Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Jan Lepel

Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer zentralisierten Weboberfläche zur vereinfachten Umsetzung individuell erstellter Systeme

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Jan Lepel

Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen - Prototyp einer zentralisierten Weboberfläche zur vereinfachten Umsetzung individuell erstellter Systeme

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

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

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

Eingereicht am: 1. Januar 2015

Jan Lepel

Thema der Arbeit
Automatisierte Erstellung und Provisionierung von ad hoc Linuxumgebungen -
Prototyp einer zentralisierten Weboberfläche zur vereinfachten Umsetzung individuell erstellter Systeme

Stichworte
Ad hoc Umgebung, automatisierter Umgebungsaufbau und Provisionierung

Kurzzusammenfassung
Dieses Dokument …
Jan Lepel

Title of the paper
TODO

Keywords
Keywords, Keywords1

Abstract
This document …

Listings

Inhaltsverzeichnis

1 Einleitung

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

Seneca formulierte 49 n. Chr. das Gefühl welches jeder kennt. Die Zeit die er hat, nicht richtig zu nutzen. Technische Neuerungen helfen uns unsere Zeit besser zu planen, mehr Zeit in andere AktivitÃ¤ten zu stecken und unsere Prioriäten zu Ã¼berdenken. Diese Arbeit beschÃ¤ftigt sich mit dem Teil-Aspekt der Informatik, der virtualisierung von Servern im Entwicklungsumfeld.

1.1 Problemstellung

Server-Virtualisierung ist in der heutigen Zeit keineswegs mehr eine Seltenheit. Sie ist eher Standard in den meißten Unternehmen.
Virtualisierung hat in vielen Bereichen den physichen Server abgelÃ¶st, denn es mÃ¼ssen keine aufwÃ¤ndigen Planung im Vorfeld getroffen werden, der Einkauf muss nicht mehr involviert werden, das Budget braucht nicht kalkuliert werden und die Frage, was in ein paar Jahren mit der Hardware passiert, wird obsolet.

Im idealfall heisst das fÃ¼r Unternehmen: Weniger Server sind gleichbedeutend mit weniger Stellfäche, mit weniger Verkabelung oder Racks. Somit ist die Konsolidierung der ehemals großen Server-Zentren für viele Unternehmen eine direkte Konsequenz. Wodurch eine Kostenreduzierung der gesamten Infrastruktur entsteht.
Nicht nur der Finanzielle Aspekt spricht oft für die Virtualisierung, sondern auch die leichte Automatisierung, die Erhöhung der Verfügbarkeit und ....
Das Verschieben von kompletten Applikationen von einem physischen Ort zu einem anderen,.... Verbesserung der Verfügbarkeit und Business Continuity. Dazu gehören Live Migration, Storage Migration, Fehlertoleranz, Hochverfügbarkeit und Ressourcen-Management. Virtuelle Maschinen können damit leicht verschoben und vor ungewünschten Auszeiten geschützt werden.
In der physischen Welt war es bisher üblich, jeder Applikation einen eigenen Server zuzuweisen.

Damit war dafür Sorge getragen, dass die einzelnen Software-Programme sauber voneinander isoliert waren. Aber das führte auch zu einem Wust von Rechnern, von denen viele noch dazu nicht optimal ausgelastet waren. Und die Kosten für diese Server-Landschaft liefen schnell aus dem Ruder. Nicht so bei Virtualisierung. Inzwischen sind auch die nötigen Funktionen und Tools vorhanden, um VMs und die in ihnen verpackten Anwendungen sauber voneinander zu trennen. CPU, Memory und Storage können exakt ausgelastet werden, die Kosten in einem solchen Modell sinken. ...

1.2 Zielsetzung

Ziel der vorliegenden Arbeit ist es, eine Softwareprodukt zu erarbeiten, die es ermöglicht zeiteffizient temporäre virtuelle Umgebungen inklusive gewünschten zugehörigen Programmen aufzubauen und deren Administration leicht verständlich und unkompliziert aufgebaut ist. Dies soll durch Verwendung von aktuellen Softwareprodukten geschehen, die fÃ¼r diesen Bereicht im Markt etabliert sind.

....

1.3 Themenabgrenzung

Diese Arbeit greift Bekannte und etablierte Softwareprodukte im auf und nutzt diese in einem zusammenhängenden Kontext. Dabei wird keine der verwendeten Software modifiziert, sondern für eine vereinfachte Benutzung kombiniert und mit einem Benutzerinterface versehen, welches die Abläufte visuallisiert und dem Benutzer die Handhabung vereinfacht.

