Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Lepel

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

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Lepel

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

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

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

Eingereicht am: 1. Januar 2015

Jan Lepel

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

Stichworte
Ad hoc Umgebung, automatisierter Umgebungsaufbau und Provisionierung

Kurzzusammenfassung
Dieses Dokument …
Jan Lepel

Title of the paper
TODO

Keywords
Keywords, Keywords1

Abstract
This document …

Listings

Inhaltsverzeichnis

1 Einleitung

> "Es ist nicht zu wenig Zeit, die wir haben, sondern es ist zu viel Zeit, die wir nicht nutzen" - Lucius Annaeus Seneca, Seneca (2005)

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

1.1 Motivation

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

1.2 Zielsetzung

Das Ziel der vorliegenden Arbeit ist es, durch inkrementelles und interatives Vorgehen eine Applikation zu modellieren, die den Anwender der Applikation bei dem Aufbau von virtuellen Umgebungen unterstützt. Je nach Wunsch des Anwenders, wird nicht nur der Aufbau einer Umgebung vereinfacht, sondern auch die direkte Installation von Programmen veranlasst. Eine Weboberfläche soll die entsprechenden Optionen zur Verfügung stellen und dem Anwender durch seine ausgewählte Funktion leiten. Große Konfigurationen oder komplizierte Einstellungen sollen dem Normalanwender abgenommen werden und geschehen im Hintergrund. Damit auch ein Sichern oder ein Zurückspielen von vorhandenen virtuellen Maschinen möglich wird,

sollten Im- und Exportfunktionen dies untzerstützen. Die Realisierung sollte auf einem zentralen Knotenpunkt stattfinden, um es mehreren Anwendern zu ermöglichen, ihre notwendige Maschine zu erstellen und zu verwalten. Kernaufgaben sollen Open-Source Anwendungen übernehmen, die in ihrem Bereich etabliert sind. Ebenfalls im Fokus steht die Leichtigkeit der Konfiguration der auszuwählenden Open-Source Anwendung. Bei der Erstellung der einzelnen Softwarekomponenten ist stehts auf das Prinzip von hoher Kohäsion und loser Kopplung zu achten. Also dem Grad der Abhängigkeit zwischen mehrere Hard-/ und Softwarekomponenten, der Änderungen an einzelnen Komponenten erleichtert. Um auch die Anwendungsoberfläche unkompliziert zu halten, soll der Anwender mit ein paar Klicks zu seinem Ziel geführt werden. Durch das Vermeiden von unnötigen Verschachtelungen oder einer Flut an Optionen und Konfigurationen, soll der Anwender in der Applikation einen Helfer für seine Tätigkeit finden.

1.3 Themenabgrenzung

Diese Arbeit greift bekannte und etablierte Softwareprodukte auf und nutzt diese in einem zusammenhängenden Kontext. Dabei werden die verwendeten Softwareprodukte nicht modifiziert, sondern für eine vereinfachte Benutzung durch eigene Implementierungen kombiniert und mit einem Benutzerinterface versehen, welches die Abläufe visualisiert und dem Benutzer die Handhabung vereinfacht. Die vorzunehmenden Implementierungen greifen nicht in den Ablauf der jeweiligen Software ein, sondern vereinfacht das Zusammenspiel der einzelnen Anwendungen.

1.4 Struktur der Arbeit

[...]

2 Grundlagen

[...]

2.1 Grundlagen der Virtualisierung

Für den Begriff Virtualisierung existiert keine allgemeingültige Definition. In der Regel wird damit der parallele Einsatz mehrerer Betriebssysteme beschrieben oder detailierter, das Ressourcen zu einer logischen Schicht zusammengefasst werden und dadurch deren Auslastung optimiert wird. So kann die logische Schicht bei Aufforderung ihre Ressourcen automatisch zur Verfügung stellen. Das Prinzip dahinter ist die Verknüpfung von Servern, Speichern und Netzen zu einem virtuellen Gesamtsystem. Daraus können sich darüber liegende Anwendungen direkt und bedarfsgerecht ihre Ressourcen beziehen.
Grundsätzlich unterscheidet man zwischen

1. **Virtualisierung von Hardware**
   die sich mit der Verwaltung von Hardware-Ressourcen beschäftigt und

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

[Bengel (2008)]

2.1.1 Virtuelle Maschine

Eine virtuelle Maschine ist nach Robert P. Goldberg

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

Wörtlich übersetzt ist also eine virtuelle Maschine ein Hardware-/Software-Duplikat eines real existierenden Computersystems, in dem eine statistisch dominante Untermenge an virtuellen

Prozessoren, Befehle im Benutzer-Modus auf dem Host-Prozessor ausführen. Dieses Duplikat kann als Software-Container betrachtet werden, der einen vollständigen Satz an emulierten Hardwareressourcen, dem Gastbetriebssystem und entsprechenden Anwendungen besteht. Durch die Containerstruktur wird eine Kapselung hervorgerufen, die es ermöglicht mehrere virtuelle Maschinen komplett unabhängig auf einem Hostsystem zu installieren und laufen zu lassen. Ist eine virtuelle Maschine fehlerhaft oder nicht mehr erreichbar, betrifft dies nicht automatisch die restlichen parallel laufenden Maschinen und stellt damit einen der charakteristischen Vorteile von virtuellen Maschinen dar. Die Verfügbarkeit. Backup-Systeme oder mehrere Instanzen einer Applikationen können so unabhängig auf einem Host ausgeführt werden, ohne sich gegegnseitig zu beeinflussen. Durch den struktuellen Aufbau einer virtuellen Maschine, ist es ebenfalls möglich, eine Maschine nach den eigenen Wünschen zu erstellen und diese im Weiteren zu replizieren.
Virtuelle Maschinen können ohne größeren Aufwand verschoben, kopiert und zwischen Hostservern neu zugeteilt werden, um die Hardware-Ressourcen-Auslastung zu optimieren. Administratoren können auch die Vorteile von virtuellen Umgebungen für einfache Backups, Disaster Recovery, neue Deployments und grundlegenden Aufgaben der Systemadministration nutzen, da das Wiederherstellen aus Speicherpunkten oder gespeicherten Abzügen, innerhalbt von Minuten zu realisieren ist.

