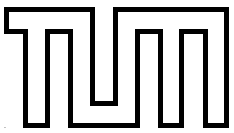


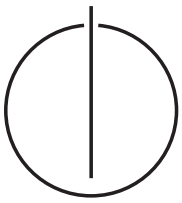
Technische Universität München

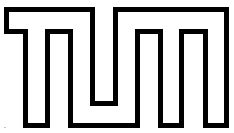
Fakultät für Informatik

Bachelorarbeit in Wirtschaftsinformatik

Überwachung von Softwarekonfigurationsständen in VMware-Clustern

Thomas Bender





Technische Universität München

Fakultät für Informatik

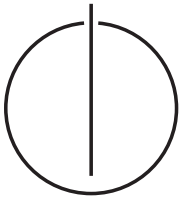
Bachelorarbeit in Wirtschaftsinformatik

Überwachung von Softwarekonfigurationsständen in VMware-Clustern

Monitoring Software Versions in VMware Clusters

Bearbeiter:	Thomas Bender
Aufgabensteller:	Prof. Dr. H.-G. Hegering Prof. Dr. D. Kranzlmüller
Betreuer:	Nils gentschen Felde Silvia Knittl

Abgabedatum: 15. Dezember 2010



Ich versichere, dass ich diese Bachelorarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 15. Dezember 2010

. . . . . .
(Unterschrift des Kandidaten)

Abstract

Virtualisierung ist ein aktuelles Thema in Wissenschaft und Industrie. Aufgrund der Leistungssteigerungen der letzten Jahre u.a. bei Prozessoren, Hauptspeicher und Festspeicher, haben viele Rechnersysteme einen Leistungsüberschuss zu verzeichnen, den man versucht effektiv durch den Einsatz Virtueller Maschinen zu nutzen.

Im Laufe der Zeit sammeln sich durch mittlerweile sehr einfache Deployment-Prozesse eine Vielzahl Virtueller Maschinen an, die jeweils ihren Nutzern und Anwendern zugeordnet sind und sich voll und ganz in deren administrativen Obhut befinden. Diese Situation führt oftmals zu aus Sicht der Software schlecht gepflegten Systemen.

Ziel dieser Arbeit ist, dass eine Datenbank der bestehenden Virtuellen Maschinen mit den jeweils installierten Software-Produkten, ihren Versionen und verfügbaren Updates und Patches automatisiert erstellt wird (Configuration Management Database, CMDB). Als Proof of Concept wird eine CMDB erstellt, die durch automatisierte Prozesse mit Softwareständen aus den Virtuellen Maschinen angereichert wird und dem Nutzer eine Oberfläche zur Verfügung stellt mit deren Hilfe er gezielt nach Software bzw. deren Version und den jeweils zugehörigen Virtuellen Maschinen suchen kann.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Übersicht über die Arbeit . . . 1
- 1.2. Begriffe . . . 2
  - 1.2.1. Virtualisierung . . . 2
  - 1.2.2. Cluster . . . 2
  - 1.2.3. Configuration Management Database (CMDB) . . . 3
- 1.3. Szenario . . . 3
  - 1.3.1. homogene Cluster . . . 4
  - 1.3.2. heterogene Cluster . . . 4
- 1.4. Motivation . . . 5
- 1.5. Ziele der Arbeit . . . 5

2. Analyse der Möglichkeiten zum Auslesen der Softwarekonfigurationsstände **6**
- 2.1. Allgemeine Möglichkeiten . . . 6
  - 2.1.1. Datenbank . . . 6
  - 2.1.2. Ordnerstruktur . . . 6
  - 2.1.3. Prozesse . . . 7
  - 2.1.4. Ausnahmen . . . 8
- 2.2. Systemspezifsche Möglichkeiten . . . 8
  - 2.2.1. VMware . . . 8
  - 2.2.2. Microsoft Windows Registrierungsdatenbank . . . 9
  - 2.2.3. Debian Package Manager . . . 10
  - 2.2.4. RPM Package Manager . . . 11
- 2.3. Zusammenfassung . . . 11

3. Konzept **12**
- 3.1. Allgemeines Modell . . . 12
- 3.2. Prozessablauf der Softwareerfassung . . . 14
- 3.3. Software Manager . . . 16
- 3.4. CMDB . . . 17
- 3.5. Zusammenfassung . . . 18

4. Praktische Umsetzung **19**
- 4.1. Virtual Update Manager . . . 19
  - 4.1.1. Konzept . . . 19
  - 4.1.2. Implementierung der CMDB . . . 20
  - 4.1.3. Systemvoraussetzungen . . . 21
  - 4.1.4. Installation . . . 22
  - 4.1.5. Aufbau des Programms . . . 22
  - 4.1.6. Grenzen des Programms . . . 23

4.1.7. Zusammenfassung 24
4.2. VM Software Manager 24
4.2.1. Konzept 24
4.2.2. Vorbereitungen und benötigte Hilfsmittel 24
4.2.3. Aufbau des Programms 25
4.2.4. Funktionsweise 26
4.2.5. Systemvoraussetzungen 27
4.2.6. Datenbankschema 27
4.2.7. Installation 28
4.2.8. Zusammenfassung 29

5. Testszenario und Testergebnisse **30**
5.1. Testszenario 30
5.2. Installation / Einrichtung 31
5.2.1. Softwarestände auslesen 33
5.2.2. Frontend testen 34
5.3. Analyse 34
5.4. Zusammenfassung 36

6. Zusammenfassung und Ausblick **37**
6.1. Zusammenfassung 37
6.2. Ausblick 37

Abbildungsverzeichnis **39**

Anhang **40**
A. CD 40

Literaturverzeichnis **41**

1. Einleitung

In diesem einführenden Kapitel wird ein kurzen Überblick über die Abschnitte dieser Arbeit verschafft. Anschließend wird eine eine gemeinsame Wissensbasis zwischen Autor und Leser durch Erklärung zentraler Begriffe und Themen geschaffen und das zugrunde liegende Szenario beschrieben. Des Weiteren wird auf die Motivation eingegangen, die zu dieser Arbeit geführt hat, und die sich daraus entwickelte Ziele dieser Bachelorarbeit erläutern.

1.1. Übersicht über die Arbeit

Einleitung In diesem ersten Teil wird der Hintergrund der zu dem Thema dieser Arbeit geführt hat, sowie die Ziele, die im Rahmen dieser Arbeit erreicht werden sollen, beleuchtet. Dabei werden relevante Begriffe und Themen erklärt und Einsatzszenarien einer CMDB vorgestellt.

Analyse Im nachfolgenden zweiten Teil findet eine Analyse der Möglichkeiten zur Erfassung der installierten Software und Versionsstände statt. Dazu werden zunächst allgemeine Ansätze skizziert um Softwarekonfigurationsstände einer Umgebung zu erfassen und gelangt dann zu den konkreten Umsetzungen dieser Konzepte in ausgewählten Betriebssystemen.

Konzept Im dritten Teil wird allgemein das Konzept dieser Bachelorarbeit zur Erfassung der Softwarestände dargestellt und der Prozessablauf spezialisiert auf VMware-Cluster erläutert. Zusätzlich werden die Anforderungen an der CMDB ausgearbeitet und das schematische Datenbankmodell erklärt.

Praktische Umsetzung Der vierte Teil beschäftigt sich mit der prototypischen Umsetzung, wobei hierfür als Datenbeschaffungs- und Frontendbasis zunächst auf den Virtual Update Manager von Florian Peugeot zurückgriffen wird. Es werden die Schwächen dieser Lösung aufgezeigt und ein neuer Prototyp, der VM Software Manager, implementiert der dann allen Anforderungen gerecht werden soll.

Testszenario und Testlauf Im vorletzten fünften Teil wird ein konkretes Testszenario erörtert und ein protokollierter Testlauf des Prototypen mit anschließender Analyse der Ergebnisse durchgeführt.

Zusammenfassung und Ausblick Im letzten Kapitel wird nochmals kurz die Arbeit rückblickend zusammengefasst sowie ein Ausblick auf sinnvolle Erweiterungen und Ergänzungen des Prototypen geschaffen.

1.2. Begriffe

Nachdem es zu jedem Begriff meist mehr als nur eine genaue Definition gibt und oftmals mit einem Thema jeder etwas anderes assoziiert, werden zunächst einige wenige grundlegende Begriffe definiert und erklärt, um ein einheitliches Verständnis sicherzustellen und somit die Grundlage für die nachfolgenden Ausführungen schaffen.

1.2.1. Virtualisierung

Je nach Anwendungsgebiet kann Virtualisierung in der Informatik sehr vielseitig verstanden werden. Ein gängiger Definitionsversuch besagt, dass Virtualisierung Methoden bezeichnet, die es erlauben Ressourcen eines Computers zusammenzufassen oder aufzuteilen. [DZ10] Diese Arbeit beschäftigt sich im speziellen mit der Softwarelösung von VMware, so dass unter Virtualisierung eine Abstraktionsschicht (auch Hypervisor genannt) verstanden wird, die es ermöglicht Betriebssystemumgebungen zu schaffen, die von der eigentlichen Hardware des Hostsystems isoliert sind. Dies wird als Hardwarevirtualisierung bezeichnet, da hierzu Teilbereiche der nativen Ressourcen (aber natürlich auch Ressourcen im Ganzen) in Form von virtueller Hardware dieser neuen Umgebung, der virtuellen Maschine, bereitgestellt werden.

Für die in diesen Umgebungen dann laufenden, so genannte Gastbetriebssysteme, gilt als Einschränkung, dass sie für den gleichen Prozessor-Typ wie das Hostsystem ausgelegt sein müssen.

Bei VMware heißt dieser Hypervisor in der aktuellsten Version ESXi (ehemals ESX).[VGb]

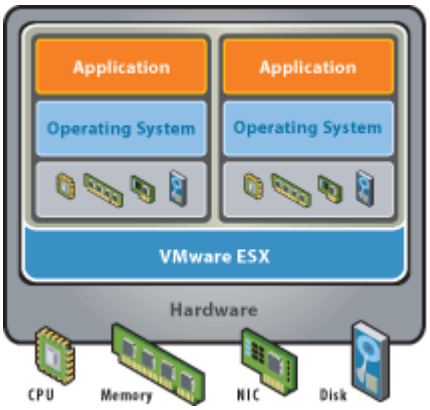


Abbildung 1.1.: VMware Hardware Virtualisierung

1.2.2. Cluster

Ein Verbund von Computern wird als Cluster bezeichnet. Dazu werden in der Regel Rechner untereinander vernetzt um somit Rechenkapazitäten oder die Verfügbarkeit zu erhöhen. Nach außen hin erscheint dieser Rechnerverbund dann oftmals nur als ein Computer.

Auch VMware bietet entsprechend die Möglichkeit durch Zusammenschluss mehrere VMware-Hostsysteme einen Cluster zu schaffen (die VMware vSphere) [VGc]. Dieser ermöglicht es hochverfügbare virtuelle Maschinen bereitzustellen, virtuelle Maschinen von einem ESXi-Host zu einem anderen zu migrieren, aber auch virtuelle Maschinen mit einer Gesamtressource aus der Kombination der Ressourcen der einzelnen Host-System zu erstellen.

Ein VMWare-Cluster erfordert neben der ESXi-Hostsystemen noch den VMware vCenter Server[VGa], der die Verwaltung und Verteilung der virtuellen Maschinen gewährleistet, sowie die Überwachung und Steuerung der ESXi-Systeme übernimmt.[Goe10]

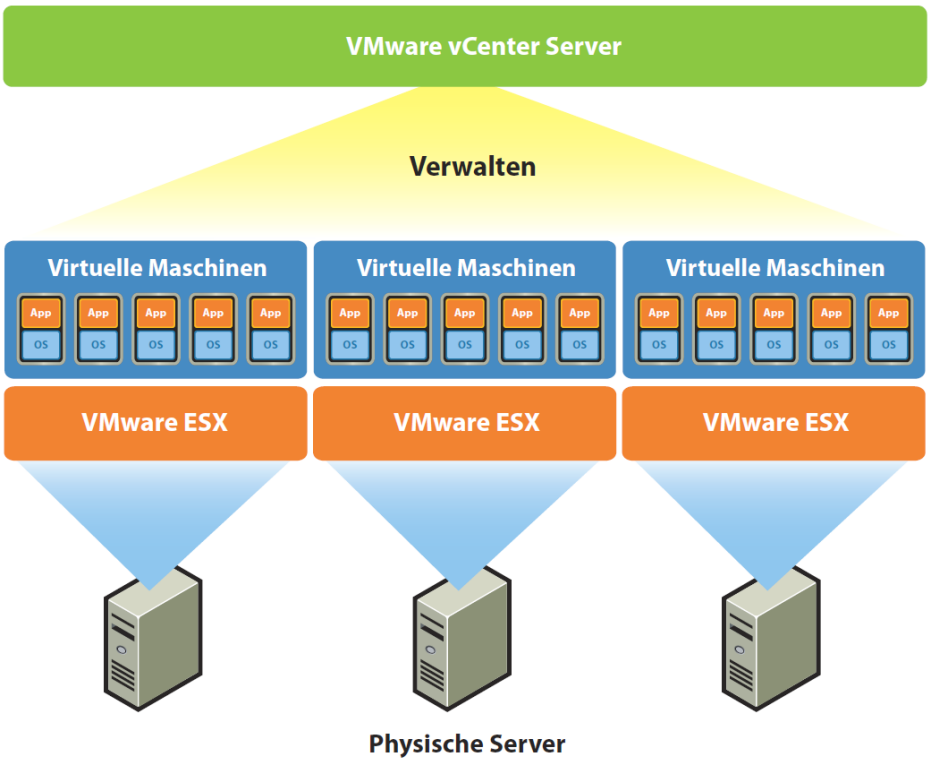


Abbildung 1.2.: VMware Cluster

1.2.3. Configuration Management Database (CMDB)

Wenn in dieser Arbeit von der Configuration Management Database (CMDB) gesprochen wird, so bezieht sich dies immer auf eine Datenbank im Sinn der IT Infrastructur Library, welche zur Verwaltung von beliebigen Betriebsmitteln (den sogenannten Configuration Items), deren Eigenschaften und gegenseitigen Abhängigkeiten dient. [?] [RB08] Da es in dieser Arbeit um die Verwaltung von Softwarekonfigurationsständen geht, spezialisieren wir den Einsatzzweck der CMDB. Als Configuration Item versteht man im Rahmen dieser Arbeit daher die virtuellen Maschinen bei denen als Eigenschaften die installierte Software sowie deren Versionsstand erfasst werden.

1.3. Szenario

Die Basis der nachfolgende Szenarien bildet der vSphere VMware-Cluster (also ein Verbund mehrere ESXi-Hostsysteme unter der Obhut des vCenter Servers). Dieser stellt eine zentrale Plattform zur Verfügung, so dass nicht mehr die einzelnen ESXi-Systeme angesprochen werden müssen, sondern die Administration über diese zentrale Stelle erfolgen kann. Entsprechend werden virtuelle Maschinen im Regelfall nicht einem ESXi-Host sondern dem Cluster

zugewiesen und der sogenannte VMware Distributed Ressource Scheduler (DRS) verteilt die virtuelle Maschinen dynamisch auf die Ressourcen der Hostsysteme. Je nach eingesetzten virtuellen Maschinen bzw. (da wir die Softwareseite von virtuellen Maschinen in dieser Arbeit betrachten) eingesetztem Betriebssysteme lassen sich diese Cluster grob in zwei Kategorien einteilen.

1.3.1. homogene Cluster

Unter einem homogenen Cluster versteht man im Allgemeinen einen Computerverbund dessen einzelne Rechner auf gleicher Hardware und gleichem Betriebssystem laufen.

