** und **Machine** und **File**. Die vierte M:N Beziehung beinhaltet einen speziellen Faktor, der eine andere Herangehensweise benötigt. Die Besonderheit hier besteht in der Rekursion, die auf die eigene Tabelle zeigt und nach **Pernul (2003)** wie folgt aufgelöst werden muss:

‘Rekursive Beziehungstypen beschreiben eine Beziehung verschiedener Entities innerhalb eines einzigen Entitytypen. Durch diesen Sachverhalt stehen wir im Relationenmodell vor dem Problem, dass nach der Transformation die Attribute des Primärschlüssels nun zweimal auftauchen. Eine Forderung des rationalen Datenbankmodells ist aber die Eindeutigkeit von Attributbezeichnungen. Aus diesem Grund muss eine der beiden Bezeichnungen - in Anlehnung an die Art der Beziehung - umbenannt werden.[...] Bei einer Kardinalität M:N muss ein zusätzliches Relationenschema erstellt werden, dass als Attribut zumindest den Primärschlüssel und den umbenannten Primärschlüssel aus dem Entitytyp-Relationenschema enthält. In dem neuen Relationenschema bilden dann beide Attributmengen gemeinsam den Primärschlüssel.’

Die Transformierung besagt, dass im ersten Schritt auch hier eine neue Tabelle entstehen muss. Da die Rekursion durch die Verwendung von Softwarepaketen entstanden ist, wird die neue Tabelle **Package** genannt. Im nächsten Schritt muss nach **Pernul (2003)**, die neue Tabelle **Package** den Primärschlüssel aus der Tabelle **Software** zwei Mal enthalten, wobei darauf zu achten ist, dass ein Schlüssel unbenannt werden muss. Das entspricht in diesem Fall der **partID**, wodurch die Umwandlung abgeschlossen ist.

Zu guter Letzt werden die 1:N Beziehungen betrachtet, die den folgenden Relationen entsprechen:

#### 1. **File** und **Option**

#### 2. **Image** und **Machine**



Um auch hier die Umwandlung korrekt vorzunehmen, müssen folgende Regeln beachtet werden:

1. Relationen, die aus dem Entitätstyp mit Kardinalität 1 gebildet wurden, bleiben erhalten.
2. Primärschlüssel aus der Relation mit Kardinalität 1 wandern als Fremdschlüssel in die N-Relation.

Als Beispiel für die Umsetzung der Regeln, werden die Tabellen **File** und **Option** herangezogen. Der Primärschlüssel aus Tabelle **Option** wird in Tabelle **File** als Fremdschlüssel eingesetzt und wird so als ein Teil der Tabelle **File**. Da die Regeln besagen, dass die Tabelle mit der Kardinalität 1 unverändert bleibt, wird an der Tabelle **Option** keine weitere Veränderung vorgenommen.

### 5.6 Client

Im Gegensatz zum Server benötigt der Client keine ausführliche Planung und Modellierung, da das angestrebte Konstrukt ein Thin Client ist. Diese Art von Clients benötigt wenig oder gar keine Anwendungsteile auf dem Client-Rechner. Auf Anwenderseite wird entsprechend nur die Weboberfläche des VM-Builders aufgerufen, damit die Funktionalitäten bereitstehen. Es wird keine lokale Datenbank oder andere Software zum Betrieb benötigt. Entsprechend ist Logik, die der Client benötigt, auf dem Server verankert. Dies gilt auch für die Darstellungen der angezeigten Webelemente. Sie werden durch die Views in den entsprechenden Komponenten der Benutzerschnittstelle realisiert und zur Verfügung gestellt. Eine detaillierte Ansicht zu der Kommunikation zwischen Client und Server, sowie deren Ablauf ist z.B. in der Laufzeitsicht (Abschnitt 5.5.2) zu finden.

### 5.7 Zusammenfassung

Die einzelnen Phasen des Entwurfs verhelfen nicht nur den Aufbau der Software besser zu verstehen, sondern ermöglichen auch eine klare Strukturierung der Anwendung. Die Kontextabgrenzung in Abschnitt 5.1 zeigt den VM-Builder inklusive seiner Umsysteme und verschafft eine erste Übersicht, während die Verteilungssicht (Abschnitt 5.2) eine Stufe heran zoomt und die Platzierung geplanten Softwarebestandteile im VM-Builder verdeutlicht. Abschnitt 5.3 beschäftigt sich mit zwei Systemarchitekturen, die die Anwendung strukturieren und Zuständigkeiten definieren. Die erste Modell der Architektur ist das Client-Server-Modell, wodurch eine erste konkrete Entscheidung der Aufgabenverteilung zwischen Client und Server getroffen wurde, während das MVC-Entwurfsmuster die Zuständigkeiten der einzelnen Schichten

bestimmt.

Die darauf folgende Übersicht der Kommunikation geht exemplarisch auf das Verhalten von Sinatra ein und zeigt die Annahme von Clientaufrufen und den Aufbau von Routen. Siehe Abschnitt 5.4.