2.1.2 Gastbetriebssystem

Ein übliches Betriebssystem wird im privilegierten Prozessor-Modus ausgeführt (Auch Kernel-Mode genannt). Dies befähigt es, die absolute Kontrolle über die vorhandenen Ressourcen zu gewinnen. Alle Anwendungen, die auf dem Betriebssystem laufen, werden im sogenannten Benutzer-Modus ausgeführt. Die Privilegien im Benutzer-Modus sind allerdings relativ eingeschränkt. Ein direkter Zugriff wird nur sehr selten und unter genau kontrollierten Bedingungen gestattet. Dies hat den Vorteil, dass kein Programm z. B. durch einen Fehler das System zum Absturz bringen kann.
Eine virtuelle Maschine (siehe 2.1.1) läuft als normaler Benutzer-Prozess im Benutzer-Modus, was zur Folge hat, dass das dort installierte Betriebssystem, das Gastbetriebssystem, folglich nicht den privilegierten Prozessor-Modus nutzen kann, wie es ein nicht virtualisiertes Betriebssystem könnte. Da weder die Anwendungen noch das entsprechende Gastbetriebssystem Kentniss dadrüber haben, dass sie in einer virtuellen Maschine laufen, müssen ausgeführte Instruktionen, die den Prozessor-Modus erfordern, entsprechend anders gehandhabt werden. Dies ist unter anderem die Aufgabe des Hypervisors (siehe 2.1.3).

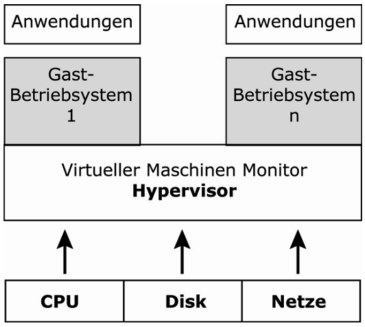


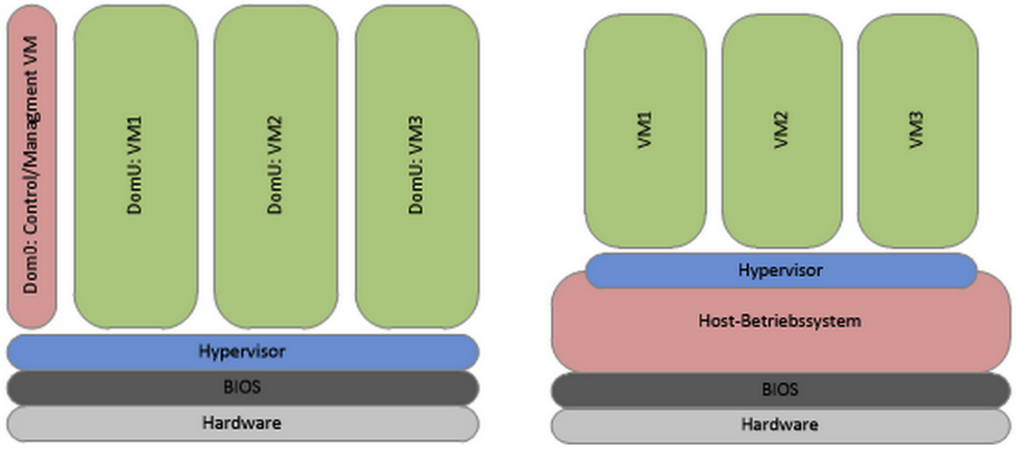
Abbildung 2.1: Betriebssystemvirtualisierung [Siegert und Baumgarten (2006)]

2.1.3 Hypervisor

Der Name des Hypervisors kann von Hersteller zu Hersteller variieren. Bei Microsoft z.B. wird er Hyper-V genannt und bei VMware als ESXi bezeichnet. Der Hypervisor, oder in der Literatur auch VMM (Virtual Machine Monitor) genannt, ist die sogenannte abstrahierende Schicht zwischen der tatsächlich vorhanden Hardware und den ggf. mehrfach existierenden Betriebssystemen. Siehe 2.1. Seine primäre Aufgabe ist die Verwaltung der Host-Ressourcen und deren Zuteilung bei Anfragen der Gastsysteme. Lösen Instruktionen, oder Anfagen eines Gastbetriebssystems eine CPU-Exception aus, weil diese im Benutzer-Modus ausgeführt werden, fängt der Hypervisor diese auf und emmuliert die Ausführung der Instruktionen (trap and emulate). Die Ressourcen des Hostsystems werden derart verwaltet, dass diese bedarfsgerecht zur Verfügung stehen, egal ob ein oder mehrere Gastsysteme laufen. Zudem zählt unter anderem E/A-Zugriffe (insbesondere Hardwarezugriffe), Speichermanagement, Prozesswechsel und System-Aufrufe.
Den Hypervisor kann man in zwei verschiedene Typen kategorisiert.

Typ 1 Hypervisor

arbeitet direkt auf der Hardware und benötigt somit kein Betriebssystem, welches zwischen ihm und der Hardware liegt. Alle darüber liegenden virtuelle 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 priviligierte Domain geregelt, die in der Dom0 läuft. Dadurch hat die priviligierte Domain die Möglichkeit andere Domains zu starten, stoppen und zu verwalten.
Der Hypervisor Type-1 verfügt selbst über die nötigen Gerätetreiber, um den virtuellen Maschinen CPU, Speicher und I/O zur Verfügung zu stellen. Durch die wegfallende



(a) Bezeichnung der linken Grafik (b) Bezeichnung der rechten Grafik

Abbildung 2.2: Hypervisor Typ 1 und 2

Schicht, das nicht benötigte Betriebssystem, gewinnt der Hypervisor Typ 1 an Performance und spart am Ressourcenverbrauch. Siehe Abbildung [2.1.3.a].

Typ 2 Hypervisor

lässt durch seine Bezeichnung als 'Hosted' erahnen, dass der Unterschied zu Typ 1 darin besteht, dass er auf einem Hostsystem aufsetzt. Also eine Schicht implementiert sein muss, die zwischen dem Hypervisor und der Hardware liegt. Siehe Abbildung [2.1.3.b]. Diese Schicht wird durch ein Betriebssystem realisiert, das dem Hypervisor den Zugang zur Hardware durch die eigenen Hardwaretreiber ermöglicht. Ist ein Betriebssystem mit einer Hardware kompatibel, ist transitiv gesehen, der Hypervisor ebenfalls mit installier- und ausführbar. Dies vereinfacht die Installation gegenüber dem Hypervisor Typ 1.
Aus Implementierungssicht gibt es für beide Hypervisoren Vor- und Nachteile. Für einige Bereiche ist die Anforderung eines Betriebssystems nur von Vorteil. Vor allem wenn es um um Hardware- und Treiber-Kompatibilität, Konfigurationsflexibilität und vertraute Management-Tools geht.
Auf der anderen Seite kann genau das zum Nachteil ausgelegt werden. Es entsteht nicht nur ein höherer Management-Aufwand um das Betriebssystem zu konfigurieren und zu verwalten, auch die Performance und der Sicherheitsaspekt leiden unter dieser zusätzlichen Schicht.