Da es sich nachfolgend um das Thema Virtualisierung dreht, ist vor allem relevant ob die Gastsysteme homogen (also gleiche virtuelle Hardware, gleiches Betriebssystem) sind. Ein derartiges Szenario wäre schnell abgehandelt, da das Konzept zur Erfassung und Verwaltung der Softwarestände auf diesen Einzelfall eingegrenzt und damit sehr einfach zu bewerkstelligen wäre. Im einfachsten Fall (vorausgesetzt, die Gastsysteme sind identisch in Bezug auf installierte Software) müsste nur eine Virtuelle Maschine (VM) erfasst und verwaltet und die anderen VMs würden durch einfaches klonen erstellt werden. Somit wäre eine Verwaltung dieser weiteren virtuellen Maschinen nicht erforderlich und die Notwenigkeit des sinnvollen Einsatzes einer CMDB in diesem Fall nur bedingt gegeben, da die Software der als Template dienenden virtuellen Maschine auch direkt mit den Bordmitteln des Betriebssystems erfasst werden könnte. Auch ist natürlich fraglich ob hierfür überhaupt der Einsatz eines VMware-Clusters sinnvoll ist, da diese Cluster meist zum Zweck haben eine einzelne Anwendung horizontal zu skalieren und somit eine Virtualisierungsschicht nicht zwingend notwendig wäre, sondern die jeweilige Applikation auch nativ auf den einzelnen Hostsystemen arbeiten könnte.

Ein typisches Beispiel für dieses Szenario wäre ein Cluster an Web-Servern auf denen jeweils die gleiche Applikation läuft um in Kombination mit einem LoadBalancer eine große Anzahl von Anfragen bearbeiten könnten, für die ein einzelner Server alleine nicht ausreichen würde.

1.3.2. heterogene Cluster

Bei einem heterogenen Cluster findet man unterschiedliche Rechner oder unterschiedliche Betriebssysteme im Verbund vor. Vor allem die verschiedenen Betriebssysteme und somit die unterschiedlichen Gegebenheiten zur Erfassung der Softwarekonfigurationsstände der einzelnen Maschinen spielen hier die entscheidende Rolle. Es lassen sich so eine Vielzahl von verschieden Einsatzbereichen abdecken (z.B. Microsoft Exchange Server auf Windowsumgebung und Webserver auf Linux Basis) ohne für jeden Fall einen einzelnen dedizierten Host vorhalten zu müssen. Ein VMware Cluster bietet hier die entsprechende Flexibilität und Komfort einen nahezu unbegrenzt ausbaubaren Ressourcenpool (1000 ESXi-Hosts und mehr) aufzubauen und je nach Anwendungsfall individuelle Umgebungen bereitzustellen. Es entsteht somit eine virtuelle Rechnerlandschaft, die schnell unübersichtlich und deren Verwaltung sehr aufwendig wird. Eine CMDB kann hier u.a. die benötigte Übersicht verschaffen und als nützliches unterstützendes Hilfsmittel für Wartung und Support dienen.

Da dieses Szenario natürlich auch den Fall des homogenen Clusters in Bezug auf Gastsysteme abdeckt (ein homogener Cluster stellt gewissermaßen eine Spezialisierung in dem Sinn dar, dass anstatt verschiedener Betriebssystemen man immer nur ein System vorfindet), wird

sich nachfolgend mit Clustern mit heterogenen Gastsystemen beschäftigt.

1.4. Motivation

Das Mooresche Gesetz besagt, dass sich die Anzahl der Schaltkreiskomponenten auf einem Computerchip mit minimalen Komponentenkosten regelmäßig verdoppelt.[Moo] Somit erlebte man in den letzten Jahren eine enorme Leistungssteigerungen der Rechnersysteme und haben oftmals einen Leistungsüberschuss zu verzeichnen, den man versucht durch den Einsatz Virtueller Maschinen effektiv zu nutzen. Es sammeln sich somit eine Vielzahl virtueller Maschinen, die jeweils ihren Nutzern und Anwendern zugeordnet sind und sich voll und ganz in deren administrativen Obhut befinden. Diese Situation führt oftmals zu aus Sicht der Software schlecht gepflegten Systemen.

Es liegt somit ein stark begründetes Interesse in der IT vor, stets einen Überblick über den Stand der jeweiligen Softwareprodukte zu haben, bezüglich möglicher Updates und Patches sowie etwaige noch nicht behobene Gefahren immer im Bilde zu sein, um so aktuelle und bestmöglich sichere Serverumgebungen vorzuhalten.

1.5. Ziele der Arbeit

Ziel dieser Arbeit ist das Konzept und die prototypische Umsetzung von Prozessabläufen in VMware-Clustern um Softwarekonfigurationsstände von Betriebssystemen in virtuellen Maschinen automatisch zu erfassen, um diese installierten Software-Produkte mit ihren Versionen und verfügbaren Updates und Patches in einer Datenbank (der CMDB) abzuspeichern.

2. Analyse der Möglichkeiten zum Auslesen der Softwarekonfigurationsstände

Um eine CMDB nutzen zu können muss diese Initial mit Daten befüllt und ständig aktuell gehalten werden. Da dies manuell natürlich (vor allem bei größeren Clustern mit vielen Virtuellen Maschinen) ein sehr zeitintensives (realistisch in der Praxis nahezu unmögliches) Vorgehen darstellt, muss der Schritt der Informationsbeschaffung weitestgehend automatisiert ablaufen und die erhaltenen Daten (Software, Versionen, Updates) sich maschinell verarbeiten lassen. Daher werden nachfolgend die Möglichkeiten betrachtet, die auf den Virtuellen Maschinen installierten Softwareprodukte sowie deren jeweilige Version auszulesen. Hierzu werden zunächst allgemeine Ansätze dargelegt und anschließend die Umsetzung in den für das angedachte Szenario üblichen Betriebssystemen betrachtet.

2.1. Allgemeine Möglichkeiten

2.1.1. Datenbank

Im besten Fall verfügt das eingesetzte Gastbetriebsystem über eine entsprechende Datenbank im weitesten Sinn, in welcher die installierten Software und deren Version aktuell vorgehalten werden. Das Betriebssytem würde dazu diverse Hilfsmittel zur Verfügung stellen, um die Installationen, Updates und Deinstallation von Software zu ermöglichen und die Konsistenz dieser Softwaredatenbank zu gewährleisten. In diesem Fall wäre eine Informationsbeschaffung sehr einfach, da lediglich eine passende Abfrage an diese Datenbank geschickt werden müsste um die gewünschten Informationen zu erhalten.

2.1.2. Ordnerstruktur

Denkbar wäre auch, dass die benötigten Informationen in Dateien bzw. Ordnerstrukturen (z.B. ein Ordner, dessen Unterordner die installierten Programme darstellen und die alle eine Datei mit Versionsinformationen enthalten) ersichtlich sind und durch auslesen und parsen die gewünschten Informationen beschafft werden können. Die meisten Softwareprodukte beherbergen in Ihren Ordnerstrukturen Dateien mit den Namen version.txt oder changelog.txt, so dass durch enstprechendes Parsen dieser Dateien die Softwareversion erfasst werden könnte. Eine Prozedur dafür könnte wie folgt aussehen:

Algorithm 2.1.1: DIRECTORYGRABBER($StartDirectory$)

while $FileInDirectory$

do
- **if** $FileIsDirectory$
  then $MERGEARRAYS(Softwarelist, DIRECTORYGRABBER(File))$
- **else if** $FilenameIsVersionfile$
  then $Softwarelist[] = PARSE(File)$

return $(S)oftwarelist$

In Zeile 1 wird eine Funktion *DIRECTORYGRABBER* erstellt, die als Parameter ein Verzeichnis erwartet, dass anaylsiert werden soll. Dabei ist die Funktion so angedacht, dass Initial ein Start-Verzeichnis übergeben wird und dieses rekursiv alle Unterordner durchläuft und die enthaltenen Dateien überprüft. In Zeile 2 holt sich diese dazu alle Inhalte des Ordners und führt auf jedes Element die Überprüfung aus ob es sich um Verzeichnis oder um eine gesuchte Datei handelt (hier wäre natürlich ein Array mit mehrere verschiedenen Dateinamen als Elemente wie version.txt oder changelog.txt nötig). Findet sich ein Verzeichnis, wird die Funktion *DIRECTORYGRABBER* rekursiv wieder aufgerufen und hangelt sich so durch jeden Ordner und Unterordner. Landet die Funktion einen Treffer, also findet sie eine der besagten Versionsdateien, übergibt sie diese Datei an einen Parser, der den Softwarekonfigurationsstand ausliest und zurückgibt. Die gefundene Softwareversion wird in einem Array gespeichert und von der Funktion mit den anderen gefunden Softwareversionen von einer Funktion *MERGEARRAYS* zusammengeführt. Am Ende erhält man somit von der Funktion ein Liste mit allen gefundenen Programmen, die entsprechend weiter verarbeitet werden kann.

2.1.3. Prozesse

Rückschlüsse auf die installierten Software und deren Version lassen sich auch durch Analyse der laufenden Prozesse ziehen. Durch spezifische Aufrufe dieser,den Prozessen zugrundeliegenden Programme, könnte man dann die jeweiligen Softwareversionen in Erfahrung bringen. An einem kurzen Beispiel wird dies nun näher aufgezeigt.

Der Befehl *top* bringt in Linuxsystemen eine Auflistung der laufenden Prozesse.

```
lb2:~# top
top - 10:28:59 up 109 days,  1:51,  1 user,  load average: 0.06, 0.08, 0.05
Tasks:  24 total,   2 running,  22 sleeping,   0 stopped,   0 zombie
Cpu(s):  1.7%us,  2.8%sy,  0.0%ni, 95.5%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   7168000k total,   162456k used,  7005544k free,        0k buffers
Swap:        0k total,        0k used,        0k free,        0k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
21805 haproxy   15   0  112m  95m  584 S    8  1.4  43:11.64 haproxy
22059 www-data  15   0 32664 6824 1568 S    1  0.1 128:05.48 nginx
    1 root      15   0 10304  744  616 S    0  0.0   0:08.85 init
11274 root      15   0    92   12    4 S    0  0.0   0:00.11 init-logger
11481 daemon    19   0  8016  520  404 S    0  0.0   0:00.00 portmap
11556 root      16   0  5896  740  580 S    0  0.0   0:14.78 syslogd
11560 root      18   0  3792  448  356 S    0  0.0   0:00.00 klogd
11567 root      18   0 48856 1192  700 S    0  0.0   0:30.13 sshd
```

Abbildung 2.1.: Liste laufender Prozesse in Linux

Mit Hilfe des Kommandos *which* gefolgt von dem Namen aus der Commando-Spalte der Ausgabe des Top-Befehls erhält man den genauen Pfad der ausführbaren Datei des entsprechendes Prozesses. Mit einem spezifischen Parameter (in diesem Fall -v) erhält man schließlich Informationen über diese Software und es zeigt sich, dass es sich in diesem Beispiel um HA-Proxy in der Version 1.4.4. handelt.

```
1 lb2:~# which haproxy
2 /usr/local/sbin/haproxy
3 lb2:~# /usr/local/sbin/haproxy -v
4 HA-Proxy version 1.4.4 2010/04/07
5 Copyright 2000-2010 Willy Tarreau <w@1wt.eu>
```

Eine Prozedur, die alle aktuelle laufende Prozesse anaylsiert, könnte wie folgt aussehen:

Algorithm 2.1.2: PROCESSGRABBER(*prozessliste*)

for each $process \in prozessliste$
do $\begin{cases} filepath = FINDPATH(process) \\ softwarelist[] = GETVERSION(filepath) \end{cases}$
return $(S)oftwarelist$

2.1.4. Ausnahmen

Der ungünstigste Fall stellt eine unstandardisierte, manuelle Installation dar (z.b. durch einfaches Kopieren der Datei in ein Verzeichnis auf dem Desktop und starten des Programms oder kompilieren ohne der von den Betriebsystem für die Softwaredatenbank zu Verfügung gestellten Hilfsmittel) und somit durch keine der oben genannten Ansätze gefunden werden kann.

Gegebenenfalls ist es möglich mit dem Wissen um derartige Spezialfälle an die gewünschte Informationen zu gelangen, indem man nach spezifischen Informationen (z.B. Ordner-/Dateinamen) suchen lässt, allerdings müssten derartige Suchpattern im worst-case für jedes Programm individuell gestaltet werden.

2.2. Systemspezifsche Möglichkeiten

Da die denkbaren Möglichkeiten sehr vielseitig sind und teilweise jeweils einen erheblich Aufwand zur Umsetzung der automatisierten Erfassung mit sich bringt, wird nachfolgend auf die gängigsten und für das angedachte Szenario denkbaren Betriebssysteme spezialisiert und die Möglichkeit der Informationsbeschaffung betrachtet.

2.2.1. VMware

Natürlich haben sich auch die Spezialisten von VMWare mit diesem Themenbereich beschäftigt und den Update Manager für vSphere etabliert.[**?**] Der Update Manager von VMware ermöglicht das automatische Update- und Patchmanagement seiner virtuellen Gastsystem. Da läge es nahe, diesen zu nutzen um an die benötigten Softwarekonfigurationsstände sowie zu Verfügung stehende Updates und Patches zu gelangen. Auch für den Update Manager bietet VMware eine entsprechende API an, die allerdings im Funktionsumfang sehr gering gehalten ist und zur Verwaltung der Update-Richtlinien und des Patchprozesses dient. Ein Auslesen der installierten Softwareprodukte auf den jeweiligen Umgebungen stellt diese leider nicht bereit, weshalb diese API nicht für den hier behandelten Fall nutzbar ist und dieser mögliche Ansatz leider nicht genutzt werden kann. Auch lässt sich der Update Manager von VMware nur auf virtuellen Maschinen mit einem Windows oder RedHat Enterprise Betriebsystem anwenden, was ihn für eine möglichst allumfassende Lösung uninteressant macht.

Eine weitere Möglichkeit im Rahmen der VMware Hilfsmittel an Softwarekonfigurationsstände zu gelangen könnte das OVF-Format darstellen.[VMw] Als wesentlicher Mitbegründer dieses offenen Standards für virtuelle Maschinen kann VMware das OVF-Format verarbeiten und virtuellen Maschinen in dieses Format exportieren und bestehende virtuelle

Maschinen, die in diesem Format vorliegen, auch mit Hilfe des vCenter Standalone Converters importieren[?]. Ein erster Blick in die Spezifikation des OVF-Formats lässt aber sogleich alle Hoffnungen trüben, da dort explizit erwähnt wird, dass in Version 1 des OVF-Formats zunächst der Schwerpunkt auf Deployprozesse gelegt wurde und die für uns relevanten Bereiche wie Softwarekonfigurationsständemanagement erst in nachfolgenden Versionen aufgenommen werden könnten.

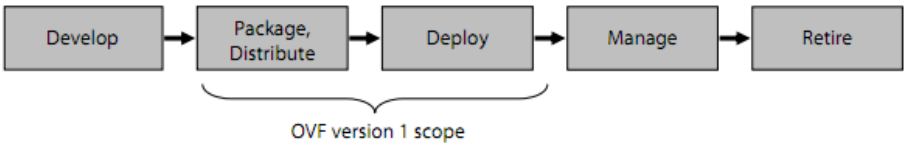


Abbildung 2.2.: Software Lebenszyklus einer Virtuellen Maschine

Ein Blick in ein beispielhaftes XML-Schema einer im OVF-Format vorliegenden virtuellen Maschine bestätigt schnell diesen Umstand. Zwar werden alle Hardware relevanten Bereiche genau genug spezifiziert und aufgeführt, allerdings finden sich kaum bis gar keine Informationen über installierte Software. Lediglich ein kleiner Abschnitt wurde dem installiertem Betriebssystem gewidmet, wobei dieser auch sehr allgemein gehalten ist.

```
1 <OperatingSystemSection ovf:id="99" vmw:osType="other26xLinuxGuest">
2       <Info>The kind of installed guest operating system</Info>
3     </OperatingSystemSection>
```

Damit fällt die Verarbeitung der Schemadatei von virtuellen Maschinen, die im OVF-Format vorliegen, als mögliche Datenquelle für die installierten Softwareversionen eines Gastsystems ebenfalls weg.