Nachdem die Basis definiert wurde, befasst sich Abschnitt 5.5 mit der Konzipierung der Serverseite, unter der Verwendung der Bausteinsicht (Abschnitt 5.5.1) und der Laufzeitsicht (Abschnitt 5.5.2). Die Bausteinsicht zeigt die Komponenten der einzelnen Schichten, deren Verbindungen zu anderen Komponenten und Abhängigkeiten zu der darunter liegenden Schicht. Während die Komponenten in den Schichten noch als Blackboxen dargestellt sind, werden sie in den einzelnen Erklärungen als Whitebox gezeigt, um eine detailliertere Ansicht auf deren Aufbau und Kommunikation zu erhalten. Die Wirkungsweise der Komponenten kann durch die Laufzeitsicht gezeigt werden, die exemplarisch Abläufe darstellt, um so die Aufgabenverteilung der involvierten Komponenten herauszustellen. Abschnitt 5.5.3 befasst sich mit dem konzeptionellen Entwurf des Datenbankschemas und dessen Umwandlung für die spätere Implementierung. Zu guter Letzt wird der Client beschrieben. Da es sich bei dem Client-Konstrukt um einen ThinClient handelt, der ohne weitere lokale Software verwendbar ist, kommt die Beschreibung in Abschnitt 5.6 mit wenig Erklärung aus.

## 6 Realisierung

Die Realisierung zeigt in wie fern die vorherige Planung der Realität stand hält und ob Änderungen in der Struktur oder Funktionalität notwendig sind. Zudem werden Entscheidungen und Problematiken verdeutlicht, die bei der Planung noch nicht zu erkennen waren. Die Implementierung wird, wie in den Anforderungen beschrieben (Kapitel 3.6.1.), mit dem Framework Sinatra und Ruby umgesetzt. Jeder Umsetzungsschritt basiert auf der Anforderungsanalyse und dem darauf aufbauenden Entwurf. Geplante Fremdsoftware wird entsprechend mit berücksichtigt und eingebunden. Zu beachten ist, dass hier ein Prototyp der Anwendung umgesetzt wird und Funktionalität entsprechend nicht komplett implementiert werden. Im späteren Kapitel Aussichten, werden weitere Funktionalitäten betrachtet, die für die Applikation praktikabel wären. Zudem wird die Applikation als ein Prototypen-Szenario aufgebaut, welches kleineren Hardwarespezifikationen und einem Anwenderzugriff zur Zeit genügt. Die Realisierung besteht aus mehreren Schritten, die aufbauend auf einander sind. Im ersten Schritt wird in 6.1 auf das Hardware-Serverkonstrukt näher eingegangen, welches der Planung und den Anforderungen entsprechend umgesetzt werden soll. Der zweite Schritt in Abschnitt 6.2 befasst sich mit den nötigen Fremdsoftware-Komponenten, wie z.B. des Webserver, seinen zugehörigen Softwarekomponenten, des Provisionierers, Virtualisierers und diversen Anderen. Schritt Drei beschäftigt sich dann mit der Implementierung der eigentlichen Applikation.

### 6.1 Hardware

Um die Voraussetzungen für die Entwicklung des 'VM-Builders' zu schaffen, wird statt auf einen Hardware-Server verzichtet und stattdessen über VirtualBox ein virtueller Server aufgebaut. Wie in der Einleitung bereits erwähnt, handelt es sich bei der Entwicklung um einen Prototypen der Applikation. Aufbauend auf diesem Aspekt, ist eine virtuelle Umgebung zum entwickeln und testen mehr als geeignet. Im Rahmen des Prototypen können gewissen Operationen bezüglich der virtuellen Maschinen ggf. noch nicht zur Verfügung stehen. Um die Aufbaugeschwindigkeit und das Verhalten auf unterschiedlicher Infrastruktur testen zu können, werden zwei virtuelle Server gleichzeitig aufgebaut. Einer Server wird lokal erstellt, der Andere auf einem VM-Host der HAW-Hamburg. In den folgenden Abschnitten wird lediglich ein

Server betrachtet. Die Unterschiede der Server, sind nur rein technischer Natur und sollen ggf. bei der Analyse des Zeitfaktors unterstützen, der für den Aufbau einer virtuellen Maschine benötigt wird.

Im lokalen Aufbau wird ein virtueller Server mit 4GB Arbeitsspeicher, 20 GB Festplattenkapazität und Ubuntu 14.04.2 realisiert. Parallel wird der Server auf HAW-Seite mit 16 GB Arbeitsspeicher ausgestattet, um die Konfiguration gegeneinander stellen zu können.

## 6.2 Fremdsoftware

Für die ordnungsgemäße Funktion des 'VM-Builders' wird auf Fremdsoftware zurückgegriffen, das sich gleichzeitig auf den zweiten Schritt der Umsetzung bezieht. In der Evaluation aus Kapitel 4. sind die ersten Annahmen getroffen worden, welche Softwarekomponente sich für bestimmte Aufgabenbereiche am besten eignet. Ebenso wurden in der Verteilungssicht aus Kapitel 5.2 weitere Softwarekomponenten empfohlen. Im Folgenden werden die Empfehlungen und evaluierten Softwarekomponenten in ihrem Einsatz beschrieben. Auf Installationsdetails und der Beschreibung von ggf. notwendiger Drittsoftware wird verzichtet.