2.2 Provisioning/Konfigurationsmanagement

TODO: WELCHER BEGRIFF IST BESSER?

Die Hauptaufgabe eines Konfigurationsmanagement-Systems, im folgenden nur noch KMS genannt, ist es, eine zuvor definierte Zustandsbeschreibung eines Hosts umzusetzen. Dies kann das Installieren von Softwarepaketen bedeuten, starten oder beenden von Diensten oder Konfigurationen erstellen/anpassen/entfernen zu lassen. In der Regel verwenden KMS eigene Agenten auf den Zielsystemen, über die die Kommunikation läuft und die Zustandsbeschreibung realisiert wird. Neuere Anwendungen wie 'Ansible' aus Kapitel 4.2.1, die Konfigurationsmanagement unterstützen, benötigen diese Agenten nicht mehr und realisieren die Kommunikation über eine SSH-Schnittstelle. Pull-basierte Tools, wie beispielsweise 'Puppet', fragen in regelmässigen Abständen ein zentrales Konfigurations-Repository ab, in dem die jeweils aktuelle Zustandsbeschreibung der Maschine gespeichert ist und sorgt dafür, dass die Änderungen auf dem Client ausgeführt werden. Es spielt keine Rolle, ob der Zielclient eine virtuelle Maschine ist oder eine standard Maschine ist. KMS sind in der Regel dazu fähig ganze Gruppen an Rechner parallel zu bearbeiten und die entsprechenden Zustandsbeschreibungen umzusetzen. Bei dem im oberen Abschnitts bereits genannten Beispiel 'Ansible', können mehrere Rechner simpel in Inventory-Dateien als Gruppen zusammengefasst werden, die dann jeweils durch 'Ansible' angesprochen werden können um entsprechende Stände an Zustandsbeschreibungen dort auszuliefern. Siehe 2.1

Listing 2.1: Beispiel Inventory-Datei

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

3 Anforderungsanalyse

Die Anforderungsanaylse 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.
Um die aus dieser Anforderungsbeschreibung hervorgehenden Kernfunktionalitäten und Qualitätsmerkmale näher zu betrachten, wird als erstes das Ziel der Anwendung beschrieben, dann die Stakeholder definiert und ihnen im Anschluss die Kernfunktionalitäten zugeordnet. Im weiteren wird auf die Qualitätsmerkmale eingegangen, die auch als nichtfunktionale Anforderungen bezeichnet werden und als Qualitätskriterium an System und Software angesehen werden können.

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 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, sollen dem Anwender in seiner Tätigkeit unterstützen. Dabei spielt das User-Interface und der Funktionsumfang der Applikation eine entscheidene Rolle. Während der Aufbau des User-Interface hilft sich mit geringem Zeitaufwand in die Applikation einzuarbeiten, vereinfacht ein übersichtliches Konfigurationsspektrum die Erstellung der gewünschte virtuelle Umgebung. Die Konfiguration einer virtuellen Maschine muss auch für unerfahrene Nutzer möglich sein und keine speziellen Kentnisse vorraussetzen. Alle virtuelle Maschinen, die zu einem Zeitpunkt aktiv sind, sollten in getrennten Instanzen laufen und von einander

unterscheidbar sein. Die Unterscheidbarkeit soll Funktionen wie den Export oder das Teilen einer Maschine mit einem Kollegen unterstützen und dem Anwender die Möglichkeit schaffen, die gewünschte virtuelle Maschine zu beeinflussen, in dem er die Umgebung abschalten oder zerstören kann.

3.2 Stakeholder

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

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

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

Abbildung 3.1: Stakeholder

3.3 Funktionale Anforderungen

Anforderungen Anwender

FA 1. Die Anwendung muss über den Browser ausführbar sein zu, ohne zusätzliche lokale Installationen auf Anwenderseite.

FA 2. Falls der Anwender keine zusätzliche Software auf der virtuellen Maschine haben möchte, muss die Anwendung eine entsprechende Option dafür anbieten.

FA 3. Die Anwendung muss dem Anwender die Möglichkeit bieten, Software mit zu installieren.

FA 4. Falls der Anwender diese Option nutzt, sollte die Anwendung Softwarekomponenten vorschlagen oder es ermöglichen eigene Dateien auszuwählen.

FA 5. Die Anwendung sollte fähig sein, dem Administrator Bearbeitungsmöglichkeiten für die vorzuschlagenden Softwarekomponenten anzubieten.

FA 6. Die Anwendung sollte dem Anwender die Option bieten, virtuelle Maschinen zu exportieren.

FA 7. Die Anwendung sollte fähig sein den Export zu komprimieren.

FA 8. Ist der Export durchgeführt worden, muss die Anwendung das mit einer Meldung auf dem Bildschirm bestätigen.

FA 9. Die Anwendung muss fähig sein, Exporte wieder importieren zu können. Falls währen der Importierung Datenfehler auftreten, muss die Anwendung den jeweiligen Fehler (Fehlerbeschreibung) auf dem Bildschirm ausgeben.

FA 10. Ist der Import erfolgreich durchgeführt worden, sollte die Anwendung eine entsprechende Meldung anzeigen.

FA 11. Wenn der Anwender eine virtuelle Maschine erstellen möchte, muss die Anwendung bei wichtigen Konfigurationsschritten, für den Benutzer sichtbare Statusmeldungen anzeigen.

FA 12. Treten Fehler bei der Erstellung einer virtuellen Maschine auf, muss das System eine Fehlermeldung ausgeben.

FA 13. Die Applikation sollte fähig sein, anderen Anwendern bereits erstellte Maschinen über das Internet und das lokale Netzwerk zur Verfügung zu stellen.

FA 14. Möchte der Anwender eine bereits erstellte und laufende virtuelle Maschine beenden, muss die Anwendung dafür eine entsprechende Option bieten.

FA 15. Falls der Anwender eine bereits erstellte virtuelle Maschine löschen möchte, muss die Anwendung ihm dafür ein Hilfsmittel bereitstellen.

Anforderungen Administrator

FA 16. Falls während des Betriebes der Anwendung Änderungen an der Konfiguration durchgeführt werden müssen, sollte die Anwendung dies nur durch einen expliziten Administratoren-Account zulassen.