2.2.2. Microsoft Windows Registrierungsdatenbank

Mit Windows NT führte Microsoft die Windows-Registrierungsdatenbank (kurz Registry) ein, die eine zentrale hierarchische Konfigurationsdatenbank darstellt und in der mittlerweile eine Vielzahl von Informationen über Windows selbst als auch Informationen von Programmen gespeichert werden [?]. Man trifft hier also auf den günstigen Fall um an die benötigen Informationen zu kommen, da eine Datenbank vorgefunden wird (die Registry), aus der die installierten Software und deren Version ausgelesen werden kann. Um mit der Registry arbeiten zu können liefert Micrsoft das Tool Registrierungs-Editor in Windows mit, dass durch einfaches Ausführen des Befehles *regedit* einen Editor für die Windows-Registrierungsdatenbank öffnet. Unter dem Pfad HKEY_LOCAL_MACHINE_Software_Microsoft \Windows \CurrentVersion \Uninstall finden sich die benötigten Information wie die installierte Software und deren Version. Dies wollen wir anhand des folgenden Screenshots kurz erläutern:

Links in der Baumstruktur finden sich in dem ausgewählten Pfad zahlreiche Unterelemente (1). Jeder dieser Ordner repräsentiert eine installierte Software, deren detaillierte Version wir nach Klick auf den Ordner im rechten Fenster erhalten. Hier sieht man z.B. die installierte Software *Microsoft .NET Framework* und dessen Version 4.0.30319 (2).

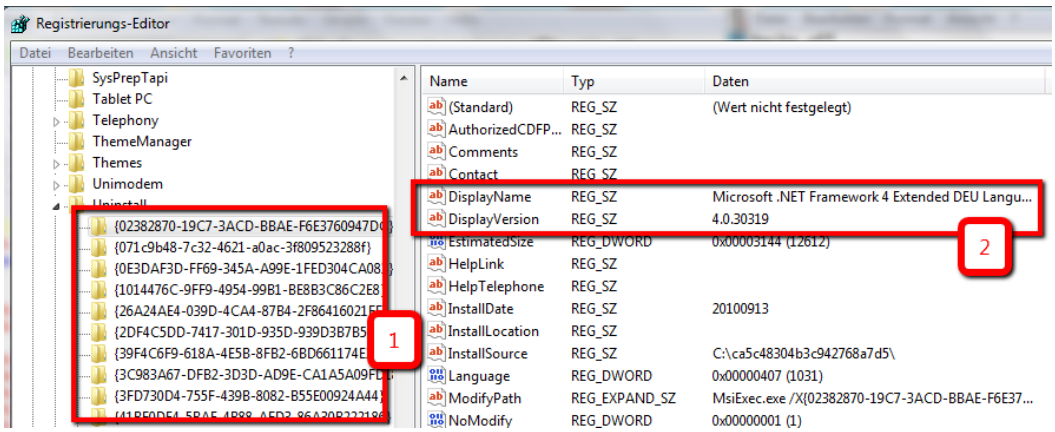


Abbildung 2.3.: Windwos Registrierungs-Editor

2.2.3. Debian Package Manager

Der Debian Package Manager mit seinem Kommando *dpkg* (Abkürzung für Debian Package) ist die Basis der Paketverwaltung des Betriebssystems Debian und das grundlegende Programm zum Installieren und Manipulieren von Debian-Paketen (Dateiendung .deb).[?] Das Paketformat wurde ursprünglich für die Debian-Linux-Distribution entwickelt und findet sich mittlerweile auch in anderen Projekten wie Fink, welches Open-Source-Pakete für Mac OS X zur Verfügung stellt, oder der OpenSolaris-Distribution Nexenta.

Das Advanced Packaging Tool (kurz *apt*) baut auf dem Debian Package Manager auf und vereinfacht Installation und Aktualisierung einer Vielzahl gängigster Softwarepakete (¿ 65.000) sowie deren automatische Konfiguration und ggf. Installation abhängiger Softwarepakete.[?] Im Regelfall werden dazu die Pakete, die als .deb Datei vorliegen, aus Softwarearchiven (Repositories) heruntergeladen. Allerdings bietet apt auch die Möglichkeit unabhängig davon Pakete, die sich nicht in einem Repositorium befinden, manuell zu installieren. Als Frontend für das Kommandozeilen Programm apt gibt es für Debian *aptitude*, welches nicht nur eine brauchbare Oberfläche zur Verwaltung der Pakete bereitstellt, sondern auch apt um fehlende Funktionen erweitert.

Daher kann man davon ausgehen, dass wir auf Debian bzw. Debian basierenden Derivaten die Möglichkeit haben, mit Hilfe der in apt enthaltenen Programmbiliotheken und Kommandozeilenprogramme an eine maschinenlesbare Liste installierte Software und deren Versionsstand zu gelangen. Eine Liste mit allen installierten Softwareprodukten erhalten wir mit dem Kommando *dpkg -l*. Das Ausführen des Kommanods dpkg -l in einer Shell einer Debian-Linux Distribution ergibt folgende Ausgabe:

```
1  root@the-successor:~# dpkg -l
2  |||/ Name                                  Version
3  +++-=====================================-=============================
4  ii  abiword                               2.8.2-2+b10
5  ii  abiword-plugin-grammar                2.8.2-2+b10
6  ii  abiword-plugin-mathview               2.8.2-2+b10
7  ii  acpi                                  1.5-2
8  ii  acpi-fakekey                          0.137-5
9  ii  acpi-support                          0.137-5
10 ii  acpi-support-base                      0.137-5
```

Um die Liste der Pakete und Updates, die in den Repositories verfügbar sind, zu aktualisieren, wird auf deb-basierenden Linuxsystemen der Befehl *apt-get update* oder *aptitute update* ausgeführt. Mit dem Befehl *apt-get upgrade* oder *aptitute upgrade* werden alle verfügbaren Updates und Patches von den Repositories heruntergeladen und die installierten Softwarepakete werden aktualisiert.

2.2.4. RPM Package Manager

Der RPM Package Manager mit seinem Kommando *rpm* war ursprünglich die Basis der Paketverwaltung des Betriebssystems RedHat (RedHat Package Manager) und das grundlegende Programm zum Installieren und Manipulieren von RPM-Paketen (Dateiendung .rpm).[**?**]

Das Paketformat wurde weiterentwickelt und unter dem Namen RPM Package Manager findet es sich mittlerweile auch in anderen Projekten wie Fedora, SuSE Linux, Mandriva etc.

Das Ausführen des Kommanods *rpm -qi rpm* in einer Shell einer SuSE-Linux Distribution ergibt die nachfolgende Ausgabe:

```
1 tigger # rpm -qi rpm
2 Name        : rpm                    Distribution: SuSE Linux 5.2 (i386)
3 Version     : 2.4.12                 Vendor: SuSE GmbH, Fuerth, Germany
4 Release     : 3                      Build Date: Tue Mar 10 01:35:47 1998
5 Install date: Fri Sep 25 18:43:41 1998 Build Host: Pascal.fs100.suse.d
6 Group       :                        Source RPM: rpm-2.4.12-3.src.rpm
7 Size        : 1163708
8 Summary     : rpm - Red Hat Package Manager
9 Description :
10 rpm (Red Hat Package Manager) is the main tool for managing software packages
      of the SuSE
11 Linux distribution. rpm can be used to install and remove software pakkages;
      with rpm it's
12 easy to update packages. rpm keeps track of all these manipulations in a
      central database.
13 This way it is possible to get an overview of all installed packages; rpm also
       supports
14 database queries.
```

Der RPM Package Manger stellt also im Prinzip von der Funktionalität ähnliche Möglichkeiten wie der Debian Package Manager bereit. Auch wenn die beiden Systeme intern teilweise sehr unterschiedlich arbeiten, so ist auch nur relevant, dass auch hier auf Betriebssystemen wie Red Hat oder Suse, die mit dem rpm Package Manger arbeiten, Hilfsmittel zur Verfügung gestellt werden, die es erlauben installierte Software auszulesen und etwaige Updates und Patches zu erfahren. Der Befehl *rpm -qa* listet alle installierten Pakete auf, die Repositories werden mit *zypper refresh* aktualisiert.

2.3. Zusammenfassung

Die Möglichkeiten wie bzw. ob überhaupt ein Betriebssystem Informationen zu installierter Software und deren Versionen bereitstellt sind nahezu unbegrenzt. Allerdings zeigt sich klar beim Betrachten der Betriebssysteme in unserer repräsentativen Auswahl , dass Möglichkeiten bzw. Hilfsmittel vom jeweiligen Betriebsystem vorgehalten werden, um Informationen über die in den Virtuellen Maschinen installierten Software und deren Softwarestände maschinell und automatisiert zu beschaffen.

3. Konzept

Nachdem im vorherigen Kapitel die Möglichkeiten aufgezeigt wurden, an die in einer Umgebung installierten Programme, Versionen und Updates zu gelangen, wird nun ein Konzept erläutert um diese Daten maschinell zu erfassen und in der CMDB automatisch zu verarbeiten. Dabei werden auch die Anforderungen an die CMDB definiert und ein allgemeines Modell für diese erarbeitet.

3.1. Allgemeines Modell

Nachfolgend wird zunächst anhand eines allgemeinen Modells dargelegt, welche Systeme für das Softwarekonfigurationsmanagement in VMware Clustern benötigt werden. Das Strukturbild zeigt anschaulich das Konzept, auf dessen einzelne Komponenten nun näher eingegangen wird.

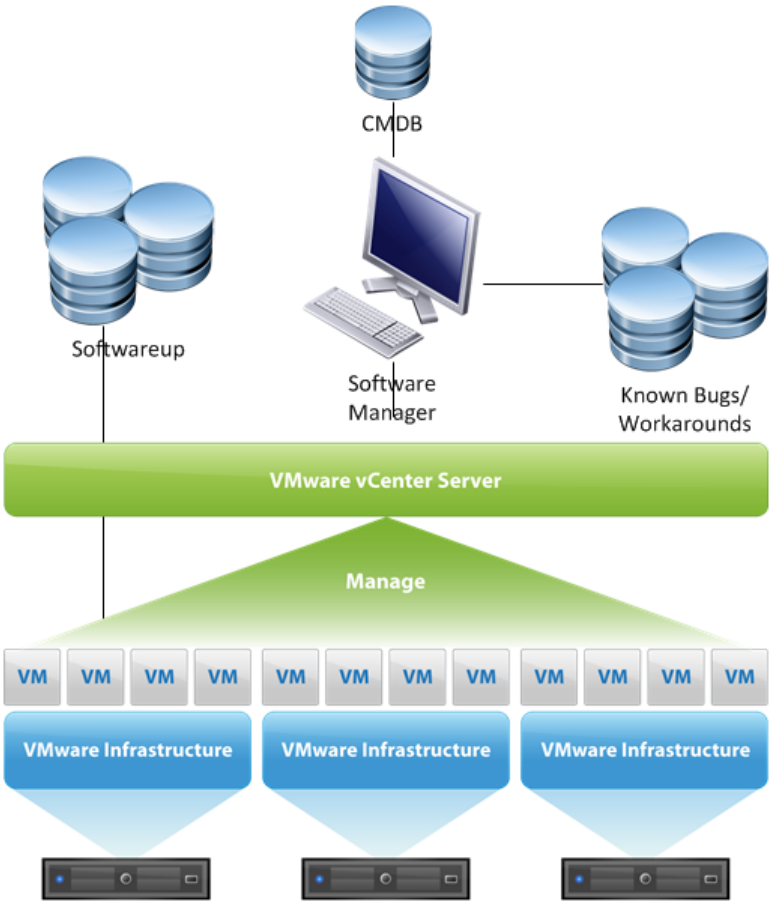


Abbildung 3.1.: Allgemeines Modell

Software Manager
Als zentrale Einheit des Prozesses fungiert der Software Manager. Diese Applikation wird benötigt um die gesamte Abwicklung des Softwarekonfigurationsmanagements zu steuern. Zu seinen Aufgaben gehören unter anderem die benötigten Daten von den virtuellen Server zu erhalten und diese in die CMDB zu speichern, sowie die Daten in der CMDB mit der Datenbank für Bugs abzugleichen um etwaige Schwachstellen präventiv und automatisch zu erkennen und ggf. weitere Schritte wie die Auslösung eines Alarms (z.B. per E-Mail Versand) zu veranlassen. Außerdem stellt er eine Oberfläche bereit, die es erlaubt Suchanfragen an die CMDB zu schicken und diese zu pflegen.

CMDB
Die Datenbank beherbergt in erster Linie Configuration Items und deren Attribute, sowie die Beziehungen zueinander. In unserem Fall sind die Configuration Items die virtuellen Maschinen aber auch die ESXi-Hosts oder vCenter Server die natürlich in Beziehung zueinander stehen (virtuelle Maschine läuft auf einem ESXi-Host bzw vCenter Server). Als Attribute werden die installierte Software mit Version sowie etwaige verfügbare Updates erfasst.

Virtuelle Maschine
Die Virtuellen Maschinen liefern Daten an den Software Manager. Hierfür treten sie über die Agenten mit dem Software Manager in Kontakt. Zu den übermittelten Daten gehört neben einer kompletten Liste aller derzeit installierten Software und deren Version (eine vollständige Liste macht es dem Software Manager möglich diese mit den bisherigen Attributen abzugleichen und so ggf. entfernte Programme zu identifizieren und in der CMDB zu archivieren) auch eine Liste der möglichen Softwareupdates. Die verfügbaren Updates und Patches (bzw. zunächst nur die Information ob und welche Updates und Patches bereitstehen), wiederum erhalten die virtuellen Maschinen von den jeweiligen Update-Servern.

Agent
(ohne Abbildung, da diese in den Virtuellen Maschinen agieren) Um Daten an den Software Manager zu schicken sollte es möglich sein, dass die virtuelle Machine wahlweise per Push (also Agent schickt Daten an den Software Manager) oder per Pull (Software Manager fragt beim Agenten nach den Daten und erhält daraufhin diese) mit dem Software Manager interagiert. Dabei ist Agent ein allgemeines Synonym für irgendeine Form von Programm oder Schnittstelle, das je nach verwendete Technik ständig auf Anfragen lauscht (Pull; z.B. auch ein einfacher SSH-Dämon, über den sich der Software Manager auf der virtuellen Maschine einloggen kann und dort die benötigten Befehle zur Datenbeschaffung ausführt) oder intervallweise angestoßen wird und dann per Push die Daten übermittelt.

Software Update/Patches
Wie in dem vorherigen Kapitel dargelegt wurde, bieten die meisten Distributionen Hilfsmittel, die es erlauben Informationen über Updates und Patches für installierte Software der jeweiligen Virtuellen Maschine von zentralen Repositories zu erhalten und ggf. von diesen herunterzuladen und zu installieren. Diesen Datenbank ermöglichen es den virtuellen Maschinen regelmäßig (zumindest aber auf Anleiern des Software Managers) auf neue Updates zu prüfen und dies dem Software Manager mitzuteilen.

Bugs
In dieser Datenbank befinden sich Informationen über bekannte noch ungepatchte Sicherheitslücken oder Fehler. Dabei steht Datenbank hier für jegliche Art der Datenbereitstellung. Dies kann ebenso eine klassische Datenbank sein wie auch ein RSS-Feed oder ein E-Mail Newsletter. Idealerweise lassen sich die darin enthalten Informationen filtern, so dass es dem Software Update Manager in Kombination mit dem Wissen um installierte Software auf den virtuellen Maschinen möglich ist betroffene Datensätze seiner CMDB zu identifizieren.

3.2. Prozessablauf der Softwareerfassung

Nachdem das allgemeine Modell betrachtet wurde, wird nun der Prozess der Datenbeschaffung näher beleuchten und detaillierter erklärt.

