



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: 28. Oktober 2015

Jan Lepel

Thema der Arbeit

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

Stichworte

Virtualisierung, Automatisierter Umgebungsaufbau, Provisionierung, Vagrant, Ansible, VirtualBox, Web-Interface, Requirement-Engineering, Sinatra

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Konzipierung und Erstellung eines Prototypen, der es Anwendern ermöglicht, schnell und effektiv virtuelle Maschinen über ein Web-Interface zu erstellen. Definierte Zustandsbeschreibungen können dabei berücksichtigt und zeitgleich ausgeführt werden.

Jan Lepel

Title of the paper

Automated Build and Provisioning of Ad Hoc Linux-Environments -
Creation of a Web-Interface Prototype for a Simplified Commissioning of Individually Created Development-Environments

Keywords

Virtualization, Automated build of Environment-Structures, Provisioning, Vagrant, Ansible, VirtualBox, Web-Interface, Requirement-Engineering, Sinatra

Abstract

This work deals with the development of a concept and the creation of a prototype, which allows users to create virtual machines quickly and effectively via a web interface. Additionally, defined state descriptions can be taken into account and executed simultaneously.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Problemstellung und Motivation	1
1.2	Themenabgrenzung	2
1.3	Zielsetzung	2
1.4	Struktur der Arbeit	2
2	Grundlagen	4
2.1	Grundlagen der Virtualisierung	4
2.1.1	Virtuelle Maschine	5
2.1.2	Gastbetriebssystem	5
2.1.3	Hypervisor	6
2.2	Konfigurationsmanagementsystem	8
3	Anforderungsanalyse	10
3.1	Zielsetzung	10
3.2	Stakeholder	11
3.3	Funktionale Anforderungen	11
3.4	Use-Cases	13
3.4.1	Business-Use-Case	14
3.4.2	System-Use-Case	15
3.5	Nichtfunktionale Anforderungen	18
3.6	Randbedingungen	26
3.6.1	Technische Randbedingungen	27
3.7	Vergleichbare Anwendungen	27
3.7.1	OpenNebula	28
3.7.2	OpenStack	28
3.7.3	Resümee	29
3.8	Zusammenfassung	29
4	Evaluation	31
4.1	Virtualisierungsprodukte	31
4.1.1	VMware Player (Plus)	31
4.1.2	Microsoft Hyper-V	32
4.1.3	Oracle VM VirtualBox	32
4.1.4	Zusammenfassung	33
4.1.5	Resümee	34

4.2	Konfigurationsmanagementsysteme	34
4.2.1	Ansible	35
4.2.2	Saltstack	35
4.2.3	Puppet	35
4.2.4	Zusammenfassung	36
4.2.5	Resümee	37
4.3	Vagrant	37
5	Softwareentwurf	38
5.1	Kontextabgrenzung	39
5.1.1	Kurzbeschreibung der externen Schnittstellen	40
5.2	Verteilungssicht	41
5.3	Systemarchitektur	42
5.3.1	Client-Server-Modell	42
5.3.2	Model-View-Controller Entwurfsmuster	44
5.4	Kommunikation	46
5.5	Server	48
5.5.1	Bausteinsicht	48
5.5.2	Laufzeitsicht	59
5.5.3	Datenbank	62
5.6	Client	66
5.7	Zusammenfassung	66
6	Realisierung	68
6.1	Server Realisierung	68
6.2	Fremdsoftware	69
6.2.1	Webserver	69
6.2.2	Phusion Passenger	69
6.2.3	VirtualBox	70
6.2.4	Ansible	70
6.2.5	Vagrant	70
6.2.6	Bundler	70
6.2.7	Datenbank	70
6.3	Implementierung	72
6.3.1	Komponenten Umsetzung	72
6.3.2	Datenbank	74
6.3.3	Funktionen	76
7	Schluss	81
7.1	Zusammenfassung und Fazit	81
7.2	Ausblick	82

8	Anhang	84
8.1	Aufbau einer virtuellen Maschine	84
8.2	Verwaltung der virtuellen Maschinen	85
8.3	Externer Zugriff	85
8.4	Import	86
8.5	Log	87
8.6	Administration	87
8.7	Software und Packages	88

1 Einleitung

1.1 Problemstellung und Motivation

Die Herausforderung klassischer IT-Umgebungen besteht oft in der dynamischen Anpassung an die Umstände eines Unternehmens. Wächst das Unternehmen oder werden neue Systeme integriert, wird die Umsetzung durch konventionelle Methoden meist aufwändig und langwierig. Die zu treffenden Maßnahmen erfordern meist die Anschaffung neuer Hardware, die exakte Festlegung der Hardwareeigenschaften und die Integration in die bestehende Infrastruktur. So entstehen unflexible Server-Konstrukte, die über Jahre meist nur einen starren Zweck erfüllen, ggf. durch nicht genutzte Auslastung Ressourcen vergeuden und nach Beendigung ihres Einsatzzweckes ausgemustert werden. Eine mögliche Lösung zur effizienteren Nutzung von Ressourcen wurde bislang durch die Verwendung von Virtualisierungslösungen erreicht. Durch die Entkopplung von der Hardware können unterschiedliche Strukturen an Maschinen parallel genutzt werden und somit mehrere Aufgaben, bei besserer Auslastung der Hardware, durchgeführt werden.

Auch wenn die Virtualisierung viele Abläufe beschleunigt hat und Umgebungen schneller bereit stehen, entsteht bei Administratoren ein hohes Zeitdefizit, durch die Vervielfältigung von Systemen, dem Anlegen von Systemzuständen, der Verwaltung von Speicherressourcen und der Bearbeitung von Anfragen nach temporären und permanenten Maschinen. Eine Entlastung dieser Tätigkeiten könnte der Schritt zu IaaS (Infrastructure-as-a-Service) bringen, um die zahlreichen Tätigkeiten zu reduzieren. Die angebotenen Automatismen helfen den Administratoren bei der Verwaltung der Basisinfrastruktur und unterstützen den Anwender mit selbstständigen Konfigurationen. Nachteilig sind allerdings die laufenden Betriebskosten, die an den IaaS-Anbieter zu entrichten sind. Auch das Herausgeben der ganzen oder teilweisen Infrastruktur (Cloud-Computing / Hybrid-Cloud) an einen Drittanbieter ist meist mit Sicherheitsbedenken verbunden. Eine alternative Herangehensweise wäre das Abgeben von Zuständigkeiten an eine interne Applikation oder den Anwender selbst. Dabei müsste die Basiskonfiguration zwar immer noch vom Administrator bereitgestellt werden, jedoch wird der Anwender so in die Lage versetzt sich in einem Teil der Infrastruktur seine eigene virtuelle Umgebung aufzubauen.

Die Einbindung der IT oder eines administrativen Weges wäre somit obsolet. Durch die daraus resultierende Autonomie des Anwenders, wäre es leichter Test- und Entwicklungsumgebungen im Aufbau zu automatisiert und den betroffenen Personenkreis flexibel agieren lassen. Möglich ist entsprechend eine Applikation, die dem Anwender Freiraum in Aufbau und Verwaltung seiner Maschinen gibt. Darüber hinaus kann diese ihm zusätzliche Funktionen hinsichtlich des Aufbaus und der Konfigurationen bieten.

1.2 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. Diese 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.

1.3 Zielsetzung

Diese Arbeit wird sich mit der Konzipierung und der Umsetzung einer Applikation auseinandersetzen, wobei der Fokus auf die Zentralisierung der Applikation, die autonome Handhabung und die Möglichkeit zeitnah individuelle Maschinen zu erstellen gelegt wird. Die Modellierung der Applikation wird, beginnend bei der Planung bis hin zur Umsetzung, iterativ und inkrementell gestaltet. Der Funktionsumfang wird nach Requirement-Engineering-Standard geplant und mit Hilfe von fiktiven Stakeholdern eingegrenzt. So wird ein lauffähiger Prototyp angestrebt, der dem geplanten Funktionsumfang entspricht.

1.4 Struktur der Arbeit

Kapitel 2 greift Grundlagen auf und erläutert Begriffe sowie Funktionsweisen der Virtualisierung. Zudem befasst sich das Kapitel mit dem Aufgabengebiet von Konfigurationsmanagement-Systemen und deren Verwendung.

Anschließend wird in Kapitel 3 eine Anforderungsanalyse erarbeitet, die den Funktionsumfang der Applikation beschreibt. Zum Ende des Kapitels hin werden die herausgestellten Funktionen mit bereits erhältlichen Applikationen verglichen und ein Fazit gezogen, bezogen auf die

zukünftige Verwendbarkeit der Applikationen.

In Kapitel 4 werden etablierte Softwarekomponenten evaluiert, die frei auf dem Markt erhältlich sind. Das Ergebnis der Evaluation soll Komponenten herausstellen, die den Funktionsumfang der angestrebten Applikation unterstützen und ggf. optimieren.

Im darauf folgenden Entwurf (Kapitel 5) wird der Grundstock für eine systematische Umsetzung gelegt, die in Kapitel 6 beschrieben wird. Kapitel 5 geht zudem auf gängige Sichten eines Softwareentwurfs ein, die das System betreffen. Jede Sicht beschreibt das System aus einer anderen Perspektive und integriert Systemarchitekturen, die am Anfang des Kapitels geplant werden. Die zusätzliche Aufteilung zwischen Client und Server sowie die Beschreibung der Kommunikation zwischen diesen beiden Akteuren, konkretisieren die Planung. Zusätzlich wird eine Zwei-Phasen-Konzipierung eines Datenbankschemas ausgearbeitet, wodurch eine detaillierte Ansicht der Tabellen-Konstrukte angestrebt wird.

Im Kapitel 7 werden die Ergebnisse dieser Arbeit nochmals zusammengefasst und mit einem persönlichen Fazit beendet. Zudem werden in einem Ausblick, zukünftig vorstellbare Erweiterungen des Funktionsumfangs und dadurch entstehende Einsatzzwecke der Applikation beschrieben.

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 diese 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 deren Funktionsweise betrachtet.

2.1 Grundlagen der Virtualisierung

Der Begriff Virtualisierung kann unterschiedlich definiert werden. Nach Christian Baun (2011) werden durch die Virtualisierung, die Ressourcen eines Rechnersystems aufgeteilt und von unabhängigen Betriebssystem-Instanzen verwendet. Durch Bündelung der Hardware-Ressourcen zu einer logischen Schicht, wird eine bessere Auslastung der Ressourcen ermöglicht. 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 darüber liegende Anwendungen direkt und bedarfsgerecht nötige Ressourcen beziehen.

Grundsätzlich wird laut Bengel (2008) 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 zum Beispiel Anwendungen und Betriebssystemen beschäftigt.

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 bei einer virtuellen Maschine, um ein Hardware-/Software-Duplikat eines real existierenden Computersystems, in der eine statistisch dominante Unter-
menge an virtuellen Prozessoren und Befehlen im Benutzer-Modus auf dem Host-Prozessor
ausführt 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 Kap-
selung erreicht, 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 dieses nicht automatisch die restlichen parallel laufenden Maschinen
und stellt damit einen der charakteristischen Vorteile von virtuellen Maschinen dar: die Ver-
fügbarkeit. Backup-Systeme oder mehrere Instanzen einer Applikation können so unabhängig
voneinander auf einem Host ausgeführt werden, ohne sich gegenseitig zu beeinflussen.

Virtuelle Maschinen können ohne größeren Aufwand verschoben, kopiert und auf unterschied-
liche Hostserver transferiert werden, um eine optimierte Hardware-Ressourcen-Auslastung zu
erhalten. Diese Eigenschaften können von Administratoren genutzt werden um z.B. effizienter
Backups erstellen zu können, Disaster-Recovery zu planen, Umgebungen für Tests/Deploy-
ments 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 Betriebssystem, welches sich auf einem standard Rechnersys-
tem befinden und einem Gastbetriebssystem besteht darin, dass üblicherweise Betriebssysteme
im privilegierten Prozessor-Modus ausgeführt werden, während dieses dem Gastbetriebssystem
verweigert wird. Der privilegierte Modus (auch Kernel-Mode genannt) befähigt das Betriebs-
system, die Kontrolle über die vorhandenen Ressourcen zu erlangen 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 von und unter genau kontrollierten Bedingungen

von einem Betriebssystem gestattet. So wird verhindert, dass laufende Anwendungen durch fehlerhafte Programmierung das System zum Absturz bringen. (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 dieses zur Folge, den privilegierten Prozessor-Modus nicht so nutzen zu können, wie es ein nicht virtualisiertes Betriebssystem könnte. Werden durch ein Gastbetriebssystems Instruktionen ausgeführt, die den privilegierten Prozessor-Modus erfordern, müssen diese Instruktionen anders gehandhabt werden, als es normalerweise der Fall wäre. Dieses zu ermöglichen ist unter anderem die Aufgabe des Hypervisors (siehe Abschnitt 2.1.3).

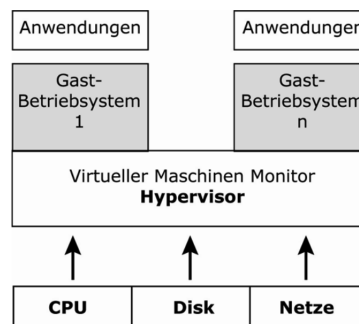


Abbildung 2.1: Betriebssystemvirtualisierung¹

2.1.3 Hypervisor

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. Wie in Abbildung 2.1 dargestellt. Dessen primäre Aufgabe ist die Verwaltung der Host-Ressourcen und deren Zuteilung bei Anfragen der Gast-systeme. 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 Gastssysteme ausgeführt werden. Zu seinen Aufgaben zählen unter anderem E/A-Zugriffe (insbesondere Hardwarezugriffe), Speichermanagement, Prozesswechsel und System-Aufrufe.

¹Bildquelle: Siegert und Baumgarten (2006)

²Bildquelle: Fleischmann (2012)

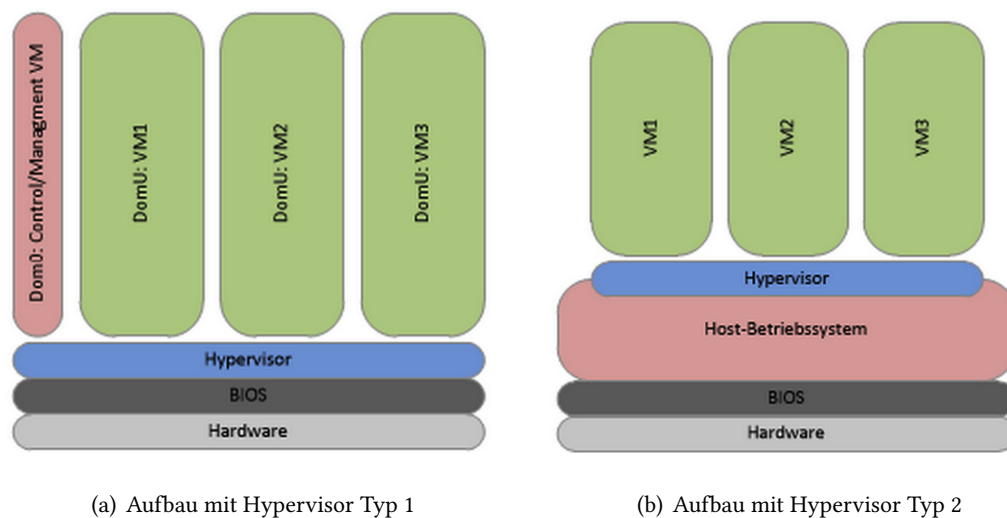


Abbildung 2.2: Hypervisor Typ 1 und 2²

Der Hypervisor kann in zwei verschiedene Typen kategorisiert werden:

Typ 1 Hypervisor

Der 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. Abbildung 2.1.3.a zeigt einen entsprechenden Aufbau mit einem Hypervisor Typ 1. [Reuther (2013)]

Typ 2 Hypervisor

Der Typ 2 Hypervisor, der auch als 'Hosted' bezeichnet wird, verdeutlicht durch dessen Bezeichnung bereits den Unterschied zu dem Typ 1 Hypervisor. Im Gegensatz zu dem Typ 1 Hypervisor, setzte der Typ 2 auf einem Hostsystem auf. Es muss also eine Schicht implementiert sein, die zwischen dem Hypervisor und der Hardware liegt. Siehe Abbil-

dung [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. Dieses vereinfacht die Installation gegenüber dem Hypervisor Typ 1.

Aus Implementierungssicht gibt es für beide Hypervisoren Vor- und Nachteile. Die Anforderung an das Vorhandensein eines Betriebssystems hat vor allem Vorteile bezogen auf Hardware- und Treiber-Kompatibilität, Konfigurationsflexibilität und der Verwendung von vertrauten Management-Tools.

