Deliver with Puppet

Anders Malmborg und Michael Haslgrübler
17. 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 [**?**] 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, das 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 Continous 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 [**?**] und [**?**] 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 commited wird, der gebaut werden kann und die automatisierten Tests besteht soll auch deployed werden. Damit können wir gewährleisten bzw. überprüfen das die Software zu jedem Zeitpunkt einsatzbereit ist und nicht nur auf einem Entwicklungs-PC funktioniert.

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

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 dass das auch hier alle Server auf Knopfdruck aktualisiert werden können und die Server einen definierten und bereits in QA getesten 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.

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 mach die Definition von Ressourcen aus, eine Ressource ist ein atomarer Typ eines Systems, es entspricht einer physisches 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.

```
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
21 $ puppet resource user puppetdemo
22 user { 'puppetdemo':
23   ensure => 'present',
24   gid    => '1004',
25   home   => '/home/puppetdemo',
26   shell  => '/bin/bash',
27   uid    => '1002',
28 }
```

Listing 1: User mit Puppet anlegen

Eine Aufzählung der vom Puppet unterstützen 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 [**?**] an. Geppeto bringt Code Completion und Syntax Highlighting mit und kommt als Standaloneapplikation oder als Plugin für eine bestehende Eclipseinstallation.

1.4 Testbox mit Vagrant

Für die Entwicklung und Tests von Puppet Modulen und Manifeste empfielt sich eine virtuelle Machine. Puppet Labs stellt sollche für VMware und VirtualBox zur Verfügung unter `http://info.puppetlabs.com/download-learning-puppet-VM.html`.

Eine andere Möglichkeit ist [**?**]. 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.

Vagrant bietet

> 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 2: 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 wechsel und eine initiale Konfiguration basierend auf der Box anlegen, siehe. Listing 3.

> 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

```
1 vagrant init debian_squeeze_64
```

Listing 3: 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 4.

```
1 vagrant up
```

Listing 4: Starten der Vagrant Maschine

Nachdem die virtuelle Machine gestartet worden ist, können wir mit ssh einsteigen, siehe. Listing 5.

```
1 vagrant ssh
```

Listing 5: 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, 160 Port 80 des Guests über Port 160 vom Host erreicht.

```
1  # Forward a port from the guest to the host, which allows for outside
2  # computers to access the VM, whereas host only networking does not.
3  config.vm.forward_port 80, 160
4  config.vm.provision :puppet do |puppet|
5    puppet.manifests_path = "~/git/puppet-demo/puppet/manifests"
6    puppet.module_path = "~/git/puppet-demo/puppet/modules"
7    puppet.manifest_file  = "user.pp"
8  end
```

Listing 6: Puppet Provisioning in Vagrantfile konfigurieren

Bei dieser Änderung müssen wir den Box neu laden, siehe. Listing 7. Der Puppet Manifest user.pp wir jetzt ausgeführt. Mit puppet ssh steigen wir und verifizieren dass der User 'puppetdemo' mit Home-Verzeichis /home/puppetdemo vorhanden ist

```
vagrant reload
notice: /Stage[main]//Node[default]/User[puppetdemo]/ensure: created
notice: /Stage[main]//Node[default]/File[/home/puppetdemo]/ensure: created
notice: Finished catalog run in 0.35 seconds

$ vagrant ssh
Welcome to your Vagrant-built virtual machine.

$ sudo su - puppetdemo
puppetdemo@precise32:~$ pwd
/home/puppetdemo
```

Listing 7: Vagrant Box neu laden

Mit vagrant provisioning kann der Puppet Manifest erneut ausgeführt werden. Ohne Änderungen im Manifest oder die virtuell Machine passiert nichts. Würde der Benutzer puppetdemo zum Beispiel entfernt, wird er wieder angelegt. Will man der Puppet Manifest in der virtuellen Machine ausführen, ist der puppet.manifests_path als /tmp/vagrant-puppet/manifest gemounted, siehe. Listing 8.

```
$ sudo puppet apply /tmp/vagrant-puppet/manifests/user.pp
No LSB modules are available.
notice: Finished catalog run in 0.04 seconds
$ sudo deluser puppetdemo
Removing user 'puppetdemo' ...
Warning: group 'puppetdemo' has no more members.
Done.
$ sudo puppet apply /tmp/vagrant-puppet/manifests/user.pp
No LSB modules are available.
notice: /Stage[main]//Node[default]/User[puppetdemo]/ensure: created
notice: Finished catalog run in 0.39 seconds
```