FA 17. Solange der Administrator angemeldet ist, sollte die Anwendung ihm die Möglichkeit bieten, den Speicherort und Name von Logdateien persistent zu ändern.

FA 18. Falls während der Änderung ein Fehler auftritt, muss die Anwendung eine Fehlermeldung auf dem Bildschirm ausgeben.

FA 19. Die Anwendung sollte dem Administrator die Option bieten, sich den Inhalt von Logdateien anzeigen zu lassen.

Definition Funktionale Anforderungen nach Rupp und die SOPHISTen (2014)

3.4 Use-Cases

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

1. Erstellung einer virtuellen Maschine
2. Export einer vorhandenen Maschine
3. Der Import einer zuvor erstellten Maschine
4. Eine virtuelle Maschine zugreiffbar für andere Anwender machen (teilen)

3.4.1 Business-Use-Case

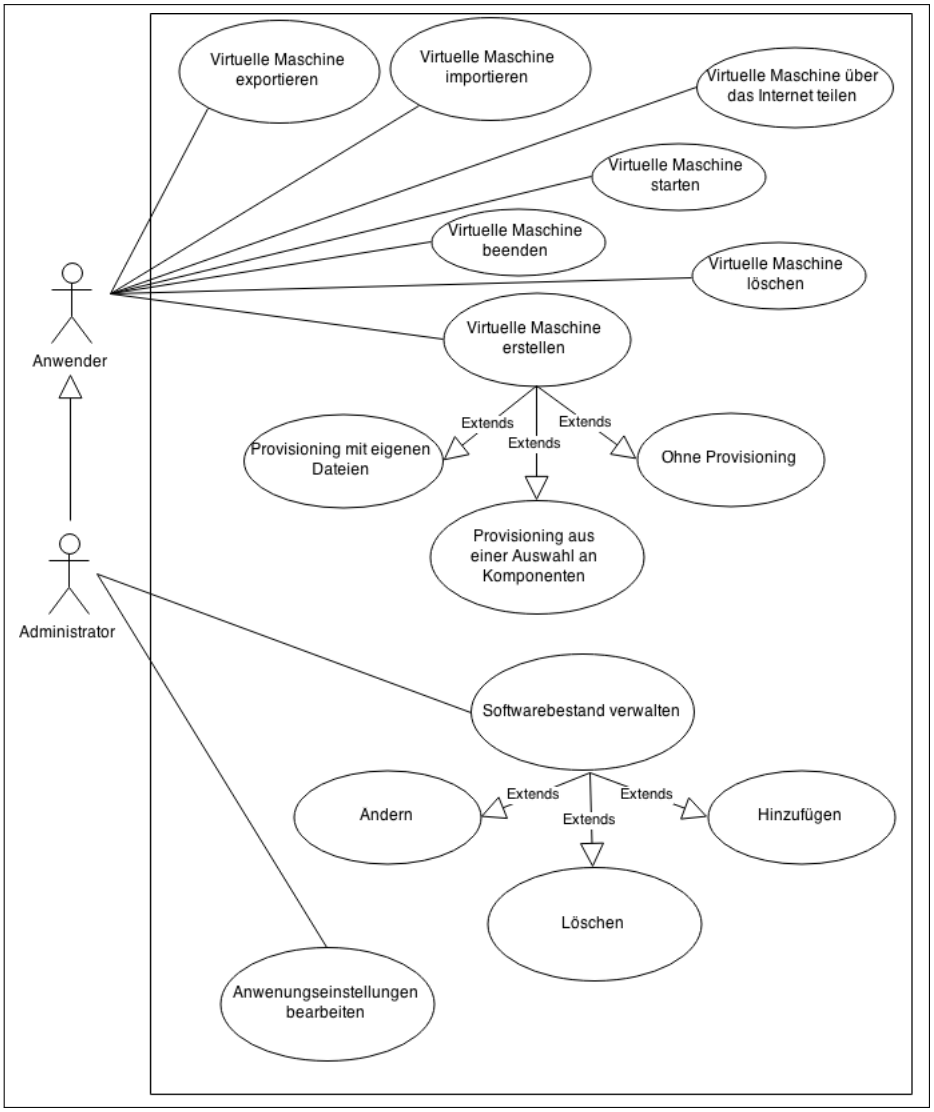


Abbildung 3.2: Titel

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 erhält eine gepackte Datei, in der alle nötigen Daten enthalten sind, die für einen erneuten Import nötig wären.
Anforderungen	FA 1, FA 3, FA 4, FA 11, FA 12, FA 13

Erfolgsszenario

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wird gebeten persönliche Konfigurationsparameter für die zu erstellende Maschine einzugeben
3. Die Applikation schlägt dem Anwender vor, Software mit auf die gewünschte Maschine zu installieren
4. Nach der entsprechenden Auswahl zeigt die Applikation den aktuellen Aufbaustatus der virtuellen Maschine
5. 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 1, FA 6, FA 7, FA 8, FA 13

Erfolgsszenario

1. Der Anwender startet die Applikation über den Browser
2. Der Anwender wählt die Exportfunktion aus und die Applikation zeigt dem Anwender die verfügbaren Maschinen an
3. Der Anwender sucht sich die entsprechende Maschine aus und mit einem weiteren Klick wird der Download mit den benötigten Dateien gestartet

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 1, FA 9, FA 10, FA 11, FA 12

Erfolgsszenario

1. Der Anwender startet die Applikation 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 erstelle 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 13
Erfolgsszenario	

1. Der Anwender startet die Applikation 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

3.5 Nichtfunktionale Anforderungen

In der Literatur findet sich keine einheitliche Definition von nichtfunktionalen Anforderungen, aber bezogen auf das Design eines Systems sind die nichtfunktionalen Anforderungen für den Architekten von besonderer Bedeutung Chung u. a. (1999).
Durch nichtfunktionale Anforderungen können neue Lösungsmöglichkeiten vorgegeben werden oder schlicht die Menge an potentiellen Designentwürfen, im Bezug auf die Funktionalitäten, reduziert werden Burge und Brown (2002).
Im Wesentlichen gibt es eine begrenze Auswahl an Definitionen und Perspektiven die im folgenden nach Rupp und die SOPHISTen (2014) zusammengefasst werden.

1. **Technologische Anforderungen**
   Die detailiertere 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 können wiederrum in detailiertere Unterpunkte unterteilt werden. Dies kann nach zwei Standards erfolgen: ISO 25000 und ISO 9126. Mittlerweile ist jedoch der ISO 9126 Standard in ISO 25000 aufgegangen. Beide Standards legen allerdings die gleiche Aussage nahe, dass Qualitätsanforderungen die Qualität des Systems und des Entwicklungsprozesses festlegen.

