TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR TECHNISCHE INFORMATIK
PROFESSUR FÜR RECHNERARCHITEKTUR
PROF. DR. WOLFGANG E. NAGEL

LCTP – Abschlussaufgabe Chef
Wintersemester 2013 / 2014

Jörg Thalheim

Dresden, June 10, 2014

Contents

1 Chef - Konfigurationsmanagementsystem (Jörg Thalheim)

1.1 Aufgabenstellung

- Analysieren Sie die Funktionsweise von Chef und gehen Sie auf Unterschiede zwischen Chef und Puppet ein.
- Wählen Sie zwei Netzwerkdienste aus, die während der Lehrveranstaltung besprochen wurden und erstellen Sie Provisionierungsvorlagen für diese.
- Beschreiben Sie, wie die Verifikation von Provisionierungsvorlagen bei der Bereitstellung von HPC-Software verwendet werden kann.

1.2 Funktionsweise von Chef

Chef ist ein Framework, welches eine automatisierte Serverkonfiguration und -verwaltung ermöglicht. Chef übernimmt dabei Aufgaben der Provisionierung (Installation der grundlegenden Dienste, Ressourcenverwaltung, Einrichtung und Konfiguration von Middleware) bis hin zum Deployment (Verteilung der eigentlichen Business-Anwendung). Der Endanwender beschreibt hierbei die Systemressourcen und ihre Zustände in der Programmiersprache Ruby. Diese Definitionen werden von dem Programm `Chef-Client` eingelesen und in notwendige Aktionen übersetzt, welche ausgeführt werden müssen, um den beschriebenen Zustand umzusetzen.
Die Gesamtheit aller Definitionen/Einstellungen nennt man das `Chef-Repo`. Ein solches untergliedert sich in mehrere `Cookbooks`. Ein Cookbook ist die Grundverwaltungseinheit in Chef. Es erfüllt einen bestimmten Teilaspekt des Systems (z.B. die Einrichtung eines Webservers Apache). Cookbooks können versioniert werden. Es können Abhängigkeiten zwischen mehreren Cookbooks angegeben werden.
Physikalische oder virtuelle Maschinen werden als `Nodes` bezeichnet. Der Node werden `Attribute`, `Rollen` und Cookbooks zugewiesen. Rollen und Cookbooks werden dazu in die sogenannte `Run-List` eingefügt. Diese gibt die Reihenfolge an, in welcher Rollen und Cookbooks angewendet werden. Rollen bieten eine Möglichkeit, Nodes zu gruppieren, welche die gleichen Aufgaben in einer Organisation erfüllen (z.B. Webserver).
Es gibt mehrere Möglichkeiten `Chef` zu betreiben:

Chef-Solo Chef-Solo ist die einfachste Ausführungsform. Alle nötigen Daten werden aus einem lokalen Verzeichnis geladen. Im Gegensatz zu `Chef-Server` und `Enterprise-Chef` wird bei Chef-Solo das Programm `chef-solo` an Stelle von `chef-client` ausgeführt. Diese Form wurde für die Umsetzung der Aufgabenstellung in Abschnitt 1.3 gewählt.

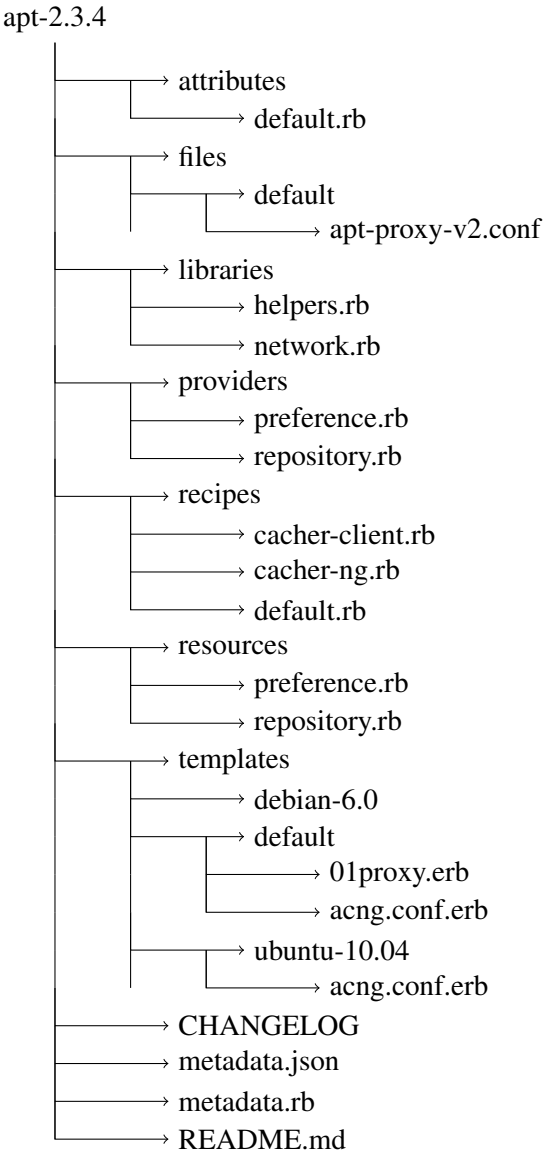
Chef-Server Hierbei authentifiziert sich `Chef-Client` über eine `REST-Api` zu einem `Chef-Server` mittels eines privaten RSA-Keys. Der Server verwaltet zentral das Chef-Repo. Der Chef-Client bekommt von diesem alle nötigen Informationen für die zu provisionierende `Node`. Chef-Server bietet eine webbasierte GUI für die Administration an. Die Attribute aller Nodes sind über die eingebaute Suchmaschine `Solr` durchsuchbar.

