

Deliver with Puppet

Anders Malmborg und Michael Haslgrübler
18. September 2012

1 Einleitung

1.1 Ausgangssituation

Eine prekäre Situation, eine mehrere Teams entwickeln, unabhängig voneinander mit agilen Methoden, eine Vielzahl von modularen Frameworks und Applikationen. Mithilfe von [Jenkins] wird stündlich kompiliert und integriert um sicherzustellen dass das alles auch zusammenpasst.

Jede Applikation beinhaltet außerdem Konfigurationen, im banalsten Fall sind das Einstellungen für den Zugriff auf eine Datenbank, im komplexesten Fall Schalter welche Teilfunktionalitäten aktivieren. Die Installation und Konfiguration gestaltet sich aber mit zunehmender Größe der Applikation jedoch so komplex, dass das initiale Setup einer Applikation schon Stunden dauern kann, auch wenn schon eine handvoll Scripts einen Großteil der Tätigkeiten automatisierten.

Eine Installation von der selben Anwendung sieht auf verschiedenen Maschinen unterschiedlich aus - **frei nach dem Motto viele Wege führen nach Rom - aber welcher ist der Beste?**

Man kann außerdem davon ausgehen, dass die gleiche Installation von der selben Anwendung auf zwei Rechner unterschiedlich aussieht – frei nach dem Motto viele Wege führen nach Rom. Zusätzlich dazu sind die Anwendungen im internationalen Einsatz, werden mehrsprachig getestet und betrieben und die laufende Entwicklung erweitert ständig die Funktionalität.

Kurz zusammengefasst, Veränderungen passieren am laufenden Band. Für die reine Softwareentwicklung ist Continuous Integration mit Jenkins und Co eine Toollandschaft entstanden welche das Problem löst. Für die Betriebsführung und Konfiguration haben wir uns nach einer Lösung umgesehen und mit [Puppet] und [Chef] Wege gefunden dieser ständigen Veränderung habhaft zu werden.

1.2 Ziel

Ziel unserer Lösung, für die Betriebsführung und Konfiguration, sollte es sein, die aktuellen Entwicklungen an den Mann bzw. Server zu bringen, vollautomatisiert. In der Entwicklung heißt das in erster Linie Softwarecode der committed wird, der gebaut werden kann und die automatisierten Tests besteht soll auch deployed werden. Damit können wir gewährleisten bzw. überprüfen dass die Software zu jedem Zeitpunkt einsatzbereit ist und nicht nur auf einem Entwicklungs-PC funktioniert.

Für die Qualitätssicherung heißt das, dass ein Softwarepaket einer Anwendung bei Übergabe von der Entwicklung nur einmal zentral hinterlegt werden muss und alle QA Server in allen möglichen Konfigurationsarten und Sprachen automatisch auf den neusten Stand der Anwendung gebracht werden.

Für den Produktiveinsatz heißt das auch hier alle Server auf Knopfdruck aktualisiert werden können und die Server einen definierten und bereits in QA getesteten Zustand sind und bleiben.

Um diese Ziele zu erreichen sind einige Veränderungen notwendig. In diesem Artikel möchten wir uns auf die notwendigen Änderungen im Entwicklungsprozess eingehen und wie wir diese mit Puppet umgesetzt haben.

Konfigurationsfehler durch Divergenz werden in allen Stages des Softwarelifecycleprozesses durch die Vollautomatisierung vermieden - **entweder funktioniert es überall oder nirgends**

1.3 Puppet Einführung

Puppet benutzt eine Domain-Specific-Language(DSL) um den Zustand eines System zu beschreiben. Der Code wird organisiert in Manifeste und Module.