### 6.2.1 Webserver

Für die Präsentation des Webinterfaces und die Kommunikation zwischen Client und Server wird auf den frei erhältlichen Apache Webserver, in der Version 2.4.7 zurückgegriffen.

Durch die freie Verfügbarkeit von Apache, wird der geforderte Open-Source-Aspekt aus 3.6.1 berücksichtigt. Zudem hat Apache einen Marktanteil von 50,51% an aktiven Webseiten, nach Statista (2015). Die Verwendung von Apache hat den entsprechenden Vorteil, dass durch den Bekanntheitsgrad, eher Fachwissen über Konfiguration und Administration bei Administratoren vorliegen könnte.

### 6.2.2 Phusion Passenger

Im Zuge der Vorbereitung der Applikation wird auf Phusions Passenger zurückgegriffen, das ein Modul für Apache darstellt. Phusions Passenger wird gerade im Bezug auf Ruby-Applikationen verwendet, um Ruby-Web-Applikationen bereitzustellen. Zudem erleichtert Passenger die Administration von Web-Applikationen und ist essenziell für den Betrieb des VM-Builders.

### 6.2.3 VirtualBox

Ein eigener Entwurf und deren Ausführung einer Virtualisierungsstrategie, wäre weder ziel führend noch praktikabel. Deshalb wurde in der Evaluation (Kapitel 4.2.3.) eine Analyse der frei zugänglichen Virtualisierer durchgeführt. VirtualBox übernimmt beim 'VM-Builder' den Part der Maschinenvirtualisierung. Wie bei Apache ist mit einer der Argumente für VirtualBox, die kostenlose Verfügbarkeit und das es Open-Source ist.

### 6.2.4 Ansible

Die Aufgabe der Provisionierung der einzelnen virtuellen Maschinen, wird Ansible übertragen. Ansible kann die gewünschte Zustandsbeschreibung direkt auf der virtuellen Maschine umsetzen, ohne einen Client dort zu installieren. Dies erleichtert nicht nur die Arbeit mit Ansible, sondern erleichtert zudem die Implementierung in den 'VM-Builder'. Wie bei VirtualBox, sind liegen die Gründe für Ansible bei dem Open-Source Gedanken und den Ergebnissen der Evaluation aus Kapitel ??.

### 6.2.5 Vagrant

Vagrant dient als Wrapper für VirtualBox und Ansible. Die Flexibilität von Vagrant ermöglicht das leichte Zusammenspiel der drei Softwarekomponenten. Da Vagrant mit anderen Provisionierern und Virtualisierern zusammenarbeiten kann, ist ein Austausch von Ansible und VirtualBox in einem begrenzten Zeitrahmen möglich.

Der 'VM-Builder' wird über Befehlsaufrufe direkt mit Vagrant kommunizieren und durch entsprechende Befehle den Aufbau, die Deaktivierung, das Sharing usw. auslösen.

### 6.2.6 Bundler

Um in Ruby-Projekten die notwendige Abhängigkeiten zu genutzten Paketen zu verwalten und ggf. deren Versionen zu spezifizieren, wird auf Bundler zurückgegriffen. Durch Bundler können einheitliche Umgebungen für Projekte geschaffen und der Wiederaufbau der gleichen Umgebung beschleunigt werden. Z.B. kann eine automatisierte Installation des 'VM-Builders' durch Bundler vereinfacht werden.

### 6.2.7 Datenbank

Für die Umsetzung der Datenbank wird auf den OR-Mapper **Datamapper** zurückgegriffen. ORM Frameworks sind gerade für objektorientierte Sprachen besonders hilfreich, da sie die

objektorientierte Welt mit der Relationalen Welt der Datenbanken in Einklang bringen. So ist es möglich programmatisch Tabellenkonstrukte zu definieren und zu erstellen. Durch den Aufruf von Datamapper wird nicht nur die Tabelle nach den Vorgaben erstellt, sondern auch die Beziehungen zwischen den Tabellen.

```
1 class Machine
2   include DataMapper::Resource
3
4   property :id, Serial
5   property :name, String, :length => 255, :required => true
6   property :ip, String, :length => 15
7   property :description, String, :length => 255
8   property :status, Integer
9
10  has n, :files
11  has n, :machinessoftwares
12  has n, :softwares, :through => :machinessoftwares
13  belongs_to :vmimage
14 end
```

Listing 6.1: ORM Framework Datamapper

Ein Beispiel ist die Definition der Tabelle *Machine*, die aus vier Attributen besteht inklusive ihrer Typ-Eigenschaften, sowie ihrer Relation zu anderen Tabellen. Programmatischer Zugriff auf die Tabellen kann über SQL-Befehle bereitgestellt werden, oder über typischerweise vorhandenen Befehle des ORM-Frameworks. Um z.B. aus der Tabelle **Machine** alle Daten einmal abzurufen genügt der Befehl 'Machine.all'.

```
1 def all_machines?
2   Machine.all
3 end
```

Listing 6.2: ORM Framework command

## 6.3 Implementierung

Nach dem Aufbau der Server und der Installation der benötigten Softwarekomponenten, steht im nächsten Abschnitt die Implementierung des Prototypen im Vordergrund. Ziel des

Prototypen ist grundlegende Funktionen zu implementieren und zu testen, in wie weit sie durch die Anforderung realisierbar sind.