Allerdings entsteht durch das vorhandene Betriebssystem nicht nur ein höherer Management-Aufwand, da dieses konfiguriert und zu verwaltet werden muss, sondern die Performance und der Sicherheitsaspekt wird unter dieser zusätzlichen Schicht eingeschränkt. [Reuther (2013)]

2.2 Konfigurationsmanagementsystem

Die Hauptaufgabe eines Konfigurationsmanagement-Systems ist es, eine zuvor definierte Zustandsbeschreibung eines Zielrechners umzusetzen. Dieses kann beispielsweise die Installation von Softwarepaketen beinhalten, das Starten oder Beenden von Diensten, oder das Erstellen/Anpassen/Entfernen von Konfigurationen. Im Allgemeinen wird dieser Vorbereitungsprozess auch 'Provisionieren' (engl. provisioning) genannt und stattet den Zielrechner mit allen Voraussetzungen für seine Aufgaben aus. In der Regel verwenden Konfigurationsmanagementsysteme eigene Agenten auf den Zielrechnern, über die die Kommunikation abgewickelt und die Zustandsbeschreibung realisiert wird. Neuere Anwendungen wie 'Ansible' (Kapitel 4.2.1) kommen ohne Agenten aus. Die Kommunikation wird im Falle von Ansible über eine SSH-Schnittstelle realisiert. Pull-basierte Tools, wie beispielsweise 'Puppet' (Kapitel 4.2.3), fragen in regelmäßigen Abständen ein zentrales Konfigurations-Repository ab, in dem die jeweils aktuelle Zustandsbeschreibung der Maschine gespeichert ist. Diese sorgen dafür, dass die Änderungen auf dem Ziel-Client ausgeführt werden. Konfigurationsmanagementsysteme können auf virtuelle und Standard Hardware-Maschinen gleichermaßen provisionieren. Zudem sind diese in der Regel dazu fähig, ganze Gruppen von Rechnern parallel zu bearbeiten und die entsprechenden Zustandsbeschreibungen umzusetzen. In dem bereits genannten Beispiel 'Ansible', können mehrere Rechner als Gruppen in Inventory-Dateien (siehe Listing 7) zusammengefasst werden. Jede Gruppe erhält einzeln die entsprechend vorgesehenen Zustandsbeschreibungen.

```
1 [atomic]
2   192.168.100.100
3   192.168.100.101
4 [webserver]
5   192.168.1.110
6   192.168.1.111
```

Listing 2.1: Beispiel Inventory-Datei³

Die Integration und Anwendung von Konfigurationsmanagement-Systemen beschleunigt nicht nur den Aufbau eines oder mehrerer Zielrechner, sondern hilft auch bei der Organisation der Softwareverteilung. [Scherf (2015)]

³Codebeispiel: Scherf (2015)

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 sind User-Interface und der Funktionsumfang primäre Ziele. 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. Diese 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

Tabelle 3.1: Stakeholder: Anwender¹

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

Tabelle 3.2: Stakeholder: Administrator²

3.3 Funktionale Anforderungen

Durch funktionalen Anforderungen werden Kernaufgaben des Systems herausgestellt und beschrieben. Dies beinhaltet dessen Dienste oder Funktionalitäten, die das System bereitstellen soll. Die Unterpunkte 'FA' listen die einzelnen funktionalen Anforderungen auf und werden in den Use-Cases (Abschnitt 3.4) erneut aufgegriffen.

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.

¹Quelle: Eigene Darstellung

²Quelle: Eigene Darstellung

- 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 darauf reagieren können.
- FA 5. Möchte der Anwender zusätzlich Dateien auf die zu erstellende virtuelle Maschine übertragen, ermöglicht dieses die Applikation.
- 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 dieses 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 dieses 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 der 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 in eine Übersicht seiner Hauptfunktionen gegliedert wird. Der Business-Use-Case spiegelt dabei ein Gesamtbild aller Funktionalitäten der Applikation wieder und grenzt diese voneinander ab. Die darauf folgenden System-Use-Cases helfen dabei Hauptfunktionalitäten zu skizzieren, die sich aus den funktionalen Anforderungen (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

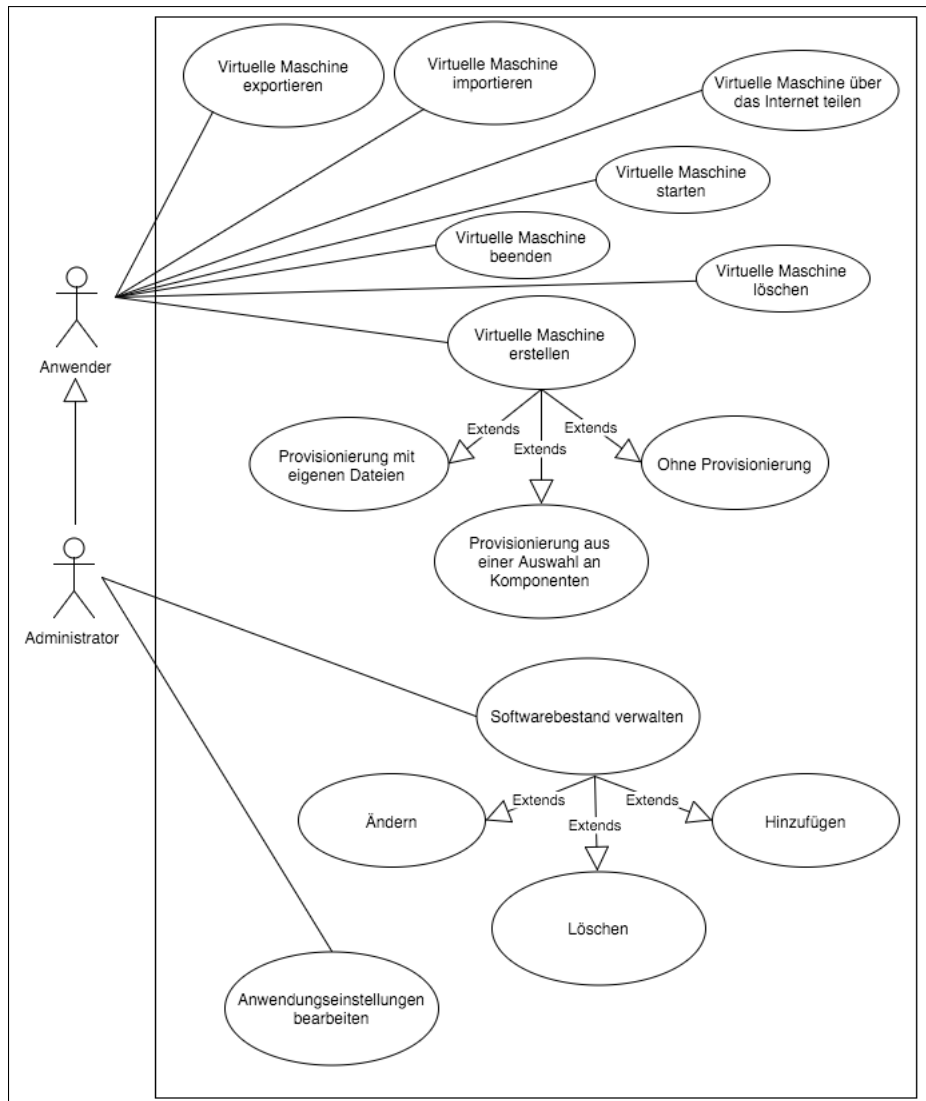


Abbildung 3.1: Business-Use-Case Übersicht³

³Bildquelle: Eigene Darstellung

3.4.2 System-Use-Case

Use-Case 1 - Virtuelle Maschine erstellen

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

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

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

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

Use-Case 3 - Virtuelle Maschine importieren

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

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

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

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

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

1. Der Administrator startet den VM-Builder über den Browser.
2. Der Administrator navigiert in das Administrationsmenü.
3. Der Administrator erstellt ein neues Softwarepaket.
4. Der Administrator gibt die entsprechenden Optionen wie z.B. Name des Softwarepakets ein.
5. Der Administrator wählt die Komponenten aus, die mitinstalliert 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 formulierte, können durch nichtfunktionale Anforderungen neue Lösungsmöglichkeiten vorgegeben werden oder eine Menge an potentiellen Designentwürfen, in 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. Dieses kann durch Anwendung von zwei Standards erfolgen: ISO 25000 und ISO 9126. Allerdings ist der ISO 9126 Standard in den ISO 25000 übernommen worden. Beide Standards sollen sicherstellen, dass die Qualität des Systems und des Entwicklungsprozesses über Qualitätsanforderungen festgelegt wird.

3. Anforderungen an die Benutzeroberfläche

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

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 Systems 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.

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 dieses 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 dieses möglichst unkompliziert erfolgen.
8. Der Zeitaufwand für den Import von virtuellen Maschinen sollte <10 Minuten betragen.
9. Die Zeit des Exports einer virtuellen Maschine sollte <5 Minuten betragen.

Anforderungen an die Benutzeroberfläche

"Mit Anforderungen an die Benutzungsoberfläche [...] wird eine Menge von Faktoren beschrieben, die sich mit dem visuellen, akustischen oder auch haptischen Erscheinen und der Bedienung des Systems befassen. Sie ergänzen die funktionalen Anforderungen, die sich an der Oberfläche zeigen und spezifizieren, wie diese funktionalen Anforderungen zu erscheinen haben." **Roland Kluge (2013)**

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 3.2). Primär steht eine Übersicht der erstellten virtuellen Maschinen im Vordergrund, inklusive derer verfügbaren Optionen und Statusinformationen.

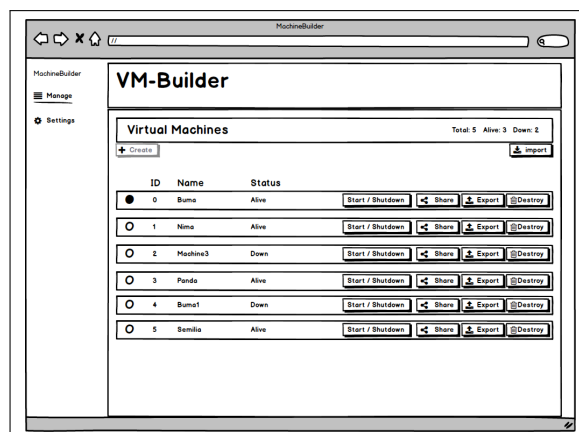


Abbildung 3.2: Entwurf der Verwaltungsoberfläche⁴

⁴Bildquelle: Eigene Darstellung

3. Der VM-Builder sollte den Oberflächendialog 'Virtuelle Maschine erstellen' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung 3.3 und 3.4). Keine umzusetzenden Anforderungen sind die genaue Größe und die Anordnung der einzelnen Elemente. Abbildung 3.3 zeigt die Menüführung des geleiteten Aufbaus einer virtuellen Maschine mit Eingabemöglichkeiten. Abbildung 3.4 zeigt den nachfolgenden Auswahlprozess an Softwarekomponenten. Zu beachten ist, dass die Eingabe der optionalen- und Pflichtparameter gegeben ist sowie die Auswahlmöglichkeit an Softwarekomponenten.

MachineBuilder

Manage
Settings

Create

Step 1 - Main Configuration

Hostname
localhost

IP Address (optional)
192.168.2.3

Step 2 - Choose a Machine

☒ Standard 32Bit Machine
☐ Standard 64Bit Machine
☒ Another one
ubuntu/trusty64
[Other 32Bit Machines](#)
[Other 64Bit Machines](#)

Abbildung 3.3: Entwurf der Aufbauoberfläche - Eingabe Parameter⁵

MachineBuilder

Manage
Settings

Create

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

☒ Apache
☒ MySQL
☒ Unzip

1. Search File 2. Unzip? 3. Target at Host

☒ Unzip

Filename Unzip Target

☒ Erp1ar Yes

Abbildung 3.4: Entwurf der Aufbauoberfläche - Auswahl der Softwarekomponenten⁶

⁵Bildquelle: Eigene Darstellung

⁶Bildquelle: Eigene Darstellung

4. Der VM-Builder sollte den Oberflächendialog 'Sharing' mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung 3.5). 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 zur Weitergabe an Dritte benötigt werden. Dieses wären z.B. Konnektivitätsmöglichkeiten und Informationen zur geteilten virtuellen Maschine.

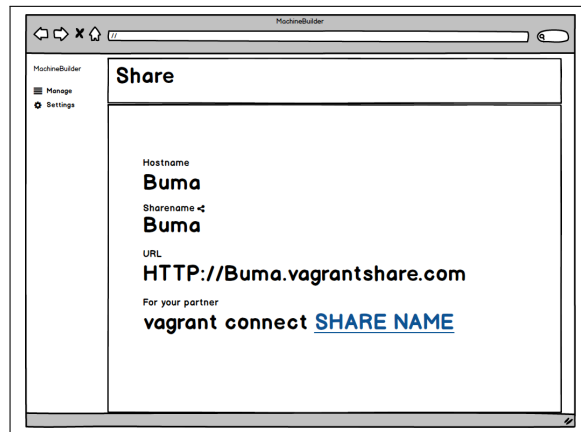


Abbildung 3.5: Entwurf der Sharing-Oberfläche⁷

⁷Bildquelle: Eigene Darstellung

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. In Abbildung 3.6 wird ein exemplarischer Aufbau der Webansicht gezeigt.

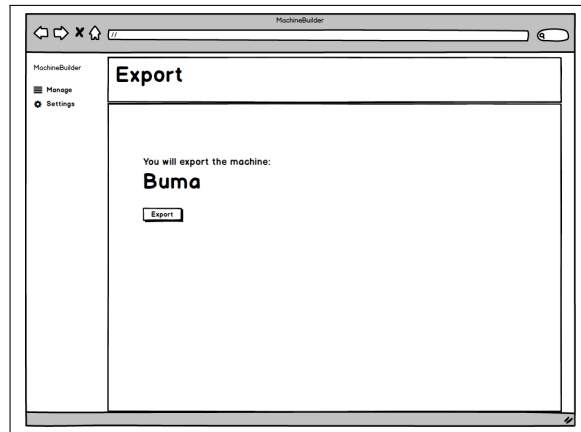


Abbildung 3.6: Entwurf der Export-Oberfläche⁸

6. Der VM-Builder sollte den Oberflächendialog 'Import' besitzen, mit dem der Anwender zuvor exportierte Dateien wieder importieren kann. So sollen Schaltflächen zum Auswählen der benötigten Dateien angeboten werden. Eine Entwurf der Oberfläche wird unter Abbildung 3.7 gezeigt.

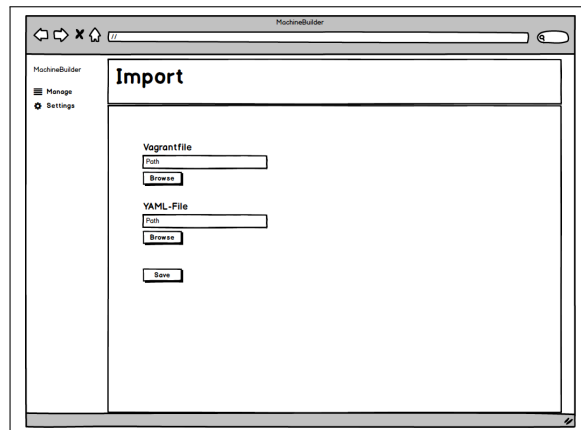


Abbildung 3.7: Entwurf der Import-Oberfläche⁹

⁸Bildquelle: Eigene Darstellung

⁹Bildquelle: Eigene Darstellung

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. Die folgenden Entwürfe (Abbildungen: 3.8, 3.9, 3.10) zeigen Vorschläge der einzelnen Seite des Menüs

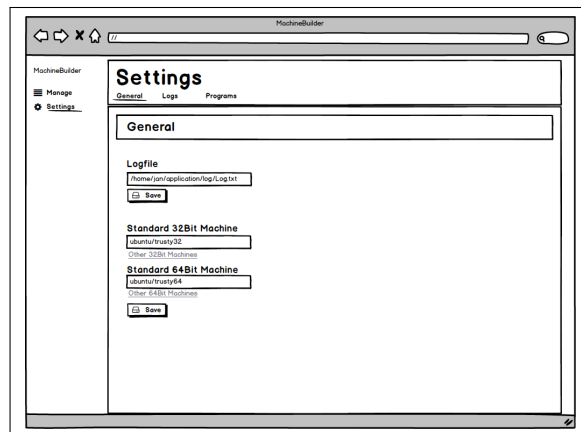


Abbildung 3.8: Entwurf der Einstellungen - Generelle Konfiguration¹⁰

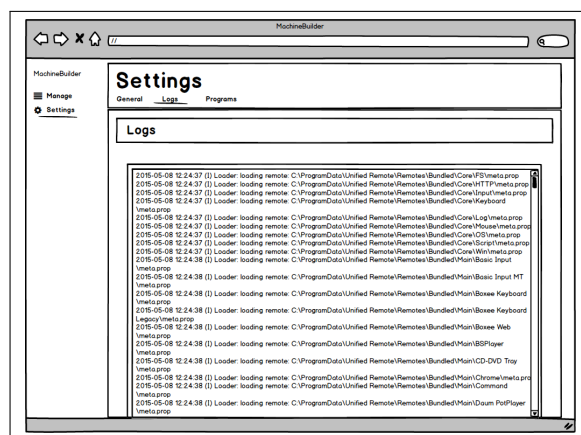


Abbildung 3.9: Entwurf der Einstellungen - Logdatei¹¹

¹⁰Bildquelle: Eigene Darstellung

¹¹Bildquelle: Eigene Darstellung

¹¹Bildquelle: Eigene Darstellung

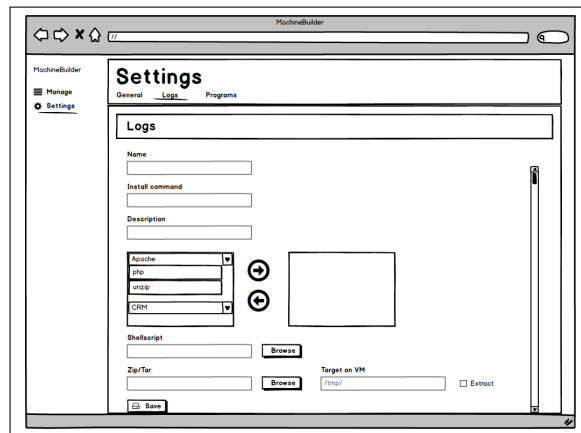


Abbildung 3.10: Entwurf der Einstellungen - Softwarekomponenten hinzufügen

3.6 Randbedingungen

Um anwendungs- und problembezogene Entwurfsentscheidungen treffen zu können, werden Faktoren analysiert, die die Architekturen der Anwendung beeinflussen oder einschränken können. Dieses 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 \(2000\)](#)] charakterisierten diese Faktoren als 'facts of life'.
- Technische Faktoren: Durch diese wird das technische Umfeld geprägt und entsprechend eingeschränkt. Diese betreffen nicht nur die Ablaufumgebung des zu entwickelnden Systems, sondern umfassen auch die technischen Vorgaben für die Entwicklung, einzusetzender Software und vorhandener IT-Systeme. [[Starke \(2014\)](#)]

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

3.6.1 Technische Randbedingungen

Randbedingungen	Erläuterung
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.