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

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

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

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

Im folgenden werden die nichtfunktionalen Anforderungen hinsichtlich der Punkte 1) 'Technologische Anforderungen' und 3) 'Anforderungen an die Benutzeroberfläche' betrachtet, da diese Zielführend.....

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, ist maximal bei 10.
3. Die Kommunikation zwischen Frontend und Backend muss nicht verschlüsselt ablaufen.
4. Softwareupdates der benutzen Softwarekomponenten müssen mit Rücksprache des Entwicklers erfolgen.

Qualitätsanforderungen

1. Die Installation und Betriebnahme 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 nachvollziebar.
4. Änderungen am vorgeschlagenen Softwarebestand müssen innerhalb von <10 Sekunden in der Anwendungsoberfläche sichtbar sein. (bezogen auf Funktionale Anforderungen FA 4.)
5. Falls das Betriebssystem auf eine höhere Version aktualisiert werden soll, muss dies ohne Änderungen am Quellcodes der Anwendung vorgenommen werden können.
6. Soll ein anderer Provisionierer verwendet werden, muss der Aufwand des Austausches bei unter einem Personentag liegen.
7. Wird angedacht weitere Grundfunktionalitäten zu implementieren, soll dies möglichst einfach erfolgen.
8. Das Importieren von virtuellen Maschinen sollte <10 Minuten betragen.
9. Die Validierung der zu importierenden Daten sollte <2 Minute betragen.
10. Der bei Import verwendete Validierungsalgorithmus muss unter 0.5 Personentagen austauschbar sein. (bezogen auf Funktionale Anforderungen FA 9.)
11. Der Export von einer virtuellen Maschine sollte <5 Minuten betragen.

Anforderungen an die Benutzeroberfläche

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

2. Die Anwendung muss den Oberflächendialog ’Virtuelle Maschine exportieren"mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ... . Keine umzusetzende Anforderung sind die genau Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

3. Die Anwendung muss den Oberflächendialog ’Virtuelle Maschine importieren"mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ... . Keine umzusetzende Anforderung sind die genau Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

4. Die Anwendung muss den Oberflächendialog ’Virtuelle Maschine erstellen - Mit Provisioning aus einer Auswahl an Komponenten"mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ... . Keine umzusetzende Anforderung sind die genau Größe und die Anordnung der einzelnen Elemente.
   BILD EINFÜGEN

5. Die Anwendung muss den Oberflächendialog ’Virtuelle Maschine über das Internet teilen"mit den folgenden Bezeichnungen und der Art der abgebildeten Elemente darstellen (siehe Abbildung ... . Keine umzusetzende Anforderung sind die genau Größe und die Anordnung der einzelnen Elemente.

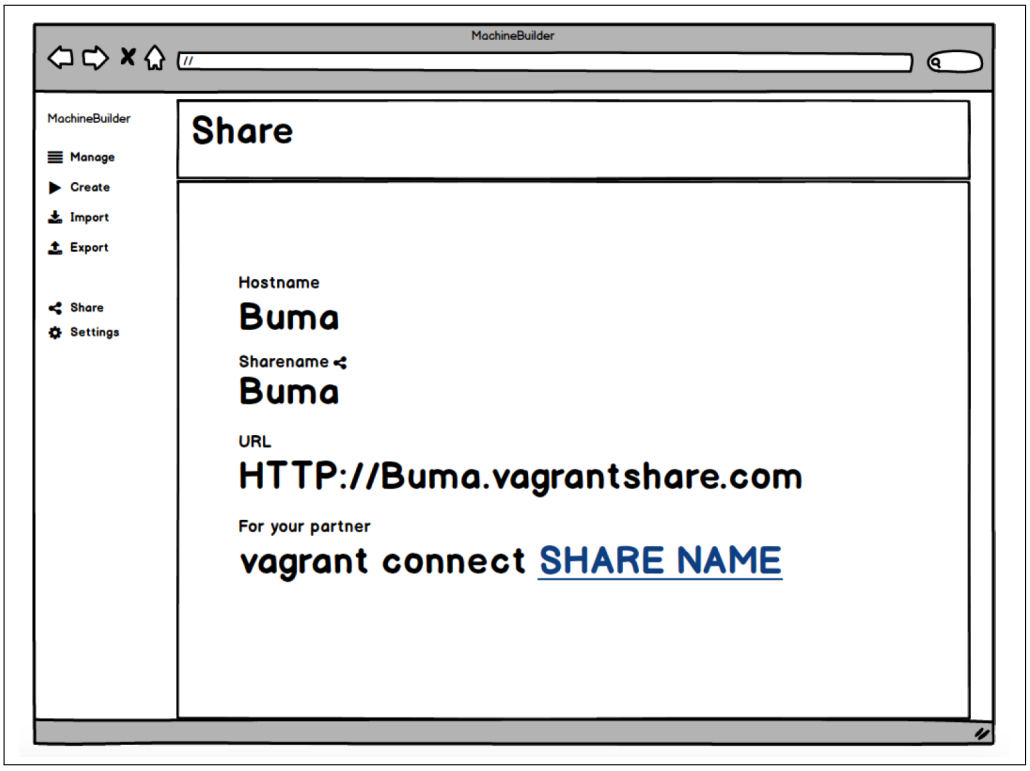


Abbildung 3.3: Titel

3.6 Randbedingungen

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

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

Da die organisatorischen-, sowie politischen Faktoren auf dieses Projekt keinen Einfluss haben, werden diese nicht weiter betrachtet.

3.6.1 Technische Randbedingungen

TODO: 64 Bit OS müssen möglich sein!

Randbedingung	Erkläuterung
Server Hardware	Die Leistung des Servers sollte entsprechend der Anforderungen genügen. Es muss möglich sein, mehrere virtuelle Maschinen gleichzeitig laufen zu lassen, ohne das es die einzelnen Maschinen beeinflusst.
Server Betriebssystem	Die Lösung sollte auf einer einem Ubuntu Server 64Bit installiert und betrieben werden.
Implementierung in Ruby	Implementiert der Anwendung erfolgt in Ruby mit dem Framework Sinatra. Da keine seperate 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 drauf 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 für ausschließlich für Mozilla Firefox.