Entsprechend stehen folgende Funktionalitäten im Mittelpunkt:

- Automatisierter Aufbau einer virtuellen Maschine
- Provisionierung
- Speichern von VM-Konfigurationen
- Softwarebundles erstellen
- Virtuelle Maschine 'sharen'

Zuvor wird ein Blick auf strukturellen Ansätze der Entwicklung geworfen, in dem allgemein auf die Komponenten-Umsetzung eingegangen wird und auf entsprechende Änderungen.

### 6.3.1 Komponenten Umsetzung

Wie bereits in der Einleitung des Kapitels erwähnt, wird der 'VM-Builder' mit Ruby inklusive dem Framework Sinatra umgesetzt. Um dem MVC-Konzept zu entsprechen, werden die definierten Komponenten aus 5.5.1 in der folgenden exemplarischen Ordner-/Dateistruktur (6.3) umgesetzt.

```
1 config.ru
2 app.rb
3 helpers/
4   persistence_handler.rb
5 models/
6   init.rb
7 routes/
8   init_controller.rb
9 views/
10  init_view.erb
```

Listing 6.3: Exemplarische Ordnerstruktur VM-Builder

Controller und Views aus der der Benutzerschnittstelle werden in den Ordnern 'routes' und 'views' erstellt. Bestandteile aus der Verarbeitungsebene und Datenebene werden in 'models' und 'helpers' gespeichert.

Da Ruby/Sinatra zulässt auf Klassen zu verzichten, sind die Controller als organisatorische Einheit konzipiert mit der entsprechenden Logik in den verwendeten Modellen.

```
1 require './models/administration_manager'
2 require_relative 'softwareadmin_controller'
3
4 get '/log' do
5   @content = admin_mgr.log_content?
6   erb :logfile
7 end
8
9 put '/log' do
10  admin_mgr.update_log_config(params['logpath'])
11  admin_mgr.create_logfile(params['logpath'])
12  redirect '/admin'
13 end
```

Listing 6.4: Controller Beispiel

Während die Views in HTML geschrieben und Helper, sowie Models als Ruby-Klassen definiert sind.

Durch die Umsetzung haben sich Unterschiede bezüglich des Entwurfs herausgestellt, die folgende Bausteinsicht ergeben haben.