Tabelle 3.3: Technische Randbedingungen¹²

3.7 Vergleichbare Anwendungen

Wenn es um den Aufbau und die Verwaltung von virtuellen Maschinen geht, steht eine Vielzahl an unterschiedlichster Software zum Angebot. Das Spektrum reicht von der Administration über Cloud-Dienste, dem Einsatz von Open-Source-Software, um ganze Rechenzentren zu verwalten oder dem Aufbau von virtuellen Maschinen im privaten Umfeld. Um einen kleinen Überblick über populäre Open-Source-Anwendungen zu erhalten, werden im nachfolgenden Teil zwei Anwendungen vorgestellt, die ähnliche Aufgaben wie der VM-Builder übernehmen könnten. Im Anschluss wird in einem Zwischenfazit (Abschnitt 3.7.3) kurz erläutert, wieso die beschriebene Software nicht für den Einsatz geeignet ist.

¹²Quelle: Eigene Darstellung

3.7.1 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 eine Art IaaS kann OpenNebula die zentrale Administration und Überwachung von virtuellen Systemen übernehmen. Kommt es zu Ausfällen von Komponenten kann OpenNebula automatisch einen Neustart auf einem anderen Hostsystem ausführen. Zudem sind dynamische Erweiterungen der Hardwareressourcen möglich. Die Kernkompetenzen liegen dabei auf schon vorhandenen Systemen im Bereich Virtualisierung, Speicherung und Vernetzung. OpenNebula kann optimal als Werkzeug für die Integration vorhandener Systeme verwendet werden, da es diverse Möglichkeiten zur Wartung und Skalierung der gegebenen Infrastruktur zur Verfügung stellt. Auch wenn OpenNebula sich selbst in das Enterprise Segment kategorisiert¹³, ist es jedoch auch für kleinere Organisationen praktikabel. Durch die Installation von wenigen Komponenten und die Steuerung über nur ein Frontend, macht die Steuerung und Konfiguration komfortabel.

3.7.2 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 diverse IT-Unternehmen OpenStack weiterentwickelten, ist die Lösung ein fester Bestandteil des Portfolios diverser bekannter 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 den Erwartungen der meisten Anwender. Ansonsten bietet OpenStack Administratoren ein großes Spektrum an Werkzeugen, um ihre Cloud-Computing Umgebung zu verwalten. Durch die vielfältigen Einsatzmöglichkeiten und die große Auswahl an Optionen, ist OpenStack ein guter 'Allrounder', mit Defiziten in der Einarbeitungszeit. [Loschwitz und Syseleven (2015)]

¹³Quelle: Llorente (2014)

3.7.3 Resümee

Eines der wichtigen Merkmale von OpenNebula ist der Schwerpunkt auf Rechenzentrums-virtualisierung inklusive bestehender Infrastruktur wodurch dieses mehr oder weniger im Enterprise-Cloud-Computing angesiedelt wird, OpenStack dagegen ein Vertreter des Public-Cloud-Computing ist. Einer der Vorteile von OpenNebula ist, dass sich eigene VM-Images per Knopfdruck auf Virtualisierungsservern und dem dazugehörigen Storage instanziierten lassen und Zuweisungen wie IP, Load-Balancing und Speicherplatz von der eigenen Verwaltungssoftware übernommen werden. 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 umfangreich in ihren Funktionalitäten und bringen eine zu hohe Einarbeitungszeit mit sich.

3.8 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 können. Die insgesamt geschaffenen Anforderungen werden im folgenden Kapitel 4. ('Evaluation') als Grundlage für die Recherche nach Produkten verwendet, die für den VM-Builder nutzbar sind. Die Ergebnisse aus der Anforderungsanalyse und der Evaluation schaffen zudem eine Grundlage für den Entwurf in Kapitel (Kapite 5).

4 Evaluation

Um herauszufinden, ob und welche Softwareprodukte der Anwendung hinzugefügt werden können, kann eine Evaluation des Marktes hilfreich sein. Dabei ist zu eruieren, ob gewünschte Funktionalitäten unter Verwendung der technischen Randbedingungen aus Kapitel 3.6.1 von externen, etablierten sowie kostenlosen Softwarekomponenten unterstützt oder sogar durchgeführt werden können.

4.1 Virtualisierungsprodukte

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

1. **VMware Player / Plus**

Der VMware Player ist VMwares preiswerte Virtualisierungslösung für Unternehmen und Privatkunden.

2. **Microsoft Hyper-V**

Microsofts Virtualisierungslösung ist Hyper-V, die ab Windows 8 oder Windows Server 2012 kostenlos zur Verfügung steht.

3. **Oracle VM VirtualBox**

VirtualBox ist ein gratis Open-Source-Tool von Oracle, das großen Funktionsumfang bietet und kostenlos angeboten wird.

4.1.1 VMware Player (Plus)

Eines der bekanntesten Unternehmen für Virtualisierungslösungen ist VMware.¹ VMware bietet für den Privatanwender den VMware Player an, der das kostenlose Pendant zur professionellen

¹Quelle: ama Direktbefragung 02/2014

Lösung VMware Workstation / VMware Fusion darstellt. Mittlerweile ist der VMware Player für Unternehmen unter dem Namen VMware Player Plus, zum Preis von 134,95 Euro zu erwerben.² Der VMware Player (Plus) unterstützt ca. 200 Gastbetriebssysteme, mit 32Bit oder 64Bit, und lässt sich auf Windows und verschiedenen Linux-Distributionen installieren. Nachteilig ist die Beschränkung auf das VMDK-Format (Virtual Machine Disk), das bei der Speicherung von virtuellen Maschinen zur Verfügung steht. Zudem fehlen Features, wie eine virtuelle Maschine in einen vorherigen Systemzustand zurückzusetzen, oder eine Maschine zu duplizieren. Allerdings werden Export- und Import Funktionen, sowie das Erstellen von momentanen Zuständen vom VMware Player unterstützt.

4.1.2 Microsoft Hyper-V

Hyper-V wurde von Microsoft in Windows 8 (oder höher) und Windows Server 2012 integriert. Die Software lässt sich über die Windows-Funktion einfach nachinstalliert. Damit dieses möglich ist, müssen zwei Faktoren erfüllt sein. Windows 8 muss als Professional-Version in der 64-Bit Version vorliegen. Der Vorteil gegenüber den Mitbewerbern besteht in der Möglichkeit mehrere virtuelle Maschinen parallel betreiben zu können ohne signifikanten Performanceverlust. Dieses erreicht Hyper-V durch SLAT einer Technik zur dynamischen RAM Verwaltung.

4.1.3 Oracle VM VirtualBox

Die im April 2005 gegründete 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 32Bit und 64Bit-Betriebssystemen unterstützt, 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. Vier gängige Formate werden von VirtualBox angeboten und ermöglichen einen leichteren Austausch von erstellten virtuellen Maschinen unter Anwendern. Ein weiterer Vorteil von VirtualBox besteht darin, dass VirtualBox kostenlos und OpenSource ist. Der Quellcode steht jedem Interessierten zur Verfügung. Im Funktionsumfang der Software enthalten sind das Importieren und Exportieren von virtuellen Maschinen, das Erstellen von Snapshots und das Klonen von virtuellen Maschinen.

²Quelle: <http://store.vmware.com>

4.1.4 Zusammenfassung

	VMware Player (Plus)	Microsoft Hyper-V	Oracle VM VirtualBox
Host-Betriebssystem	Windows, diverse Linux Distributionen	Windows 8 Pro 64 Bit (oder höher); Microsoft Server 2012	Windows, Linux, Mac OS X, FreeBSD und Solaris
Gast-betriebssysteme	Mehr als 200 Gast-Betriebssysteme	Ab Windows XP mit SP 3 (oder höher), Diverse Linux Distributionen, FreeBSD	Diverse Linux Distributionen, Windows, FreeBSD, MacOS X, IBM OS/2
64-Bit-Gast-Betriebssystem	Ja	Ja	Ja
Dateiformate für virtuelle Disks	VMDK	VHDX	VMDK, VHD, Parallels Hard Disk, OVF
Snapshots	Nein	Nein	Ja
Klonen	Nein	Nein	Ja
Export von virtuellen Maschinen	Ja	Ja	Ja
Preis	134,96 Euro für Unternehmen; Kostenlos für Privatanwender	Kostenlos	Kostenlos

Tabelle 4.1: Zusammenfassung evaluierter Softwareprodukte³

³Quelle: Eigene Darstellung

4.1.5 Resümee

Auch wenn VMware zu den bekannteren Herstellern gehört und der VMware Player (Plus) eine Vielzahl an Funktionen bietet, macht die Unterscheidung zwischen einer Unternehmensvariante und einer für Privatanwender die zukünftige Benutzung aufwändiger. Sollte die zu erstellende Anwendung in einem kommerziellen Umfeld betrieben werden, so muss auf den VMware-Player Plus zugegriffen werden. Dadurch sind ggf. Änderungen an der Implementierung vorzunehmen sind und die entsprechenden Lizenzkosten zu entrichten. 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 ist ein negativer Punkt.

Microsofts Lösung stellt sich, im Rahmen dieser Arbeit, von den Grundanforderungen her als nicht nutzbar heraus. Da Hyper-V Windows als Host-System benötigt, widerspricht dieses den gestellten technischen Randbedingungen aus Kapitel 3.6 - Server Betriebssystem). Oracles VirtualBox hingegen vereint viele für die geforderten Funktionen unterstützende und hilfreiche Aspekte. Die Vielzahl an unterstützenden Dateiformaten ist für die Umsetzung der geforderte Exportfunktion ein positiver Faktor. Da VirtualBox für Privatanwender und Firmenkunden kostenlos ist, hat der Lizenzierungsaspekt keine Relevanz. VirtualBox lässt sich auf diversen Linux-Umgebungen installieren und erfüllt einen weiteren Punkt aus den technischen Randbedingungen (siehe Kapitel 3.6.1). VirtualBox ist durch seinen Funktionsumfang in der Lage, für wichtige Funktionen in der angestrebten Applikation eingesetzt zu werden. Dieses hat den Vorteil, dass die Funktionsbestandteile nicht kostenaufwändig und langwierig implementiert werden müssen.

4.2 Konfigurationsmanagementsysteme

Wie in Kapitel 2.2 beschrieben, helfen Konfigurationsmanagementsysteme bei der Umsetzung von sogenannten Zustandsbeschreibungen eines Hostes. Konfigurationsmanagementsysteme können erforderliche Software installieren, Dienste und Applikationen starten/beenden, Konfigurationen ändern/erstellen/löschen und dieses gleichzeitig auf einem oder mehreren Zielrechnern. Durch die Anwendung von Konfigurationsmanagementsystemen 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.

4.2.1 Ansible

Die Hauptdesignidee bei dem in Python geschriebenen Programm Ansible ist es, Konfigurationen mit wenig Aufwand durchführen zu können. Dabei werden weder aufwändige Deployment-Scripts benötigt noch komplizierte Syntax 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 einfache YAML Struktur und können somit schon vorgefertigt als Template gespeichert und wiederverwendet werden. [ScriptRock (2014)]

4.2.2 Saltstack

Saltstack oder kurz 'Salt' ist wie Ansible in Python entwickelt worden. Zur Kommunikation mit den Clients wird ein Mastersystem benötigt und Agenten, als Minions bezeichnet, die auf den Zielclients installiert werden müssen. Die eigentliche Kommunikation wird über eine ZeroMQ messaging lib in der Transportschicht aufgebaut, wodurch die Verständigung zwischen Master und Minions vereinfacht wird. Dadurch verschnellert sich zwar die Kommunikation, jedoch ist diese dadurch unsicherer, als die SSH Kommunikation von Ansible. ZeroMQ bietet nativ keine Verschlüsselung an, kann aber Nachrichten über IPC, TCP, TIPC oder Multicast transportieren. Salts Skalierbarkeit ist für Administratoren ein relevanter Faktor. Diese macht es möglich mehrere Ebenen an Mastern zu erstellen, um eine bessere Lastverteilung zu erhalten. Für die eigenen Konfigurationsdateien benutzt Salt das YAML-Format. Allerdings müssen Befehle, die in der Konsole ausgeführt werden, in Python oder PyDSL verfasst werden. [ScriptRock (2014)]

4.2.3 Puppet

Der größte Vertreter auf dem Markt im Bereich Konfigurationsmanagement-Systeme ist 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 den Nachteil mit sich bringt, dass neben dem Lernen der

Puppet Befehlen auch Ruby Kenntnisse erforderlich sind. 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 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)]

4.2.4 Zusammenfassung

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

Tabelle 4.2: Zusammenfassung evaluierter Konfigurationsmanagementsysteme⁴

⁴Quelle: Eigene Darstellung

4.2.5 Resümee

Der Unterschied in den verglichenen Konfigurationsmanagementsystemen 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. Wie Puppet benötigt Staltstack für die Clients extra Software, um die gewünschte Zustandsbeschreibung durchzuführen, was einen höheren Zeitaufwand und eine Fehlerquelle mehr bedeutet. Ansible ist das neuste entwickelte Softwareprodukt unter den Konfigurationsmanagement-Systemen, kann aber einfacher eingerichtet werden und benötigt keine weiteren Ergänzungen auf der Clientseite. Dadurch wird auf den Zielmaschinen ausschließlich die erforderliche Software installiert, der gesamte Installationsprozess beschleunigt und der gesamte Zeitaufwand der Konfiguration minimiert.

4.3 Vagrant

Ein weiteres Produkt, das die zu erstellende Applikation unterstützen wird, ist Vagrant. Dabei handelt es sich um ein Softwareprojekt, welches 2010 von Mitchell Hashimoto und John Bender 2010 entwickelt 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 virtuellen Maschine geschieht über das 'Vagrantfile', in dem Parameter wie IP-Adresse konfiguriert und Konfigurationsmanagementsysteme hinzugeschaltet werden können. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, kann dieses über eine Versionskontrollen (z.B. Git oder Subversion) versioniert und an andere Anwender verteilt werden.

Um eine Virtualisierung vorzunehmen, verwendet Vagrant standardmäßig VirtualBox. Alternativ werden Amazon-Web-Services und VMware-Fusion ebenfalls unterstützt und sind einfach zu integrieren. Zudem werden von Vagrant Funktionen zur Teamarbeit angeboten, die einen entfernten Zugriff auf eine virtuelle Maschine lassen. Diese Möglichkeit des Zugriffs auf eine virtuelle Maschine, ermöglicht es Teams an unterschiedlichen Standorten, auf die gleiche Maschine zuzugreifen. [(Peacock (2013))]

Die zusätzlichen Funktionen von Vagrant und das Vereinen von Konfigurationsmanagement und Virtualisierer in einem Produkt, unterstützen den Entwicklungsprozess der Applikation. Funktionen, wie der entfernte Zugriff auf eine Maschine, können 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 Herausforderung bei der Erstellung eines Softwareentwurfs ist eine Softwarearchitektur zu entwerfen, die die zuvor erarbeiteten funktionalen- (Kapitel 3.3) und nichtfunktionalen Anforderungen (Kapitel 3.5) berücksichtigt, einschließlich der Berücksichtigung von Einflussfaktoren, wie definierte Randbedingungen (Kapitel 3.6). Der Softwareentwurf ist eine Richtlinie, die 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 verwendet werden, die Kontextabgrenzung, Bausteinsicht, Laufzeitsicht und die Verteilungssicht. Diese Sichten werden im folgenden Softwareentwurf verwendet und näher beschrieben.

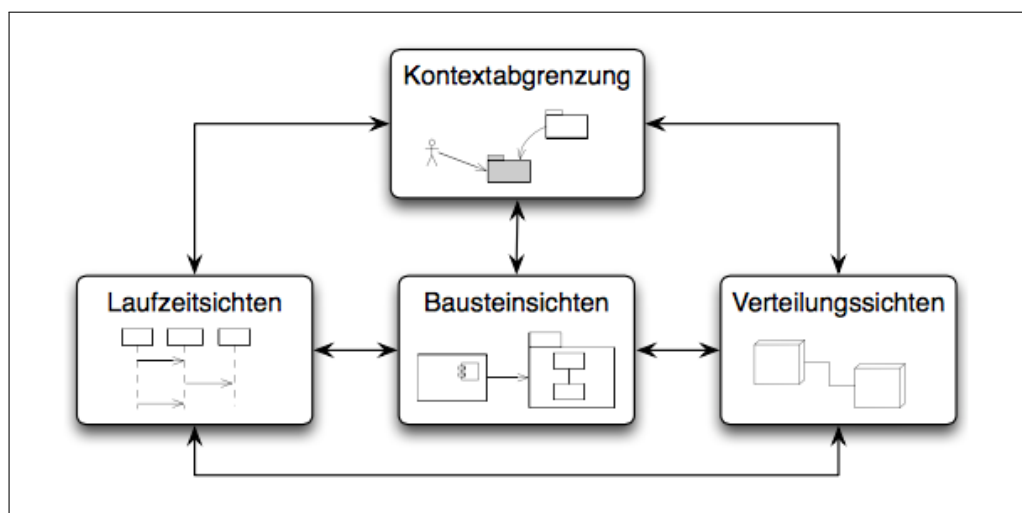


Abbildung 5.1: Vier Arten von Sichten¹

¹Bildquelle: Starke (2014)

5.1 Kontextabgrenzung

Die Kontextabgrenzung beschreibt die Einbettung des Systems in dessen 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 dieses weiter zu bündeln, können Teile der Hauptfunktionalitäten einzelner Produkte, die in Kapitel 4 betrachtet wurden, übernommen werden. VirtualBox kann das Erstellen, Exportieren und den Import von virtuellen Maschinen übernehmen. Das Konfigurationsmanagement-System Ansible ist darauf ausgelegt, mit bekannten Virtualisierungslösungen zusammenarbeiten zu können und übernimmt somit die Anforderung nach automatisierter Softwareinstallation. Um die beiden Anwendungen zu kombinieren, kann Vagrant als Wrapper eingesetzt werden, der zusätzliche Funktionen, wie das Teilen (Sharen) einer Maschine, mitbringt. Logik und Oberfläche werden durch den VM-BUILDER bereitgestellt, der in den folgenden Abschnitten näher konzipiert wird.