Abbildung 3.4: Titel

3.7 Zusammenfassung

4 Evaluation

Um herauszufinden, ob und welche Softwareprodukte der Anwendung hinzugefügt werden können, muss eine Evaluation der vorhanden Produkte, die auf dem Markt sind durchgeführt werden. Es ist zu eruieren, ob gewünschte Funktionalitäten unter Verwendung der technischen Randbedingungen aus Kapitel 3.6.1 - Unterpunkt 'Femdsoftware' von externen Softwarekomponenten unterstützt oder sogar durchgeführt werden können, die etabliert und kostenlos sind.

4.1 Virtualisierungsprodukte

Virtualisierungsprodukte oder Tools helfen virtuelle Maschinen für temporäre zwecke aufzubauen. Beispielsweise um ein alternatives Betriebssystem zu testen, Softwareprodukte auszuprobieren oder Entwicklungsumgebungen zu erstellen. Die wohl derzeit (Stand Mitte 2015) bekanntesten und etabliertesten Virtualisierungslösungen werden von von drei verschiedenen Herstellern repräsentiert und zum Teil als Freeware angeboten.

1. **VMware Player / Plus**
   Preiswerte aber dennoch kostenpflichtige Virtualisierungslösung für Unternehmen. Source-Code.

2. **Microsoft Hyper-V**
   Microsofts Virtualisierungslösung, die nur in Windows 8 oder höher lauffähig ist.

3. **Oracle VM VirtualBox**
   Gratis Open-Source-Tool von Oracle mit großem Funktionsumfang und frei verfügbar.

4.1.1 VMware Player (Plus)

Eins der bekanntesten Unternehmen für Virtualisierungslösungen ist VMware. VMware bietet für den Privatanwender den VMware Player an, der das kostenlose Pendant zur professionellen Lösung VMware Workstation / VMware Fusion darstellt. Mittlerweile ist der VMware Player in zwei Teile aufgeteilt worden und ist für Unternehmen unter dem dem Namen VMware Player

Plus, für den Preis von 90 Euro zu erwerben. Der VMware Player (Plus) unterstützt ca. 200 Gast-Betriebssysteme, egal ob 32Bit oder 64Bit und lässt sich auf Windows und verschiedenen Linux-Distributionen installieren. Virtuelle Maschinen lassen sich nur als VMDK-Format (Virtual Machine Disk) speichern, was die Wahl der Dateiformate enorm einschränkt. Ausserdem sind Snapshots möglich und versteht sich damit nicht auf das praktische Feature, eine virtuelle Maschine in einen vorherigen Systemzustand zurückzuversetzen. Genauso fehlt die Funktion des Klonens, also des Duplizierens einer Maschine gänzlich. Funktionen wie der Export und Import werden allerdings von dem VMware Player unterstützt.

4.1.2 Microsoft Hyper-V

Microsoft hat in Windows 8 und höhere Hyper-V integriert, das über Windowsfunktionen einfach nachinstalliert werden kann. Allerdings müssen gewissen Faktoren gegen sein, damit dies möglich ist. Windows 8 muss als Professional Version vorliegen und als 64 Bit Version. Hyper-V hat einen großen Vorteil gegenüber der Konkurrenz, denn es können mehrere virtuelle Maschinen gleichzeitig nebeneinander laufen gelassen werden, ohne Performance Probleme. Dies erreicht Hyper-V dank SLAT, einer Technik zur dynamischen RAM Verwaltung.**[TODO: Weitere Infos suchen in der BA]**

4.1.3 Oracle VM VirtualBox

Die im April 2005 entstandene Virtualisierungssoftware von Oracle (erst InnoTek Systemberatung GmbH) eignet sich für Windows, Linux, Mac OS X, FreeBSD und Solaris als Hostsystem. Als Gastsysteme wird eine Vielzahl an x64- als auch x86-Betriebssysteme unterstützt, wie diverse Linux Distributionen, Windows von 3.11 an, Mac OS X, IBM OS/2 und FreeBSD.
VirtualBox lässt dem Anwender viele Freiheiten im Speichern seiner virtuellen Umgebung. Etwa vier gängige Formate werden angeboten und ermöglichen einen leichteren Austausch unter Anwendern. Ein weiterer Vorteil ist, dass VirtualBox komplett kostenlos und OpenSource ist. Der Quellcode ist für jeden interessierten verfügbar. Der angebotene Funktionsumfang beinhaltet das Importieren sowie Exportieren von Maschinen und das Erstellen von Snapshots. Das Klonen von virtuellen Maschinen ist ebenfalls kein Problem.

4.1.4 Zusammenfassung

Die wichtigsten Faktoren im Bezug auf die zu entwickelnde Anwendung zusammengefasst.

	VMware Player (Plus)	Microsoft Hyper-V	Oracle VM VirtualBox
Host-Betriebssystem	Windows, diverse Linux Distributionen	Windows 8 Pro 64 Bit	Windows, Linux, MacOS X
Gast-Betriebssysteme	Mehr als 200 Gast-Betriebssystemen		Diverse Linux Distributionen, Windows, FreeBSD, Mac OS 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
Preis	90 Euro für Unternehmen	Kostenlos	Kostenlos

4.1.5 Fazit

Auch wenn VMware zu den bekannteren Herstellern gehört und der VMware Player (Plus) eine vielzahl an Funktionen bietet, macht die Unterscheidung zwischen der Unternehmensvariante und der für Privatanwender, die zukünftige Benutzung zu kompliziert. Sollte die zu erstellende Anwendung in einem nicht privaten Umfeld betrieben werden, so muss auf den VMware Player Plus zugegriffen werden, ggf Änderungen an der Implementierung vorgenommen werden und die entsprechenden Lizenskosten beachtet werden. Ausserdem schränkt nicht nur die kleine Auswahl an Dateiformaten für den Export von Maschinen, die Funktionsweise ein, sondern auch das nicht vorhanden sein von Snapshot- und Klon Funktionen. Microsofts Lösung stellt sich von den Grundanforderunger her, als nicht nutzbar heraus, da Hyper-V Windows als Grundlage benötigt und dies nicht den gestellten Anforderungen an die zu erstellenden Software genügt. Oracles VirtualBox vereinigt hingegen viele Aspekte, die für die geforderten Funktionen unterstützend und hilfreich sind. Die Vielzahl an unterstützenden Dateiformaten, kann für die geforderte Exportfunktion interessant sein. Da VirtualBox für Privatanwender

und Firmenkunden kostenlos ist, fällt der Lizensierungsaspekt weg. VirtualBox lässt sich ebenfalls mit den technischen Randbedingungen aus Kapitel 3.6.1 in einklang bringen, da es sich auf diversen Linux Umgebungen installieren lässt. Ggf. könnte der Punkt, dass VirtualBox Open-Source ist, für spätere Weiterentwicklungen interessant sein. VirtualBox könnte durch seinen Funktionsumfang wichtige Funktionen in der angestrebten Applikation übernehmen. Dies hätte den Vorteil, dass die Funktionsbestandteile nicht aufwändig implementiert werden müssten.