Der genaue Prozessablauf ergibt sich wie folgt:

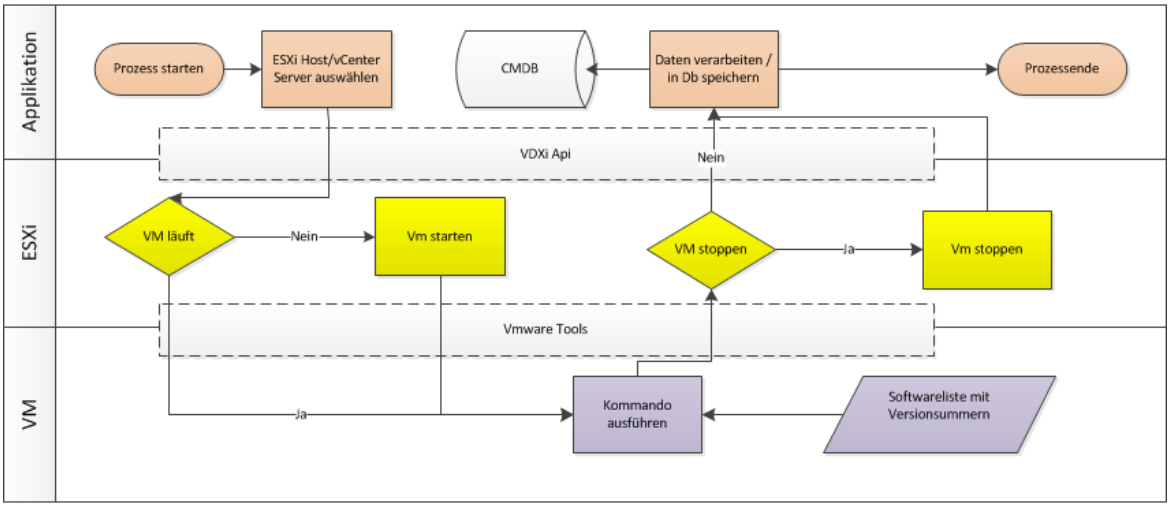


Abbildung 3.2.: Liste laufender Prozesse in Linux

Schritt 1 Verbindung zum Host aufbauen
VMware bietet für die Kommunikation mit dem vCenter Server (aber genauso auch direkt an die ESXi Server) die VIX API, die es erlaubt sich mit dem Clustermanager zu verbinden und eine Vielzahl von Aktionen in Bezug auf die virtuellen Maschinen auszuführen. Um mit den virtuellen Maschinen in Verbindung zu treten, muss man sich zunächst mit dem zuständigen vCenter Server verbinden und mit einem Benutzernamen und Passwort authentifizieren. Anschließend können über die Schnittstelle Befehle an die Gastsysteme geschickt werden.

Schritt 2 Mit virtueller Maschine verbinden
Ein möglicher Befehl der VIX API in Bezug auf die virtuellen Maschinen ist die Möglichkeit diese zu starten oder zu stoppen. Dies ist insofern wichtig, da natürlich nicht zu einer ausgeschalteten Umgebung verbunden werden kann und somit keine Befehle auf dieser ausführen werden können.

Nun kann es natürlich den Fall geben, dass eine virtuellen Maschine bewusst ausgeschaltet ist um nicht in Konflikt mit anderen virtuellen Maschinen zu geraten. Ein klassisches Beispiel wäre ein Cold-Backupsystem, dass im Falle des Ausfalls eines System gestartet werden würde und dessen Funktion übernimmt. Wären nun beide System gleichzeitig Online könnte dies unter anderem zu Inkonsistenz von Daten führen, da man nicht mehr wüsste, auf welches System man nun zugreifen soll, oder aber auch dazu, dass keines der Systeme mehr erreichbar ist, weil sie beide Versuchen würden eine gemeinsame Ressource wie ein IP für sich zu beanspruchen.

Der Prozess muss also, nachdem er geprüft hat ob eine Maschine online oder offline ist, wissen ob er gegebenenfalls eine deaktivierte Maschine starten darf um dort mit dem Prozess fortzufahren. Läuft die Maschine kann man sich hier ebenso mit Benutzername und Passwort zu dem Gastbetriebssystem verbinden (diesmal allerdings Benutzer, die in dem Betriebsystems definiert wurden und nicht wie beim vCenter mit einem in der vSphere angelegten User). Voraussetzung dafür ist, dass dem Gastbetriebssystem die VMware-Tools installiert wurden.

Schritt 3 Kommando auf der virtuellen ausführen
Nun geht es an den entscheidenden Schritt eines der Kommandos aus dem vorherigen Kapitel auf der virtuelle Maschine auszuführen um so an die gewünschten Informationen zu gelangen. Da auf dem Gastsystem VMware Tools installiert sind (sonst hätten man sich erst gar nicht in die virtuelle Umgebung einloggen können), besteht über die VIX API (die diesen Befehl dann wiederum mit Hilfe der VMware Tools auf dem jeweiligen System ausführt) die Möglichkeit beliebige Kommandozeilenbefehle auf dem Gastbetriebssystem ausführen zu lassen. Hier kommen je nach Betriebssystem individuelle Befehle zum Einsatz, die aus der Softwaredatenbank des jeweiligen Systems die aktuellen Stände ausgibt bzw. verfügbare Updates und Patches anzeigt.

Schritt 4 Softwarelsite an Host übetragen
Nachdem nun die benötigten Informationen in der ausgewählten Umgebung auflistet wurden, müssen diese auf den Software Manager übertragen und dort weiterverarbeiten werden. Dies lässt sich problemlos dadurch realisieren, dass die Ausgaben der Softwaredatenbank-Hilfsmittel in der virtuellen Maschine abfangen und diese in eine temporäre Datei geschrieben wird. Diese temporäre Datei wird schließlich von dem System heruntergeladen und auf deb CMDB-Server übetragen. Diese Funktionalität der Übertragung von Dateien (sowohl der Upload auf die virtuellen Maschine, als auch der Download von der virtuellen Maschine) stellen die VMware-Tools in Kombination über die VIX API ebenso zur Verfügung.

Schritt 5 Rückgabe bearbeiten
Der letzte Schritt beinhaltet die Aufbereitung der Ausgabedaten der jeweiligen Sytemhilfsmittel, die meist in einem proprietären Format (z.B. Trennzeichen strukturierte Tabellen) vorliegen. Ein Grabber läuft nun über alle Dateien und schickt sie an einen Parser, der aus diesen raw-Daten die entsprechenden Informationen extrahiert. Da je nach Betriebssystem die Ausgabe der Softwareliste unterschiedlich ausfallen kann muss je nach Gastsystem ein spezialisierter Parser sich der Aufgabe annehmen. Die so gewonnen Attribute lassen sich nun in der CMDB abspeichern und ihrem jeweiligen Configuration Item zuweisen.

Diese Schritte zeigen exemplarisch den Ablauf zum Erfassen der Daten bei einer ausgewähl-

ten virtuellen Maschine. Sollen mehrere Systeme abgefragt werden, wiederholt sich nach Schritt 1, der Verbindung zum Host, Schritt 2, 3 und 4 für jede virtuelle Maschine und erst dann werden die Daten final in Schritt 5 verarbeitet.

3.3. Software Manager

Wie schon im ersten Abschnitt des Kapitels teilweise angedeutet kommt dem Software Manager eine bedeutende, zentrale Aufgabe zu, da er den ganzen Prozess des Softwarekonfigurationsmanagements von der Datenholung und dem Abspeichern, über zahlreiche Abfragefunktionalitäten über die Datenbestände der CMDB bis hin zum Datenabgleich mit unseren sogenannten Bugs-Daten durchführen muss. Daher sollen nachfolgend die Anforderungen an den Software Manager festhalten und anschließend der genauen Prozessablauf betrachtet werden.

Steuerung des Datenbeschaffungsprozesses
Natürlich ist einer der wichtigsten Anforderungen an die Applikation, dass diese den Prozessablauf gemäß Abbildung 1.1 koordiniert. Dazu muss er die Verbindungen zu dem vCenter Server und virtuellen Maschinen herstellen können. Hierfür ist zwingend erforderlich, dass die Applikation Zugriff auf die Zugangsdaten zu den vCenter Server und den einzelnen virtuellen Maschinen erhält und den genauen Datastorage kennt, da dieser für die Funktionen der VIX API benötigt wird um eine virtuelle Maschine gezielt ansprechen zu können.

Der Prozess der Datenbeschaffung sollte in regelmäßigen Abstände automatisch erfolgen, so dass der Softwarestand der CMDB stets so aktuell wie möglich gehalten wird.

Verarbeitung der virtuellen Maschinen Softwarelisten
Im Anschluss an den Datenbeschaffung müssen die aus dem Prozess hervorgegangenen Daten mit einem Parser verarbeitet werden. Da die Softwarelisten je nach Umgebung aus der sie entstammen, sehr unterschiedlich aufgebaut sein können muss der Parser je nach Betriebssystem angepasste Suchkriterien verwenden um die benötigten Informationen aus den Rückgaben der virtuellen Maschinen zu erhalten. Dazu benötigt er die genaue Information welche Datei zu welchem System gehört und welches Betriebssystem diese beherbergt. Die so erhaltenen Softwarestände, sowie etwaige Informationen über verfügbare Updates und Patches schreibt der Parser im Anschluss in unsere CMDB.

Kommunikation mit der CMDB
Eine fundamentale Voraussetzung für das Softwarekonfigurationsmanagement ist es, dass unsere Applikation mit der CMDB kommunizieren kann. Hierzu gehören neben dem herstellen einer Datenbankverbindung vor allem auch die Möglichkeit Datensätze neu hinzuzufügen und verändern zu können, sowie gezielte Abfragen an die Datenbank zu schicken und die Rückgaben zu verarbeiten.

Ein typisches Einsatzszenario wäre z.B. dass ein Bugreport für eine bestimmt Software X mit Version Y bekannt gegeben wird und ein entsprechender Patch hierfür bereitgestellt wurde. Der Administrator möchte also von der CMDB wissen, auf welche VM diese Kriterien zutreffen um dort den betreffenden Patch einspielen zu können. Somit muss die CMDB in diesem Beispiel dem User die Möglichkeit geben entsprechende Filter zu setzten um nur die passenden Datensätze (CI) zu erhalten.

Abgleich der Daten mit den Known-Bugs
In der Praxis schützt das Wissen um die Aktualität der eigenen Systeme nur wenig vor etwaigen neu aufgetreten Sicherheitslücken. Da es bisweilen Tage bis Wochen dauern kann, bis zu einem Problem ein Patch oder Update bereitgestellt wird ist es für die Sicherheit einer IT-Landschaft unumgänglich über aktuelle Geschehnisse in der Branche informiert zu sein und daraus resultierende etwaige Gefährdungen für die eigenen System abschätzen zu können. Da dies bei einer Vielzahl von Systeme kaum noch zu bewerkstelligen ist bedarf es eines Automatismus, der sich Informationen aus diversen Informationsquellen selbständig beschafft und die erhaltenen Informationen mit den eigenen Daten abgleicht um Sicherheitslücken identifizieren zu können. Auch diese Aufgabe muss die Applikation bewerkstelligen und dabei flexibel genug sein um verschiedene Arten von Quellen ansprechen zu können.

Bereitstellung des Frontends
Um System-Administratoren ein nützliches Tool an die Hand zu geben, mit dem sie die virtuellen Maschinen verwalten und die Softwareversionen gezielt Abfragen, sowie etwaige Ausnahmen manuell eintragen zu können, wird ein Frontend benötigt, dass es dem Benutzer erlaubt schnell und intuitiv Datensätze einsehen und bearbeiten zu können. Des weiteren sollte dieses Frontend die aus dem Datenabgleich mit der Known-Bugs-Datenbank hervorgegangen Ergebnisse einzusehen.

3.4. CMDB

Die Grundanforderung an die Datenbank ist natürlich, dass mit der CMDB die Softwarestände der einzelnen Virtuellen Maschinen und Hostsystemen (ESXi, VCenter) erfasst und verwaltet werden können. Dabei stellen die Hostsysteme die nach ITIL benötigen CIs dar, deren Beziehung untereinander die Datenbank ebenso abbilden können muss. Die jeweilig installierte Software ist ein Attribut, welches einem CI zugeordnet wird.[GO09] Eine weitere wichtige Anforderung an die Datenbank stellt die Archivierung dar. Da es für Wartung und Support wichtig ist, welche Softwareveränderungen in einer Umgebung erfolgten muss die Datenbank alte Datensätze weiterhin vorhalten und darf diese nicht löschen. Ebenso müssen diese alten Datensätze in Beziehung zu den neuen Daten gebracht werden können um im Bedarfsfall die Veränderung genau analysieren zu können. [GB08] Aus den Anforderungen an die Datenbank lässt sich das folgende vereinfachte Datenschema herleiten:

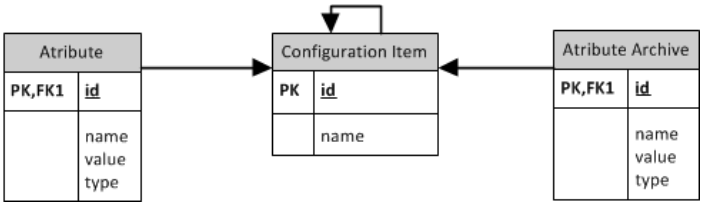


Abbildung 3.3.: stilisiertes CMDB Schmea

Configuration Items
Hier werden sowohl die virtuellen Maschinen, als auch die ESXi oder vCenter gespeichert. als zwingend erforderliche Attribute ergeben sie gemäß dem Konzept, die Zugangsdaten (Be-

nutzername und Passwort) für das jeweilige System sowie bei den virtuellen Maschinen der datastore und der imagename bzw bei den ESXi-Hosts und vCenter Server eine IP-Adresse oder DNS-Name. Fundamental für den problemlosen Einsatz des Softwarekonfigurationsmanagements-Konzepts ist ebenso, dass die Beziehung der Systeme untereinander festgehalten wird (welche virtuelle Maschine befindet sich auf welchem ESXi-Host/vCenter Server)

Attributes
Neben den schon genannten *Pflichtattributen* für jedes Configuration Item muss die Datenbank eine im Idealfall unbeschränkte Anzahl weiterer Attribute zu jedem Item speichern können, um so sämtliche installierte Programme sowie etwaige Updates festzuhalten. Um die verschiedenen Attribute identifizieren zu können sollte zu jedem Attribut noch das Merkmal Typ festgehalten werden (z.b. Update, installierte Software), um bei den Abfragen und der Ausgabe nur die wirkliche gewünschten Ergebnisse zu erhalten.

Attributes-Archiv
In diesem Speicherbereich werden alte Softwaredatensätze mit deren jeweiligen Eintragungsdatum geschrieben, sobald sie aus den Attributes Datenbestand gelöscht werden. Dies kann einerseits der Fall sein, wenn eine Software in der Umgebung deinstalliert wurde oder aber wenn eine neuere Version installiert wurde. Dabei bedeutet dies nicht, dass die Daten zwingend aus dem Datenbestand der Attribute entfernt werden müssen, da auch eine gemeinsame Datenbasis sowohl gelöschter als auch noch gültiger Attribute denkbar ist. in diesem Fall müsste darauf geachtet werden, dass die *gelöschten* Datensätze ein spezielles Merkmal erhalten um Sie von den anderen Attributen unterscheiden zu können.

3.5. Zusammenfassung

In diesem Kapitel wurde ein Konzept aufgezeigt mit dem das Softwarekonfigurationsmanagement in VMware Clustern bewerkstelligt werden kann. Dabei wurde zunächst anhand eines allgemeinen Modells Grundgedanken zu den mitwirkenden Komponenten gemacht und deren jeweilige Rolle festgehalten. Anschließend haben wurde ein Prozessablauf definiert um die Softwarestände sowie Updates und Patches der virtuellen Maschinen zu erlangen. Dank zahlreicher Hilfsmittel seitens der zu erwartenden Betriebssysteme und VMware lässt sich ein lückenloser, automatisierbarer Ablauf ermöglichen. Auch wurden die Anforderungen an die Applikation definiert, die zur Steuerung des Prozesses eingesetzt wird und Grundüberlegungen zur CMDB festgehalten und ein vereinfachtes Datenbankmodell für diese erstellt. Das Konzept stellt die wichtige Grundlage für das nächste Kapitel dar, in welchem sich nun um die praktische Umsetzung dieser theoretischen Überlegungen gekümmert wird.