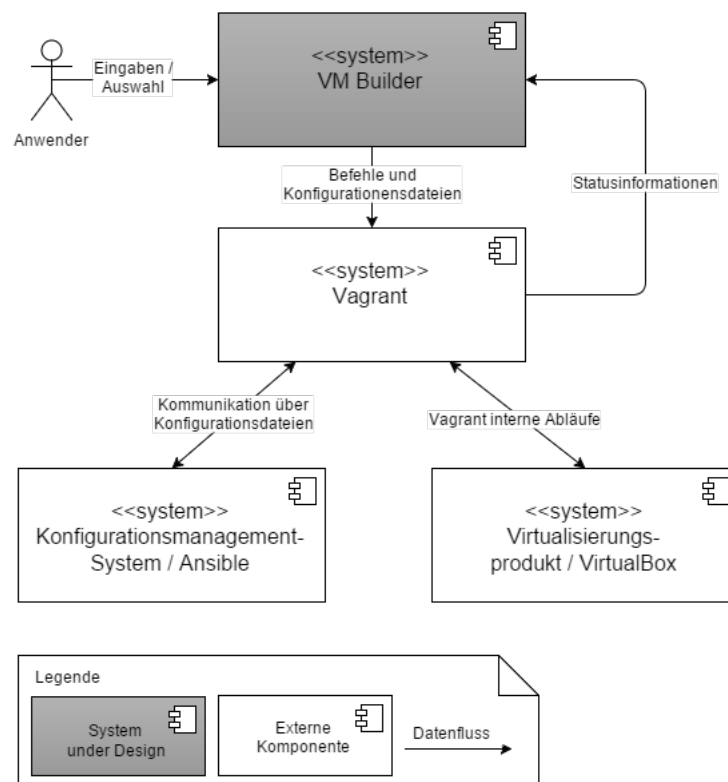


Abbildung 5.2: Kontextsicht²

²Bildquelle: Eigene Darstellung

5.1.1 Kurzbeschreibung der externen Schnittstellen

Eingaben / Auswahl	Der Anwender tätigt Eingaben und wählt unter bereitgestellten Optionen aus. Diese werden direkt von der Applikation verarbeitet.
Befehle und Konfigurationen	Die Applikation erstellt nötige Konfigurationsdateien für Vagrant und Ansible und leitet Befehle für die Weiterverarbeitung an Vagrant weiter.
Kommunikation über Konfigurationsdateien	Vagrant ruft über die erstellten Konfigurationsdateien den Konfigurationsmanager Ansible auf, um die virtuelle Maschine in den beschriebenen Zustand zu überführen.
Vagrant interne Abläufe	VirtualBox erstellt diese 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, inklusive der oben genannten Produkte, wird auf einem Server betrieben, der zentralisiert positioniert ist und über HTTP, FTP und SSH erreichbar ist. Dem Anwender wird von der Applikation eine Weboberfläche zur Verfügung gestellt, die es ermöglicht Eingaben zu tätigen und Optionen auszuwählen, um eine virtuelle Maschine zu erstellen oder zu verwalten. Damit die Applikation auf dem Server betrieben werden kann, 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 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 ausgewählte Image und leitet den dazugehörigen Download des Betriebssystems ein. Ist das Image auf dem Server schon vorhanden, wird dieses verwendet. Durch die Hilfe von VirtualBox und ggf. Ansible wird die zu erwartende virtuelle Maschine komplettiert.

5.2 Verteilungssicht

Um die Beschreibung aus 5.1.1 der Kontextabgrenzung visuell zu unterstützen, wird die Verteilungssicht herangezogen. Starke (2011) beschreibt die Verteilungssicht dabei wie folgt:

"Die Verteilungssicht zeigt die Verteilung von Systembestandteilen auf Hard- und Softwareumgebungen. Diese Sicht 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."

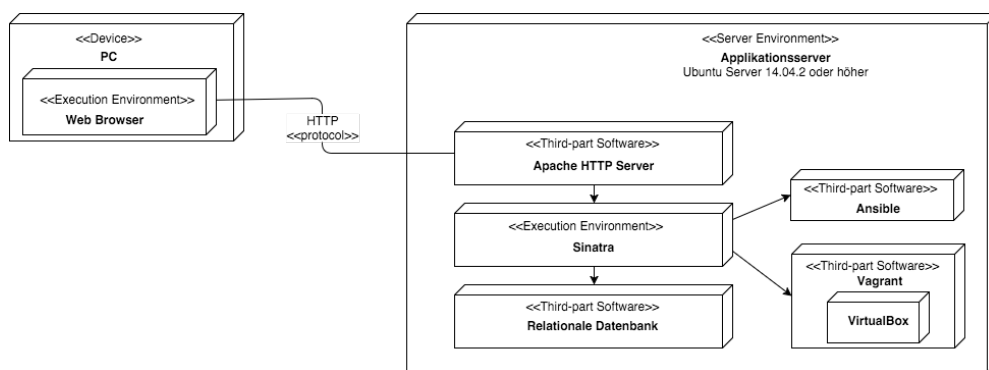


Abbildung 5.3: Verteilungssicht des VM-Builders³

Die genaue Platzierung der Softwarekomponenten aus Abschnitt 5.1.1 verhilft zu einer besseren Umsetzung der kompletten Softwarestruktur und verdeutlicht die Beziehung zwischen den einzelnen Komponenten. Unter den Abschnitten 3.5 und 3.6.1 wurden technologische Anforderungen und entsprechende Randbedingungen beschrieben und definiert. Die dortige Beschreibung definiert, dass die zu verwendende Software frei verfügbar und kostenlos sein muss. Aus diesem Grund wird bei dem Betrieb des Servers Ubuntu verwendet, wobei hier die aktuelle Version (Stand Juni 2015) vorausgesetzt wird. Als Webserver kann Apache eingesetzt werden, der mittels dem zustandslosen HTTP-Protokoll mit dem Client kommuniziert. Die benötigte relationale Datenbank kann durch MySQL umgesetzt werden. Diese wird primär für das Speichern von Konfigurationen einzelner virtueller Maschinen genutzt, dem speichern derer individuellen Konfigurationen und zur Persistierung der angebotenen Softwarekomponenten. In dem Execution-Framwork wird der VM-Builder ausgeführt, der wiederum an die Drittanbieter Ansible und Vagrant angebunden ist.

³Bildquelle: Eigene Darstellung

5.3 Systemarchitektur

Nachdem die Umsysteme, deren Verteilung und der technische Aufbau des VM-Builders konzipiert sind, wird die Struktur der eigentlichen Anwendung geplant werden. 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 diese sich in einigen Punkten überschneiden und ergänzen, entstehen bei der Verwendung mehrerer Architekturmuster keine Widersprüche.

5.3.1 Client-Server-Modell

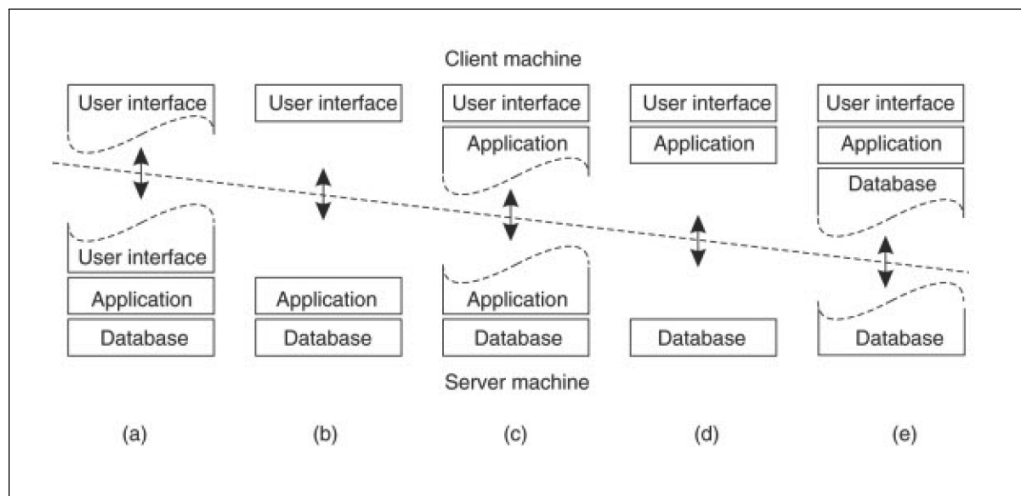
Um eine möglichst gute strukturelle Verteilung der Aufgaben und Dienstleistungen des VM-Builders auf physikalischer Ebene zu erhalten, wird auf das Client-Server-Modell zurückgegriffen. Durch das Client-Server-Modell wird die Aufgabenverteilung innerhalb einer Applikation strukturiert sowie die Zentralisierung von Prozessorenleistung und gemeinsamen Diensten. [Schäfer (2009)]

Die eindeutige 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 Erforderliche, 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 als Schichtenmodell bezeichnet. Eine Schicht ist entweder ein physikalischer Rechner oder mindestens ein Systemprozess, der eine bestimmte Aufgabe übernimmt. Die einfachste Anordnung dieser Schichten besteht darin, diese auf zwei Computer, den Client und den Server, zu verteilen (2-Tier Model)⁴. Dabei können die Schichten wie in Abbildung 5.4 auf der folgenden Seite dargestellt, zwischen Client und Server verteilt sein. [Tanenbaum und van Steen (2007)]

⁵Bildquelle: Tanenbaum und van Steen (2007)

Abbildung 5.4: Client-Server-Anordnungen⁵

Die in Abbildung 5.4 (a) bis (c) dargestellten Varianten gehören zu der Kategorie Thin-Clients, während die Varianten (d) und (e) zu den sogenannten Fat-Clients gezählt werden. Das Konzept des VM-Builders wird eines der Thin-Client-Konstrukte verwenden, um die Client-Seite zu planen.

Einer der Vorteile von Thin-Clients ist, dass weniger bis keine Client-Software auf die Seite des Anwenders gebracht werden muss. Die Problematik, die bei der Verwendung von Software auf Clientseite entsteht, ist die schwerere Administration des Clients und dessen höhere Anfälligkeit für Fehler. [Tanenbaum und van Steen (2007)]

Die, im Rahmen dieser Arbeit, angestrebte Client-Server-Anordnung ist die in Abbildung 5.4 (a) gezeigte Variante. Der Client soll so minimal wie möglich gehalten werden, um dem Anwender einen schnellen Seitenaufbau zu ermöglichen und lokale Installationen zu vermindern. 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 wider, 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) wird in Applikationen ver-

wendet, in denen zusätzlich Logikverarbeitung auf Anwender-Seite ausgeführt werden soll, wogegen die Planung des VM-Builders sprechen würde. So ist die Wahl von Variante (a) als Thin-Client am praktikabelsten.

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-Entwurfsmuster (auch MVC genannt) findet beim Entwurf grafischer Benutzungsoberfläche anwendung, d.h. bei Mensch-Maschine-Interaktionen. Dazu wird das System, das Interaktionen anbietet und ausführt, in drei Verantwortlichkeiten unterteilt:

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 diese 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 nichts mit der Verteilung des Systems zu tun.

⁶Bildquelle: [Wikipedia \(2015\)](#)

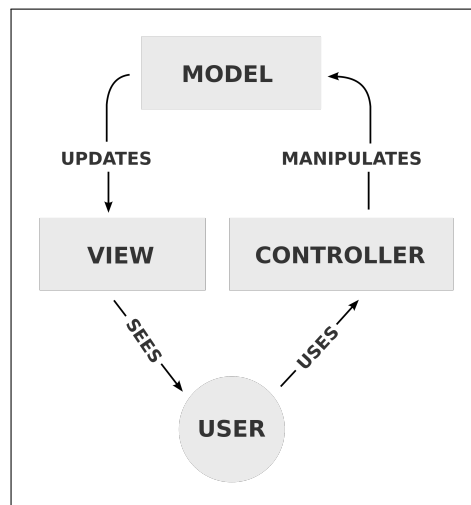


Abbildung 5.5: Model-View-Controller⁶

Abbildung 5.5 zeigt die Kommunikation zwischen den Bestandteilen des Model-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 erhaltenen Daten, in dem diese z.B. an den persistenten Speicher weitergeleitet werden oder Berechnungen stattfinden.
4. Die resultierenden Ergebnisse / Änderungen werden über die View sichtbar gemacht.

Das MVC-Entwurfsmuster entkoppelt das User-Interface von der Verarbeitungsebene. Es trennt die View (Repräsentation) und das Model (Fachlichkeiten) von einander, da die Änderungshäufigkeit beider Ebenen unterschiedlich ausfallen können. Durchschnittlich ändern sich z.B. Windows-Oberflächen etwa alle zwei Jahre, Fachlichkeit ca. 10 bis 15 Jahren. Das MVC-Entwurfsmuster ermöglicht als eine Modernisierungsmaßnahme den Austausch der Oberfläche, ohne die Fachlichkeiten ändern zu müssen. [Masak (2009)]

Da das geforderte Framework Sinatra dieses Konzept nativ umsetzen kann, lässt sich das Entwurfsmuster einfach in der Programmierung anwenden. In Sinatra werden die Views separiert von den sogenannten Routen (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. Die Abbildung 5.6 zeigt die Integration des MVC-Entwurfsmusters in das Schichtenmodell.

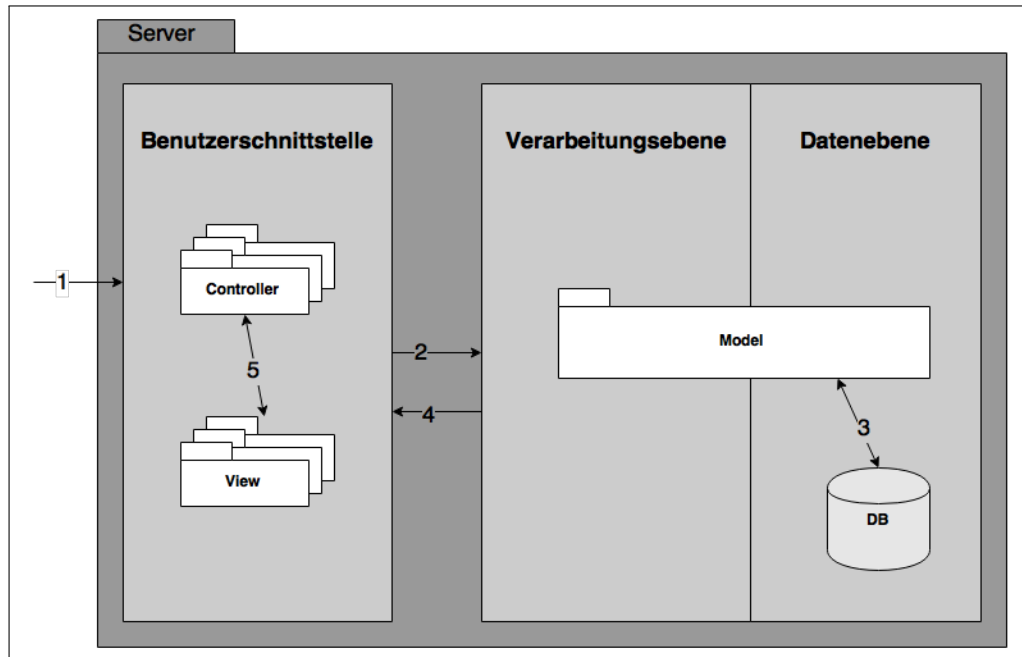


Abbildung 5.6: Model-View-Controller im 3-Schichten-Modell⁷

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 zeilenorientiert. Dementsprechend ist eine Nachricht leicht zu erstellen und auf Serverseite entsprechend leicht auszuwerten. Ist ein Server mit der Anfragebearbeitung fertig, sendet er in der Regel den Status (war die

⁷Bildquelle: Eigene Darstellung

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	Definition
GET	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 Sinatra⁸

5.5 Server

Wie in Kapitel 5.3 beschrieben, wird der Server nach den dort geplanten Entwurfsmustern 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

Starke (2011) beschreibt in seinem Werk die Bausteinsicht wie folgt:

SSie zeigt die statische Struktur des Systems, seinen Aufbau aus Softwarebausteinen sowie deren Beziehungen und Schnittstellen untereinander. Jeder dieser Bausteine wird letztlich durch eine Menge (selbst erstelltem oder zugekauftem) Quellcode implementiert."

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

⁸Quelle: Sinatra (2015)

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.