Enterprise-Chef/Hosted-Enterprise-Chef Enterprise-Chef bietet zusätzlich zu den Funktionen der Opensource-Version Chef-Server eine rollenbasierte Benutzerverwaltung, bessere Überwachung,

eine verbesserte Weboberfläche sowie Push-Deployment an. Während bei Hosted-Enterprise-Chef die Firma Chef die Infrastruktur des Chef-Server betreibt und die Skalierung des Dienstes übernimmt, befindet sich im Falle von Enterprise-Chef der Server in der eigenen Organisation [Che14b].

1.2.1 Aufbau eines Cookbooks

Figure 1: Ordnerstruktur eines Cookbooks am Beispiel des apt-Cookbooks dargestellt.



Die Verzeichnisnamen und die Datei `metadata.rb` sind fest vorgeben. Jedes Verzeichnis hat seine eigene Funktion. Dies hat den Vorteil, das man sich schnell in neuen Cookbooks zurecht findet.

attributes Attribute sind einfache Schlüssel-Wert-Beziehungen und setzen Standardwerte für das Cookbook. Die Schlüssel sind hierarchisch organisiert. In der Regel ist die höchste Ebene der Name des Cookbooks (z.B. `normal.mysql.client.packages`). Werte können Strings, Zahlen oder Arrays sein. Die gesetzten Attribute können in Rollen, Nodes oder von anderen Cookbooks überschrieben werden. Hierfür werden die Attribute mit den verschiedenen Prioritäten `default`,

`force_default`, `normal` und `override` (aufsteigender Wertigkeit) gesetzt. Attribute mit einer höheren Priorität überschreiben den Wert von Attributen mit einer niedrigeren Priorität.

files In diesem Verzeichnis können statische Dateien eingefügt werden, welche auf dem Zielsystem in das entsprechende Verzeichnis kopiert werden können.

libraries In diesem Pfad können Hilfsfunktionen und Spracherweiterungen definiert werden.

resources Ressourcen beschreiben die Bestandteile eines Systems. Eine Ressource kann z.B. eine Datei, ein Prozess oder ein Paket sein. Man beschreibt, welchen Zustand (Action in Chef genannt) diese Ressource haben soll und Chef versucht, diesen Zustand herzustellen. Chef liefert bereits viele wichtige Ressourcen mit. In Cookbooks können darüber hinaus eigene Ressourcen definiert werden.

providers Während Ressourcen nur die Schnittstelle mit allen Attribute beschreiben, die gesetzt werden können, ist der Provider eine konkrete Implementierung. Deswegen muss für jede Ressource mindestens ein Provider existieren. Es kann mehrere Provider für eine Ressource geben, um zum Beispiel mehrere Plattformenvarianten oder Betriebssysteme abdecken zu können (z.B. bei Paketmanagern oder Initsystemen - .1). In eigenen Cookbooks erstellte Ressourcen/Provider nennt man LWRP (englische Abkürzung für `Lightweight Resources and Providers`).

recipes In Recipes werden Ressourcen instantiiert, welche nötig sind, um das gewünschte Ziel zu erreichen. Dabei können Abhängigkeiten zwischen Recipes angegeben werden.

definitions Ressourcen, welche häufiger in verschiedenen Recipes in ähnlicher Form benötigt werden, können in eine `Definition` ausgelagert werden. Ein Beispiel ist das Generieren eines SSH-Schlüssels für verschiedene Nutzer auf dem System. Für komplexere Konstrukte sollten jedoch LWRPs (siehe oben) bevorzugt werden.

templates Häufig werden dynamisch generierte Dateien benötigt, um zum Beispiel Konfigurationsdateien zu erzeugen. In Chef wird für diesen Zweck die Templatesprache eRuby (Embedded Ruby) verwendet. In ERB-Templates wird Rubyquellcode ausgeführt, der sich zwischen den Tags `<%` und `%>` befindet. Dies erlaubt es einerseits den Inhalt von Variablen oder den Rückgabewert von Methoden einzufügen, andererseits können in Templates Kontrollstrukturen wie If-Statements und Schleifen verwendet werden.

metadata.rb In der Datei `metadata.rb` kann der Name des Cookbook, die eigene Version, eine Beschreibung sowie Abhängigkeiten zu anderen Cookbooks angegeben werden.

Listing 1: Beispiel ERB-Template:

```
Diese Zeile wird beim Rendern ohne Aenderung uebernommen
<%# Ein Kommentar%>

Diese Node heisst: <%= @node.name %>

<% if node[:platform] == "ubuntu" -%> <%# Bedingte Anweisung %>
Diese Zeile erscheint auf Ubuntu-basierten Nodes.
```

```
<% else %>
Diese Zeile erscheint auf nicht Ubuntu-basierten Nodes.
<% end -%>

<%# Listet in einer Schleife alle Blockdevices des Node auf %>
<% @node.block_device.each do |block_device, attributes| %>
<%= block_device %>: <%= attributes.join(", ") %>
<% end %>
```

1.2.2 Ablauf einer Provisionierung

Der genaue Ablauf wurde der Onlinedokumentation ([Che14a]) von Chef entnommen. Wie schon zu Beginn erwähnt, wird die Provisionierung von einem Programm namens `Chef-Client` durchgeführt. Je nach gewählter Umgebung kann dieser unterschiedlich gestartet werden:

- periodisch vom Scheduler `Cron`
- permanent als Systemdienst (z.B. bei Enterprise Chef)
- manuell (z.B. bei Vagrant - siehe 1.3)