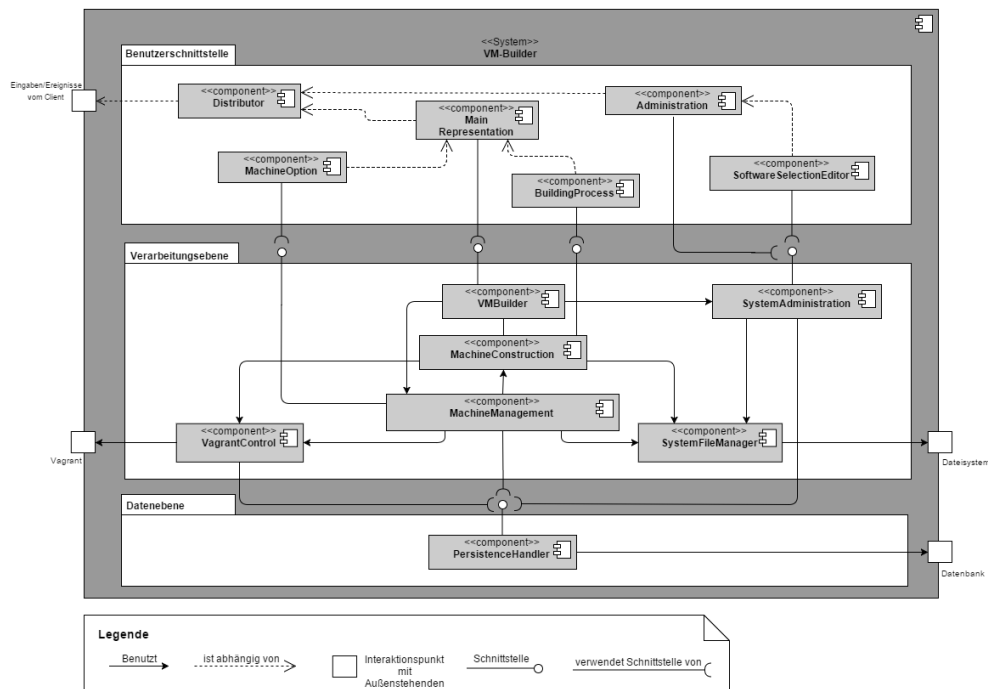


Abbildung 6.1: Bausteinsicht Level 1

Abgesehen von der Entstehung neuen Beziehungen zwischen Komponenten ist eine der wesentlichen Änderungen das Wegfallen des ConfigurationfileHandlers. Seine Aufgabe bestand im lesen und beschreiben einer Konfigurationsdatei, die Konfigurationen des gesamten 'VM-Builders' beinhalten sollte. Da in der Konzeption eine Datenbank mit eingeplant wurde, sind die Konfigurationen dort hinein übernommen worden. Dies hat den Vorteil, dass es eine zentrale Steuerung für den persistenten Speicher besteht und Änderungen, sowie das Hinzufügen von neuen Konfiguration deutlich vereinfacht wird.

### 6.3.2 Datenbank

Wie bereits erwähnt, wird im Gegensatz zur Planung, ein Tabellenkonstrukt angewendet, um die Einstellungen zu persistieren. Dies spart zu erstellende Komponenten und zentralisiert den Zugriff und die Datenhaltung. Zur Umsetzung von Konfigurationstabellen werden in der Regel zwei Konzepte vorgeschlagen:

#### 1. Name-Value-Pair Table

Name-Value Pair Tabellen bestehen nur aus zwei Spalten. Eine für den Konfigurations-

Typen und die Andere für den zugehörigen Wert. Kommen neue Konfigurationen dazu, werden die einfach an die Tabelle angehängen.

1	ConfigOption		Value
2	-----+		
3	CompanyName		ACME Inc.
4	StartFullScreen		true (or 1, or Y, ...)
5	RefreshSeconds		20
6	...		...

Listing 6.5: Name-Value-Pair Table

**Vorteil**

- a) Neue Konfigurationen benötigen keine Rekonfiguration des Tabellenschemas
- b) Schlanke Tabellenstruktur. Neue Konfigurationen werden einfach angehängen

**Nachteil**

- a) Jede Konfiguration hat den gleichen Typ
- b) Alles wird als String/Varchar gespeichert
- c) Für die Arbeit mit den Werten der Konfigurationstabelle, sollte bekannt sein, welcher Typ sich eigentlich hinter dem Wert verbirgt

**2. Single-Row Table**

Bei Single-Row Tabellen wird die Struktur gedreht. Jede Konfiguration besteht aus einer eigenen Spalte. Die darunter liegende Zeile beinhaltet die Werte.

1							
2	CompanyName		StartFullScreen		RefreshSeconds		...
3	-----+-----+-----+-----						
4	ACME Inc.		true		20		...

Listing 6.6: Single-Row Table

**Vorteil**

- a) Jede Spalte hat bekommt den entsprechenden Datentyp
- b) Durch den exakt angegebenen Typ jeder Spalte, kann in der Programmierung besser mit den Werten umgegangen werden

- c) Jeder Spalte können leichter Standardwerte zugeordnet werden

### **Nachteil**

- a) Das Tabellenschema muss geändert werden, wenn neue Einstellungen hinzugefügt werden
- b) Die Tabelle kann schnell unleserlich werden, wenn zu viele Einstellungen in der Tabelle enthalten sind

In der Umsetzung wurde das Konzept der '**Name-Value-Pair Table**' verwendet. Auch wenn dieses Konzept weniger Vorteile als '**Single-Row**', spricht das leichtere Einpflegen von neuen Konfigurationseigenschaften und die bessere Lesbarkeit, für das Konzept.

### **6.3.3 Funktionen**

Die Realisierung aller geplanten Controller, Models und Views hat das Funktionsspektrum der Applikation fast vollständig erfüllt.

So sind folgende Funktionen durch die bisherige Implementierung erfüllt worden:

#### **Erstellung einer virtuellen Maschine**

Die Komponente 'Main-Representation', die die zentrale Steuerung darstellt, kommuniziert kann die BuildingProcess-Komponente aufrufen in der die Darstellung für den geleiteten Aufbau einer virtuellen Maschine gehalten wird und die Schnittstelle zu der entsprechenden Logik. Der Aufbauprozess beinhaltet die Benennung der gewünschten Maschine, Optionen wie IP-Adresse und die Auswahl an Softwarekomponenten. Echtzeitinformation über den Aufbauprozess sind nicht implementiert. Allerdings im Logfile nachlesbar. VagrantControl liefert in Kombination mit dem SystemFileManager die Befehle für den Aufbau einer Maschine. Der SystemFileManager bietet eine Methode für direkte Systemaufufe an, mit der VagrantControl Vagrant Befehle absetzen kann.

#### **Provisioning**

Die Provisionierung wurde in den Aufbau mit integriert und wird durch die Komponente MachineConstruction realisiert. Um genauer zu sein, durch die Klasse 'YamlBuilder'. Die ausgewählte Software, wird durch diese Klasse in eine YAML-Datei übersetzt und mit Einstellungen aus der Konfigurations-Tabelle angereichert. Beinhaltet die Auswahl an Software Unterkomponenten und/oder Dateien, die auf den Zielhost übertragen werden sollen, wird die YAML-Datei automatisch entsprechend angepasst.