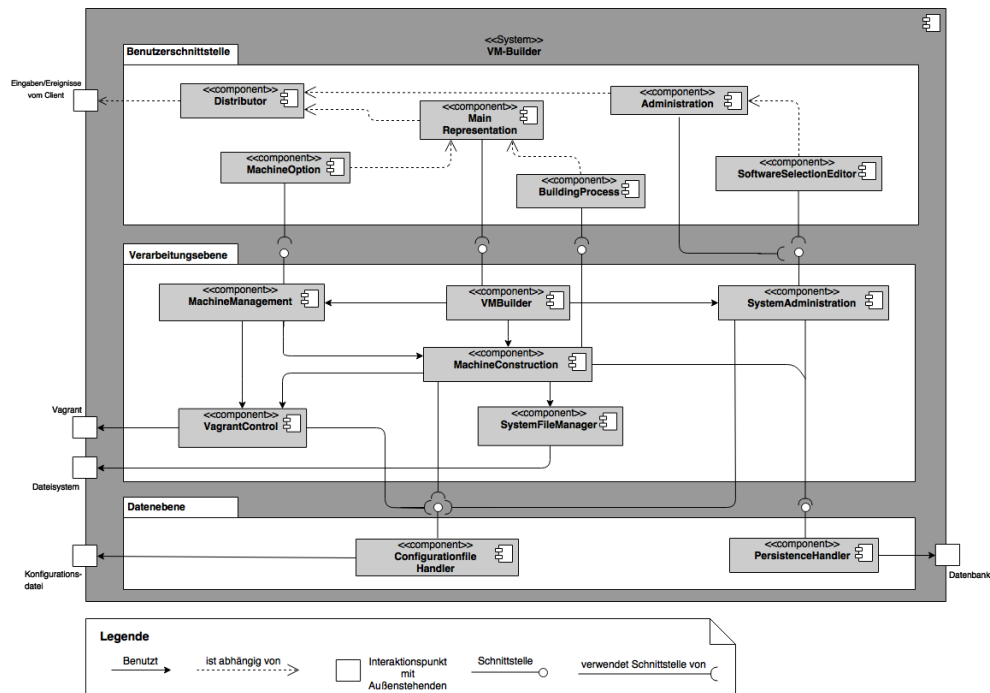


Abbildung 5.7: Bausteinsicht Level 1⁹

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.

⁹Bildquelle: Eigene Darstellung

5.5.1.1 Benutzerschnittstelle

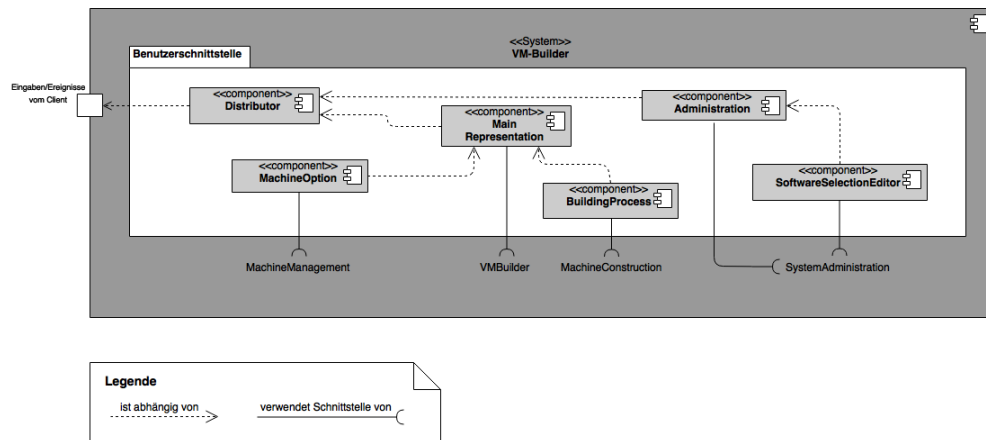


Abbildung 5.8: Benutzerschnittstelle¹⁰

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

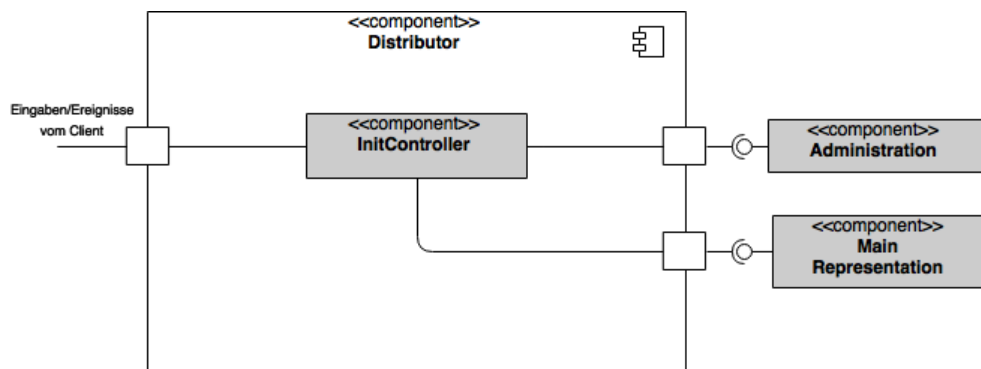


Abbildung 5.9: Komponentensicht Distributor¹¹

Die primäre Zuständigkeit der **Distributor**-Komponente liegt im Empfang der eingehenden Kommunikation und der Weiterleitung an den entsprechenden Controller. Da die Administration des VM-Builders eine gänzlich andere Hauptfunktion ist als der Aufbau inklusive der

¹⁰Bildquelle: Eigene Darstellung

¹¹Bildquelle: Eigene Darstellung

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

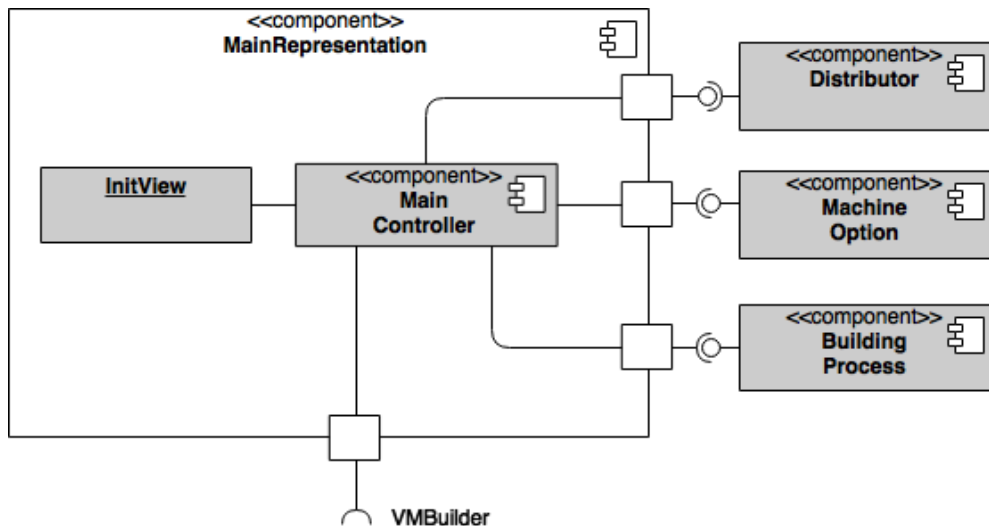


Abbildung 5.10: Komponentensicht VMBUILDER¹²

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

¹²Bildquelle: Eigene Darstellung

¹³Bildquelle: Eigene Darstellung

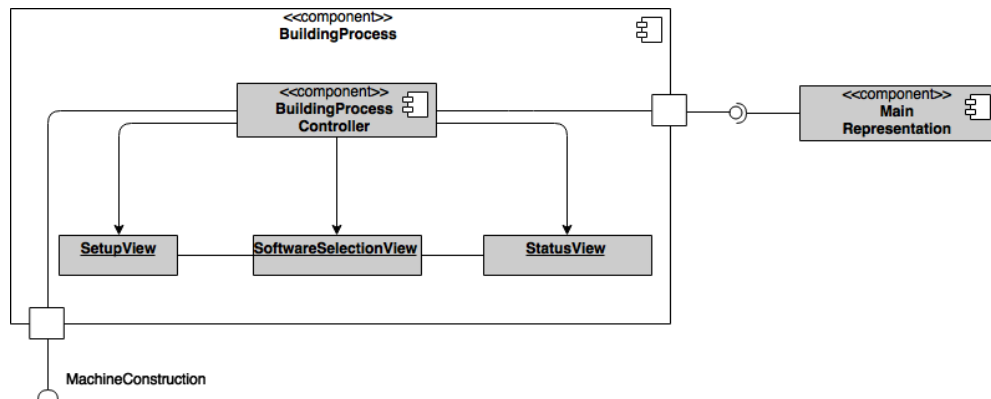


Abbildung 5.11: Komponentensicht BuildingProcess¹³

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 Software-SelectionView dem Anwender eine Auswahl an zu installierenden Softwarekomponenten und Paketen.

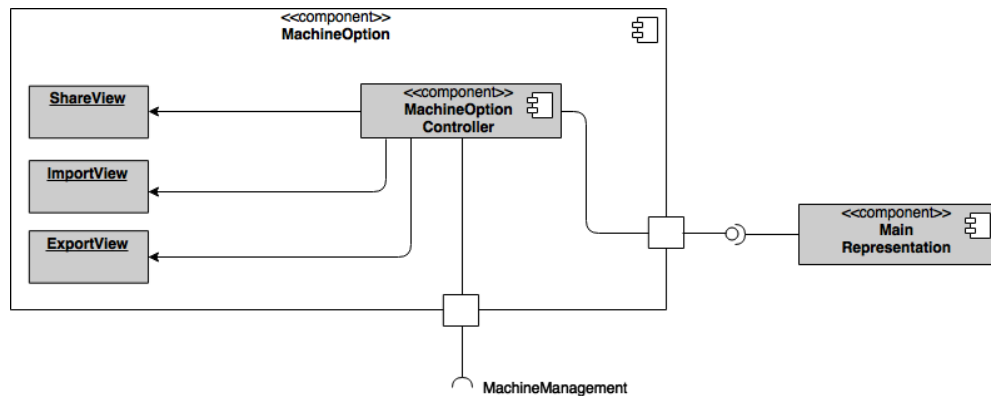
3. StatusView

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.

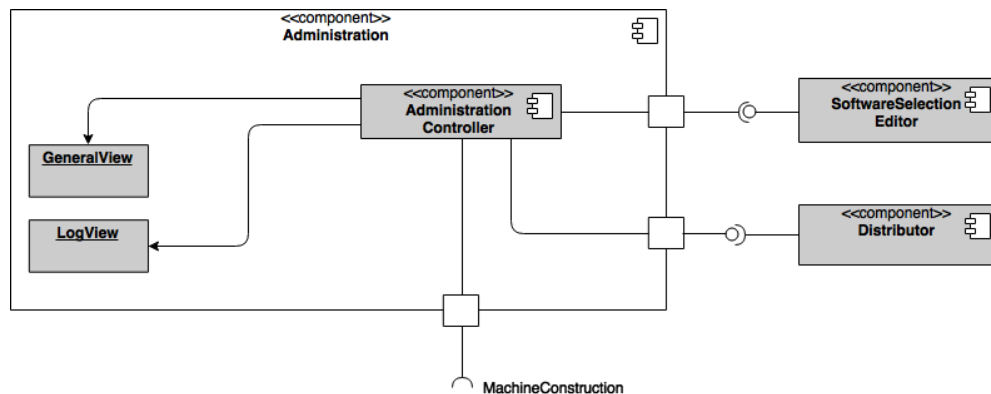
MachineOption-Komponente

¹⁴Bildquelle: Eigene Darstellung


Abbildung 5.12: Komponentensicht MachineOption¹⁴

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 Administration¹⁵

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

¹⁵Bildquelle: Eigene Darstellung

für die Visualisierung der generellen Einstellungen, der Anzeige von Logdateien und des Softwareeditors zuständig. Der Softwareeditor ist als eigenständige Komponente angeschlossen, 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

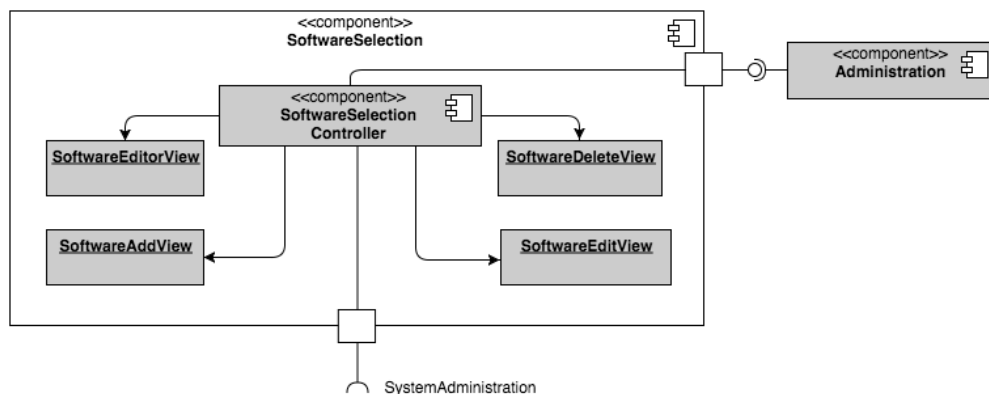


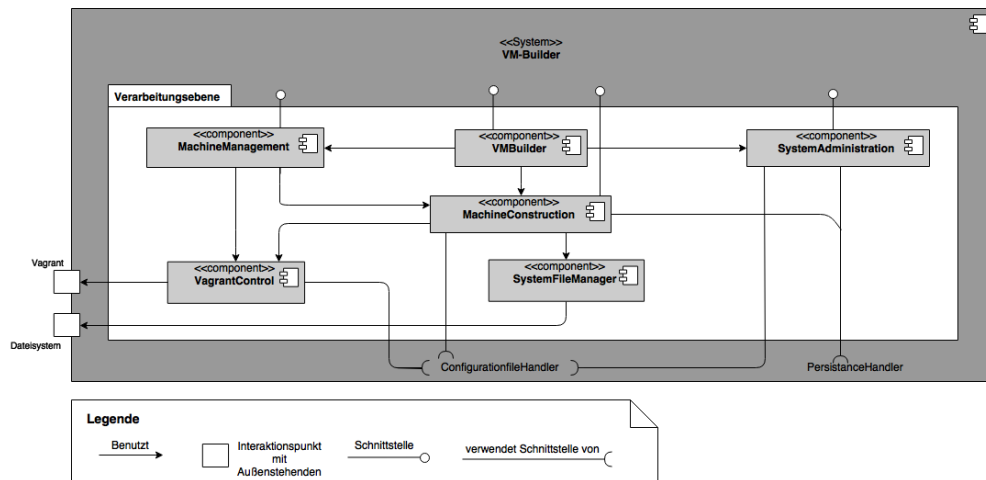
Abbildung 5.14: Komponentensicht SoftwareSelection¹⁶

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

¹⁶Bildquelle: Eigene Darstellung

¹⁷Bildquelle: Eigene Darstellung

Abbildung 5.15: Ansicht der Verarbeitungsebene¹⁷

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.

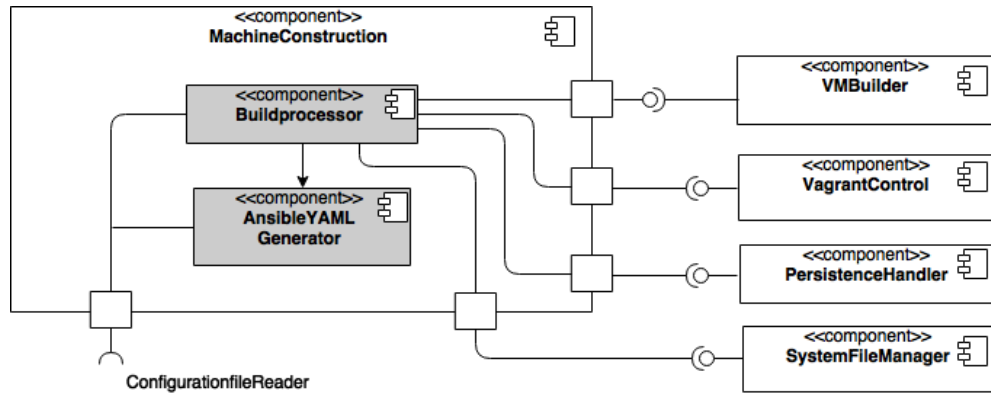
SystemFileManager-Komponente

Die Zuständigkeit des **SystemManagers** besteht im Zugriff auf das Dateisystem. Durch den **SystemFileManager** werden Standard-Ordner-Funktionen des Betriebssystems ermöglicht sowie Kopier-, Duplizierungs- und Erstellungsoperationen. Diese Komponente ist gerade in 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. Zudem werden diese für die Generierung von Fehlermeldungen benötigt und verwendet. Da Vagrant nur mit einem gültigen Vagrantfile lauffähig ist, übernimmt **VagrantControl** auch das Erstellen dieser.

MachineConstruction-Komponente

Abbildung 5.16: Komponentenansicht MachineConstruction¹⁸

Der Aufbau einer virtuellen Maschine wird durch die **MachineConstruction**-Komponente realisiert. Genauer durch das Zusammenspiel zwischen **Buildprocessor** und **AnsibleYAML-Generator**. Eine Zweiteilung der beiden Komponenten ist in Bezug auf ihre Zuständigkeiten wichtig. So sind das Auswechseln der Komponenten und das Erweitern von Funktionalitäten einfacher. Der **AnsibleYAMLGenerator** ist die Komponente, die es Ansible ermöglicht zu wissen, welche Software provisioniert werden soll, während der Aufgabenbereich des **Buildprocessors** 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 entsprechenden Softwarevorschläge, die der Benutzer auf der virtuellen Maschine installieren kann. Der Provisionierer 'Ansible' arbeitet mit YAML-Dateien, die Konfigurationseigenschaften enthalten und den Aufbau einer virtuellen Maschine unterstützen. Die YAML-Dateien sind optional, da diese nur für die Installation von Softwarekomponenten benötigt werden. Der **AnsibleYAMLGenerator** baut aus den Informationen, die er aus dem **Buildprocessor** erhält,

¹⁸Bildquelle: Eigene Darstellung

die richtige Struktur und den inhaltlichen Kontext der YAML-Datei. Durch den Zugriff auf die **ConfigurationfileHandler**-Komponente, die sich in der Datenebene befinden, erhalten Buildprocessor und AnsibleYAMLGenerator Grundkonfigurationseinstellungen, die zum Aufbau benötigt werden.