Als erstes lädt dieser Prozess seine Konfiguration aus der Datei `client.rb`. In dieser stehen beispielsweise die URL des Chef-Server und der Name des Node. Letzteres ist wichtig, um die Node in Chef einordnen zu können und die richtigen Einstellungen zuzuweisen. Alternativ kann der Name auch von der Bibliothek Ohai gesetzt werden, in dem auf den Hostnamen oder der FQDN (Fully Qualified Domain Name) zurückgegriffen wird. Ohai sammelt systemrelevante Daten wie Details über Hardwarekomponenten (Anzahl der CPUs, Größe und Art des RAMs, Netzwerkanbindung, Festplatten/SSDs, ...), Informationen über die Plattform (Art des Betriebssystems und sowie dessen Version, installierte Anwendungssoftware) und die laufenden Prozesse. Diese Informationen sind durch eigene Ohai-Plugins erweiterbar und können im Provisionierungsprozess genutzt werden, um weitere Entscheidungen zu treffen. Sie sind darüber hinaus auch auf dem Server gespeichert und für andere Clients abrufbar.
Nach dem alle Einstellungen eingelesen sind, verbindet sich der Chef-Client mit dem Chef-Server. Die Authentifizierung erfolgt über dem vorher auf dem Node abgelegten RSA-Schlüssel. Für Administratoren gibt es einen Validator-Key. Mit diesem kann ein Node auf dem Server registriert werden und so ein Client-Schlüssel generiert werden.
Anschließend werden zuvor gesetzte Attribute und die Run-List vom Server übertragen. Im 1. Durchlauf oder bei Verwendung von Chef-Solo sind diese Daten nicht vorhanden. Stattdessen kann eine Datei im JSON-Format angegeben werden, um die Attribute und die Run-List für die Node zu spezifizieren. Außerdem ist es möglich eine Run-List auf dem Chef-Server einzustellen, welche ausgeführt wird, wenn die Node keine eigene Run-List besitzt.
Durch Auswertung der eingebunden Rollen und Recipes werden die benötigen Cookbooks ermittelt. Der Client fordert eine Liste aller darin enthaltenen Dateien und deren Checksumme an. Alle geänderten oder neuen Dateien werden heruntergeladen und lokal gespeichert.
Nun werden die Attribute zurückgesetzt und aus den Cookbooks, Rollen und dem Node neu generiert und entsprechend ihrer Priorität gesetzt. Die Ressourcen aus den Cookbooks werden geladen und in der

Ressource-Collection zusammengefasst. Nachdem alle Definitionen und Bibliotheken geladen wurden, werden schließlich die Recipes verarbeitet. Die darin erstellten Ressourcen beschreiben das System. Für jede Ressource wird der Zustand festgelegt.
Im nächsten Schritt folgt das sogenannte `Converging` (englisch für Angleichen). Es werden alle Ressourcen Schritt für Schritt abgearbeitet. Dabei wird für jede Ressource der für die Plattform zugehörige Provider ausgewählt. Dieser überprüft den aktuellen Zustand der Ressource und verändert falls notwendig das System, um den Sollzustand zu erreichen. Zum Schluss überträgt Chef-Client die aktualisierten Attribute auf den Server, von welchem sie in `Solr` indexiert werden.
Es besteht die Möglichkeit, Handler vor oder nach der Provisionierung auszuführen. Diese können im Fehlerfall Benachrichtigungen an das Monitoringssystem oder per Email verschicken. In letzten Abschnitt (1.4.2) wird dieser Mechanismus genutzt, um Tests auszuführen.

1.2.3 Vergleich mit puppet

Historischer Kontext Ein ebenfalls weit verbreitetes Konfigurationsmanagementsystem ist Puppet. Es ist das Ältere der beiden Projekte. Während das erste Puppet-Release bereits im Jahre 2005 von den Puppet Labs veröffentlicht wurde, erschien Chef erst 4 Jahre später im Jahr 2009. Chef wurde stark von Puppet beeinflusst. Der Erfinder von Chef, Adam Jacob, war selbst langjähriger Puppetnutzer, bevor er Chef schrieb. Seine damalige Firma betreute als Unternehmensberater mehrere Firmen bei der Provisionierung der Infrastruktur bis hin zum Deployment der Anwendung. Dabei kam Puppet zum Einsatz. Mit steigender Anzahl der Kunden, wuchs nach Aussagen von Adam Jacob der Aufwand bei der Verwaltung der Puppet-Konfiguration. Diese mussten häufig für jeden Kunden stark angepasst oder neu geschrieben werden. Aus diesem Grund begann er an ein neues Deploymentsystem zu schreiben. Damals trug es noch den Namen `Marionette`. Dabei verwendete er, wie schon bei Puppet, die Programmiersprache Ruby zur Implementierung des Clients. Ein wichtiges Designziel seines neuen System war es, bessere Abstraktionsmöglichkeiten zu schaffen, um damit die Wiederverwendbarkeit zu erhöhen (Quelle: [Che14c]). Anzumerken ist, dass seit der damals veröffentlichten Puppetversion (0.24.5) neue Funktionen und Spracherweiterungen zu Puppet hinzugefügt wurden, um dieses Problem zu adressieren. ([PL14b])

Sprache Während bei Chef die Konfiguration in Ruby geschrieben wird, besitzt Puppet eine eigene Konfigurationssprache. Puppet's Sprache ist im Gegensatz zu allgemeinen verwendeten Sprachen (engl. General-Purpose-Languages, kurz GPL) wie Ruby, Java oder C/C++ eine domänspezifische Sprache (engl. Domain-Specific-Language - DSL). Eine DSL ist eine speziell für den Anwendungszweck geschriebene und optimierte Sprache. Sie enthält häufig Elemente und Ausdrücke, welche es erlauben, Probleme der Anwendungsdomäne effizient zu lösen. Es wird häufig auf umfangreiche Standardbibliotheken und Sprachkonstrukte verzichtet, die in GPLs üblich sind. Puppet's Sprache wurde an das Konfigurationsformat der Überwachungssoftware Nagios angelehnt ([PL14c]). Sie ist deklarativ gehalten und soll möglichst einfach erlernbar sein (auch für Administratoren ohne programmiertechnischen Hintergrund). Der Schwerpunkt liegt auf der Beschreibung von `Ressourcen`. Die Sprache besitzt Kontrollstrukturen wie Case- und If-Statements. Es gibt Datentypen wie `Strings`, `Booleans`, `Arrays`, `Reguläre Ausdrücke` und `Hashes`. Diese können in Variablen gespeichert werden. Die Standardbibliothek von Puppet stellt Funktionen bereit, um auf diesen Datentypen einfache Operationen auszuführen. Allerdings