### **VM-Konfigurationen speichern**

Sobald eine virtuelle Maschine erfolgreich aufgebaut wurde, Name und optionale Einstellungen der Maschine, sowie ausgewählten Softwarekomponenten durch Transaktionen im Datamapper gespeichert. Durch die verwendeten Transaktionen, wird vermieden, dass inkonsistente Konfigurationen gespeichert werden und sichergestellt, dass diese auch nur im Erfolgsfall gesichert werden.

Solange die Maschine durch keinen Anwender gelöscht wird, bleiben die gespeicherten Parameter bestehen.

### **Softwarebundles erstellen**

Softwarebundles helfen dabei, Software mit Abhängigkeiten zu anderen Softwarekomponenten und Dateien zu erstellen und im System zu hinterlegen. So wird die Möglichkeit geschaffen, Software zu installieren, die mehrere Installationsschritte benötigt. Die SystemAdministration-Komponente hält dafür die entsprechende Implementierung vor. Da die Implementierung organisatorisch zur Administration gehört, wurde sie in diesem Paket platziert. Auch hier wird die Speicherung der Einstellungen durch Transaktionen bewerkstelligt, um keine Inkonsistenzen zu verursachen. Dem Anwender wird die Möglichkeit geboten, durch seine Eingaben zu entscheiden, was für eine Software gespeichert wird. Wählt er keine Relationen zu anderen Softwarekomponenten aus und lässt er die Möglichkeit aus Dateien zu transferieren, wird automatisch nur eine standard Software zur Auswahl hinzugefügt. Erst durch die Auswahl von Relationen und/oder Dateien, wird es ein Softwarebundle. Die Umsetzung der Auswahl geschieht in dem Model 'SoftwareEditor' der seine Entscheidung an den 'PersistenceHandler' weitergibt.

### **VM-Sharing**

Das 'Sharing' einer bestehenden virtuellen Maschine wird durch die interne Implementierung von Vagrant übernommen. Die Zuständigkeit um die Funktion in den 'VM-Builder' zu übernehmen und auszulösen, übernimmt die Klasse 'VagrantControl'. Wie auch bei dem Aufbau einer virtuellen Maschine, verwendet 'VagrantControl' die Funktion des 'SystemfileManagers' um den entsprechenden Befehl abzusetzen und an Vagrant selber weiterzuleiten. Das Teilen oder 'Sharing' einer virtuellen Maschine wird allerdings durch die Portsperrung des jeweiligen Netzwerkes eingeschränkt. Ist der Port 4567 gesperrt, wird durch Vagrant eine Fehlernachricht generiert und das Teilen der Maschine nicht ausgeführt. Bei erfolgreichem 'Sharing' wird ein SSH-Share erstellt. Die SSH Variante hat den Vorteil, dass Port und ggf. Passwort optional eingerichtet werden können, durch Angaben von erweiterten Parametern.

## 7 Schluss

Der Schluss dieser Arbeit fasst noch einmal alle Kapitel in Kürze zusammen und bildet ein Fazit im Bezug auf das entwickelte Projekt. Der danach folgenden Ausblick, geht auf Funktionen und Anwendungsgebiete der Applikation ein, die zukünftig realisiert werden könnten.

### 7.1 Zusammenfassung und Fazit

Das Ziel dieser Arbeit war es eine Applikation zu entwickeln, die einen schnellen und unkomplizierte Aufbau von virtuellen Maschinen ermöglicht, sowie deren Verwaltung gepaart mit einer leicht erlernbaren Steuerung. Dies erfolgte durch eine detaillierte Anforderungsanalyse in Kapitel 3. In der Anforderungsanalyse wurden die funktionalen- und nichtfunktionalen Anforderungen eruiert und durch die eigenen Anforderungen an die Applikation definiert. Die daraus entstandenen Use-Case formten die ersten Details bezogen auf die Verwendbarkeit und stellten die primären Funktionen heraus. Darauf folgende Mockups verhalfen den Applikationskontext in eine erste Form zu bringen und den Funktionen weiter Abläufe festzuhalten und den Aufbau der Applikation zu verdeutlichen.