Ein Manifest ist ein Puppet "Program". Module sind für die Puppet-EntwicklerIn ähnlich wie Libraries für eine Java-EntwicklerIn. Ein Großteil der Manifeste und Module machen die Definition von Ressourcen aus, eine Ressource ist ein atomarer Typ eines Systems, es entspricht einer physischen Identität eines Computersystems. Ein Beispiel für eine solche Ressource, wäre ein Benutzer oder eine Datei. In , siehe. Listing 1 wird der Ressource `user puppetdemo` und dazugehörige Ressource `file /home/puppetdemo` definiert. Der Aufruf `sudo puppet apply manifest/user.pp` führt es aus und muss mit 'sudo' aufgerufen werden damit Verzeichnis und Benutzer angelegt werden können. Mit `puppet resource user puppetdemo` wird die Informationen zum neu angelegten Benutzer ausgegeben., siehe. Listing 2

```

1 $ cat manifest/user.pp
2 node default {
3
4     user {
5         'puppetdemo' :
6             ensure => present,
7             home  => '/home/puppetdemo',
8             shell => '/bin/bash',
9     }
10    file {
11        '/home/puppetdemo' :
12            ensure => 'directory',
13            owner  => 'puppetdemo',
14            group  => 'puppetdemo',
15    }
16 }
17 $ sudo puppet apply manifest/user.pp
18 notice: /Stage[main]/Node[default]/User[puppetdemo]/ensure: created
19 notice: /Stage[main]/Node[default]/File[/home/puppetdemo]/ensure: created
20 notice: Finished catalog run in 0.36 seconds

```

Listing 1: User mit Puppet anlegen

```

1 user { 'puppetdemo':
2     ensure => present,
3     gid    => '1004',
4     home   => '/home/puppetdemo',
5     shell  => '/bin/bash',
6     uid    => '1002',
7 }

```

Listing 2: Anzeige der Benutzerinformation in Puppet

Eine Aufzählung der vom Puppet unterstützten Ressourcen wird mit `puppet resource --types` ausgegeben. Die Ressourcen sind im Core Types Cheat Sheet http://docs.puppetlabs.com/puppet_core_types_cheatsheet.pdf gut beschrieben.

Für die Entwicklung von Puppet Module und Manifeste bietet sich [Geppeto] an. Geppeto bringt Code Completion und Syntax Highlighting mit und kommt als Standardanwendung oder als Plugin für eine bestehende Eclipse-Installation.

1.4 Testbox mit Vagrant

Für die Entwicklung und Tests von Puppet Modulen und Manifeste empfiehlt sich eine virtuelle Maschine. Puppet Labs stellt solche für VMware und VirtualBox zur Verfügung unter <http://info.puppetlabs.com/download-learning-puppet-VM.html>.

Eine andere Möglichkeit ist [Vagrant]. Vagrant ist eine Konfigurationstool für die Verwaltung von Virtuellen Maschinen mit VirtualBox. Es kann in weiterer Folge auch Puppet,

Chef oder Shell Scripts benutzen kann um die virtuelle Maschine zu konfigurieren. Analog zu Virtualbox kann Vagrant via Paketmanager für Linux installiert werden oder von den entsprechenden Downloadseiten runtergeladen werden. Wir verwenden für diesen Artikel Vagrant.

Eine Liste mit vorgefertigten Vagrant Boxen gibt es übrigens auf <http://www.vagrantbox.es/>

Nachdem Vagrant und VirtualBox installiert worden sind, können wir eine Box zum Testen aufsetzen. In unserem Fall verwenden wir eine 64 Bit Version von Debian Squeeze. Diese wurde von uns für diesen Artikel neu erstellt und beinhaltet eine Minimalinstallation mit den für Vagrant üblichen Vorbereitungen: SSH Key Setup, VirtualBox Guest Additions, Puppet und Ruby. http://vagrantup.com/v1/docs/base_boxes.html

```
1 vagrant box add debian_squeeze_64 http://dl.dropbox.com/u/937870/VMs/squeeze64.box
```

Listing 3: Download der Vagrant Box

Nachdem dem Download steht uns jetzt die Vagrant Box `debian_squeeze_64` zur Verfügung. Nun können wir in ein beliebiges Verzeichnis wechseln und eine initiale Konfiguration basierend auf der Box anlegen, siehe. Listing 4.

```
1 vagrant init debian_squeeze_64
```

Listing 4: Vagrant initialisieren

Diese initiale Konfiguration beinhaltet alles an was Vagrant zum Konfigurieren und Starten der Maschine braucht, es sind keine weiteren Einstellungen mehr nötig und wir können diese starten, siehe. Listing 5.

```
1 vagrant up
```