ist es nicht möglich, Schleifen auszuführen. (Diese Funktion ist momentan als `experimentell` markiert). Funktionen können nicht direkt in Puppet's Sprache definiert werden. Stattdessen werden diese als Erweiterung des Parsers in Ruby implementiert, was wiederum den Nachteil hat, dass dafür eine weitere Sprachen erlernt werden muss. Manche Unternehmen und Organisationen greifen bevorzugt auf Puppet zurück, weil es einfacher ist, neue Mitarbeiter ohne Rubykenntnisse in diesem Framework zu schulen. Andere wiederum bevorzugen die Flexibilität von Ruby. Facebook gab dies als einen der Grund an für einen Umstieg im Jahre 2013 von `CFEngine2` auf `Chef 11` [Che13b].

Communities Das strukturelle Gegenstück zu `Cookbooks` in Chef ist das `Modul` in Puppet. Diese werden in der Nutzergemeinschaft entwickelt. Da Puppet älter ist, ist anzunehmen, dass hierfür mehr Module zur Verfügung stehen, als Cookbooks für Chef. Die primäre Distributionsquelle ist Puppet-Forge, in dem **2206** Modul zur Verfügung stehen (Stand: 31.03.2014). Für Chef gibt es eine ähnliche Community-Seite mit **1368** Modulen, (Stand: 31.03.2014 - ermittelt über die API). Zu einer weiteren wichtigen Quelle hat sich die Plattform Github für beide Projekte entwickelt. Für einen Vergleich wurde die Anzahl der Suchtreffer für Projekte, die die Begriffe "Chef" und "Puppet" in der Suchmaschine auf Github herangezogen. Github filtert Forks (Abspaltungen) von Projekten aus den Suchergebnissen heraus und schlüsselt die Ergebnisse nach Programmiersprache auf. Es wurden alle Sprachen in beiden Projekte mit weniger als 100 Suchtreffer aus Übersichtlichkeitsgründen nicht in das Diagramm übernommen (siehe Tabelle 2). Eine Stichproben der Ergebnisse, dass die Suchtreffer sich überwiegend mit den eigentlichen Projekten Chef und Puppet beschäftigen. Anzumerken ist, dass Zielgruppe von Puppet überwiegend Systemadminstratoren aus besteht, während Chef auch von vielen Entwicklern genutzt wird. Letztere verwenden bevorzugt Github.

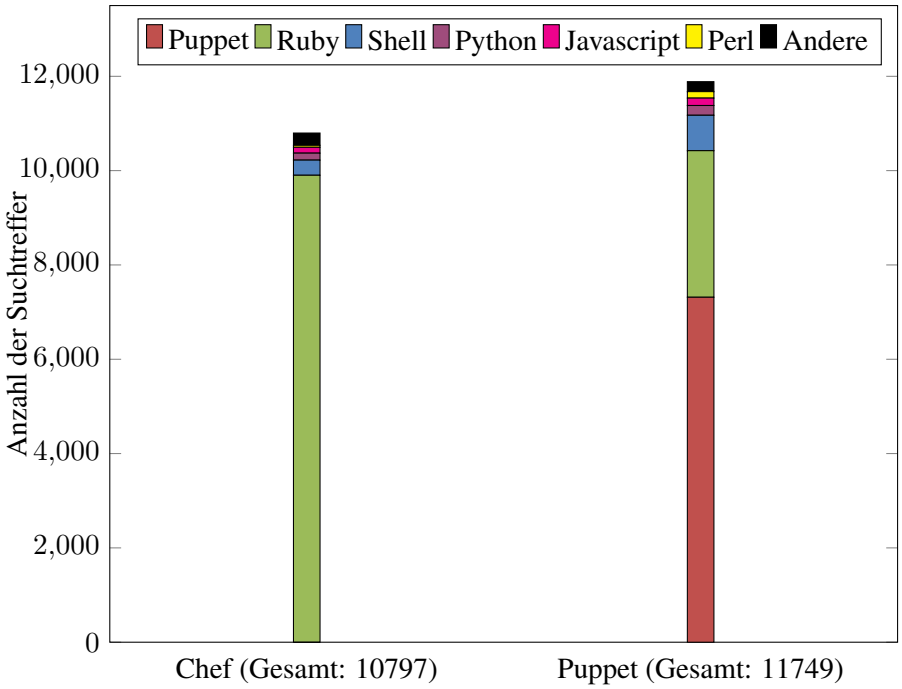


Figure 2: Anzahl der Suchtreffer auf Github aufgeschlüsselt nach Programmiersprache für die Begriffe "Chef" und "Puppet".

Table 1: Ausgangsdaten für das Diagramm

Sprache	**Ruby**	**Puppet**	**Shell**	**Python**	**Javascript**	**Perl**	PHP	Java	VimL	CSS	C	C++
Chef	9,902	-	321	148	124	42	56	88	-	31	48	37
Puppet	3,108	7,315	751	207	157	137	82	42	64	23	-	-

Eine weitere wichtige Statistik für Opensource-Projekte ist die Anzahl der Abonnenten auf den jeweiligen Mailinglisten. Engagierte und aktive Nutzer/Entwickler abonnieren häufig diese, wodurch sich die Größe der Community qualitativ vergleichen lassen.

chef@lists.opscode.com Community-Mailingliste, 1620 Abonnenten, Quelle: [Che14d], Stand: 31.03.2014

chef-dev@lists.opscode.com Entwickler-Mailingliste, 652 Abonnenten, Quelle: [Che14e], Stand: 31.03.2014

puppet-users@googlegroups.com Community-Mailingliste, ~7000 Abonnenten, Quelle: [PL14a], Stand: 01.04.2014

Die Anzahl der verfügbaren Module, veröffentlichte Githubprojekte und der Abonnenten auf den Mailinglisten weisen darauf hin, dass Puppet nach wie vor eine größere Community hat.