SystemAdministration-Komponente

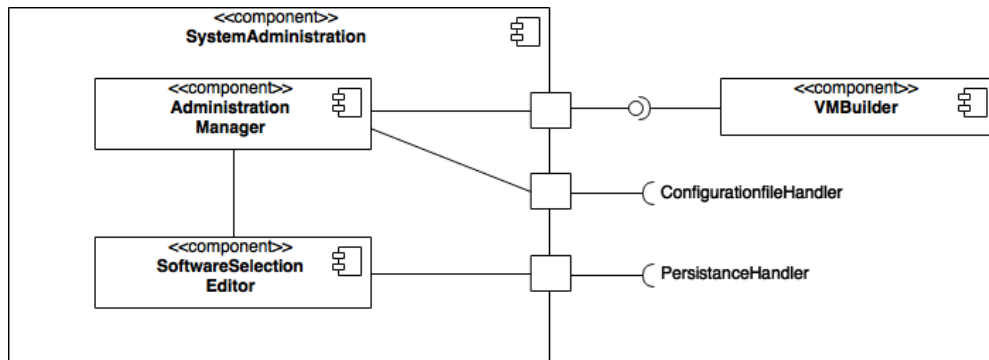


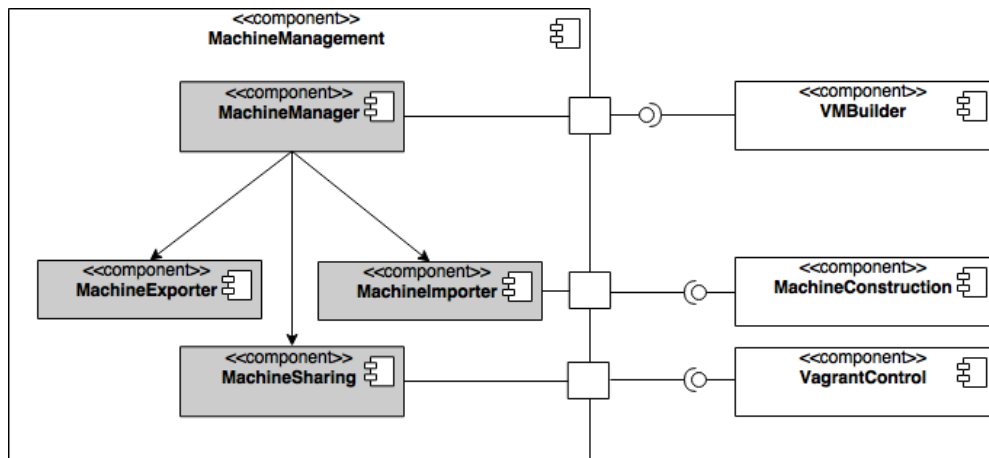
Abbildung 5.17: Komponentenansicht SystemAdministration¹⁹

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 **PersistanceHandler** 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 Skripte hinzugefügt.

MachineManagement-Komponente

¹⁹Bildquelle: Eigene Darstellung

²⁰Bildquelle: Eigene Darstellung

Abbildung 5.18: Komponentenansicht MachineManagement²⁰

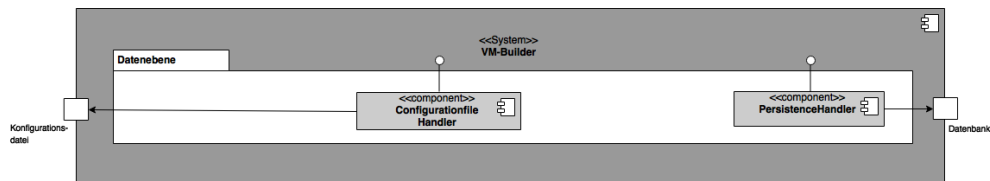
Um dem Anwender Optionen für die bereits bestehenden virtuellen Maschinen anbieten zu können, gibt es den **MachineManager**. Dieser agiert als Verwalter für Optionen, die auf virtuelle Maschinen ausgeführt werden können. Wie in Abbildung 5.15 werden Optionen wie Export-, Import- und das Sharing von 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-Komponente 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

²¹Bildquelle: Eigene Darstellung

Abbildung 5.19: Ansicht der Datenebene²¹

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

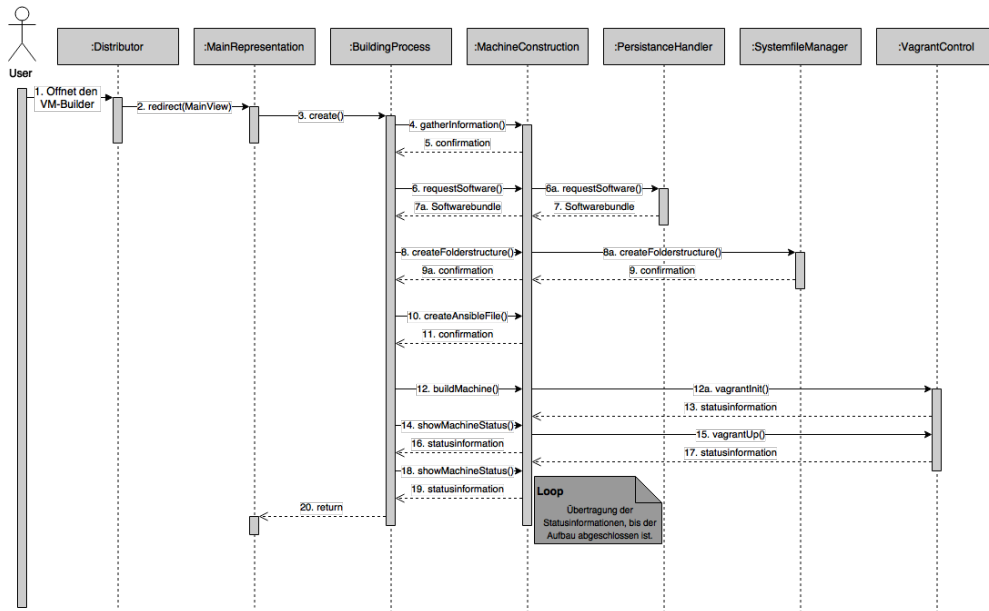


Abbildung 5.20: Laufzeitsicht eines VM-Aufbaus²²

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.

²²Bildquelle: Eigene Darstellung

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 bereithalten, 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

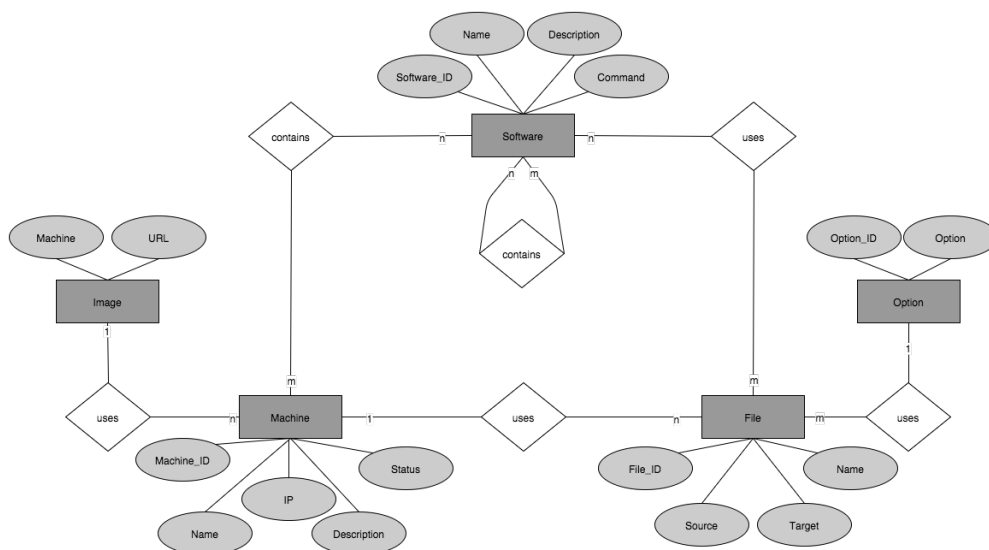


Abbildung 5.21: Entity-Relationship-Model²³

²³Bildquelle: Eigene Darstellung

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

²⁴Bildquelle: Eigene Darstellung

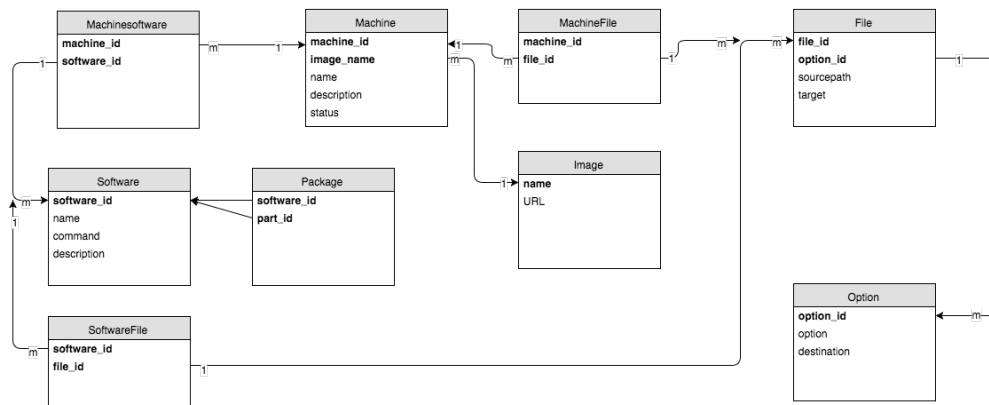


Abbildung 5.22: RelationalDatabaseModel²⁴

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 das 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 umbenannt 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 Thin-Client handelt, der ohne weitere lokale Software verwendbar ist, kommt die Beschreibung in Abschnitt 5.6 mit wenig Erklärung aus.

6 Realisierung

Dieses Kapitel wird die Umsetzung und die damit verbundene Realisierung des VM-Builders beschreiben. Die Umsetzung geschieht unter der Beachtung der zuvor durchgeführten Planung aus Kapitel 3 und der damit verbundenen Integration von Dritt-Anwendungen. Ein wichtiger Aspekt des Kapitels ist die Verdeutlichung von Unterschieden, die zwischen Planung und der Umsetzung auftreten. Die Realisierung selbst wird in erster Linie ein Prototyp des VM-Builders anstreben, wobei auch seine Implementierung sich ganz nach dem Entwurf richtet, der in Kapitel 5 entstanden ist. Um die Anforderungen aus dem technischen Randbedingungen (Abschnitt 3.6.1) zu erfüllen, geschieht die Umsetzung der Applikation in einer Kombination aus dem Framework Sinatra und Ruby und wird in der IDE 'RubyMine' entwickelt.

Die Realisierung wird in drei Schritte unterteilt:

1. Aufbau des Servers (Abschnitt 6.1)
2. Installation und Beschreibung der Fremdsoftware (Abschnitt 6.2)
3. Erstellung und Implementierung des VM-Builders (Abschnitt 6.2)

Die drei Schritte beinhalten nicht nur Details zu ihrer Umsetzung, sondern auch die Unterschiede zum Entwurf.

6.1 Server Realisierung

Um die Voraussetzungen für die Entwicklung des VM-Builders zu schaffen, wird der erste Schritt der Aufbauphase angegangen und eine Entwicklungsplattform vorbereitet. Anstelle eines Hardware-Servers, wird zunächst auf einer virtuellen Maschine entwickelt. Wie bereits in der Einleitung erwähnt, handelt es sich bei der Entwicklung um einen Prototypen der Applikation und genau auf diesem Aspekt aufbauend, ist eine virtuelle Umgebung zum entwickeln und testen mehr als geeignet.

Zeitgleich wird eine identische Umgebung auf einem VM-Host der HAW aufgebaut, um später Unterschiede in der Aufbaugeschwindigkeit der virtuellen Maschinen messen zu können. Einfacher halber wird in den folgenden Abschnitten lediglich ein Server betrachtet, damit

keine Unklarheiten auftreten. Die Unterschiede der Server sind nur rein technischer Natur und unterscheiden sich weder in Applikationsversionen, noch im Aufbau.

Der lokale virtuelle Server wird mit 4GB Arbeitsspeicher, 20 GB Festplattenkapazität und Ubuntu 14.04.2 umgesetzt. Der Server auf HAW-Seite, wird mit 16 GB Arbeitsspeicher ausgestattet und einem schnelleren Prozessor. Das Betriebssystem ist ebenfalls Ubuntu 14.04.2.

6.2 Fremdsoftware

Eines der Ergebnisse des Entwurfs war die Verwendung von Fremdsoftware, wodurch gewisse Teile des Funktionsspektrums übernommen werden sollen. Der zweite Schritt der Realisierung wird sich mit den einzelnen Softwareelementen beschäftigt, die für den Betrieb des VM-Builders eingesetzt werden sollen. In der Evaluation aus Kapitel 4 sind die ersten Annahmen getroffen worden welche Softwarekomponente sich für bestimmte Aufgabenbereiche eignen könnten, um sie dann in der in der Verteilungssicht (Abschnitt 5.2) provisorisch zu integrieren. Vorteile der verwendeten Applikationen sind ihre OpenSource Eigenschaften und der damit verbundene Verzicht auf weitere Kosten oder Investitionen. Die einzelnen Applikationen werden noch einmal kurz vorgestellt, wobei auf Installationsdetails verzichtet wird.

6.2.1 Webserver

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

Es bietet sich an, einen etablierten Webserver zu nutzen, gerade wenn er den geforderten Open-Source-Aspekt erfüllt. Zudem besitzt Apache einen Marktanteil von 50,51% an aktiven Webseiten ([Statista \(2015\)](#)), wodurch das Argument der Etablierung weiter untermauert wird. Durch diesen Faktor kann zudem davon ausgegangen werden, dass die Weiterentwicklung weiter anhält, sowie Supportmöglichkeiten auch noch zukünftig vorhanden sein werden.

6.2.2 Phusion Passenger

Im Zuge der Vorbereitung auf die Applikation, wird auf Phusions Passenger zurückgegriffen, das als ein Modul für Apache konzipiert wurde. Phusions Passenger wird primär bei 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 zielführend noch praktikabel. Deshalb wurde in der Evaluation (Kapitel 4.1.3) eine Analyse der frei zugänglichen Virtualisierer durchgeführt, wodurch letztendlich das Ergebnis VirtualBox als geeigneten Kandidaten herausstellte. VirtualBox übernimmt beim VM-Builder die Funktion der Maschinenvirtualisierung und somit den automatisierten Aufbau der Umgebung.

6.2.4 Ansible

Damit die erstellte Umgebung mit Software bestückt werden kann, wird diese Aufgabe Ansible übertragen. Ansible kann die gewünschte Zustandsbeschreibung des Anwenders, direkt auf der virtuellen Maschine umsetzen, ohne dort einen extra Client installieren zu müssen. Dies erleichtert nicht nur die Arbeit mit Ansible, sondern erleichtert zudem die Integration von Ansible in den VM-Builder.

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 Abhängigkeiten nutzen, Pakete zu verwalten und ggf. deren Versionen zu spezifizieren, kann auf Bundler zurückgegriffen werden. Durch Bundler können einheitliche Umgebungen für Projekte geschaffen und der Wiederaufbau der gleichen Umgebung beschleunigt werden. Durch seine Eigenschaft, Ruby-Komponenten sofort und automatisch installieren zu können, ist Bundler prädestiniert für die Umsetzung einer automatisierten Installation des VM-Builders.

6.2.7 Datenbank

Um die objektorientierte Welt mit der Relationalen Welt der Datenbanken in Einklang zu bringen, können OR-Mapper helfen. Durch ihre Übersetzung von Quellcode, können program-

matisch Tabellenkonstrukte erzeugt und Beziehungen zwischen den gewünschten Tabellen definiert werden. Datamapper ist eines der Beiden populärsten ORM Frameworks für Ruby und wird die beschriebenen Aufgaben im VM-Builder übernehmen.

Ein Beispiel für die Funktionsweise von Datamapper, ist die Definition der Tabelle **Machine**, die aus vier Attributen besteht inklusive ihrer Typ-Eigenschaften, sowie ihrer Relation zu anderen 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 Datamapper¹

Programmatrischer 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 command²

¹Codebeispiel: Eigene Darstellung

²Codebeispiel: Eigene Darstellung

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-Builder³

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 Beispiel⁴

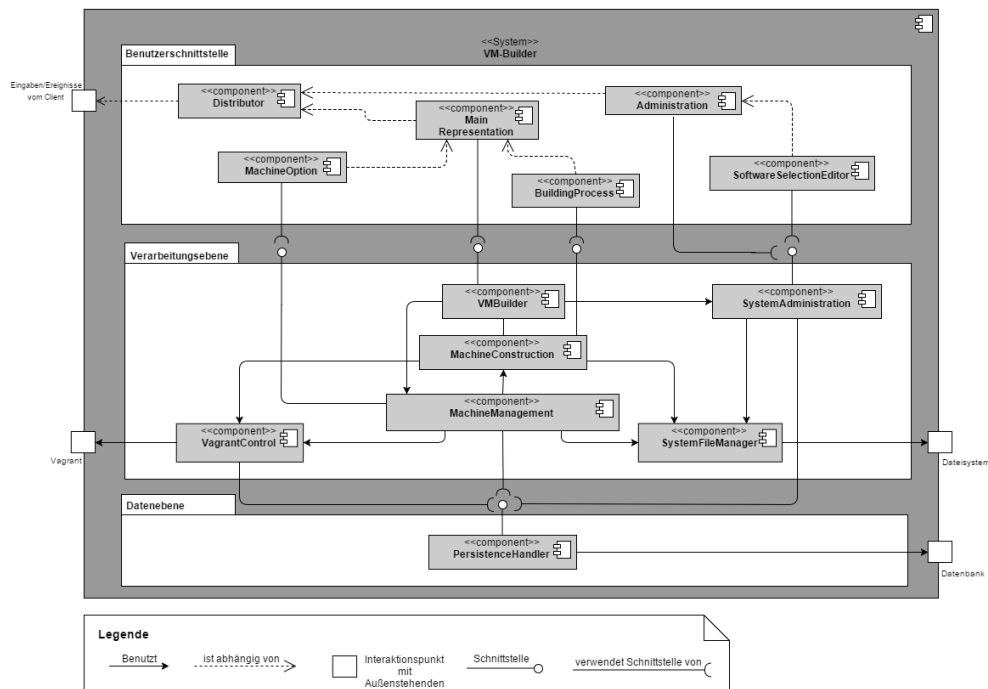
Während die Views in HTML geschrieben und Helper, sowie Models als Ruby-Klassen definiert sind.