4. Praktische Umsetzung

Nachdem im vorherigen Kapitel der theoretische Prozess der Datenerfassung der Aufbau und die Anforderungen an die Applikation und die CMDB definiert wurden, soll nachfolgend nun eine prototypische Umsetzung erfolgen. Zunächst wurde dazu der Ansatz verfolgt ein bestehendes Programm, den Virtual Update Manager, auszubauen.

4.1. Virtual Update Manager

Als Ausgangsprodukt diente der Virtual Update Manager, welcher im Rahmen einer Diplomarbeit von Florian Preugschat zum 31.März 2010 an der Ludwig Maximilians Universität München enstand.

Die Entscheidung den Virtual Update Manager als Grundlage für den Prototypen zu verwenden hängt damit zusammen, dass die technische Implementierung der Kommunikation mit den ESXi Hostsystemen und der Informationsbeschaffung aus den Gastsystemen schon weitgehend vorhanden war.

Außerdem hängen Softwarestanderfassung und Softwareupdate zusammen, so dass auch aus praktischen Gründe dieser Schritt sehr sinnvoll erschien.

4.1.1. Konzept

Der Virtual Update Manger wurde speziell entwickelt um Updates für Gastsysteme auf einem ESXi-Host durchzuführen. Entsprechend bedarf es einiger Anpassungen um Softwarestände der virtuellen Maschinen in einem Cluster zu erfassen.

Zunächst musste der Prozess des Auslesen der installierten Software erweitert werden, da das Grundsystem vorsah, zunächst einen ESXi-Host auszuwählen und dann dessen virtuelle Maschinen anzusprechen. In einem Cluster mit mehreren ESXi-Systeme hätte dies bedeutet, jeden Host zunächst auszuwählen und dann den Erfassungsprozess anzustoßen. Eine sehr zeitraubende Tätigkeit. Die Funktionalität wurde daher angepasst, so dass nun sequentiell zu allen als aktiv gekennzeichneten Servern verbunden wird und von allen laufenden Virtuellen Maschinen die Softwarestände ausgelesen werden.

Des weiteren wurde der Bereich *Installierte Software* gänzlich erneuert, da dieser anfangs nur die rudimentäre Funktion enthielt für eine ausgewählte virtuelle Maschine die unverarbeitete Liste installierter Produkte einsehen zu können. Es wurde ein Parser implementiert, der aus den Rückgaben der Gastsysteme die Software und Version extrahiert und diese in die CMDB schreibt. Weiter wurde die Möglichkeit geschaffen, Datensätze manuell in die CMDB zu schreiben und die angezeigte Liste der CMDB mit verschiedenen Abfragen zu filtern. Auch werden alte Datensätze nicht überschrieben, so dass sich vorher installierte Versionen nachverfolgen lassen.

4.1.2. Implementierung der CMDB

Als Datenbanksystem wurde SQLite ausgewählt. SQLite unterstützt als relationales Datenbanksystem gängige SQL Sprachkonstrukte und kann seine Tabellen in nur einer Datei abspeichern. Da es ohne Server-Software auskommt, bietet es den entscheidenden Vorteil, dass so der Virtual Update Manager direkt aus dem Ordner heraus gestartet werden kann (nach Anpassung der server.xml) und somit das Programm theoretisch auch von einem beliebigen Medium wie USB-Stick oder CD (dann natürlich nur mit Lesezugriff auf die Datenbank) sofort und ohne Installation gestartet und genutzt werden kann.

Für eine erste Version wurde ein bewusst einfaches Schema für die CMDB gewählt, das sich folgendermaßen präsentiert:

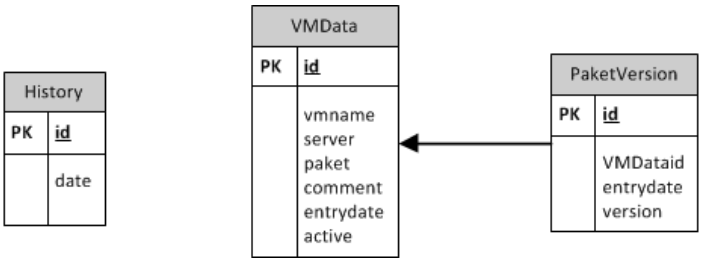


Abbildung 4.1.: CMDB Datenschema

uHistory
Diese Tabelle dient gewissermaßen als Logtabelle und speichert derzeit nur Uhrzeit und Datum jedes Updatelaufes.

- data: Uhrzeit und Datum eines Updatelaufes

VMData
In dieser Tabelle werde alle installierten Softwareprodukte zu einer virtuellen Maschinen gespeichert.

- server: Name bzw IP des Hosts
- vmname: Namen der virtuellen Maschine
- paket: Name des Softwarepakets
- comment: Kommentar zu dem Softwarepaket (liefert das Paketverwaltungsprogramm der jeweiligen Distribution)
- entrydate: Datum an dem der Datensatz angelegt wurde
- active: Defaultmäßig mit status *online* belegt; ist für spätere Archivierungsfunktion angedacht

PaketVersion
Hier werden die jeweiligen Versionen zu einer Software gespeichert.

- version: Softwareversion eines Programms
- entrydate: Datum an dem der Datensatz angelegt wurde

Wenn man diesem einfachen Schema die ursprünglichen Anforderung aus Kapitel 3 gegenüberstellt erkennt man, dass trotz der sehr einfachen Strukturen die gewünschten Funktionalitäten abgebildet werden können. Sowohl Archivierung wurde bedacht als auch das Abspeichern der Configuration Items Host und Virtuelle Maschine sowie deren Beziehung und die Zuordnung der Attribute (derzeit werden nur Softwareprodukte als Attribute den Configuration Items zugeordnet, da andere Attribute wie Benutzername und Passwort im Virtual Update Manager in der jetzigen Version nicht benötigt werden(siehe hierzu auch Systemvoraussetzungen). Natürlich ist sofort zu erkennen, dass dieses Datenbankschema nicht normalisiert ist, aber für einen ersten Test genügt dieses Schema.

4.1.3. Systemvoraussetzungen

Für den Betrieb des Virtual Update Managers sind mindestens folgende Punkte erforderlich:

- Microsoft Windows ab XP (32-Bit / 64-Bit)
- Microsoft .NET Framework 2.0 (oder höher)
- ca. 50 MB freier Platz auf der Festplatte
- VIxapi (mind. Version 1.10 für vCenter Server/Vsperhe ESXi 4.1)

Für Windows Gastsysteme sind die folgenden Voraussetzungen notwendig:

- Microsoft Windows ab XP (32-Bit / 64-Bit)
- Microsoft .NET Framework 2.0 (oder höher)
- VMware Tools
- Spezieller Benutzer *vUpdUser* mit lokalen Admin-Rechten und password *Update201!*
- Microsoft Windows Update Client
- ca. 15 MB freier Platz auf der Festplatte

Für Linux Gasysteme wird vorausgesetzt:

- Debian Linux 5 (Lenny, 32-Bit / 64-Bit)
- VMware Tools
- Spezieller Benutzer *vUpdUser* mit lokalen Admin-Rechten und password *Update201!*
- apt, at, sudo
- ca. 15 MB freier Platz auf der Festplatte

Der spezielle Benutzer muss Mitglied der folgenden Gruppen sein: sudo, root. Des Weiteren benötigt der spezielle Benutzer die sudo-Berechtigung OHNE Passworteingabe. Dazu muss die Konigurationsdatei /etc/sudoers die folgenden Einträge beinhalten:

```
1 vUpdUser ALL =( ALL) ALL
2 % sudo ALL= NOPASSWD : ALL
```

4.1.4. Installation

Wie schon in Abschnitt 4.2 erwähnt, lässt sich das Programm direkt aus dem Ordner starten und benötigt keine weitere Installation. Lediglich die Datei server.xml (eine XML-Datei) mit selbsterklärender Struktur muss angepasst und um die entsprechend ESXi-Hostsysteme erweitert werden. Die Datei *VmWareData.db3* sollte für den aktuellen Windowsuser Schreibrechte vorhanden sein um Änderungen an der CMDB zuzulassen. Sollten diese Rechte nicht gesetzt sein, ist nur lesender Zugriff möglich. Sind alle Anpassungen gemacht lässt sich der Virtual Update Manager durch einfaches Ausführen der Datei *vUpdManager.exe* starten.

4.1.5. Aufbau des Programms

Das Programm ist einfach strukturiert und entsprechend Bedienerfreundlich gehalten. Im wesentlichen findet sich im linken Bereich des Fensters die Navigation, während im rechten größeren Teil die Darstellung der Inhalte der ausgewählten Menüpunkte erfolgt. Eigentlich relevant ist dabei zunächst nur der Menüpunkt *Installierte Programme* unter dem Hauptnavigationspunkt *Virtuelle Maschinen*. Nach einem Klick auf diesen Menüpunkt ergibt sich folgende Ansicht:

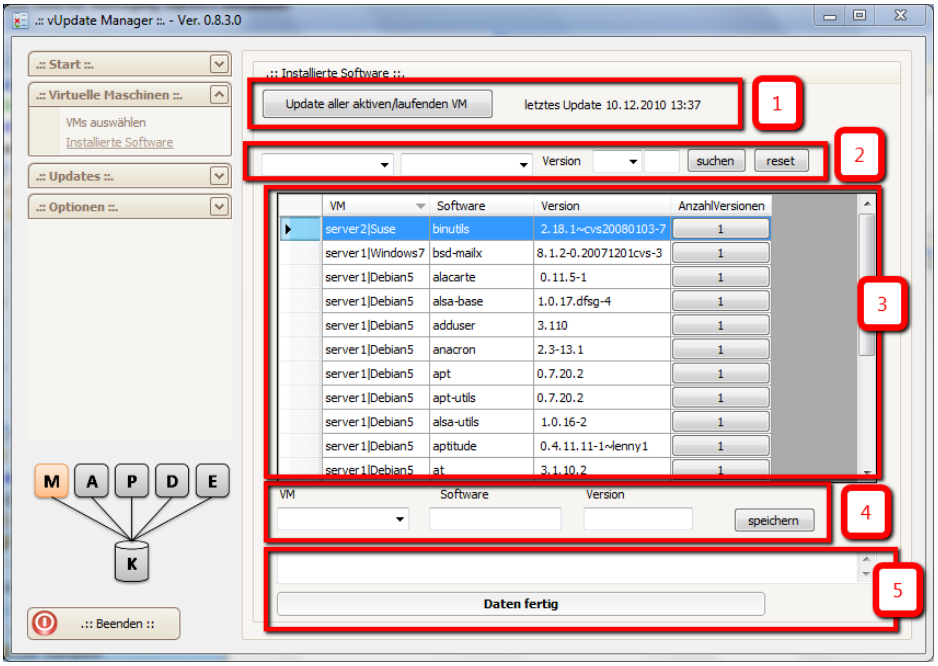


Abbildung 4.2.: Der Virutell Update Manager

Softwarestand (1)
In dem obersten Bereich wird dem User Datum und Uhrzeit des letzten Updates der CMDB angezeigt. Außerdem hat er hier die Möglichkeit den Updateprozesse anzustoßen und somit die CMDB auf den neusten Stand zu bringen.

Suchfunktion/Filter (2)
Dem Benutzer wird eine flexible Suche angeboten, mit der er alle installierten Softwareprodukte wahlweise pro virtuelle Maschine anzeigen lassen kann. Auch kann er gezielt nach einer

Software suchen und dabei noch optional festlegen, ob die Version dieser Software größer, kleiner, gleich einer bestimmten Version sein soll. Durch Klick auf den Button *suchen* wird die in Bereich 3 angezeigte Tabelle gemäß der eingestellten Kriterien gefiltert und es werden nur noch die relevanten Datensätze angezeigt.

Liste der Softwareprodukte (3)
In diesem Bereich werden dem Nutzer zunächst alle installierten Softwareprodukte aufgelistet. Die Tabelle lässt sich durch Klick auf die Tabellenspalten entsprechend sortieren. In der Spalte *Anzahl Versionen* ganz rechts erhält man einen Einblick ob früher eine andere Softwareversion installiert war. In diesem Fall wäre hier eine Zahl größer 1 (z.b. 3 wenn es vorher zwei andere Versionen gab) zu sehen und durch Klick auf diese würde sich ein kleines Fenster öffnen, dass die vorherigen Versionen samt Änderungsdatum auflistet.

Software manuell hinzufügen (4)
Wie in Kapitel 2 angesprochen kann es immer Ausnahmefälle in Bezug auf installierte Software geben, die nicht von den automatischen Routinen erfasst werden. Um diese Programme dennoch in der CMDB zu erfassen besteht hier für den Nutzer die Möglichkeit einer virtuellen Maschinen manuell eine Software und eine Version zuzuweisen.

Updatestatus (5)
Im unteren Bereich des Fenster erhält man Informationen über den Status des Updates, wenn dieses mit dem Button in Bereich 1 gestartet wurde.

4.1.6. Grenzen des Programms

Aufgrund seiner C#.net Grundlage benötigt der Virtual Update Manager eine Windowsumgebung. Dies stellt allerdings im praktischen Einsatz bisweilen eine erhebliche Einschränkung dar, da sich vor allem im Bereich der Systemadministrierung in Rechenzentren häufig Linuxumgebungen vorfinden.

Auch zeigte sich in ersten Tests, dass das Update der Softwarelisten der virtuellen Maschinen mit zunehmender Anzahl Hosts und Gastsystemen schnell auch mal mehrere Minuten in Anspruch nehmen kann, was sich auf die relativ unperformante API von VMWare zurückführen lässt. Da sinnvoll allerdings nur mit aktuellen Softwareständen gearbeitet werden kann muss man vor jedem Einsatz des Programms zunächst einige Minuten für die Updates einplanen. Ein schnelles *Mal eben Nachschauen* entfällt somit. Auch nicht zu vernachlässigen ist der Punkt der dezentralen Datenhaltung der Konfiguration des Programms. Sollten also neuen ESXi oder vCenter Server hinzugefügt werden oder z.B. nur die Logindaten eines Hosts geändert werden muss dies bei jeder Installation vorgenommen werden.

Ein weiteres großes Problem bereitete die DevExpress Library, die zur Gestaltung der GUI zum Einsatz kam. Hierbei handelt es sich um eine lizenzpflichtige Software, deren aktueller Paketpreis bei rund 800,00 US-Dollar liegt. Auf Anfragen war die Firma freundlicherweise aber bereit zumindest eine auf diese Bachelorarbeit beschränkte Lizenz auszustellen. Somit konnte zumindest das Programm soweit weiterentwickelt und kompiliert werden. Natürlich liegt es da nahe, auf diese Library zu verzichten und die Standard-Oberfläche von C#.net als GUI zu verwenden. Allerdings durchzieht sich die Verwendung dieser Library durch alle Bereich des Programms und ist in jeder KLasse tief verwurzelt. Ein Ersetzen dieser Bibliothek stellt somit einen immensen Aufwand dar. Zukünftige Entwickler, die sich mit

diesem Programm auseinandersetzen wollen, werden nicht um den Erwerb der Lizenz herum kommen.