Funktionsweise Anstelle von Recipes werden in Puppet Manifeste geschrieben. Das sind Dateien, die auf den Suffix .pp enden und sich in dem Ordner `manifests` im Modul befinden. Jedes Manifest definiert eine Klasse, eingeleitet durch das Schlüsselwort `class`. Der Namen dieser Klasse wird aus dem Modulnamen und dem Manifest-Namen gebildet. Wenn das Modul `foo` das Manifest `bar` enthält, ist der Name der Klasse `foo::bar`. Eine Ausnahme bildet das Manifest `init.pp`, bei dem die Klasse nur `bar` lauten würde. Diese Benennungskonvention wurde in Chef übernommen, um Recipes in Cookbooks zu referenzieren. Allerdings werden in Recipes keine separaten Objekt definiert und der ganze Inhalt der Datei bildet das Recipe.
Eine Klasse in Puppet kann über Parameter konfiguriert werden. Parameter werden im Kopfteil der Klasse deklariert und können Standardwerte besitzen. Chef besitzt mit `Attributen` ein vergleichbares Konzept. Allerdings werden Attribute getrennt von den Recipes definiert und sie werden dem Node-Objekte zugewiesen. Die Attribute stehen somit allen Recipes zu Verfügung und können an verschiedenen Stellen überschrieben werden. In Puppet 3 wurde diese Trennung von Code und organsationsspezifischen Daten durch die Erweiterung `Hiera` ebenfalls eingeführt. Klassenparameter werden automatisch in der `HieraDB` gesucht und gegebenenfalls überschrieben. In älteren Versionen von `Puppet` wurden Einstellungen für die Nodes in der zentralen `site.pp`-Manifest verwaltet. Hiera ersetzt die `site.pp` weitest gehend. Durch die Funktion `hiera_include` können Klassen im Hiera-Backend gesetzt werden (ähnlich der Run-List in Chef).
Ressourcen heißen in Puppet `Types`. Puppet liefert wie Chef bereits eine Reihe von Types mit. Diese werden Core-Types genannt. Wie auch in Chef können Types in Puppet mehrere plattformspezifische Provider besitzen. Es ist möglich, eigene Types zu definieren, auch Custom-Types genannt (Ähnlich LRWP in Chef). Die Implementierung der Types/Provider erfolgt in Ruby im Verzeichnis `lib/puppet`.

Die Zustände einer Ressource können in Puppet über das Setzen des Parameters `ensure` festgelegt werden (vergleichbar mit `action` in Chef).
Ein weiteres häufig genutztes Entwurfsmuster, um Ressourcen zu gruppieren, ist der `Defined-Type`. Dieser ist das Äquivalent zur aus Chef bekannten `Definition`. Ein Defined-Type kann im Gegensatz zum Custom-Type auch direkt in der Puppet-Sprache mit dem Schlüsselwort `define` erstellt werden.
Vor der eigentlichen Provisionierung werden Informationen über das System zu gesammelt. Dabei wird auf die Bibliothek Facter zurückgegriffen. In frühen Versionen von Chef wurde die gleiche Bibliothek verwendet, bevor später Ohai integriert wurde.
Puppet nutzt die gleiche Template-Syntax wie Chef, welche in Quellcodelisting 1 vorgestellt wurde, um Dateien auf dem System zu generieren. Der einzige Unterschied bei Chef ist die Funktion für verschiedene Plattformen und -versionen verschiedene Template-Varianten der gleichen Datei im Cookbook vor halten zu können. Die Varianten werden durch Unterordner im Verzeichnis `templates/` unterschieden (z.B. `templates/windows` oder `templates/ubuntu-12.04`). Falls kein der Plattform entsprechende Ordner existiert sucht Chef im Verzeichnis `templates/default`.
Ein wesentlicher Unterschied zwischen Puppet und Chef ist die Reihenfolge der Ausführung von Ressourcen. Chef überprüft die Ressourcen in der Reihenfolge, in der sie in der Run-List und in den Recipes geladen werden. Puppet sortiert Ressourcen derartig um, dass möglichst wenig Veränderungen am System vorgenommen werden müssen um den gewünschten Zustand zu erreichen. Zum Beispiel, wenn an mehreren Stellen eine Konfiguration für einen Dienst verändert wird, sollte dieser nur einmal neu gestartet werden müssen. Bei Puppet spricht man von modellbasiertem Konfigurationsmanagement, während Chef ein codebasiertes Konfigurationsmanagement ist. Da manche Ressourcen voneinander abhängen, kann durch die Angabe der Parameter `before` und `require` die Reihenfolge festgelegt werden. Über die Parameter `notify` und `subscribe` können darüber hinaus Ressourcen aktualisiert werden, wenn sich eine Abhängigkeit geändert hat (z.B. kann ein Dienst neu gestartet werden, wenn sich die dazu gehörige Konfiguration verändert hat). In Chef kann Letzteres über die Parameter `notifies` und `subscribes` angegeben werden.