### 7.2 Ausblick

# Abbildungsverzeichnis

2.1	Betriebssystemvirtualisierung . . . . .	6
2.2	Hypervisor Typ 1 und 2 . . . . .	7
3.1	Stakeholderdefinition . . . . .	11
3.2	Business-Use-Case Übersicht . . . . .	14
3.3	Entwurf der Verwaltungsoberfläche . . . . .	21
3.4	Entwurf der Aufbauoberfläche - Eingabe Parameter . . . . .	22
3.5	Entwurf der Aufbauoberfläche - Auswahl der Softwarekomponenten . . . . .	22
3.6	Entwurf der Sharing-Oberfläche . . . . .	23
3.7	Entwurf der Export-Oberfläche . . . . .	24
3.8	Entwurf der Import-Oberfläche . . . . .	25
3.9	Entwurf der Einstellungen - Generelle Konfiguration . . . . .	26
3.10	Entwurf der Einstellungen - Logdatei . . . . .	26
3.11	Entwurf der Einstellungen - Softwarekomponenten hinzufügen . . . . .	27
3.12	Technische Randbedingungen . . . . .	28
5.1	Vier Arten von Sichten ( <a href="#">Starke (2014)</a> ) . . . . .	39
5.2	Kontextsicht . . . . .	40
5.3	Verteilungssicht des VM-Builders . . . . .	42
5.4	Client-Server-Anordnungen ( <a href="#">Tanenbaum und van Steen (2007)</a> ) . . . . .	44
5.5	Model-View-Controller ( <a href="#">Wikipedia (2015)</a> ) . . . . .	45
5.6	Model-View-Controller im 3-Schichten-Modell . . . . .	47
5.7	Bausteinsicht Level 1 . . . . .	50
5.8	Benutzerschnittstelle . . . . .	51
5.9	Komponentensicht Distributor . . . . .	51
5.10	Komponentensicht VMBuilder . . . . .	52
5.11	Komponentensicht BuildingProcess . . . . .	53
5.12	Komponentensicht MachineOption . . . . .	54
5.13	Komponentensicht Administration . . . . .	54

5.14	Komponentensicht SoftwareSelection . . . . .	55
5.15	Ansicht der Verarbeitungsebene . . . . .	56
5.16	Komponentenansicht MachineConstruction . . . . .	57
5.17	Komponentenansicht SystemAdministration . . . . .	58
5.18	Komponentenansicht MachineManagement . . . . .	59
5.19	Ansicht der Datenebene . . . . .	60
5.20	Laufzeitsicht eines VM-Aufbaus . . . . .	61
5.21	Entity-Relationship-Model . . . . .	63
5.22	RelationalDatabaseModel . . . . .	65
6.1	Bausteinsicht Level 1 . . . . .	75

# Listings

2.1	Beispiel Inventory-Datei . . . . .	8
5.1	Routen in Sinatra ( <a href="#">Sinatra (2015)</a> ) . . . . .	48
6.1	ORM Framework Datamapper . . . . .	72
6.2	ORM Framework command . . . . .	72
6.3	Exemplarische Ordnerstruktur VM-Builder . . . . .	73
6.4	Controller Beispiel . . . . .	74
6.5	Name-Value-Pair Table . . . . .	76
6.6	Single-Row Table . . . . .	76



# Literaturverzeichnis

- [Balzert 2011] BALZERT, Helmut: *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. 3. Aufl. 2012. Spektrum Akademischer Verlag, 9 2011. – URL <http://amazon.de/o/ASIN/3827417066/>. – ISBN 9783827417060
- [Bengel 2008] BENDEL, Gunther: *Masterkurs Parallele und Verteilte Systeme: Grundlagen und Programmierung von Multiprozessoren, Multiprozessoren, Cluster und Grid (German Edition)*. 2008. Vieweg+Teubner Verlag, 6 2008. – URL <http://amazon.de/o/ASIN/3834803944/>. – ISBN 9783834803948
- [Burge und Brown 2002] BURGE, Janet E. ; BROWN, David C.: *NFR's: Fact or Fiction?* / Computer Science Department WPI, Worcester. URL <http://web.cs.wpi.edu/~dcb/Papers/CASCON03.pdf>, 2002. – Forschungsbericht
- [Hall 2013] HALL, Daniel: *Ansible Configuration Management*. Packt Publishing, 11 2013. – URL <http://amazon.com/o/ASIN/1783280816/>. – ISBN 9781783280810
- [Harris und Haase 2011] HARRIS, Alan ; HAASE, Konstantin: *Sinatra: Up and Running*. 1. O'Reilly Media, 12 2011. – URL <http://amazon.com/o/ASIN/1449304230/>. – ISBN 9781449304232
- [Hock 2012] HOCK, Jürgen: *OpenNebula - The Open Source Solution for Data Center Virtualization* / Hochschule Mannheim Institut für Softwaretechnik und Datenkommunikation, Mannheim. URL [http://baun-vorlesungen.appspot.com/SEM12/Dokumente/CLCP\\_SEM\\_SS2012\\_OpenNebula\\_Ausarbeitung.pdf](http://baun-vorlesungen.appspot.com/SEM12/Dokumente/CLCP_SEM_SS2012_OpenNebula_Ausarbeitung.pdf), 2012. – Forschungsbericht
- [Jarosch 2002] JAROSCH, Helmut: *Datenbankentwurf: eine beispielorientierte Einführung für Studenten und Praktiker*. Braunschweig Wiesbaden : Vieweg, 2002. – ISBN 3528058005
- [Llorente 2014] LLORENTE, Ignacio M.: *OpenNebula vs. OpenStack: User Needs vs. Vendor Driven*. 2014. – URL <http://opennebula.org/opennebula-vs-openstack-user-needs-vs-vendor-driven/>