Ein weiterer störender Faktor ist, dass derzeit im Virtual Update Manager eingesetzte Konzept, dass auf den virtuellen Maschinen die Einrichtung eines speziellen Benutzers für den Login auf den virtuellen Maschinen erfordert. Vor allem auf dem Debiantestsystem erforderte dies mehrere Eingriffe und gelang erst nach einigen Versuchen

4.1.7. Zusammenfassung

Die prototypische Implementierung einer CMDB für Softwarekonfigurationsstände in virtuellen Maschinen in VMWare Clustern auf Basis des Virtual Update Managers zeigt auf wie beindruckend einfach sich unter Zuhilfenahme der von VMWARE bereitgestellten Hilfsmitteln der Prozess der Datenbeschaffung realisieren und sinnvoll einsetzen lässt. Allerdings machten sich die erwähnten Schwächen des Programms schon kurz nach ersten Tests im praktischen Einsatz bemerkbar, weshalb dieser erste Prototyp verworfen wurde und auf eine zentralisierte, betriebssystemunabhängige Lösung umgestellt wurde, die im nachfolgenden Abschnitt vorgestellen wird.

4.2. VM Software Manager

4.2.1. Konzept

Um das Problem der Bindung an ein bestimmtes Betriebssystem zu umgehen und weitere Funktionen wie zentrale Datenhaltung und stets aktuelle Datensätze zu ermöglichen wurde auf eine serverseitige Umsetzung umgestellt, die über ein Webinterface im Browser nutzbar sein sollte. Diese Lösung sollte regelmäßig automatisch die Softwarekonfigurationsstände der virtuellen Maschinen holen, diese mit seiner CMDB abgleichen und Veränderungen erfassen. Auch bietet sich im Zuge dessen an, die in der CMDB vorgehalten Softwarestände gegen diverse Bugreport- Seiten zu matchen um so mögliche Sicherheitslücken automatisch identifizieren zu lassen.

4.2.2. Vorbereitungen und benötigte Hilfsmittel

Als Programmiersprache wurde in diesem Fall PHP gewählt, da es Rapid Prototyping ermöglicht und sich über Jahre im Markt für Webapplikationen etablierte. Entsprechend findet diese Sprache eine große Unterstützung und Zustimmung im Webbreich.

Um schnelle Entwicklung und stets gut strukturierten und wartbaren Code zu gewährleisten, wurde als Grundlage für das neue System auf das Framework CakePHP gesetzt, welches unter der MIT-Lizenz verfügbar ist. Durch das Model-View-Controller Konzept dieses Frameworks ist eine klare Trennung der verschiedenen Schichten unseres Modells (Daten, Applikation, Präsentation) möglich und erlaubt es uns so im Bezug auf das tatsächlich eingesetzte Datenbanksystem für die CMDB flexibel zu bleiben und mit den Anforderungen zu wachsen. Außerdem stellt CakePHP zahlreiche Libraries zur Verfügung mit denen sich Standardaufgaben schnell und effizient realisieren lassen und so die Entwicklungszeit zum ersten Prototypen drastisch verkürzt.

4.2.3. Aufbau des Programms

Die Oberfläche des Frontends wurde klar und schlicht gehalten um größtmögliche Browser-Kompatibilität und klare, selbsterklärende Strukturen zu schaffen. Im Wesentlichen besticht stets der gleiche einfache Aufbau.

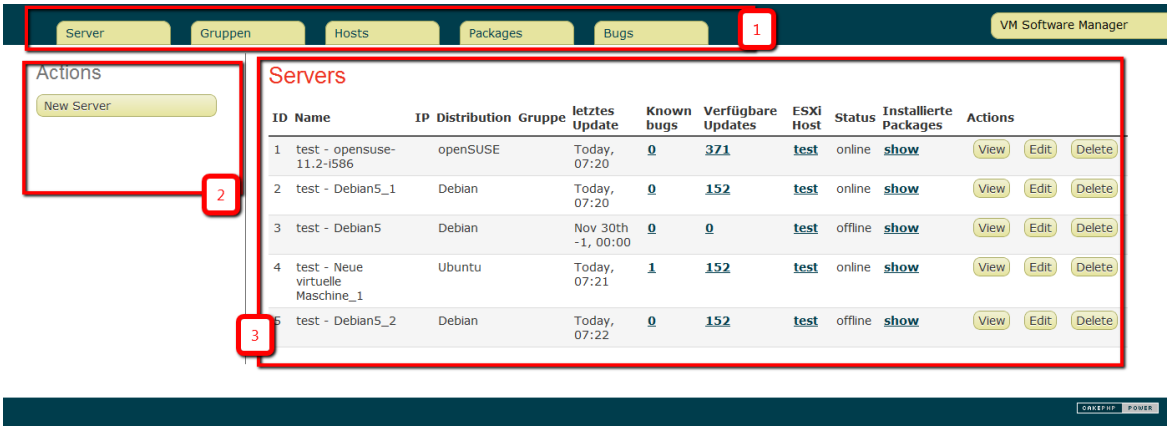


Abbildung 4.3.: Der VM Software Manager

Hauptnavigation(1)
Im oberen Bereich der Seite findet sich die stets sichtbare Hauptnavigation, die es ermöglicht schnell auf alle Bereich zu gelangen.

Subnavigation(2)
Links am Rand findet sich eine je nach Bereich spezifische Navigation in der zumeist die Option vorhanden ist, einen neuen Eintrag anzulegen.

Content(3)
In diesem Bereich erfolgt die Ausgabe der angeforderten Informationen; zumeist in einfacher Tabellenform.

Um einen kurzen Umriss des Funktionsumfangs zu geben werden nachfolgenden die verschiedenen Bereiche, die über die Hauptnavigation direkt angesteuert werden können erläutert.

Server
Der Benutzer hat hier die Möglichkeit seine virtuellen Maschinen aufzulisten, zu bearbeiten und zu löschen, sowie neue virtuelle Maschinen in den automatischen Softwareerfassungsprozess hinzuzufügen. Die Serverliste liefert dem Benutzer auf Anhieb einen Überblick über den Zustand seiner virtuellen Maschinen (online/offline). Weitere Spalten in der Serverliste zeigen zudem mögliche Gefahren (known Bugs) oder anstehende Wartungsarbeiten (verfügbare Updates) an.

Groups
Hier lassen sich Gruppen anlegen mit denen virtuellen Maschinen zusammengefasst werden können. Dieser Bereich ist derzeit noch sehr rudimentär, da die Gruppenlogik bisher nur in der Paketsuche implementiert wurde. Für die Zukunft ist angedacht auch in den Softareer-

fassungszyklen gezielt nur einzelne Gruppen abfragen zu können.

Packages
In diesem Bereich kann nach Softwarepaketen bzw. nach virtuellen Maschinen, die diese Softwarepakete installiert haben, gesucht werden. Dabei stehen verschieden Filterungsmöglichkeiten zur Verfügung, so dass gezielte Anfragen an die CMDB gestellt werden können. So kann in das Textfeld *Version* durch Vergleichsoperatoren eine genaue Suche über die Versionsnummer vorgenommen werden.

Hosts
Der Benutzer hat hier die Möglichkeit seine ESXi-Hosts oder vCenter Server aufzulisten, zu bearbeiten und zu löschen, sowie neue Hostsysteme hinzuzufügen. Fügt man einen neuen Host hinzu werden nach dem Speichern automatisch die auf diesem Host registrierten virtuellen Maschinen zu den *Servern* hinzugefügt.

Bugs
In diesem Bereich laufen die Bugreports des Grabbers auf.

4.2.4. Funktionsweise

Nachdem im vorherigen Abschnitt der Aufbau des Frontend kurz dargestellt wurde, soll es nun um die Funktionalitäten gehen und der genauen Aufbau und Ablauf.

Der Perl-Wrapper

Um den im Konzept vorgestellten Prozessablauf reibungslos sicherzustellen, wird auf die VMWare VIX Api zurückgegriffen. Leider gibt es (noch) keine Bibliothek für PHP die den Zugriff auf diese Schnittstelle ermöglichen wurde. Allerdings gibt es diese für Perl, so dass durch Aufrufe von Perlskripten aus PHP heraus indirekt auf die Schnittstelle zugegriffen werden kann. Der klassische Weg dazu wäre wie folgt:

```
1 <?php
2 $applikation = '/lokaler/pfad/zu/meiner/vixapi.pl';
3 $output = shell_exec('\$applikation');
4 echo '$output';
5 ?>
```

Dabei wir mit dem Befehl *shell_exec* das Perlskript aus PHP heraus aufgerufen und die Ausgabe des Skripts in einer PHP-Variable geschrieben. Eine wesentlich eleganter Lösung bietet das PECL-Modul Perl, dass es erlaubt Perl-Befehle direkt in PHP auszuführen und direkt auf die Werte der Perl-Variablen zuzugreifen, wodurch das aufwendige Parsen der Ausgabe, wie es in vorherigem Beispiel notwendig wäre, entfällt. Eine Liste aller laufenden virtuellen Maschinen ließe sich dadurch wie folgt erlangen:

```
1 function findRunningVMs() {
2   $this->perl->eval('@vmlist = FindRunningVMs($hosthandle, 300);');
3   $vmlist = $this->perl->array->vmlist;
4   return $vmlist;
5 }
```

Softwarekonfigurationsmanagement

Da der Prozess der die installierten Programme und möglichen Updates aus den virtuellen Maschinen ausliest auch einige Zeit in Anspruch nehmen kann muss sichergestellt werden, dass kein neuer Prozess angestoßen wird solange der Alte noch läuft um die Datenkonsistenz nicht zu gefährden. Daher wird in der Datenbank der Start und Ende eines Prozess festgehalten. Jeder neu startende Prozess prüft nun zunächst in der Datenbank ob es noch laufende Prozesse gibt und bricht ggf. seinen Durchlauf ab. Auch hat dieses Logging den Vorteil, dass sich die Prozessdauer genau bestimmen lässt und etwaige Fehler schon frühzeitig anhand von länger andauernden Prozessdurchläufen erkennen lassen. In dem Prozess zur Datenerfassung der virtuellen Maschinen war vorgesehen, dass ausgeschaltete Umgebungen im Prozessdurchlauf hochgefahren und anschließend wieder runtergefahren werden. Dabei gilt es aber zu beachten, dass dies nicht für alle inaktiven Maschinen gelten darf, da es sonst zu etwaigen Problemen kommen könnte. Daher ist es möglich bei jedem Server zu definieren, ob dieser gestartet werden soll bei dem Prozessdurchlauf oder nicht. Das ausführende Programm weiß so im Prozessdurchlauf, ob er eine ausgeschaltete Maschine starten darf oder nicht Die Problematik, dass jedes Betriebssystem bzw. dessen Tools meist ähnliche aber dennoch sehr verschiedene Software Liste zurückgeben, wird durch ein individuelles Parsing realisiert. Dazu kann bei jedem Server hinterlegt werden, um welches Betriebssystem es sich handelt um dann beim Parsen je nach Betriebssystem den passenden RegEx-String auszuwählen, der gegen Softwareliste gemacht wird und man so stets gleichgeformte Rückgabedaten erhält.

4.2.5. Systemvoraussetzungen

Für die Datenbank wird der MYSQl-Server 5.x benötigt. Als Storage-Engine für die Tabellen wurde MyIsam gewählt, da es gegenüber InnoDB einfacher in der Installation und Wartung ist und schnelle Abfragen ermöglicht. Ein beliebiger Webserver mit PHP5 Unterstützung und aktiviertem ModRewrite ist für die Nutzung des Frontends erforderlich. Sollte nicht Apache2 als Webserver zum Einsatz kommen müssen die .htaccess Dateien gegebenenfalls angepasst werden. Um Prozesse regelmäßig automatisch ausführen zu können muss ebenso PHP5-CLI installiert sein. Php5 benötigt die Library Mysql sowie die PECL Extension mod-Perl.

Auf Clientseite ist lediglich ein Internetbrowser in einer einigermaßen aktuellen Version erforderlich. Da auf grafische Spielereien und Javascript-Verwendung verzichtet wurde sollte das Frontend auf jedem gängigen Browser nutzbar sein.

4.2.6. Datenbankschema

Die in dem Konzept ausgearbeiteten und beim Virtual Update Manager schon rudimentär umgesetzten Anforderungen an eine CMDB sollen weiter gelten. Da der VM Software Manger einen erweiterten Funktionsumfang in Bezug auf Softwarekonfigurationsmanagement als der Virtual Update Manager bietet ist auch das Datenbankschema entsprechend komplexer.

Auf dem Schaubild wurden logisch zusammenhängende Komponenten gruppiert um eine deutliche Abgrenzung der CMDB zu ermöglichen.

Die Tabellen Log und cron_log werden von dem im Hintergrund ablaufenden Datenerfassungsprozess befüllt und ermöglichen so, dass der Prozessablauf lückenlos nachvollzogen werden kann, was vor allem im Fehlerfall eine erhebliche Hilfe darstellen kann.

Die CMDB findet sich in den Tabllen servers und esxi_hosts als Configuration Items sowie in der Tablle packages und updates als Attribute wieder. Hierbei wurde bewusste eine

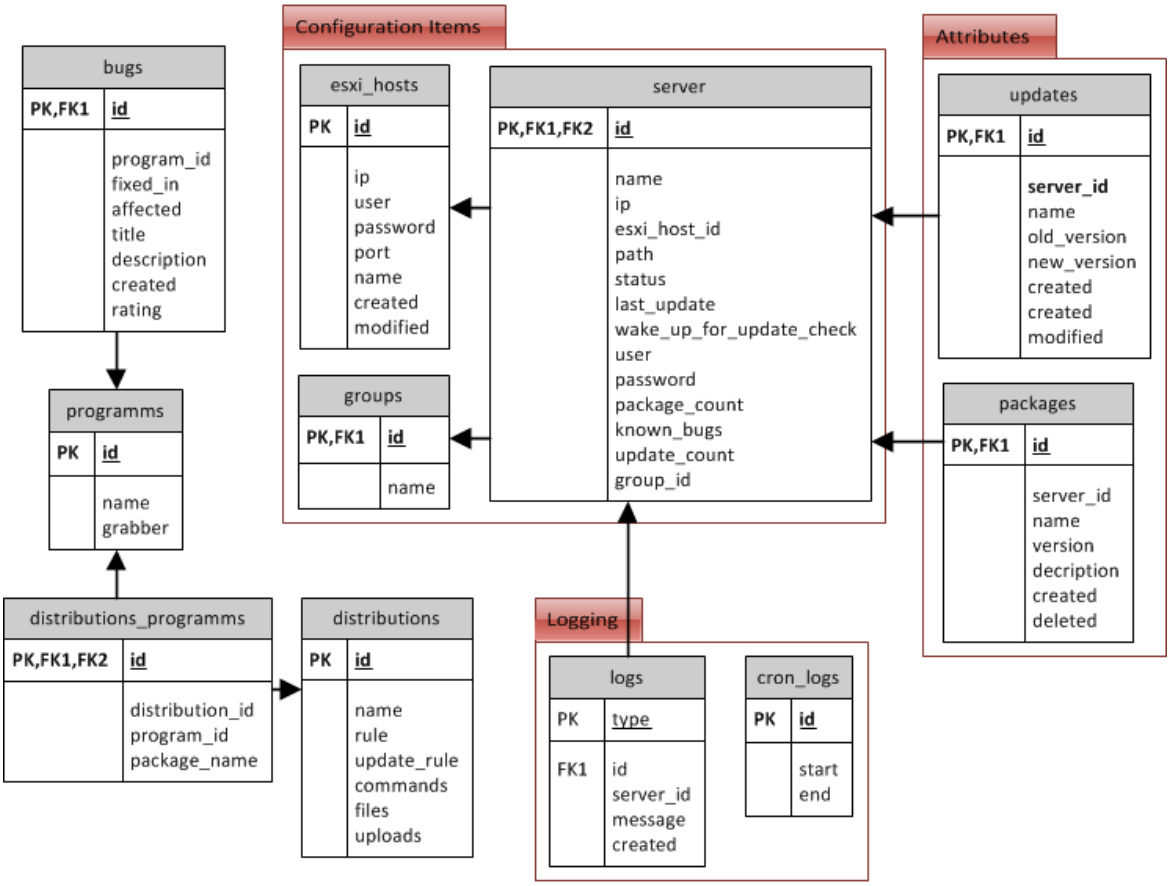


Abbildung 4.4.: Datenbankschema VM Software Manager

getrennte Tabelle für server und Hosts gewählt, um dem regelmäßigen Datenbeschaffungsprozess die Abfragen zu erleichtern und die Struktur übersichtlich zu halten.