Architektur Wie auch Chef bietet Puppet verschiedene Betriebsmodi. Im einfachsten Fall wird mit dem Befehl `puppet apply` ein lokales Manifest geladen werden (vergleichbar mit Chef-Solo). Das Äquivalent zum Chef-Server in Chef ist bei Puppet der Puppet-Master, zu welchem sich der Client `Puppetd` verbindet und mittels SSL-Zertifikaten authentifiziert. In der Standarteinstellung setzt Puppetmaster auf den verhältnismäßig einfachen Webserver `WEBrick` Dieser skaliert allerdings nicht auf mehrere CPU-Kerne, da nur ein Prozess und Thread gestartet wird. Für Installationen mit mehr als 10 Knoten werden Passenger oder Mongrel als Applikationsserver empfohlen, wobei Nginx als Load-Balancer fungiert. Ein beliebter Ansatz zum Skalieren größerer Cluster ist das Verwalten der Manifeste in einem Git-Repository, wobei ein Cron-Job periodisch die Manifeste vom Git-Server lädt und Puppet ausführt.
Während Chef-Server bis zur Version 10 wie Puppetmaster in Ruby geschrieben war, wurde der API-Teil von Chef-Server in Version 11 in der Programmiersprache Erlang neu geschrieben. Die Zahl der Nodes, die von einem Server bedient werden, soll sich dabei vervierfacht haben und kann somit bis zu 10.000 Nodes bedienen (Quelle: [Che13a]). Für Puppet wurden keine Statistiken gefunden, die eine Aussage darüber treffen, wie viele Nodes pro Server betreut werden können. Allerdings ist anzunehmen, dass die Anzahl der Server, bedingt durch die genutzte Architektur, kleiner ist als bei Chef.

Zu den, von offiziell von Chef unterstützten, Plattformen gehören Windows, Mac OS X, verschiedene Linuxderivate (Debian, Ubuntu, Redhat, ...) und Solaris. Puppet bietet breiteren Support und unterstützt zusätzlich Free- und OpenBSD sowie HP-UX und AIX.

Résumé Zusammenfassend lässt sich feststellen, dass Chef und Puppet den gleichen Funktionsumfang bieten. Die Grundkonzepte sind ähnlich, so ein Anwender des einen Systems mit wenig Aufwand auch das andere System lernen kann. Die beiden Firmen, Puppet Labs und Chef, entwickeln beide ihr Produkt stetig weiter und bieten kommerziellen Support. Während Puppet auf den klassischen Systemadministrator abzielt, Chef spricht den Trend der DevOps-Kultur an, bei welcher Administration und Entwicklung stärker ineinander über gehen.

1.3 Einrichtung der Netzwerkdienste

Für die Provisionierung der Netzwerkdienste wurde Vagrant verwendet. Dies ist ein Programm, um schnell und reproduzierbar virtuelle Maschinen für Virtualbox und andere Virtualsierungslösungen zu erstellen und zu starten. Die Einstellungen hierfür werden in der Datei `Vagrantfile` hinterlegt, welche Vagrant beim Start einliest. Vagrant kann Chef beim Erstellen von virtuellen Maschinen integrieren. Zum Einsatz kam das Betriebssystem Ubuntu in der Version 12.04. Das Basisimage hierfür wurde von `Chef`, der gleichnamigen Firma, bereitgestellt. Für die Kommunikation mit Vagrant wurde die virtuelle Netzwerkkarte `eth0` konfiguriert. Ein weitere Karte (`eth1`) wird für das interne virtuelle Netzwerk zwischen den VMs zum Betreiben der Netzwerkdienste benötigt.
Vagrant bietet keine Optionen, ein virtuelles Netzwerk zu erstellen, ohne das jeder VM eine IP-Adresse fest oder DHCP unmittelbar nach dem Start zugewiesen wird. In dem genannten Netzwerk sollte allerdings DHCP von dem Head-Node bereit gestellt werden. Deswegen waren zusätzliche Kommandozeilenargumente an den Befehl `VBoxManage` im Vagrantfile nötig, welches von Vagrant genutzt wird um Virtualbox zu verwalten. Dies schränkt die Nutzung allerdings auf den Hypervisor Virtualbox ein.
Des weiteren wird Ruby auf dem Host benötigt, um beispielsweise die Tests ausführen zu können. Auf Unix-Ähnlichen Systemen kann man diese Programmiersprache mit dem Befehl:

```
$ curl -sSL https://get.rvm.io | bash -s stable
```

installiert werden. Auf dem Betriebssystem Windows kann auf den RubyInstaller zurückgegriffen werden. Um die benötigten Ruby-Bibliotheken zu installieren, müssen folgende 2 Befehle im Projektverzeichnis ausgeführt werden:

```
$ gem install bundler
$ bundle install
```

Zur Verwaltung der externen und selbst geschriebenen Cookbooks wurde die Abhängigkeitsverwaltung Berkshelf verwendet. Bei diesem werden die zu ladenden Cookbooks und die gewünschte Version in einer Datei namens Berksfile angegeben (vergleichbar mit Bundler und Gemfiles in Ruby). Berkshelf unterstützt dabei verschiedene Quellen (per API von der Communityseite von Chef, Git, lokal) und kann Abhängigkeiten zu anderen Cookbooks auflösen. Um die Cookbooks initial zu laden, muss der Befehl:

```
$ berks install
```

im Projektverzeichnis ausgeführt werden.
Für das Zusammenspiel mit Vagrant gibt es das Plugin vagrant-berkshelf, so dass die von Berkshelf verwalteten Cookbooks auch in Vagrant zur Verfügung stehen.

Für bestimmte Funktionen, wie Gemeinsame Ordner (shared folders) zwischen VM und Host, müssen die `virtualbox-client-modules` in der VM installiert sein. Diese sind in vielen Images bereits vorhanden, die es für Vagrant gibt. Allerdings muss die Virtualbox-Version des Host mit der Version in der VM übereinstimmen. Abhilfe schafft das Vagrantplugin vagrant-vbguest. Beim Start der VM installiert das Plugin die gleiche Version des Modul in der VM. Wenn Virtualbox mit Linux als Host-System ausgeführt wird, muss das Kernelmodule `vboxdrv` geladen sein. Manche Linux-Distributionen installieren dieses Module bereits während der Installation von Virtualbox. Auf Mac OS X und Windows sind keine weiteren Schritte notwendig.
Beide Plugins werden mit den Befehlen:

```
$ vagrant plugin install vagrant-vbguest
$ vagrant plugin install vagrant-berkshelf
```