——Im folgenden wird darauf eingegangen, ob und wie es möglich ist, durch Verwendung von aktuellen Softwareprodukten, sowie deren automatisiertem zusammenspiel, eine eigenstÃ¤ndige Software zu entwicklen, die zeitsparend agiert—-

1.4 Struktur der Arbeit

2 Grundlagen

3 Die Software

3.1 Idee und Anforderung

3.2 Ubersicht der Komponenten

3.2.1 Vagrant

Durch die Verwendung von Vagrant wird es mÃ¶glich durch wenig Aufwand eine schnell verfÃ¼gbare Umgebung aufzubauen. Diese Arbeit bezieht sich in der Entwicklungsphase ausschliesslich auf eine Ubuntu 32Bit Version, die Online von Vagrant bereitgestellt wird. Somit wird sichergestellt, dass diese vorgefertigte virtuelle Box frei von Schadsoftware ist. Ausserdem wird damit garantiert, dass die angebotene Platform immer identisch ist. Da Vagrant auf jedem System installiert werden kann, OS X; Windows; Linux Distrubutionen, hervorragend mit diversen Provisionierern zusammenarbeitet und VM-Tools wie VirtualBox, VMWare und AWS unterstÃ¼tzt, wird es zu einem guten Allrounder. Gerade die Provisionierung von Software auf die zu startende virtuelle Maschine, macht Vagrant fÃ¼r das vorliegende Projekt attraktiv. Zwar dauert der Aufbau im Gegensatz zu Docker nicht Sekunden sondern Minuten, aber die Provisionierung ist mit das entscheidene. Ausserdem ist die hÃ¤ndische Konfiguration und die Begrifflichkeiten eingÃ¤ngiger und auf das Projektvorhaben besser zugeschnitten.

3.2.1.1 Vergleich zu Docker

Im Zusammenhang mit ad hoc Umgebungen, wird der Fokus in Diskussion auf Docker und Vagrant gelegt. Der Grundgedanke bei beiden Applikationen ist der Gleiche, aber man muss sich bewusst sein, was das Ziel der Anwendung sein soll. Beide Applikationen haben wie alles ihre Vor- und Nachteile, aber in einem sind beide Virtualisierungs-Tools gleich. Die zentrale Steuerung zum Aufbau einer Maschine, geschieht Ã¼ber eine einziges Konfig-File.
Docker ist ein Linux-only VE (Virtuel Environment)-Tool und arbeitet im Gegensatz zu Vagrant mit Operating-System-Level Virtualisierung, auch Linux Containern (LxC) genannt. WÃ¤hrend Vagrant Hypervisor-basierten virtuellen Maschinen aufbaut. FÃ¼r die sogenannten Container nutzt Docker spezielle KernelfunktionalitÃ¤ten, um einzelne Prozesse in Containern

voneinander zu isolieren.
Dadurch wird fÃ¼r den Benutzer der Eindruck erweckt, dass fÃ¼r Prozesse die mit Containern gestartet werden, diese auf ihrem eigenen System laufen wÃ¼rden. Docker wird in drei Teile unterteilt. Die Arbeitsweise ist nicht wie bei einer VM (virtuelle machine) bei der ein virtueller Computer mit einem bestimmten Betriebsystem, Hardware-Emulation sowie Prozessor simuliert wird. Ein VE ist quasi eine leichtgewichtige virtuelle Maschine. In einem Container ausgefÃ¼hrte Prozesse greifen gemeinschaftlich auf den Kernel des Hostsystems zu. Durch die KernelnamensrÃ¤ume(cnames) werden die ausgefÃ¼hrten Prozesse von einander isoliert. Allerdings ist gerade die Isolation der Prozesse, durch den gemeinsam genutzten Kernel etwas geringer, als bei der Benutzung durch einen Hypervisor.
Durch die zugrunde liegende Architektur von Vagrant, wird Vagrant gerne fÃ¼r immer gleich aufbauende Entwicklungsumgebungen benutzt. Die vielzahl an unterstÃ¼tzten Betriebsystemen macht es fÃ¼r viele Benutzer attraktiver. Allerdings kooperieren Vagrant und Docker auch hervorragend zusammen. Vagrant verspricht durch die leichtere Handhabung und Konfiguration eher das, was fÃ¼r das vorliegende Projekt entscheidend ist.

ZITAT: Docker wird in diversen Varianten als LÃ¶sung fÃ¼r das Setup und die Kapselung lokaler Entwicklungsumgebungen, als temporÃ¤r verfÃ¼gbarer Service im Rahmen von Integrationstests und als Laufzeitumgebung auf Produktionsservern eingesetzt. Gerade der temporÃ¤re Charakter von Docker-Containern wird fÃ¼r einen anderen Anwendungsfall interessant: als dynamisch erzeugte Buildsysteme mit einer projektspezifischen Buildumgebung.