4.2.7. Installation

Um den SW Update Manager zu installieren kopiert man sämtliche Dateien und Ordner in das Webroot-Verzeichnis des Webservers. Als neuer Webroot sollte aus Sicherheitsgründen der Ordern app/webroot definiert werden.

Die Datenbank muss initial mit dem SQL-Dump vmsm.sql im Ordner SQL befüllt werden.

Danach muss die Datei app/config/database.php entsprechend den MYSQL-Server Daten angepasst werden und für PHP Schreibrechte auf das Verzeichnis app/tmp und dessen Unterverzeichnisse eingerichtet werden.

Folgender Befehl sollte in regelmäßigen Abständen (z.B. stündlich) aus dem Stammverzeichnis des installierten Programms ausgeführt werden, da dieser die Softwarelisten aktualisiert und somit die CMDB aktuell hält:

```
1 cake/console/cake Vmcheck
```

Ist dies alles erfolgt kann das Programm durch Öffnen der Web-Adresse oder IP-Adresse des Webserver im Browser genutzt werden.

4.2.8. Zusammenfassung

Der Schritt in Richtung zentralisierte Datenbank mit Weboberfläche erwies sich durch die Umstellung auf die Skriptsprache PHP als der richtige Ansatz. Kombiniert mit dem Rapid-prototyping Framework CakePhp konnte die um einiges mächtigere Anwendung in kürzester Zeit implementiert und problemlos ständig erweitert und angepasst werden. Die zahlreichen neuen Funktion wie regelmäßige automatische Updates und der Bugmatcher gewährleisten eine stets aktuelle Sicht auf die Softwarestände und lassen mögliche Problemfälle schon früh erkennen. Die Lösung hält somit notwendige Hilfsmittel bereit um Systeme (im speziellen virtuelle Maschinen) und deren Software zu überwachen und zu verwalten und unterstützt so den Administrator bei seinen alltäglichen Aufgaben.

5. Testszenario und Testergebnisse

Dieses Kapitel beschäftigt sich mit der Einrichtung einer Testumgebung, um die Funktionalitäten des entwickelten VM Software Manager in einem exemplarischen Testlauf zu beobachten, und der anschließenden Analyse der erhaltenen Testergebnisse.

5.1. Testszenario

Für die Testumgebung stehen zwei Hostsysteme in einem VM-Ware Cluster bereit. Die Rechner verfügen über 64-Bit Intel Celeron 440 CPUs mit 2.0 Ghz, 4 GB DDR 266 Arbeitsspeicher und 80 GB Sata-Festplatte die an dem Onboard Sata Controller des Mainboards Asrock ConRoe1333 D667 Mainboard Ver 1.0.1 angeschlossen sind. Über die Intel Express PCI Netzwerkkarten werden diese mit einem 100 Mbit Switch verbunden und ermöglichen über einen zusätzlich an dem Switch angeschlossen Router mit DHCP-Server den Zugang zum Internet. Auf diesen Computern wird der VMware Hypervisor vSphere 4.1. installiert. Ein dritter PC (64-Bit Intel Celeron 440 CPU mit 2.0 Ghz, 4 GB DRR 266 Arbeitspeicher und 80 GB Sata-Festplate, Asus PFSD Mainboard) soll die Aufgabe des Cluster-Managers übernehmen. Dazu installiert man zuerst Windows 2008 Server 64-Bit, um anschließend VMware vCenter Server ver 123.321 installieren zu können. Im Anschluss daran installiert man auf einem beliebigen Windows Client (Die Eckdaten des Systems sind im nachfolgenden nicht relevant und daher ungenannt) den VMware vSphere Client zur Konfiguration des Cluster. Nach dem verbinden mit dem vCenter werden die beiden ESXi-Hostsysteme einem neuen Cluster hinzugefügt.

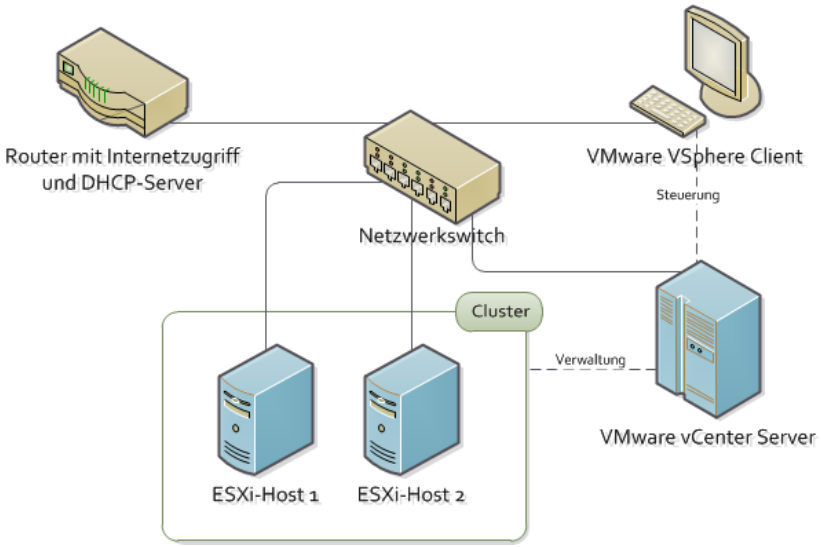


Abbildung 5.1.: Testsenzario

5.2. Installation / Einrichtung

Die Installation der Debian und SuSE Gastsysteme wird über vorgefertigte Appliances vorgenommen, welche im VMWare Appliance Place[?] heruntergeladen können. Nach dem Download dieser virtuellen Maschinen im VMware-Format installiert man sie mit dem vCenter Standalone Converter auf dem ESXi-Host. Nach dem starten der vorgefertigten Appliances ist für das Testszenario keine weitere Konfiguration notwendig. Auch die benötigen VMware Tools sind hier schon vorinstalliert.

Die Windows Gastsysteme werden von einer CD mit Standardeinstellungen installiert. Aufgrund der von VMware unterstützten Hardware der Host-Systeme und der damit in den VMs bereitgestellten virtuellen Hardware ist keine Installation von Treibern notwendig, da Windows passende Treiber mitbringt bzw. diese durch anschließend Installation der VMwareTools geupdatet werden.

Durch Klonen der Debian virtuellen Maschinen (diese werden dann für die Installation de VM Software Managers verwendet) ergibt sich das in Abildung 6.1 dargestellte Szenario.

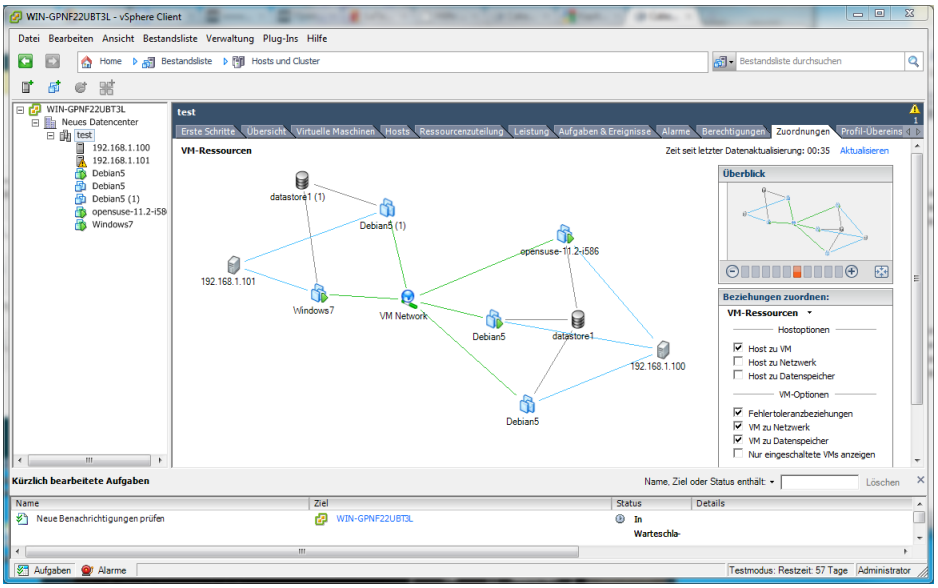


Abbildung 5.2.: Testcluster

In einem der Debiansysteme wird nun gemäß 4.4.2 der VM Software Manager installiert. Als Grundlage hierfür wird der Apache2 Webserver sowie MYSQL-Server 5 als Datenbank verwendet. Die Installation aller benötigten Pakete lässt sich in Debian mit dem folgenden Befehl einfach realisieren.

```
1 apt-get install apache2 php5 php5-cli php5-dev libapache2-mod-php5 php-pear
    mysql-server && pecl install perl
```

Die Installation läuft bis auf die kurzen Nachfragen nach der Vergabe des MYSQL-root-Passwort vollständig selbständig ab und ist in einer Minute abgeschlossen (Breitband Internetverbindung vorausgesetzt).

Da das Apache Rewrite-Module benötigt wird muss es mit *a2enmod rewrite* aktiviert werden sowie die Einstellungen AllowOverite und webroot in der Apache Config angepasst werden

```
server:~#vi /etc/apache2/sites-enabled/000-default

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/app/webroot/
        <Directory /var/www/app/webroot/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride All
                Order allow,deny
                allow from all
        </Directory>


        ErrorLog /var/log/apache2/error.log

</VirtualHost>
%
```

Nun wird der Apache Server neu gestartet damit die Änderungen übernommen werden.

```
/etc/init.d/apache2 restart
```

Danach wird der SQL-Dump in die Datenbank eingespielt um die benötigte Struktur zu erschaffen. Dies geschieht mit dem nachfolgenden Befehl:

```
mysql -uroot -proot vmsm < /var/www/SQL/vmsm.sql
```

Nun muss die Datei */var/www/app/config/databse.conf* angepasst werden um dem Programm den Zugriff auf die Datenbank zu ermöglichen. Die Datei besteht aus einem selbsterklärendem PHP-Array, in dem die Mysql-Benutzerdaten eingetragen werden.

Nun werden noch die Rechte des tmp-Verzeichnisses der Applikation angepasst um so das Ablegen temporärer Dateien zu gewährleisten.

```
chmod -R 777 /var/www/app/tmp
```

Zum Abschluss werden die CronJobs eingerichtet, die dafür sorgen, dass die die CMDB und die Bugs-Datenbank aktualisiert wird.

```
server:~#crontab -e
0 * * * cake/console/cake Vmcheck
```

Durch Aufrufen der IP des Debiansystem im Browser kann die ordnungsgemäße Installation überpruft werden, es wird der noch leere VM Software Manager angezeigt

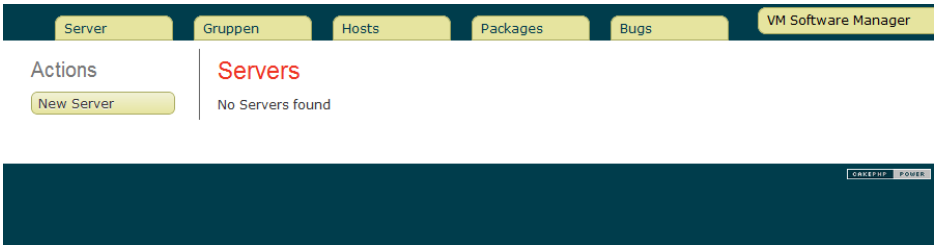


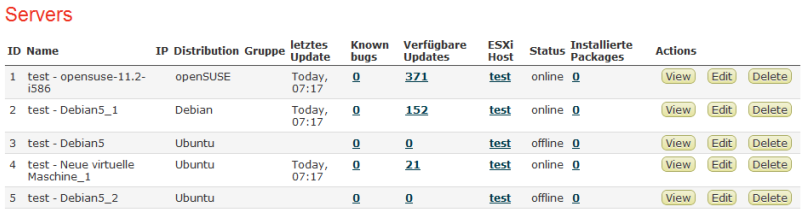
Abbildung 5.3.: VM Software Manager frisch nach der Installation

5.2.1. Softwarestände auslesen

Um den ersten Testlauf zu beginnen muss zunächst ein Hostsystem angelegt werden. Dazu auf *Neuer Server* im Bereich *Host* klicken und die Daten des vCenter Servers eingeben. Nach dem Abspeichern werden automatisch die auf dem Host/dem Cluster registrierten virtuellen Maschinen importiert. Sobald dies abgeschlossen ist finder sich nun unter *server* eine Auflistung der virtuellen Maschinen. Um den Softwarekonfigurationsmanagement-Prozess erstmalig starten zu können, müssen wir vorher noch die gerade importieren virtuellen Maschinenen bearbeiten und sie einer Distribution zuordnen (damit der Parser weiß welches Muster er anwenden muss) und die Logindaten für die Maschinen hinterlegen (damit ein Login und damit die Beschaffung der Softwarelisten möglich ist). Ist dies erfolgt kann entweder auf den nächsten Cron-Durchlauf gewartet werden oder der Prozess auch händisch iniziert werden in dem im Stammordner der VM Software Manager Installation der folgenden Befehl ausgeführt wird:

```
1 server:~#cake/console/cake Vmcheck
```

Wenn nun das Frontend erneut aufgerufen wird sollte sich in etwa folgendes Bild zeigen:

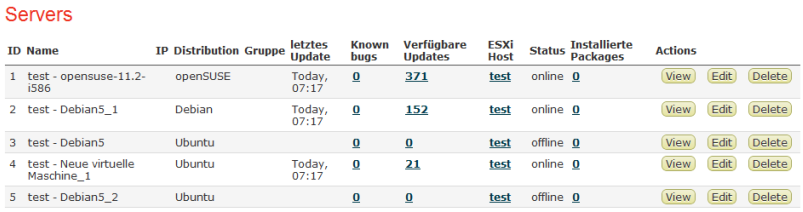


Servers

ID	Name	IP	Distribution	Gruppe	letztes Update	Known bugs	Verfügbare Updates	ESXi Host	Status	Installierte Packages	Actions
1	test - opensuse-11.2-i586		openSUSE		Today, 07:17	0	371	test	online	0	View Edit Delete
2	test - Debian5_1		Debian		Today, 07:17	0	152	test	online	0	View Edit Delete
3	test - Debian5		Ubuntu			0	0	test	offline	0	View Edit Delete
4	test - Neue virtuelle Maschine_1		Ubuntu		Today, 07:17	0	21	test	online	0	View Edit Delete
5	test - Debian5_2		Ubuntu			0	0	test	offline	0	View Edit Delete

Abbildung 5.4.: VM Software Manager nach dem ersten Prozesslauf

Nachdem nun die ersten Daten in das System eingespielt wurden sollten weitere Tests gemacht werden. Hierzu wird die Konsole einer virtuelle Maschine im vSphere Client ausgewählt um diese dann zu aktualisieren. Der Befehl apt-get update && apt-get upgrade installiert neue Softwarepakte/Updates und bringt das System somit auf den neusten Stand. Wir bereiten gleich noch einen weiteren Test vor indem wir bei einem der inaktiven Debian Servern editieren und den Hacken bei dem Feld *wakeup_for_update_check* speichern. Nun stoßen wir den Aktualisierungsprozess erneut an. Sobald der Prozess durchgelaufen ist und wir wieder ins Frontend gehen müsste sich in etwa das folgende Bild bieten:



Servers

ID	Name	IP	Distribution	Gruppe	letztes Update	Known bugs	Verfügbare Updates	ESXi Host	Status	Installierte Packages	Actions
1	test - opensuse-11.2-i586		openSUSE		Today, 07:17	0	371	test	online	0	View Edit Delete
2	test - Debian5_1		Debian		Today, 07:17	0	152	test	online	0	View Edit Delete
3	test - Debian5		Ubuntu			0	0	test	offline	0	View Edit Delete
4	test - Neue virtuelle Maschine_1		Ubuntu		Today, 07:17	0	21	test	online	0	View Edit Delete
5	test - Debian5_2		Ubuntu			0	0	test	offline	0	View Edit Delete

Abbildung 5.5.: VM Software Manager nach dem zweiten Prozesslauf

In der Serverübersicht sollte bei dem frisch aktualisierten Debiansystem die Ziffer 0 in der Spalte der *verfügbaren updates* stehen und bei dem eigentlich inaktiven Debian Server sollten

nun Pakete hinterlegt sein und das Datum des letzten Checks entsprechend gesetzt sein, da dieser kurzzeitig hochgefahren und die Softwarelisten ausgelesen wurden, bevor dieser danach dann vom dem Aktualisierungsprozess wieder runterfahren wurde.