4.2 Konfigurationsmanagement-Systeme

Wie in Kapitel 2.2 beschrieben, helfen Konfigurationsmanagement-Systeme bei der Umsetzung von sogenannte Zustandsbeschreibungen eines Hostes. Kurzgefasst, Konfigurationsmanagement-Systeme können gewünschte Software auf Zielrechner(n) installieren, Dienste und Applikationen starten/beenden, Konfigurationen ändern/erstellen/ löschen und wenn gewünscht, dies alles auf einmal auf einem oder mehrere Zielrechnern. Durch die Anwendung von Konfigurationsmanagement-Systeme, 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 Ansible ist es, Konfigurationen so leicht wie möglich durchführen zu können. Es werden weder aufwändigen Deployment-Scripts benötigt, noch komplizierte Syntaxen verwendet. Ansible wird nur auf der Maschine installiert, die die Infrastruktur verwaltet und kann von dort auf die gewünschten Maschinen zugreifen. Die Clients benötigen weder eine lokale Ansible Installation noch andere spezielle Softwarekomponenten. (Hall (2013)) Die Kommunikation zwschen 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 durchzufüren. Windows wird in der aktuellen Version 1.9.1 nicht Unterstützt. Zustandsbeschreibungen werden in YAML Syntax ausgeführt und in sogannte Playbooks geschrieben. Playbooks haben eine simple YAML Struktur und können somit schon vorgefertigt als Template gespeichert und wieder verwendet werden (ScriptRock (2014))

4.2.2 Saltstack

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

4.2.3 Puppet

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

4.2.4 Zusammenfassung

Die wichtigsten Faktoren im Bezug auf die zu entwickelnde Anwendung zusammengefasst.

	Vagrant	Saltstack	Puppet
Host-Betriebssystem	Diverse Linux Distributionen	Diverse Linux Distributionen	Linux Distributionen, Windows
Client-Betriebssysteme	Diverse Linux Distributionen, Windows	Diverse Linux Distributionen, Windows	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

4.2.5 Fazit

Der größte Unterschied in den verglichenen Konfigurationsmanagement-Systeme, besteht im Kontext, in dem man es einsetzen möchte. Alle drei Anwendungen erfüllen die Grundbedingung, auf Linux lauffähig zu sein und diverse Linux-Distributionen als Client bespielen zu können. Wobei Puppet sich eher im größeren Umfeld hervorragend eignet. Wenn es also darum geht, mehrere Clients mit auf einmal zu verwalten und zu konfigurieren. Wie Puppet benötigt auch Staltstack extra Software für die Clients, um die gewünschte Zustandsbeschreibung durchzuführen, was einen gewissen Zeitaufwand und eine Fehlerquelle mehr bedeutet. Ansible ist zwar der unbekanntere Vertreter unter den Konfigurationsmanagement-Systeme, kann aber leicht installiert werden und benötigt keine weitere Installation auf den Clients. Durch das wegfallen der Clientinstallation, wird auf den zu erstellenden Maschinen, nur das Installiert, was gewünscht wird. So kann der gesamte Installationsprozess beschleunigt werden und und minimiert den gesamten Zeitaufwand.

4.3 Vagrant

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

John Bender 2010 ins Leben gerufen wurde. Vagrant ist ein Entwicklungswerkzeug, das als Wrapper zwischen Virtualisierungssoftware wie VirtualBox und Konfigurationsmanagement-System fungiert. Das command-line Interface und die einfache Konfigurationssprache helfen virtuelle Maschinen schnell zu konfigurieren und zu verwalten. Die Konfiguration einer Maschine geschieht über das 'Vagrantfile', in dem Parameter wie IP Adresse konfiguriert und Konfigurationsmanagement-Systeme hinzuschaltet werden können. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, bedeutet das für den Anwender, dass er es einfach mit anderen Kollegen über Versionskontrollen (z.B. Git oder Subversion) teilen kann. Im Punkto Konfigurationsmanagement-Systeme, wird dem Benutzer die Freiheit gegeben, auf eine vielzahl an bekannten Konfigurationsmanagement-Systeme zurückzugreifen. Unter anderem auch Ansible, Salt und Puppet.
Um die Virtualisierung vorzunehmen, greift Vagrant standardmäßig auf VirtualBox zurück. Allerding werden auch Amazon Web Services und VMware Fusion unterstützt. Teamarbeit wird durch eine 'sharing' Option unterstützt, die mit Version 1.5 implementiert wurde. Das Teilen eine Maschine ermöglicht es Teams an gemeinsamen und entfernten Standorten auf die gleiche Maschine zuzugreifen. (Peacock (2013))

5 Softwareentwurf

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

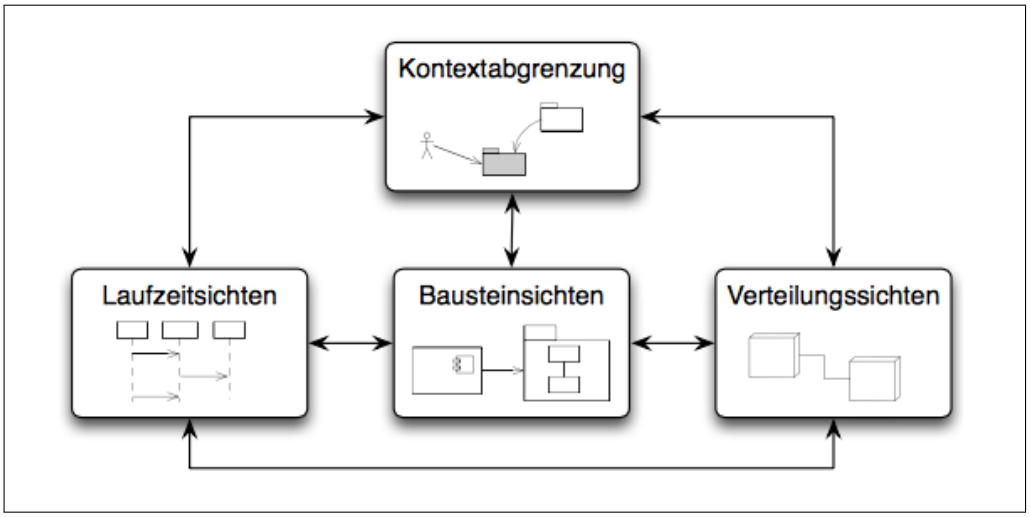


Abbildung 5.1: Vier Arten von Sichten (Starke (2014))