installiert.
Gestartet wird die VM mit dem Befehl:

```
$ vagrant up
```

Während des ersten Starts wird die VM mit Chef provisioniert. Später kann Chef erneut mit Befehl:

```
$ vagrant provision
```

gestartet werden.
Die Netzwerkdienste sollen die Protokolle DHCP, DNS und NTP bereitstellen. Es wird wie im Praktikum zwischen `Head-Nodes` und `Compute-Nodes` unterschieden. Die Head-Node bietet die genannten Dienste an. Die Compute-Nodes fordern auf dem internen Netzwerk per DHCP eine IP-Adresse an und nutzen den DNS- und NTP-Dienst der ihr zugewiesenen Head-Node.
Die Attribute für die Rollen und den Nodes wurden in JSON-Dateien in den Verzeichnissen `roles/` und `nodes/` abgelegt. Es gibt je eine Rollen-Datei für Compute-Nodes und Head-Nodes. In der aktuellen Konfiguration erzeugt Vagrant eine Head-Node mit der FQDN `node0.lctp` und zwei Compute-Nodes (`node1.lctp` und `node2.lctp`).
Es wurden fünf Cookbooks geschrieben:

bind Für Bereitstellung des DNS-Dienstes wird Named aus dem BIND-Packet installiert. Das Cookbook richtet diesen Dienst ein und fügt die in den Attributen konfigurierten DNS-Einträge zu den entsprechenden Zonen hinzu.

dhcp Dieses Cookbook richtet den ISC-DHCP-Server ein. Neben der Zuordnung von festen IP-Adressen zu Nodes, kann ein DNS-Server und ein NTP-Server festgelegt werden.

lctp-network Dieses Cookbook ist ein Wrapper um das network_interfaces Cookbook. Wrapper-Cookbooks werden häufig dazu benutzt um bestehende Cookbooks aus anderen Quellen um Funktionalität zu erweitern. Für Compute-Nodes aktiviert das Cookbook für die DHCP in dem virtuellen Netzwerk. Im Falle eines Head-Nodes wird eine statische IP-Adresse gesetzt, der DNS-Server auf localhost festgelegt und IP-Forwarding sowie Masquerading via iptables für den Router-Betrieb aktiviert.

ntp Dieses Cookbook richtet den NTP-Deamon ein, welcher die Zeit zwischen den einzelnen Nodes synchronisiert.

main Dieses Cookbook fasst alle oben genannten Cookbooks für Compute- und Head-Node zusammen. Man könnte dies prinzipiell auch in den jeweiligen Rollen erledigen. Rollen haben allerdings den Nachteil, dass diese im Gegensatz zu Cookbooks nicht versionierbar sind und (bei Chef-Server) über alle Umgebungen identisch sind. Somit ist eine Trennung zwischen Test- und Produktivumgebung schwierig.

Es wurden folgende externen Cookbooks verwendet:

apt aktualisert die lokalen Paketlisten und den Paketcache.

network_interfaces verwaltet Debian's Netzkonfiguration

minitest-handler Sammelt alle Tests in den Cookbooks und führt diese nach der Provisionierung aus (siehe 1.4.2).

1.4 Verifikation

Wie auch in der Softwareentwicklung müssen Konfigurationssysteme getestet werden. Dies gestaltet sich oft als schwierig, weil nicht immer eine exakte Kopie des aktuellen Produktionssystems zur Verfügung steht. Mit steigender Komplexität steigt der Aufwand, Cookbooks manuell zu testen. Im Folgenden werden verschiedene Möglichkeiten aufgeführt, wie dies automatisiert werden kann.
Die erste und einfachste Methode ist der Befehl:

```
$ knife cookbook test [COOKBOOKS...]
```

Das Kommandozeilenprogramm `knife` ist ein Teil von Chef. Es ist das primäre Verwaltungsprogramm für Chef-Administratoren. Der Unterbefehl `cookbook test` überprüft den Ruby-Quellcode und die Templates des Cookbooks auf Syntaxfehler. Allerdings treten viele Fehler erst zur Laufzeit auf, im Besonderen da Ruby dynamisch typisiert ist und der Compiler beispielsweise Tippfehler in Methoden und Variablennamen nicht erkennen kann. Ein anderes Programm ist foodcritic. Es führt eine statische Codeanalyse ähnlich JSlint oder Perl::Critic auf der eigenen Codebasis durch. Dabei wird der Ruby-Quellcode gegen einen Regelsatz getestet, um so häufige Programmierfehler zu erkennen oder um Code-Konventionen innerhalb eines Projekts einzuhalten. Dieser Regelsatz kann durch eigene Regeln erweitern werden.

1.4.1 Chefspec

Chefspec baut auf das in Ruby verbreitete Testframework RSpec auf. Chefspec erweitert dabei RSpec um die Funktion, Cookbooks zu laden und stellt spezielle Matcher (RSpec-Terminologie für Assertions) bereit, um diese zu testen. Wie bereits in Abschnitt 1.2.2 erwähnt, gibt es zwei Phasen bei der Ausführung von Chef. Bei Chefspec wird der Provisionierungsprozess nur bis zur Convergingphase durchlaufen. Die eigenen Tests überprüfen nur die erzeugten `Ressourcen`. Dies hat den Vorteil, das Tests sehr schnell durchlaufen werden, da keine Änderungen an einem System vorgenommen werden müssen. Dies hat Vorteile beim Entwickeln, weil man auf diese Weise schnell Feedback bekommt. Das Zusammenspiel mehrerer Cookbooks lässt sich dadurch gut testen. Außerdem ermöglicht es, verschiedene Konfigurationen/Betriebssysteme zu testen, ohne das diese (zeit)aufwendig aufgesetzt werden müssen. Da Chefspec allerdings zu keinen Zeitpunkt Code auf dem System ausführt, sind weitere Integrationstest unerlässlich. Der folgende Test wurde aus dem selbst geschriebenen NTP-Cookbook (1.3) entnommen.