5.2.2. Frontend testen

Um zu überprüfen ob die Pakete eines Servers korrekt erkannt und diesem zugewiesen wurden klickt man auf die Zahl der *Installierten Pakete* in der Serverübersicht. Nun gelangt man auf eine Seite in der alle installierten Softwareversionen aufgelistet werden.

Eine ähnliche Ansicht erreicht man durch Klicken auf *Packages* in der Hauptnavigation. der wesentliche Unterschied hierbei ist, dass dort verschieden Filterfunktionen zur Verfügung stehen. Testweise wird in die obenstehende Zeile *Apache* eingegeben.

Nach einem Klick auf Suche erhält man folgende Ausgabe der installierten Pakete:

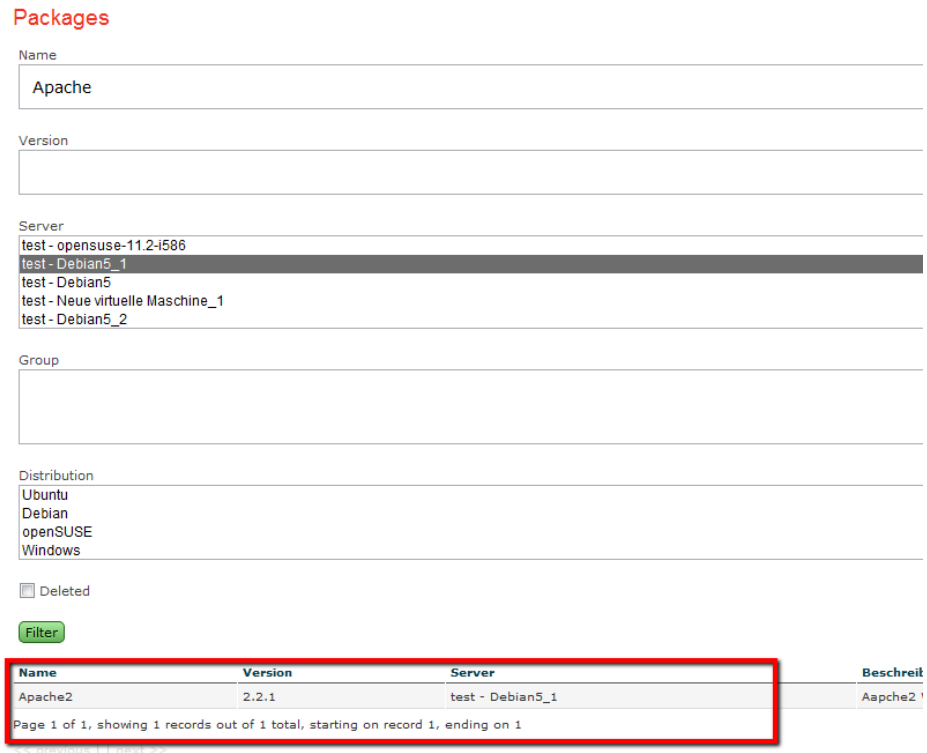


Abbildung 5.6.: Suchen im VM Software Manager

Wird die Filterung verändert indem beispielsweise noch die Distribution SuSE ausgewählt wird werden natürlich nach Klick auf Suche keine Pakete mehr angezeigt.

5.3. Analyse

Nach der Durchfürung des kleinen aber erfolgreichen Testlaufs sollte eine Auswertung stattfinden damit einen reibungslosen Einsatz des System nichts mehr im Weg steht. Der cron-job schreibt sowohl in die Logdatei /var/www/app/tmp/logs/vm_ckeck.txt als auch Zeilenweise seine Meldungen in die Datenbank. Es empfiehlt sich die Tabelle *cron_logs* stets im Auge

zu behalten, da etwaige drastische Anstiege der Bearbeitungszeit im Cronlauf ein Indiz für mögliche Fehler im Datenerfassungsprozess sein könnte.

Die Logdatei /var/www/app/tmp/logs/log.txt hat nach dem Testlauf unter anderem folgenden Inhalt (es wurden Teilbereiche entfern, da die Logdatei mehrere tausend Zeilen erzeugt hat):

```
2010-12-02 19:40:45 Vm\_check: Finished Cronjob ...
2010-12-12 19:33:56 Vm\_check: Found 36 Apache Bugs on its website
2010-12-12 19:40:19 Vm\_check: Powering off VM Test-Debian5\_2
2010-12-12 19:40:28 Vm\_check: Searching for new package or update information on VM Test-Debian5\_2
2010-12-02 19:30:33 Vm\_check: Updated package found gpg-pubkey-56b4177a -> 3dbdc284
2010-12-02 19:30:34 Vm\_check: Updated package found gpg-pubkey-3dbdc284 -> 9c800aca
2010-12-02 19:30:35 Vm\_check: Updated package found gpg-pubkey-9c800aca -> 307e3d54
2010-12-02 19:30:36 Vm\_check: Searching for new package or update information on VM 100er-Debian5\_1
2010-12-02 19:30:48 Vm\_check: Updated package found adduser-3.109 -> 3.110
2010-12-02 19:35:07 Vm\_check: Start grabbing Vms of 100er (192.168.1.100)
2010-12-02 19:35:08 Vm\_check: Finisehd grabbing Vms of 100er (192.168.1.100)
2010-12-02 19:35:08 Vm\_check: Searching for new package or update information on VM 100er-opensuse-11.2-i586
2010-12-02 19:35:20 Vm\_check: Updated package found gpg-pubkey-307e3d54 -> a1912208 on 100er-opensuse-11.2-i586
2010-12-02 19:35:21 Vm\_check: Updated package found gpg-pubkey-a1912208 -> 3d25d3d9 on 100er-opensuse-11.2-i586
2010-12-02 19:35:21 Vm\_check: Updated package found gpg-pubkey-3d25d3d9 -> 7e2e3b05 on 100er-opensuse-11.2-i586
2010-12-02 19:35:22 Vm\_check: Updated package found gpg-pubkey-7e2e3b05 -> 0dfb3188 on 100er-opensuse-11.2-i586
2010-12-02 19:35:22 Vm\_check: Updated package found gpg-pubkey-0dfb3188 -> 56b4177a on 100er-opensuse-11.2-i586
2010-12-02 19:35:23 Vm\_check: Updated package found gpg-pubkey-56b4177a -> 3dbdc284 on 100er-opensuse-11.2-i586
2010-12-02 19:35:24 Vm\_check: Updated package found gpg-pubkey-3dbdc284 -> 9c800aca on 100er-opensuse-11.2-i586
2010-12-02 19:35:24 Vm\_check: Updated package found gpg-pubkey-9c800aca -> 307e3d54 on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update Botan (3522 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (1597 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (1708 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (1774 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (2017 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (2273 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (2774 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (2595 found on 100er-opensuse-11.2-i586
2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (3132 found on
```

```
    100er-opensuse-11.2-i586
30 2010-12-02 19:35:25 Vm\_check: Available update MozillaFirefox (3422 found on
    100er-opensuse-11.2-i586
31 2010-12-12 19:33:56 Vm\_check: Powering on VM Test-Debian5\_2
32 2010-12-02 19:30:45 Vm\_check: Starting Cronjob ..
```

Der auf den ersten Blick erschlagende Datenverhau lässt sich sehr einfach analysieren und belegt einen Reibungslosen unseren Testlauf.

Im Detail beginnt jeder Prozess mit *Starting Cronjob* und beendet seinen Lauf mit *Finishing Cronjob*. Somit lässt sich genau bestimmen, wie lang der Prozess lief und was in dieser zeit alles genau gemacht wurde.

Auch können wir in zeile 3 und 31 erkennen, dass das starten und anschließende ausschalten den ruhenden Maschine funktoniert.

Am Ende des Prozesses werden die Bug-Datenbanken angesprochen. Wie wir hier in zeile 2 erkennen können hat dies auch funktoniert und die Daten von 36 bekannten Apache-Bugs in unseren Datenbank aufgenommen.

5.4. Zusammenfassung

In diesem kurzen Testlauf wurde gezeigt, wie die Software installiert wird, wie sie bedient wird und vor allem aber auch, dass sie funktioniert. Es konnten die Softwareversionen sämtlicher virtueller Maschinen erfasst werden und nach entsprechenden VMs mit bestimmter Software gesucht werden. Auch wurde gezeigt, dass die Versionierung funktioniert, in dem ein Update ausgelöst wurde und anschließend erneut die Softwarestände angesehen wurden.

6. Zusammenfassung und Ausblick

In diesem letzten Kapitel der Bachelorarbeit möchte ich meine Arbeit kurz zusammenfassen und eine kleinen Ausblick über denkbare Erweiterungen des Konzepts und der praktischen Umsetzung geben

6.1. Zusammenfassung

Ziel dieser Arbeit war es, Softwarekonfigurationsstände in VMware-Clustern zu erfassen und in einer CMDB zu verwerten.

Nach einer anfänglichen Begriffserklärung gingen wir zu der Analyse über, und betrachteten verschiedene Möglichkeiten Softwarestände zu erfassen. Diese Vorüberlegungen flossen in ein Konzept zum automatischen Erfassen der Daten in einer CMDB ein. Nach dem theoretischen Teil dieser Arbeit ging es um die praktische prototypische Umsetzung des Konzepts. Zunächst taten sich große Probleme mit dem zuerst gewählten Ansatz auf. Doch durch das frühzeitige Erkennen der Grenzen des ersten Prototypens war noch genug Zeit vorhanden einen neuen Ansatz zu erarbeiten. Nach kurzer Zeit konnte dann ein voll funktionstüchtiger Prototyp erfolgreich getestet werden.

6.2. Ausblick

Wie einleitend im Rahmen der Motivationsdarlegung für diese Arbeit schon dargelegt entwickeln sich Virtualisierungssysteme und damit speziell VMware als Marktführer mit rasender Geschwindigkeit und ringen den bisher herrschenden Nur-Host Systemen immer mehr Marktanteile ab. Somit gewinnt auch das komplexere Thema Clustering im Bereich Virtualisierung immer mehr an Oberhand. Es ist also abzusehen, dass Tools zur automatischen Erfassung speziell für VMware Cluster immer mehr an Bedeutung gewinnen, was auch das von VMware selbst bereitgestellte Update Manager Tool zeigt (welcher allerdings derzeit nur für RedHat und Windows Systeme verfügbar ist und sich auf reines Auslesen der Softwareversionen und Updaten dieser beschränkt). Natürlich konnte im Rahmen dieser Arbeit nicht allumfassend jedes Detail behandelt werden und auch mussten wir uns auf spezielle (wenn auch sehr verbreitete) Gastbetriebssysteme beschränken um den Rahmen nicht zu sprengen.

Auch wenn diese Bachelorarbeit hier sein Ende findet, so gilt dies nicht für die Entwicklung des VM Software Managers. In den letzten Monaten konnten zahlreiche Erfahrungen gesammelt werden, wobei ich kurz drei wünschenswerte Funktionalitäten erläutern möchte.

Erweiterung der unterstützten Systeme
Derzeit ist vor allem der Prozess des Erfassens der Softwarekonfigurationsstände auf eine Handvoll System beschränkt. Denkbar ist in jedem Fall eine Erweiterung des Tools um weitere Linux Systeme wie SuSE oder Gentoo oder auch anderer unixartiger Systeme wie freeBSD oder openBSD.

Automatisches erkennen des Gastsystems
Derzeit müssen leider nach jedem Anlegen eines neuen Hostes die importierten virtuellen Maschinen noch bearbeitet und einer Distribution zugewiesen werden. Ein sehr sinnvolle und Fehler reduzierende Maßnahme wär es die automatische Erkennung des Gastbetriebssystems zu implementieren und somit einen weiteren Schritt in Richtung Automatisierung machen.

Ausbau der Bug-matching-Funktion
Als kleines Gimmick entstand zunächst der Grundgedanke die erfassten Programme auf mögliche Schwachstellen anhand eines Datenbankabgleiches zu prüfen und den Administrator entsprechend drauf hinzuweisen. In der Praxis entwickelt sich diese Funktonalität immer mehr und mehr zu einem der wichtigsten Features des VM Software Managers weswegen in diese Richtung in jedem Fall weiter entwickelt werden sollte.

Auch wird die Praxis zeigen, welche Funktionen oder spezielle Workflows in dem Tool noch gefordert sind oder verändert und erweitert werden müssen. Erste Test im realen Betrieb mit 6 VMware Servern ergaben eine sehr positive Resonanz, so dass zu hoffen ist, dass das Tool im Form eines open-source Projekts weiter entwickelt und zur Marktreife gebracht wird.

Abbildungsverzeichnis

1.1. VMware Hardware Virtualisierung 2
1.2. VMware Cluster 3

2.1. Liste laufender Prozesse in Linux 7
2.2. Software Lebenszyklus einer Virtuellen Maschine 9
2.3. Windwos Registrierungs-Editor 10

3.1. Allgemeines Modell 12
3.2. Liste laufender Prozesse in Linux 14
3.3. stilisiertes CMDB Schmea 17

4.1. CMDB Datenschema 20
4.2. Der Virutell Update Manager 22
4.3. Der VM Software Manager 25
4.4. Datenbankschema VM Software Manager 28

5.1. Testsenzario 30
5.2. Testcluster 31
5.3. VM Software Manager frisch nach der Installation 32
5.4. VM Software Manager nach dem ersten Prozesslauf 33
5.5. VM Software Manager nach dem zweiten Prozesslauf 33
5.6. Suchen im VM Software Manager 34

Anhang

A. CD

Sämtlicher Quellcode und ggf. fertig kompilierte Programme sowie zusätzliche ein PDF dieser Arbeit finden sich auf beiligender CD-ROM.

Literaturverzeichnis

[DZ10] Dennis Zimmer, Bertram Wöhrmann: *VMware ESX/ESXi 4*. Galileo Computing, Bonn, 3. Auflage, 2010.

[GB08] Gerard Blokdijk, Ivanka Menken: *Configuration Management Best Practice Handbook: Building, Running and Managing a Configuration Management Data Base, CMDB - Ready to use supporting documents bringing ITIL Theory into Practice*. Emereo Pty Ltd, Newstead, 2008.

[GO09] Glenn O'Donnell, Carlos Casanova: *CMDB Imperative*. Prentice Hall, Upper Saddle River, 2009.

[Goe10] Goepel, Ralph: *Praxishandbuch VMware vSphere 4*. OReilly, Köln, 2010.

[Moo] Moore, Gordon: *MOORE'S LAW*. `http://www.intel.com/about/companyinfo/museum/exhibits/moore.htm`.

[RB08] Ralf Buchsein, Frank Victor, Holger Günther Volker Machmeier: *IT-Management mit ITIL® V3*. Vieweg+Teubner Verlag, Wiesbaden, 2. Auflage, 2008.

[VGa] VMWARE GLOBAL, INC: *VMware vCenterTM Server*. `http://www.vmware.com/de/products/vi/vc/`.

[VGb] VMWARE GLOBAL, INC: *VMware vSphere HypervisorTM (ESXi)*. `http://www.vmware.com/de/products/vsphere-hypervisor/`.

[VGc] VMWARE GLOBAL, INC: *VMware vSphereTM*. `http://www.vmware.com/de/products/vsphere/`.

[VMw] VMware: *OVF Whitepaper specification*. `http://www.vmware.com/pdf/ovf_whitepaper_specification.pdf`.