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

³Quelle: Eigene Darstellung

⁴Codebeispiel: Eigene Darstellung

⁵Bildquelle: Eigene Darstellung


Abbildung 6.1: Bausteinsicht Level 1⁵

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. Anstelle der Konfigurationsdatei, werden die Konfigurationen in der Datenbank gepflegt. Die daraus entstehenden Vorteile sind eine zentrale Steuerung für den persistenten Speicher und Änderungen, sowie das Hinzufügen von neuen Konfiguration sind deutlich vereinfacht worden.

6.3.2 Datenbank

Wie bereits erwähnt, wird im Gegensatz zur Planung, ein Tabellenkonstrukt angewendet, um Konfigurationen 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	Start		true (or 1, or Y, ...)
4	FullScreen		true (or 1, or Y, ...)
5	Resolution		720
6

Listing 6.5: Name-Value-Pair Table⁶

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	Start		FullScreen		Resolution		...
3	-----+-----+-----+-----						
4	true		true		20		...

Listing 6.6: Single-Row Table⁷

Vorteil

- a) Jede Spalte hat bekommt den entsprechenden Datentyp

⁶Abbildungsquelle: Eigene Darstellung

⁷Abbildungsquelle: Eigene Darstellung

- 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 realisiert worden:

Erstellung einer virtuellen Maschine

Der Aufbau einer virtuellen Maschine kann, entgegengesetzt zur Planung, von überall aus aufgerufen werden. Die Planung sah ursprünglich vor, dass eine Maschine nur von der Hauptseite aus erstellt werden kann. Ein Optionsmenü, welches am oberen Rand platziert wurde, stellt dem Anwender den Aufbau und diverse andere Funktionen zur Verfügung. Die Realisierung des Aufbaus geschieht hauptsächlich über die BuildingProcess-Komponente. Sie verwaltet nicht nur die für sie bestimmten HTTP-Requests, sondern hält auch die Darstellung vor, die den Aufbau betreffen. Der Aufbauprozess selbst beinhaltet die Benamung der gewünschten Maschine, Optionen wie IP-Adresse, die Auswahl an Softwarekomponenten und die Möglichkeit Dateien auf dem Zielhost zu speichern. Echtzeitinformation über den Aufbauprozess sind nicht implementiert. Allerdings in einem separaten Logfile nachlesbar. VagrantControl liefert in Kombination mit dem SystemFileManager die Befehle für den Aufbau einer Maschine. Der SystemFileManager bietet eine Methode für direkte Systemaufrufe an, mit der VagrantControl Vagrantspezifische Befehle absetzen kann. Sobald der Anwender alle nötigen Informationen

eingetragen hat und dies im Webformular bestätigt wurde, werden eine Reihe an aufbauprozessen eingeleitet. Der VM-Builder legt im ersten Schritt alle nötigen Informationen in der Datenbank ab. Dabei garantieren Transaktionen die konsistente Speicherung aller nötigen Informationen. Danach wird ein eigenes Verzeichnis angelegt, in dem aufbaurelevante Dateien, sowie Logdateien abgespeichert werden. In diesem Zusammenhang werden das Vagrantfile und eine YAML-Datei als aufbaurelevant bezeichnet, da sie die Kernkonfigurationen der zukünftigen virtuellen Maschine beinhalten. Das Vagrantfile wird durch die VagrantControl-Komponente erzeugt und wird dabei dynamisch an die Auswahl des Anwenders angepasst. Benötigt der Anwender keine Software und keine Dateien auf seiner Maschine, wird im Vagrantfile der Provisionierer (Vagrant) abgeschaltet. Ähnlich funktioniert der Aufbau der YAML-Datei für Ansible. Sie wird ebenfalls dynamisch an den Kontext der Auswahl angepasst und von dem YamlBuilder erzeugt. Die dynamische Anpassung soll Fehlinformationen in den Dateien verhindern und eine spätere Erweiterung an Funktionen vereinfachen. Falls der Anwender Dateien hochladen wollte, werden diese nun in das angelegte Verzeichnis der Maschine geladen. Der letzte Schritt beinhaltet den Bau der Maschine, wodurch auch die Zustandsumsetzung erfolgt. Sobald die Maschine erstellt wurde, werden die ausgewählten Softwarepakete installiert und die Dateien auf der Maschine platziert. Danach startet der VM-Builder die virtuelle Maschine hoch, so dass der Anwender sie sofort nutzen kann. Der ganze Aufbauprozess ist ab dem ersten Schritt im Verwaltungsmenü des VM-Builders nachzuverfolgen, da jeder Aufbauschritt in der Logdatei gespeichert wird.

Provisioning

Wie im oberen Abschnitt bereits beschrieben, ist die Provisionierung ebenfalls umgesetzt worden. Sie wurde in den Aufbau mit integriert und wird durch die Komponente MachineConstruction realisiert. Um genauer zu sein, durch die Klasse 'YamlBuilder'. Dateien und Software werden durch diese Klasse in eine YAML-Datei übersetzt und mit Einstellungen aus der Konfigurations-Tabelle angereichert. Beinhaltet die ausgewählte Software ein oder mehr Packages, werden diese in ihre einzelnen Komponenten aufgelöst und so in der YAML-Datei hinterlegt. Ähnlich agiert die Komponente bei Dateien. Der Anwender hat die Möglichkeit eine Vielzahl an Dateien auf die Zielmaschine zu bringen, worauf hin die YAML-Datei dynamisch angepasst wird.

VM-Konfigurationen speichern

Der Aufbau einer virtuellen Maschine beinhaltet zeitgleich das Speichern ihrer Konfiguration. Jede getroffene Auswahl des Anwenders wird über Transaktionen in der

Datenbank hinterlegt. Durch die Verwendung von Transaktionen wird vermieden, dass inkonsistente Konfigurationen gespeichert werden und sichergestellt, dass diese auch nur im Erfolgsfall gesichert werden.

Packages erstellen

Packages können vom Anwender ähnlich wie Softwarekomponenten erstellt werden. Einer der großen Unterschiede liegt allerdings im Konfigurationsspektrum und ihrer eigentlichen Verwendung. Packages werden in der Datenbank wie Softwarekomponenten geführt, allerdings mit einer extra Kennzeichnung. So stehen sie zur Auswahl beim Maschinenaufbau, werden aber vom System als Package erkannt. Packages können aus diversen Softwarekomponenten bestehen und aus einer Vielzahl von Dateien. So wird es möglich größere Softwareprojekte zu erstellen und dem Anwender zur Verfügung zu stellen. Die SystemAdministration-Komponente hält dafür die entsprechende Implementierung vor. Da die Implementierung organisatorisch zur Administration gehört, wurde sie auch in diesem Paket platziert. Wie bei der Speicherung von Konfigurationen, wird auch hier mit Transaktionen gearbeitet, um keine Inkonsistenzen zu verursachen. Um ein Bild einer solchen Transaktion zu bekommen zeigt das Codebeispiel in Abbildung 6.7 den Vorgang des Speicherns eines Softwarebundles.

```
1 def save_software_bundle(program, command, desc, selection)
2   Software.transaction do |t|
3     begin
4       Software.first_or_create(:name => program,
5                               :command => command,
6                               :description => desc,
7                               :package => 1)
8
9       selection.each do |software|
10        Package.first_or_create(
11          :source_id => get_software_id(program),
12          :sub_id => get_software_id(software))
13      end
14    rescue
15      t.rollback
16    end
17  end
```

18 **end**

Listing 6.7: Beispiel einer Transaktion - Speichern eines Packages⁸

Werden nicht alle Bedingungen erfüllt, verwirft die Transaktion alle Änderungen und führt ein **Rollback** aus. Daraus folgt, dass die Datenbank auf den Stand zurück geholt, wie er vor der Ausführung der Transaktion war.

VM-Sharing

Das 'Sharing' einer bestehenden virtuellen Maschine wird durch die interne Implementierung von Vagrant übernommen. Jede Maschine, die in der Hauptansicht zu sehen ist, kann von dort aus freigegeben werden. Wie jede weitere Option einer Maschine, wird die Share-Option durch den AdministrationController angesteuert und anschliessend durch die Logik der Option ermöglicht. In diesem Fall wird die Logik von der MachineShare-Komponente bereitgestellt. Die Voreinstellung des 'Sharings' ist auf HTTP gesetzt, wodurch ein Webzugriff auf die Maschine ermöglicht wird. Optional kann in der Implementierung auf einen SSH Zugriff gewechselt werden, der direkten Zugriff auf die Maschine erlaubt. So wäre beispielsweise simultaner Zugriff auf eine Maschine möglich, wodurch Teamarbeit von mehreren Standorten kein Problem darstellt. Die Zuständigkeit der Umsetzung erfolgt über 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.

Import

Der wesentliche Bestandteil einer virtuellen Maschine, die über den VM-Builder aufgebaut wird, sind zwei Dateien. Das Vagrantfile, welches generelle Konfigurationen über die Maschine beinhaltet und die Yaml-Datei, die für die Zustandsumsetzung auf der Maschine benötigt wird. Werden diese beiden Dateien in den VM-Builder geladen, kann daraus der Aufbau der Maschine abgeleitet und die Provisionierung der Umgebung übernommen werden. Die Importierte Maschine, wird im System wie jede andere Maschine geführt und kann somit auf die gleichen Optionen zugreifen. Der Import wird durch den MachineOptionController verwaltet, während dieser auf die Logik im MachineImporter zurückgreift. Damit der Import im System umgesetzt werden kann, verwendet die Klasse MachineImporter die Komponenten der MachineConstruction. Entsprechende Aufbauinformationen werden aus dem MachineImporter extrahiert und

⁸Codebeispiel: Eigene Darstellung

an die `MachineConstruction` weitergeleitet. Dort geht dann der normale Aufbauprozess von statten.

Export

Der Export befähigt den Anwender seine Maschine vom Server herunterzuladen und als `vmdk`-Datei zu speichern. Wie auch bei allen anderen Optionen, wird die Annahme des HTTP-Requests vom `MachineOptionController` verarbeitet. Die Logik, die in der Klasse `'MachineExport'` beherbergt ist, veranlasst Vagrant die gewählte Maschine zu packen. Das entstandene Paket besteht dabei aus folgenden Dateien:

```
1 | -- Vagrantfile
2 | -- box-disk1.vmdk
3 | -- box.ovf
4 | -- metadata.json
```

Listing 6.8: Inhalt eines Export⁹

Eine Veranschaulichung der Applikation lässt sich im Anhang (8) finden. Dort werden die einzelnen Funktionen noch einmal kurz aufgelistet und visualisiert.

⁹Codebeispiel: [Vagrant \(2015\)](#)

7 Schluss

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

7.1 Zusammenfassung und Fazit

Das Ziel dieser Arbeit war die Entwicklung einer Applikation, die einen schnellen und unkomplizierten Aufbau von virtuellen Maschinen ermöglicht sowie deren Verwaltung gepaart mit einer leichten Steuerung. Die Applikation sollte dem Anwender ermöglichen, eigenständig seine virtuellen Aufbauten zu erstellen und zu verwalten. Die Planung der Applikation (in der Arbeit VM-Builder genannt), 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-Cases formten, bezogen auf die Verwendbarkeit, die ersten Details und stellten die primären Funktionen heraus. Mockups verhalfen den Applikationskontext in eine erste Form zu bringen und Optik, Bedienbarkeit und Funktionen zu vereinen. Die anschließende Evaluation von Open-Source-Anwendungen (Kapitel 4) ergab verschiedene Applikationen, die den Funktionsumfang unterstützen. Vagrant stellte sich als hervorragender Lieferant für diverse Funktionalitäten heraus. Für die Zustandsumsetzung auf der gewünschten Maschine, war Ansible die erste Wahl. So wurden etablierte und vorhandene Anwendungen in die Planung mit integriert, um das Funktionsspektrum zu verbessern.

Der Entwurf in Kapitel 5 basiert auf der vorhergegangenen Anforderungsanalyse und greift deren Aspekte auf, um die programmatische Umsetzung zu definieren. Im Softwareentwurf wurden beschlossen, die Applikation nach bestimmten Systemarchitekturen zu gliedern. So wurde das Client-Server-Modell und das MVC-Entwurfsmuster erarbeitet und bei der Konzeptionierung der einzelnen Komponenten berücksichtigt. Anschließend wurden verschiedene Sichten verwendet, die unterschiedliche Blickwinkel auf die Applikation geben und damit bei der Planung unterstützend wirkten. Im Laufe des Entwurfs stellte sich die Klärung der Datenbankstruktur als expliziter Punkt heraus, die durch zwei typische Datenbanksichten

entworfen wurde. Zuerst wurde das Entity-Relationship-Model (auch als ERM bezeichnet) als Darstellungsform gewählt, um Beziehungen und Attribute der Entitäten darzustellen. Das darauf folgende relationale Datenbank-Modell basiert auf den Ergebnissen des ERMs und lehnt sich an die tatsächliche Umsetzung an. Relationen zwischen Tabellen wurden dort genauer betrachtet und ggf. nach gewissen Regeln aufgelöst. Das Resultat sind neue Koppeltabellen (Zwischentabellen), die bei der Umsetzung zu berücksichtigen sind.

Bei der Umsetzung des VM-Builders in Kapitel 6 stellten sich teilweise Unterschiede zur Planung heraus. Eine der größten Änderungen betraf den Wegfall der geplanten Konfigurationsdatei. Sie sollte Einstellungen des VM-Builders beinhalten und durch entsprechende Komponenten ausgelesen und beschrieben werden. Konfigurationen werden zukünftig in der bereits bestehenden Datenbank verwaltet. So wird der Implementierungsaufwand verringert und nur ein persistenter Speicher verwendet. Die Entsprechende Umsetzung der Konfigurationstabellen ist in Abschnitt 6.3.2 nachzulesen. Die geplante Struktur der Komponenten konnte beibehalten werden, wobei sich teilweise neue Assoziationen zwischen diesen herausstellte, die in der Planung nicht berücksichtigt wurden. Die im Entwurf beschriebenen Architekturstile konnten gut übernommen werden und halfen bei der Umsetzung. Der komplette Entwurf, beschleunigte nicht nur die Umsetzung, sondern verhalf zu einer übersichtlichen Strukturierung. Auch die Vorgaben, die Komponenten mit hoher Kohäsion und niedriger Kopplung zu entwerfen, kam der Umsetzung zugute. Alle Komponenten können leicht ausgetauscht werden und erfüllen ihren alleinigen Zweck. Die GUI wurde optisch den Mockups angepasst, um den Prototypen übersichtlich und einfach steuerbar zu machen. Alle geplanten Funktionen des VM-Builders sind im Testumfeld voll funktionsfähig. Im zweiten Aufbau, der im HAW-Umfeld stattgefunden hat, wurde das Teilen von Maschinen mit anderen Anwendern durch die Netzwerkbeschränkungen unterbunden. Komplikationen wie diese, lassen sich aber durch konkrete Einstellungen in den verwendeten Softwarekomponenten beheben.

7.2 Ausblick

Bei der Entwicklung und Planung der Applikation sind weitere Ideen und Anwendungszwecke entstanden, die allerdings in der zeitlichen Begrenzung und mit der vorhandenen Arbeitskraft nicht umzusetzen waren. Vorstellbar wäre die Erweiterung des Funktionsumfangs, so dass ganze Infrastrukturen nachgebildet werden könnten. Das bedeutet, dass mehrere Server, inklusive ihrer Softwarebestandteile, nachgebaut werden könnten und das auf Knopfdruck. Die Basis dafür bietet der jetzige Entwicklungsstand schon an. Eine entsprechende Erweiterung wäre durch ein umfangreiches Konfigurationsmenü und die Anpassung der Komponenten

möglich, die für die Kreierung der Konfigurationsdateien nötig sind. Die zugrunde liegenden Softwarekomponenten sind in der Lage, mehrere Maschinen gleichzeitig zu erstellen und sie unabhängig voneinander zu provisionieren. Extras wie unterschiedliche und explizite IP-Adressen zu vergeben, ist z.B. eine minimale Konfigurationsangelegenheit. Da Ansible fähig ist, vorgefertigte Zustandsbeschreibungen (Playbooks) zu verwenden, könnten auch zuvor geschriebene Zustandsbeschreibungen nachträglich in der zu erstellenden virtuellen Infrastruktur umgesetzt und für spätere Zwecke abgelegt werden. Zudem wäre eine Erweiterung denkbar, die automatisiert produktionsnahe Umgebungen erstellt und gleichzeitig den aktuellen Datenbestand zur Verfügung stellt. So hätten Tester und Entwickler schneller und effektiver einen Abzug des Produktionsumfeldes, um ihren Tätigkeiten besser nach zu gehen. Zu beachten wäre bei allen Aufbauten, die Integration der Schnittstellen zwischen den verwendeten Systemen.