Listing 5: Starten der Vagrant Maschine

Nachdem die virtuelle Maschine gestartet worden ist, können wir mit `ssh` einsteigen, siehe. Listing 6. Unter Windows ist dieser Befehl derzeit nicht verfügbar und man muss deshalb mit Tools wie [Putty] darauf zugreifen.

```
1 vagrant ssh
```

Listing 6: Mit `ssh` in der Vagrant Maschine einsteigen

Jetzt aktivieren wir die Puppetkonfiguration für unseren Vagrantbox. Vagrant liegt im Verzeichnis wo `vagrant init` durchgeführt haben, eine Datei `Vagrantfile`. Öffne die Datei und lösche die Kommentarzeichen für die Sektion `config.vm.provision :puppet`. Weiter tragen wir unter `puppet.manifests_path` und `puppet.module_path` ein wo Manifeste und Module liegen. Als `puppet.manifest_file` tragen wir `user.pp` ein. Zum testen von der Apacheinstallation wird mit `config.vm.forward_port 80, 8080` Port 80 des Guests über Port 8080 vom Host erreicht. Statt 8080 kann auch ein beliebig anderer Port gewählt werden, falls dieser schon belegt ist, jedoch sollte die Portnummer größer 1024 sein, da sonst Root-Rechte notwendig wären.

```
1 config.vm.forward_port 80, 8080
2 config.vm.provision :puppet do |puppet|
3   puppet.manifests_path = "~/git/puppet-demo/puppet/manifests"
4   puppet.module_path = "~/git/puppet-demo/puppet/modules"
5   puppet.manifest_file = "user.pp"
6 end
```

Listing 7: Puppet Provisioning in `Vagrantfile` konfigurieren

Die Vagrant Boxen werden unter Unix in installiert. `$HOME/.vagrant.d/boxes`
Falls man dies ändern will kann man die Umgebungsvariable `VAGRANT_HOME` setzen:

```
export VAGRANT_HOME=$HOME/vagrant_home
```

Bei dieser Änderung müssen wir den Box neu laden, siehe. Listing 8. Der Puppet Manifest `user.pp` wird jetzt ausgeführt. Mit `puppet ssh` steigen wir und verifizieren dass der User `'puppetdemo'` mit Home-Verzeichnis `/home/puppetdemo` vorhanden ist

```

1 vagrant reload
2 notice: /Stage[main]//Node[default]/User[puppetdemo]/ensure: created
3 notice: /Stage[main]//Node[default]/File[/home/puppetdemo]/ensure: created
4 notice: Finished catalog run in 0.35 seconds
5
6 $ vagrant ssh
7 Welcome to your Vagrant-built virtual machine.
8
9 $ sudo su - puppetdemo
10 puppetdemo@precise32:~$ pwd
11 /home/puppetdemo

```

Listing 8: Vagrant Box neu laden

Mit `vagrant provisioning` kann der Puppet Manifest erneut ausgeführt werden. Ohne Änderungen im Manifest oder die virtuelle Maschine passiert nichts. Würde der Benutzer `puppetdemo` zum Beispiel entfernt, wird er wieder angelegt. Will man der Puppet Manifest in der virtuellen Maschine ausführen, ist der `puppet.manifests_path` als `/tmp/vagrant-puppet/manifest` gemounted, siehe. Listing 9.

```

1 $ sudo puppet apply /tmp/vagrant-puppet/manifests/user.pp
2 No LSB modules are available.
3 notice: Finished catalog run in 0.04 seconds
4 $ sudo deluser puppetdemo
5 Removing user 'puppetdemo' ...
6 Warning: group 'puppetdemo' has no more members.
7 Done.
8 $ sudo puppet apply /tmp/vagrant-puppet/manifests/user.pp
9 No LSB modules are available.
10 notice: /Stage[main]//Node[default]/User[puppetdemo]/ensure: created
11 notice: Finished catalog run in 0.39 seconds

```

Listing 9: Puppet apply im Box

Der Vagrant-Box wird mit `vagrant suspend` angehalten, `vagrant resume` startet ihn wieder. Will man ganz von vorne anfangen, entfernt `vagrant destroy` den Box. Die Datei `Vagrantfile` wird erhalten, damit kann man mit `vagrant up` den Box inklusive Provisioning mit Puppet wieder aufsetzen.