Informationen: http://www.scriptrock.com/articles/docker-vs-vagrant ; https://entwickler.de/online/docker-basics-system-level-virtualisierung-125514.html

3.2.2 Ansible

3.2.2.1 Vergleich zu Salt

Da Ansible mit Salt am ehesten verwandt ist, wird auf Puppet und Chef im weiteren nicht weiter eingegangen. Puppet und Chef wiedersprechen dem vorgesehenen Konzept der leichten Konfiguration.

Salt ist wie Ansible in Python entwickelt worden. Da Salt zur Kommunikation mit seinen Clients Agenten (Minions) benÃ¶tigt, ist dies wiederrum schwer zu automatisieren. Zwar kann Vagrant mit Salt zusammenarbeiten, allerdings nicht nativ. Salt wÃ¤re eine gute Alternative, wenn nicht nur einen Provisioner haben mÃ¶chte, sondern auch ein remote execution framework. Salt implementiert eine ZeroMQ messaging lib in der Transportschicht, was die

Kommunikation zwischen Master und Minions im Gegensatz zu Chef und Puppet vervielfacht. Dadurch ist die Kommunikation zwar schnell, aber nicht so sicher, wie bei der SSH Kommunikation von Ansible. ZeroMq bietet keine native VerschlÃ¼sselung und transportiert Nachichten Ã¼ber IPC, TCP, TIPC und Multicast.

http://www.scriptrock.com/articles/ansible-vs-salt

3.3 Struktur und Zusammenspiel

3.4 Konfiguration

3.5 Webkomponente

3.6 Datenbank

3.7 Virtualisierung

3.8 Provisionierung

3.9 Kommunikation der einzelnen Komponenten

3.10 Vagrant

Bei Vagrant handelt es sich um ein Softwareprojekt, welches 2010 von Mitchell Hashimoto und John Bender 2010 ins Leben gerufen wurde. Der primäre Gedanke hinter diesem Projekt ist es, gerade Entwicklern und Teams eine schnelle und unkomplizierte Möglichkeit zu bieten, virtuelle Maschinen und Landschaften zu erstellen.
Vagrant ist also ein mächtiges Werkzeug zum virtuallisieren, der sonst oft lokalen Entwicklungsumgebungen. Gerade Teamarbeit wird dadurch vereinfacht, denn die gewünschten Maschinen können mit den gleichen Konfigurationen, Komponenten, Infrastrukturen und Bibliotheken erstellt werden.
Um die gewünsche Maschine zu visuallisieren, greift Vagrant auf VirtualBox zurück. VirtualBox ist Oracles Freeware Pandant zur kommerziellen VMware Produktlinie. Wenn gewünscht, kann auf VMware Fusion oder Amazon Web Services zurückgegriffen werden.
Die Konfiguration einer Maschine geschieht über das "Vagrantfile"; in dem Parameter wie IP Adresse konfiguriert oder Provisionierer hinzuschaltet. Da das Vagrantfile in einer Ruby Domain Specific Language geschrieben wird, bedeutet das für den Anwender, dass er es einfach mit anderen Kollegen über Versionskontrollen (z.B. Git oder Subversion) teilen kann. Bei den Provisionierern wird dem Benutzer die Freiheit gegeben, auf Bekannte wie Chef, Puppet oder Ansible zurückzugreifen.

3.11 Konfiguration

Vagrantfile beinhalten die Konfiguration jeder Vagrantmaschine. Sogar jeder Vagrant-"Landschaft". Das in Ruby Syntax geschriebene Konfigurationsfile, wird automatisch über den Befehl "vagrant initïm gewÃ¼nschten Ordner generiert oder manuell Ã¼ber jeden Editor erstellt werden. Auf Linux-Systemen ist darauf zu achten, dass der gewÃ¼nschte Ordner über entsprechende Berechtigungen verfügt. Manuelle Erweiterung des Vagrantfiles wird durch die Rubysyntax zusätzlich vereinfacht.

3.12 Ansible

3.13 Passenger

3.14 Sidekiq

4 Schluss

4.1 Zusammenfassung

4.2 Fazit

See also ?.

Literaturverzeichnis

[Seneca 2005] Seneca: *Von der KÃ¼rze des Lebens*. Deutscher Taschenbuch Verlag, 11 2005. – URL http://amazon.de/o/ASIN/342334251X/. – ISBN 9783423342513

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

Hamburg, 1. Januar 2015 Jan Lepel