8 Anhang

Dieses Kapitel zeigt Auszüge aus dem Prototypen und gibt kurze Einblicke in die Funktionsweise. Die Abläufe werden entsprechend erklärt und sollen bei der Verdeutlichung der Arbeitsweise helfen.

8.1 Aufbau einer virtuellen Maschine

Create a new machine

Machine name

IP address

Description

Image

- ☒ Standard 32 bit machine
- ☐ Standard 64 bit machine
- ☐ Enter Machine

Software

	Name	Command
<input type="checkbox"/>	unzip	Command: <code>sudo unzip</code>
<input type="checkbox"/>	zip	Command: <code>sudo zip</code>
<input type="checkbox"/>	apache	Command: <code>sudo apt-get install apache2 apache2-doc</code>
<input type="checkbox"/>	CRM-Bundle	Command: <code></code>

Files

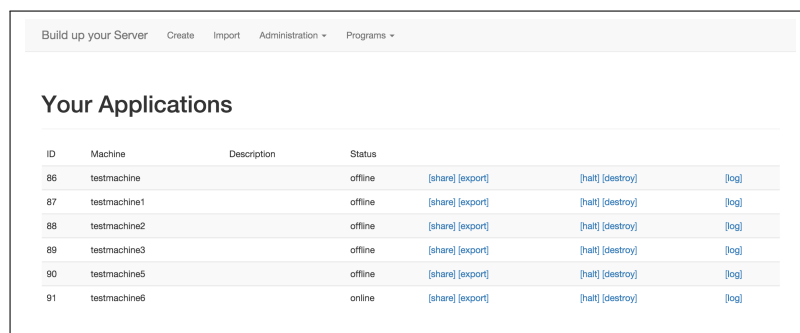
Abbildung 8.1: Menü zum Erstellen einer virtuellen Umgebung

Der Aufbau einer virtuellen Umgebung wird durch das Menü in Abbildung 8.1 ermöglicht. Der Anwender bekommt hier die Möglichkeit, Konfigurationsparameter zu erfassen und eine Auswahl des Betriebssystems zu treffen. Die Konfigurationsparameter umfassen die Vergabe eines individuellen Namens, der IP-Adresse und einer optionalen Beschreibung der Maschine. Danach kann er sich entscheiden, ob und welche Softwarekomponenten er auf der Zielmaschine verwenden möchte. Die Auswahl der Softwarekomponenten kann einzelne Softwarebestandteile beinhalten und Gruppierungen von Software. Die Gruppierung ermöglicht das Konfigurieren von ganzen Installationsabläufen und das Einbringen von externer Software. Das CRM-Bundle

aus Abbildung 8.1, ist so eine Gruppierung an Software und beinhaltet mehrere Softwarekomponenten.

Zu guter letzt wird dem Anwender angeboten, beliebig viele Dateien mit zu übergeben, die dann durch den VM-Builder auf den Zielhost gebracht werden. Nach dem Aufbau der virtuellen Umgebung, wird der Anwender in eine Übersicht geleitet, die ihm den Status seiner und aller aktuell vorhandenen Maschinen anzeigt. Zu sehen in Abschnitt 8.2, Abbildung 8.2.

8.2 Verwaltung der virtuellen Maschinen



ID	Machine	Description	Status			
86	testmachine		offline	[share]	[export]	[halt] [destroy] [log]
87	testmachine1		offline	[share]	[export]	[halt] [destroy] [log]
88	testmachine2		offline	[share]	[export]	[halt] [destroy] [log]
89	testmachine3		offline	[share]	[export]	[halt] [destroy] [log]
90	testmachine5		offline	[share]	[export]	[halt] [destroy] [log]
91	testmachine6		online	[share]	[export]	[halt] [destroy] [log]

Abbildung 8.2: Übersicht aller aktuellen virtuellen Maschinen

Die Verwaltung der einzelnen Umgebungen, wird über das Hauptmenü der Anwendung möglich. Nicht nur Maschinenbezogene Informationen werden über das Menü angezeigt, sondern auch Optionen für die einzelnen Maschinen angeboten. Über die **Share**-Funktion, wird die Maschine für einen externen Zugriff vorbereitet und ermöglicht so den Zugriff über das Internet. Mehr Informationen zu der Share-Option unter (8.3). Der Export einer virtuellen Maschine kann durch die entsprechende Funktion eingeleitet werden. Das Stoppen, sowie das Herausnehmen der Umgebung aus dem System, wird ebenfalls explizit für jede Umgebung angeboten. Die **Log**-Funktion ermöglicht das Nachlesen des bisherigen Maschinenstatus. Alle Aktionen, die auf die entsprechende Maschine angewandt wurden, sind dort nachlesbar. Sei es der Aufbaufortschritt, die installierte Software oder das Beenden der Maschine.

8.3 Externer Zugriff

Unterschiedliche Zugriffsmöglichkeiten auf eine virtuelle Maschine, helfen bei kooperativer Arbeit von Mitarbeitern. Gerade wenn die involvierten Mitarbeiter nicht im gleichen Gebäude oder Land sitzen. Deshalb kann die Sharing-Option angewählt werden, die die Maschine im

The screenshot shows the 'Share your Machine' page in the Vagrant Share web interface. At the top, there is a navigation bar with links: 'Build up your Server', 'Create', 'Import', 'Administration', and 'Programs'. The main heading is 'Share your Machine'. Below this, the interface displays the following information:

- Hostname:** testmachine6
- Sharename:** testmachine6
- URL:** HTTP://testmachine6.vagrantshare.com
- For your Partner:** vagrant connect superb-vicuna-4426.vagrantshare.com

Abbildung 8.3: Externe Zugriffsoptionen auf eine virtuelle Maschine

laufenden Betrieb so konfiguriert, dass auf sie über einen Link zugegriffen werden kann. Der Anwender selbst ist dabei immer noch in der Lage seine Maschine zu verwenden, während der oder die anderen Mitarbeiter von extern darauf arbeiten.

8.4 Import

The screenshot shows the 'Import your Machine' page in the Vagrant Share web interface. At the top, there is a navigation bar with links: 'Build up your Server', 'Create', 'Import', 'Administration', and 'Programs'. The main heading is 'Import your Machine'. Below this, the interface displays the following form fields:

- Machine Name:** A text input field.
- Vagrantfile:** A button labeled 'Datei auswählen' and a status indicator 'Keine ausgewählt'.
- YAML-File:** A button labeled 'Datei auswählen' and a status indicator 'Keine ausgewählt'.
- Save:** A blue button at the bottom right.

Abbildung 8.4: Import durch Konfigurationsdateien

Wie bereits in der Umsetzung des Imports aus Kapitel 6.3.3 erwähnt, werden für den Aufbau einer virtuellen Maschine zwei wesentliche Dateien benötigt:

- Das Vagrantfile, welches generelle Konfigurationen enthält und z.B. Informationen über das zu installierende Betriebssystem beinhaltet.
- Die Yaml-Datei, die für die Zustandsumsetzung benötigt wird.

Sind diese Dateien hochgeladen und hat der Anwender dies mit einem Klick bestätigt, wird der Aufbauprozess eingeleitet, wodurch die importierte Maschine in das System integriert wird.

8.5 Log

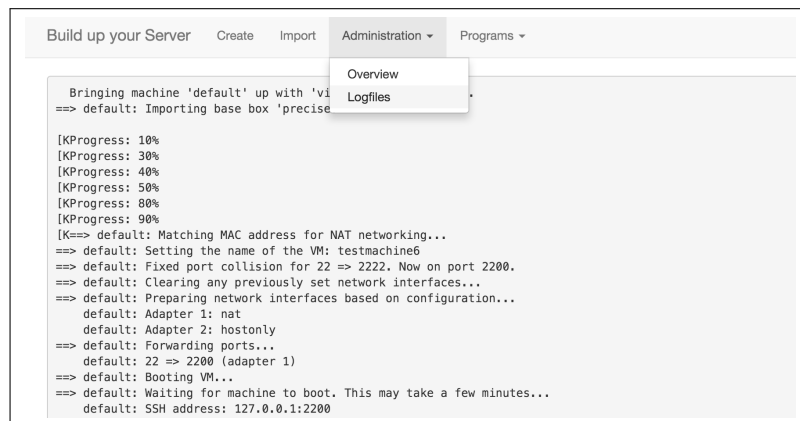


Abbildung 8.5: Allgemeine Logdatei

Jede Maschine verwendet eine eigene Logdatei, um vergangene Vorgänge besser nachvollziehen zu können. Zudem werden alle Aktionen zu jeder Maschine in einer allgemeinen Log-Datei protokolliert, die über eine administrative Oberfläche einsehbar ist. Die individuellen Logdateien beinhalten grundsätzlich alle Aktionen, die auf jener Maschine ausgeführt wurden. Zudem wird der komplette Aufbauprozess dort festgehalten, inklusive der Softwarebestandteile und Dateien, die auf der virtuellen Maschine provisioniert wurden.

8.6 Administration

Build up your Server Create Import Administration Programs

Administration

Global - Configuration

App install path

VM install path

Upload-Folder

Save

Ansible - Configuration

Hosts

Abbildung 8.6: Administration des VM-Builders (Teil 1)

Sudo

Save

Vagrant - Configuration

User

Password

Save

Logfile - Configuration

General

Machines

Save

Abbildung 8.7: Administration des VM-Builders (Teil 2)

Die Administration des VM-Builders erfolgt über ein eigenes Menü, welches über die Auswahl am oberen Rand ausgewählt werden kann. Das Menü bietet dem Anwender unterschiedlichste Konfigurationsmöglichkeiten an, die die Verwaltung von Dateien, der virtuellen Maschinen, der Logdateien und der integrierten Fremdsoftware ermöglichen. In der Endversion des VM-Builders, sollte der Zugriff allerdings nur einem Administrator möglich sein, weswegen eine Zugriffsverwaltung eine optionale Möglichkeit wäre.

8.7 Software und Packages

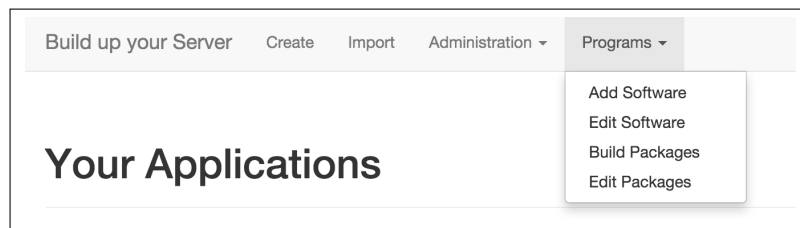


Abbildung 8.8: Administration von Software und Packages

Um eine Softwareauswahl im VM-Builder anzubieten, diese zu administrieren und ggf. Softwarepakete schnüren zu können, werden Optionen angeboten, die diese Tätigkeiten unterstützen. Wie in Abbildung 8.8 gezeigt wird, kann der Anwender über das entsprechende Optionsmenü, Software und Packages hinzufügen und editieren. Jede neu hinzugefügte Softwarekomponente und jedes Package, ist sofort einsatzbereit und wird dem Anwender beim Aufbau einer Maschine angeboten. Während Softwarekomponenten nur aus Linux-Typischen **apt-get install**-Befehlen bestehen, können Softwarepakete (oder Packages genannt) unterschiedliche Konfigurationen beinhalten. Softwarepakete erweitern die Funktionspalette des VM-Builders, indem sie es ermöglichen, mehrere Softwarekomponenten zu vereinen und zusätzliche Dateien mit anzuhängen.

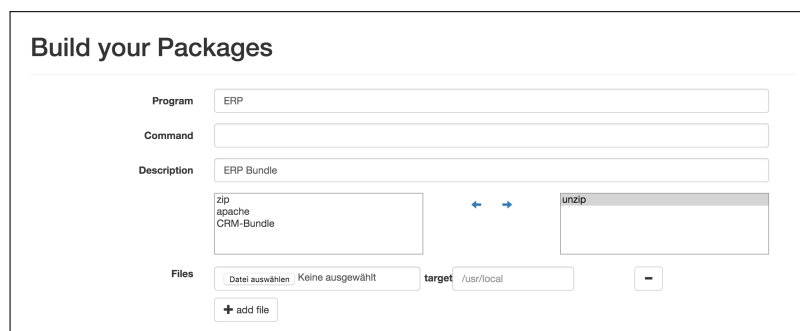


Abbildung 8.9: Erstellung eines Packages

Ein Softwarepackage kann durch das Konfigurationsmenü aus Abbildung 8.9 erstellt werden. Jede ausgewählte Software, wird in der Reihenfolge der Auswahl im späteren Installationsprozess installiert. Zudem können Dateien mitgegeben und in den gewünschten Pfad des Zielhosts kopiert werden. Die Erweiterung der Packageskonfiguration wäre ein wichtiger Punkt für die zukünftige Anwendung der Applikation. Denkbare Erweiterungen wären die Integration von Schnittstellen, die im System gebaut werden könnten und eine Auswahl an Dateioptionen. Die

Erstellung von Softwarepaketen könnte so zukünftig dabei helfen, Komplettinstallationen von Teilsystemen zu übernehmen und erst zu ermöglichen.

Abbildungsverzeichnis

2.1	Betriebssystemvirtualisierung	6
2.2	Hypervisor Typ 1 und 2	7
3.1	Business-Use-Case Übersicht	14
3.2	Entwurf der Verwaltungsoberfläche	21
3.3	Entwurf der Aufbauoberfläche - Eingabe Parameter	22
3.4	Entwurf der Aufbauoberfläche - Auswahl der Softwarekomponenten	22
3.5	Entwurf der Sharing-Oberfläche	23
3.6	Entwurf der Export-Oberfläche	24
3.7	Entwurf der Import-Oberfläche	24
3.8	Entwurf der Einstellungen - Generelle Konfiguration	25
3.9	Entwurf der Einstellungen - Logdatei	25
3.10	Entwurf der Einstellungen - Softwarekomponenten hinzufügen	26
5.1	Vier Arten von Sichten	38
5.2	Kontextsicht	39
5.3	Verteilungssicht des VM-Builders	41
5.4	Client-Server-Anordnungen	43
5.5	Model-View-Controller	45
5.6	Model-View-Controller im 3-Schichten-Modell	46
5.7	Bausteinsicht Level 1	49
5.8	Benutzerschnittstelle	50
5.9	Komponentensicht Distributor	50
5.10	Komponentensicht VMBuilder	51
5.11	Komponentensicht BuildingProcess	52
5.12	Komponentensicht MachineOption	53
5.13	Komponentensicht Administration	53
5.14	Komponentensicht SoftwareSelection	54
5.15	Ansicht der Verarbeitungsebene	55

5.16	Komponentenansicht MachineConstruction	56
5.17	Komponentenansicht SystemAdministration	57
5.18	Komponentenansicht MachineManagement	58
5.19	Ansicht der Datenebene	59
5.20	Laufzeitsicht eines VM-Aufbaus	60
5.21	Entity-Relationship-Model	62
5.22	RelationalDatabaseModel	64
6.1	Bausteinsicht Level 1	74
8.1	Menü zum Erstellen einer virtuellen Umgebung	84
8.2	Übersicht aller aktuellen virtuellen Maschinen	85
8.3	Externe Zugriffsoptionen auf eine virtuelle Maschine	86
8.4	Import durch Konfigurationsdateien	86
8.5	Allgemeine Logdatei	87
8.6	Administration des VM-Builders (Teil 1)	88
8.7	Administration des VM-Builders (Teil 2)	88
8.8	Administration von Software und Packages	89
8.9	Erstellung eines Packages	89

Listings

2.1	Beispiel Inventory-Datei ¹	9
5.1	Routen in Sinatra ²	47
6.1	ORM Framework Datamapper ³	71
6.2	ORM Framework command ⁴	71
6.3	Exemplarische Ordnerstruktur VM-Builder ⁵	72
6.4	Controller Beispiel ⁶	73
6.5	Name-Value-Pair Table ⁷	75
6.6	Single-Row Table ⁸	75
6.7	Beispiel einer Transaktion - Speichern eines Packages ⁹	78
6.8	Inhalt eines Export ¹⁰	80

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 Multicoreprozessoren, 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
- [Christian Baun 2011] CHRISTIAN BAUN, Viktor M.: *Cluster-, Grid- und Cloud-Computing*. 2011. – URL http://www.baun-vorlesungen.appspot.com/CGC11/Skript/fohlen_cgc_vorlesung_13_SS2011.pdf
- [Fleischmann 2012] FLEISCHMANN, Andreas: *Konfigurationsmanagement mit Ansible*. April 2012. – URL <http://www.s-inn.de/blog/image.axd?picture=2012%2F5%2FHypervisor-Type-1-vs-Type-2.png>
- [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](#), 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
- [Roland Kluge 2013] ROLAND KLUGE, SOPHIST G.: *Schablonen für alle Fälle*. 2013. – URL https://www.sophist.de/fileadmin/SOPHIST/Publikationen/Broschueren/SOPHIST_Broschuere_MASTeR.pdf
- [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
- [Scherf 2015] SCHERF, Thorsten: *Konfigurationsmanagement mit Ansible*. 2015. – URL <http://www.admin-magazin.de/Online-Artikel/Konfigurationsmanagement-mit-Ansible>
- [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>
- [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
- [Vagrant 2015] VAGRANT: *CREATING A BASE BOX*. 2015. – URL <https://docs.vagrantup.com/v2/virtualbox/boxes.html>

- [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

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

Hamburg, 28. Oktober 2015 Jan Lepel