2 Apache Webserver und Applikation mit HTML und JavaScript mit Puppet installieren

Nachdem wir die Grundlagen von Puppet und Vagrant jetzt gelernt haben, wird jetzt eine Apache Webserver installiert. Da es wahrscheinlich ist, dass Apache für andere Applikationen auch genutzt wird, wird der Puppet Code dafür in einem Module abgelegt. Module sind wiederverwendbare Einheiten vom Code und Data. Die Struktur eines Modules ist auf <http://docs.puppetlabs.com/learning/modules1.html> beschrieben, siehe. Listing 10. Siehe auch Puppet Module Cheat Sheet docs.puppetlabs.com/module_cheat_sheet.pdf.

```

1 my_module
2 |-- files
3 |   '-- my_file.txt
4 |-- manifests
5 |   '-- init.pp
6 '-- templates
7   '-- my_template.erb

```

Listing 10: Puppet Module Struktur

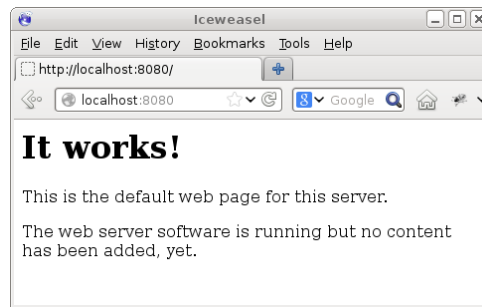


Abbildung 1: Zugriff auf den Webserver der Vagrant-Maschine

Auf die gleiche Ebene wie das manifests-Verzeichnis legen wir modules an. Darunter `apache/manifests/init.pp`, siehe. Listing 11.

```

1 puppet
2 |-- manifests
3 |   '-- user.pp
4 '-- modules
5   '-- apache
6     '-- manifests
7       '-- init.pp

```

Listing 11: Verzeichnisstruktur für den apache-Module

`modules/apache/manifests/init.pp` ist recht kompakt und zeigt auf einen Blick einen großen Vorteil mit Puppet. Die Ressourcen `package` und `service` verstecken die Komplexität und plattformsspezifische Vorgehen für Installation von Paketen und Starten von Diensten, siehe. Listing 12.

```

1 class apache {
2     package {
3         'apache2' :
4             ensure => present,
5     }
6     service {
7         'apache2' :
8             ensure => running,
9             require => Package["apache2"]
10    }
11 }

```

Listing 12: Inhalt von `modules/apache/manifests/init.pp`

Um das Modul in einem Manifest zu verwenden reicht lediglich `include apache`, siehe. Listing 13.

```

1 include apache

```

Listing 13: Inhalt von `manifests/setupapache.pp`

In `Vagrantfile` ändern wir jetzt die Zeile `puppet.manifest_file = "user.pp"` auf `puppet.manifest_file = "setupapache.pp"` und geben `vagrant provision` ein, siehe. Listing 14.

```

1 $ vagrant provision
2 [default] Running provisioner: Vagrant::Provisioners::Puppet...
3 [default] Running Puppet with /tmp/vagrant-puppet/manifests/setupapache.pp...
4 notice: /Stage[main]/Apache/Package[apache2]/ensure: ensure changed 'purged' to 'present'
5 notice: Finished catalog run in 97.25 seconds

```

Listing 14: `vagrant provision` für Apache

Autoren

Anders Malmborg



hat jahrezehntelange Erfahrung in Applikations und Produktentwicklung im C++ und JavaEE und arbeitet als IT Freelancer im automotive Bereich.

Michael Haslgrübler



hat mehrjährige Erfahrung in JavaEE Entwicklungsumfeld in der Automotive und Immobilienbranche. Er administriert seit Jahren einen Linux-Root-Server für diverse Kunden.

Literatur

[Chef] Chef. Chef is a systems integration framework, built to bring the benefits of configuration management to your entire infrastructure.

[Geppeto] Geppeto. Eclipse plugin for developing puppet modules and manifests.

[Jenkins] Jenkins. An extendable open source continuous integration server.

[Puppet] Puppet. Puppet is it automation software that helps system administrators manage infrastructure throughout its lifecycle, from provisioning and configuration to patch management and compliance.

[Putty] Putty. Ein ssh client für windows.

[Vagrant] Vagrant. Virtualized development for the masses.