Listing 2: Chefspec-Test für das NTP-Cookbook

```
require_relative '../spec_helper'

describe 'ntp::default' do
  let(:chef_run) do
    ChefSpec::Runner.new do |node|
      node.set["ntp"]["subnets"] = ["::1", "127.0.0.1", "172.28.128.0␣mask␣
          255.255.255.0␣nomodify␣notrap␣nopeer"]
    end.converge(described_recipe)
  end

  it "should␣setup␣ntp" do
    chef_run.should install_package("ntp")
    chef_run.should render_file("/etc/ntp.conf").with_content("172.28.128.0")
  end
end
```

Im `chef_run`-Block wird dem fiktiven Node Attribute zugewiesen und das zu testende Cookbook ausgeführt. Das Ergebnis wird in diesem Beispiel in dem Objekt `chef_run` gespeichert. Gegen dieses Objekt wird getestet, ob bestimmte `Ressourcen` korrekt initialisiert wurden. In diesem Fall wird überprüft, ob das Paket `ntp` installiert werden soll und ob das Subnetz in dem Template in der Konfigurationsdatei `/etc/ntp.conf` richtig gesetzt wird.
Die Tests werden mit dem Befehl `rspec` ausgeführt. Wenn keine weiteren Argumente angegeben sind, führt dieses Programm alle Dateien unterhalb des Ordners `spec` aus, dessen Dateinamen auf `_spec.rb` enden.
Um alle drei oben genannten Testmethoden gleichzeitig ausführen zu lassen, wurde ein Rakefile geschrieben. Rake ist das in Ruby geschriebene Äquivalent zu Make, welches ein verbreitetes Buildprogramm auf UNIX-ähnlichen Plattformen ist. Die Tests werden durch den Befehl:

```
$ rake test
```

ausgeführt.
Dieser muss innerhalb Projektverzeichnisses aufgerufen werden.

1.4.2 Minitest-Handler

Minitest-Handler hingegen wird nach jedem Provisionierungsdurchgang ausgeführt. Im Gegensatz zu Chefspec nutzt es das Minitest-Framework, welches schon mit Ruby mitgeliefert wird. Um Minitest-Handler zu nutzen, muss das Recipe aus `Minitest-Handler-Cookbook` als erstes Recipe in der Node geladen werden. Minitest-Handler durchsucht beim Durchlauf in jedem anderen Cookbook, in den Unterordnern in `files/` nach dem Verzeichnis `test` und lädt alle Tests aus diesem Verzeichnis. Über die Zeile:

```
describe_recipe "ntp::default" do #
#...
end
```

wird angeben, zu welchem Test das Recipe gehört (In diesem Fall das Recipe aus dem NTP-Cookbook). Wenn das entsprechende Recipe von dem Node ausgeführt wird, wird der dazugehörige Test nach dem Provisionierungsdurchlauf ebenfalls ausgeführt. Minitest-Handler erweitert RSpec um nützliche

Methoden, um den Status des Systems zu überprüfen. Nachfolgend ist ein Beispiel aus dem Bind-Cookbook, welches in Abschnitt 1.3 erwähnt wurde:

Listing 3: `Minitest`-Test für das Bind-Cookbook

```
describe_recipe 'bind::default' do
  it "starts␣the␣named␣daemon" do
    service("bind9").must_be_running
  end
  it "should␣resolve␣dns" do
    assert_sh("dig␣localhost␣@localhost")
  end
end
```

Die Methode `assert_sh` überprüft den Exit-Code eines Befehls und schlägt fehl, wenn dieser ungleich der Zahl Null ist, während die `service`-Methode den Status eines Systemdienst überprüft. Weitere Testmethoden sind zum Beispiel das Überprüfen von Verzeichnissen, Inhalte von Dateien oder Mountpoints. Viele Fehler werden in der Regel schon von den Provider erkannt und festgestellt. Minitest-Handler kann dies Erweitern um protokollspezifische Tests durchzuführen oder das Testen von Funktionalität bestimmter Dienste.

1.5 Zusammenfassung

Chef

References

[Che13a] CHEF: *Opscode Unleashes New Generation of Chef*. http://www.getchef.com/press-releases/opscode-unleashes-new-generation-of-chef/. Version: Februar 2013

[Che13b] CHEF: *Scaling systems configuration at Facebook - Phil Dibowitz*. http://www.youtube.com/watch?v=SYZ2GzYAw_Q. Version: April 2013

[Che14a] CHEF: *About the chef-client Run*. http://docs.opscode.com/essentials_nodes_chef_run.html. Version: März 2014

[Che14b] CHEF: *Enterprise-class features and support*. http://www.getchef.com/enterprise-chef/#features-and-support. Version: März 2014

[Che14c] CHEF: *History of Chef: What's in a Name?* http://www.youtube.com/watch?v=Ia2ItmjRsw8&feature=plcp. Version: März 2014

[Che14d] CHEF: *Opscode Mailing Lists*. http://lists.opscode.com/sympa/info/chef. Version: März 2014

[Che14e] CHEF: *Opscode Mailing Lists*. http://lists.opscode.com/sympa/info/chef-dev. Version: März 2014

[PL14a] PUPPET-LABS: *Anfrage auf Twitter*. https://twitter.com/puppetlabs/status/450760644329881600. Version: März 2014

[PL14b] PUPPET-LABS: *Docs: History of Puppet Language Features*. http://docs.puppetlabs.com/guides/language_history.html#puppet-language-features-by-release. Version: März 2014

[PL14c] PUPPET-LABS: *Docs: Language: Basics*. http://docs.puppetlabs.com/puppet/latest/reference/lang_summary.html#compilation-and-catalogs. Version: März 2014

.1 Initsystem

Prozess, der in einem Betriebsystem alle nachfolgenden Prozesse verwaltet und startet.