- [Loschwitz und Syseleven 2015] LOSCHWITZ, Martin ; SYSELEVEN: *OpenStack*. 2015. – URL <http://www.golem.de/news/openstack-viele-brauchen-es-keiner-versteht-es-wir-erklaeren-es-1503-112814.html>
- [Masak 2009] MASAK, Dieter: *Der Architekturreview: Vorgehensweise, Konzepte und Praktiken (Xpert.press)*. 2010. Springer, 11 2009. – URL <http://amazon.de/o/ASIN/3642016588/>. – ISBN 9783642016585
- [Peacock 2013] PEACOCK, Michael: *Creating Development Environments with Vagrant*. Packt Publishing, 8 2013. – URL <http://amazon.de/o/ASIN/1849519188/>. – ISBN 9781849519182
- [Pernul 2003] PERNUL, Günther: *Datenbanken im Unternehmen : Analyse, Modellbildung und Einsatz*. München Wien : Oldenbourg, 2003. – ISBN 3486272101
- [Rechtin und Maier 2000] RECHTIN, Eberhardt ; MAIER, Mark: *The Art of Systems Architecting, Second Edition*. 0002. Crc Pr Inc, 6 2000. – URL <http://amazon.de/o/ASIN/0849304407/>. – ISBN 9780849304408
- [Reuther 2013] REUTHER, Claus: *Virtualisierung - VMware und Microsoft im Vergleich*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 1 2013
- [Rhett 2015] RHETT, Jo: *Learning Puppet 4*. 1. O'Reilly Vlg. Gmbh and Co., 8 2015. – URL <http://amazon.de/o/ASIN/1491907665/>. – ISBN 9781491907665
- [Rupp und die SOPHISTen 2014] RUPP, Chris ; SOPHISTEN die: *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*. 6., aktualisierte und erweiterte Auflage. Carl Hanser Verlag GmbH und Co. KG, 10 2014. – URL <http://amazon.de/o/ASIN/3446438939/>. – ISBN 9783446438934
- [Schäfer 2009] SCHÄFER, Werner: *Softwareentwicklung - inkl. Lerntest auf CD: Einstieg für Anspruchsvolle (Master Class)*. 1. Addison-Wesley Verlag, 12 2009. – URL <http://amazon.de/o/ASIN/3827328519/>. – ISBN 9783827328519
- [ScriptRock 2014] SCRIPTROCK: *Ansible vs. Salt*. Januar 2014. – URL <https://www.scriptrock.com/articles/ansible-vs-salt>
- [Seneca 2005] SENECA: *Von der Kürze des Lebens*. Deutscher Taschenbuch Verlag, 11 2005. – URL <http://amazon.de/o/ASIN/342334251X/>. – ISBN 9783423342513

- [Siegert und Baumgarten 2006] SIEGERT, Hans-Jürgen ; BAUMGARTEN, Uwe: *Betriebssysteme: Eine Einführung*. überarbeitete, aktualisierte und erweiterte Auflage. Oldenbourg Wissenschaftsverlag, 12 2006. – URL <http://amazon.de/o/ASIN/3486582119/>. – ISBN 9783486582116
- [Sinatra 2015] SINATRA: *Sinatra Einführung*. Juni 2015. – URL <http://www.sinatrarb.com/intro-de.html#Bedingungen>
- [Starke 2011] STARKE, Gernot: *Software-Architektur kompakt (IT kompakt)*. 2nd Printing. Spektrum Akademischer Verlag, 3 2011. – URL <http://amazon.de/o/ASIN/3827420938/>. – ISBN 9783827420930
- [Starke 2014] STARKE, Gernot: *Effektive Softwarearchitekturen: Ein praktischer Leitfaden*. 6., überarbeitete Auflage. Carl Hanser Verlag GmbH und Co. KG, 1 2014. – URL <http://amazon.de/o/ASIN/3446436146/>. – ISBN 9783446436145
- [Statista 2015] STATISTA: *marktanteil-der-meistgenutzten-webserver*. 2015. – URL <http://de.statista.com/statistik/daten/studie/181588/umfrage/marktanteil-der-meistgenutzten-webserver/>
- [Tanenbaum und van Steen 2007] TANENBAUM, Andrew S. ; STEEN, Maarten van: *Verteilte Systeme: Prinzipien und Paradigmen (Pearson Studium - IT)*. 2., aktualisierte Auflage. Pearson Studium, 11 2007. – URL <http://amazon.de/o/ASIN/3827372933/>. – ISBN 9783827372932
- [Wikipedia 2015] WIKIPEDIA: *Model-view-controller* — *Wikipedia - The Free Encyclopedia*. 2015. – URL <https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/MVC-Process.svg/2000px-MVC-Process.svg.png>
- [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

# **Anhang**

## **3 Mockups**

### **3.1 Aufbauprozess**

### **3.2 Optionen**

### **3.3 Einstellungen**

*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