5.1 Kontextabgrenzung

Die Kontextabgrenzung beschreibt die Einbettung des Systems in seine Umgebung sowie die wesentlichen Teile der umgebenden Infrastruktur. Die ermittelten Anforderungen aus Kapitel 3.3 und **??** haben ergeben, dass die Hauptfunktionalitäten aus erstellen, exportieren, importieren, teilen und dem Provisioning von virtuellen Maschinen bestehen. Um dies weiter zu bündeln, können Teile der Hauptfunktionaliäten von bestimmten Produkten übernommen werden. Frei erhältliche Virtualisierungsprodukte können das Erstellen, Exportieren und den Import von virtuellen Maschinen übernehmen. Software-Provisioner sind meist darauf ausgelegt, mit bekannten Virtualisierungslösungen zusammen zu arbeiten und übernehmen somit die gewünschten Anforderung nach automatisierter Softwareinstallation. Der Kern der Anwendung organisiert das zusammenspiel der einzelnen Softwareprodukte und stellt dem Anwender die entsprechende Benutzeroberfläche zur Verfügung. Die Anwendung selbst, inklusive der oben genannten Produkte, laufen auf einem gemeinsamen Server, der zentralisiert angesprochen werden kann.

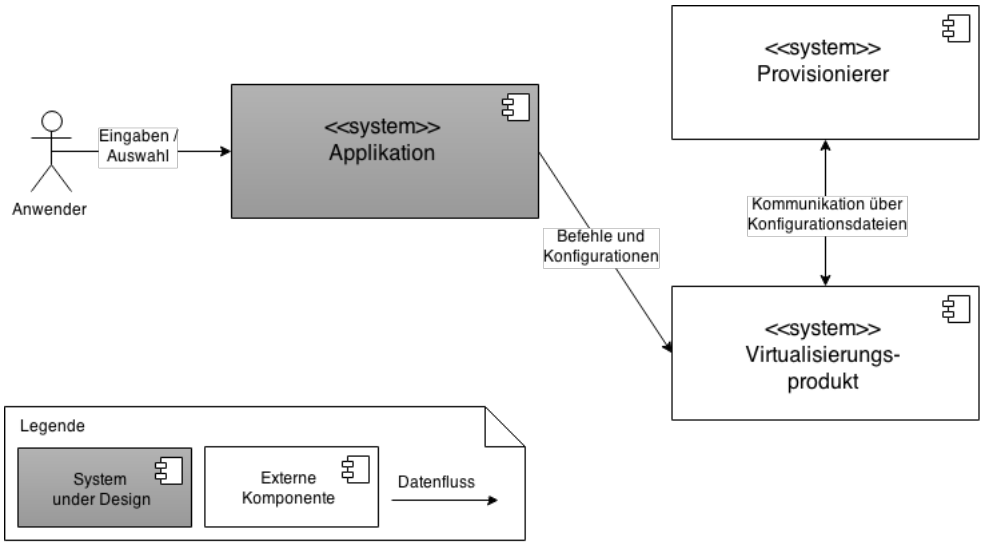


Abbildung 5.2: Bildunterschrift

Kurzbeschreibung der externen Schnittstellen

Eingaben / Auswahl	Der Awender tätigt Eingaben und wählt unter Optionen aus, die von der Anwendung bereitgestellt werden. Diese werden direkt von der Applikation verarbeitet
Befehle und Konfigurationen	Die Applikation erstellt nötige Konfigurationsdateien und leitet Befehle für das Virtualisierungsprodukt weiter
Kommunikation über Konfigurationsdateien	Das Virtualisierungsproduk ruft über die erstellen Konfigurationsdateien auf den Provisionierer zu. **[TODO MUSS NOCH BEARBEITET WERDEN]**.

6 Sichten

6.1 Bausteinsicht

6.2 Laufzeitsicht

6.3 Verteilungssicht

6.4 Zusammenfassung

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] Bengel, 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

[Chung u. a. 1999] Chung, Lawrence ; Nixon, Brian A. ; Yu, Eric ; Mylopoulos, John: *Non-Functional Requirements in Software Engineering (International Series in Software Engineering)*. 2000. Springer, 10 1999. – URL http://amazon.de/o/ASIN/0792386663/. – ISBN 9780792386667

[Hall 2013] Hall, Daniel: *Ansible Configuration Management*. Packt Publishing, 11 2013. – URL http://amazon.com/o/ASIN/1783280816/. – ISBN 9781783280810

[Peacock 2013] Peacock, Michael: *Creating Development Environments with Vagrant*. Packt Publishing, 8 2013. – URL http://amazon.de/o/ASIN/1849519188/. – ISBN 9781849519182

[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

[Rhett 2015] Rhett, Jo: *Learning Puppet 4*. 1. O'Reilly Vlg. Gmbh and Co., 8 2015. – URL http://amazon.de/o/ASIN/1491907665/. – ISBN 9781491907665

[Rupp und die SOPHISTen 2014] Rupp, Chris ; SOPHISTen die: *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil.* 6., aktualisierte und erweiterte Auflage. Carl Hanser Verlag GmbH und Co. KG, 10 2014. – URL http://amazon.de/o/ASIN/3446438939/. – ISBN 9783446438934

[ScriptRock 2014] ScriptRock: *Ansible vs. Salt.* Januar 2014. – URL https://www.scriptrock.com/articles/ansible-vs-salt

[Seneca 2005] Seneca: *Von der Ku"rze des Lebens.* Deutscher Taschenbuch Verlag, 11 2005. – URL http://amazon.de/o/ASIN/342334251X/. – ISBN 9783423342513

[Siegert und Baumgarten 2006] Siegert, Hans-Jürgen ; Baumgarten, Uwe: *Betriebssysteme: Eine Einführung.* überarbeitete, aktualisierte und erweiterte Auflage. Oldenbourg Wissenschaftsverlag, 12 2006. – URL http://amazon.de/o/ASIN/3486582119/. – ISBN 9783486582116

[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

[Zörner 2012] Zörner, Stefan: *Softwarearchitekturen dokumentieren und kommunizieren: Entwürfe, Entscheidungen und Lösungen nachvollziehbar und wirkungsvoll festhalten.* Carl Hanser Verlag GmbH und Co. KG, 5 2012. – URL http://amazon.de/o/ASIN/3446429247/. – ISBN 9783446429246

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

Hamburg, 1. Januar 2015 Jan Lepel