Listing 8: Puppet apply im Box

Der Vagrant-Box wird mit vagrant suspend angehalten, vagrant resume startet ihn wieder. Will man ganz von vorne anfagnen, 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

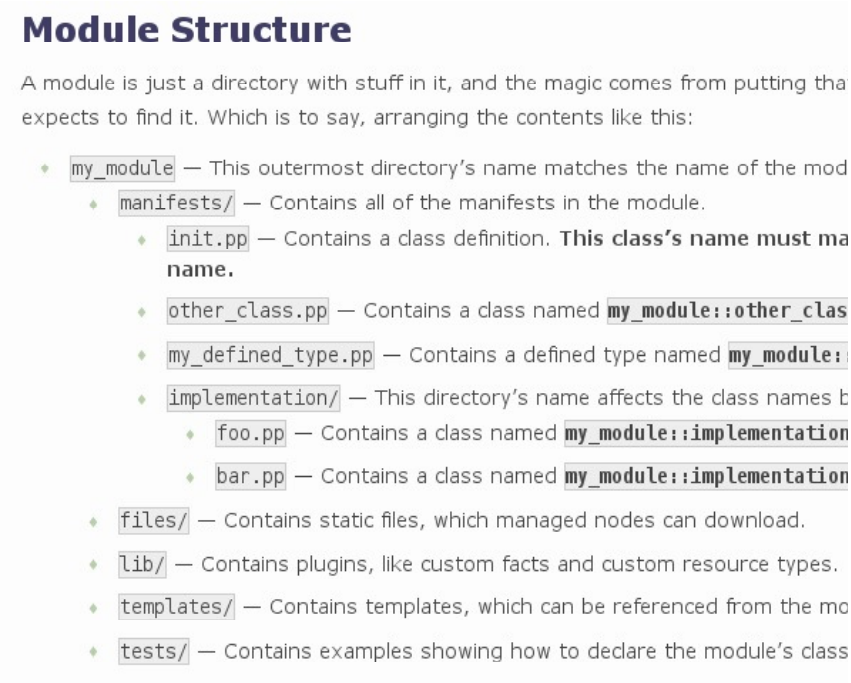
Nachdem wir die Grundlagen von Puppet und Vagrant jetz gelernt haben, wird jetzt eine Apache Webserver installiert. Da es wahrscheinlich ist, dass Apache für andere Applikationen auch genutzt wird, wird der Puppet Code dafür in einem Module abgelegt. Module sind wiederverwendbare Einheiten vom Code und Data. Die Struktur eines Modules ist auf `http://docs.puppetlabs.com/learning/modules1.html` beschrieben, siehe. Abbildung 1. Siehe auch Puppet Module Cheat Sheet `docs.puppetlabs.com/module_cheat_sheet.pdf`.

Auf die gleiche Ebene wie der manifest-Verzeichnis legen wir modules an. Darunter apache /manifests/init.pp, siehe. Listing 9.

```
```

Listing 9: Verzeichnisstruktur für den apache-Module

modules/apache/manifests/init.pp ist recht kompakt und zeigt auf einen Blick eine große Vorteil mit Puppet. Die Ressourcen package und service verstecken die Komplexität und platformspezifische Vorgen für Installation von Pakete und Starten von Dienste, siehe. Listing 10.



Module Structure

A module is just a directory with stuff in it, and the magic comes from putting tha
expects to find it. Which is to say, arranging the contents like this:

- `my_module` — This outermost directory's name matches the name of the mod
  - `manifests/` — Contains all of the manifests in the module.
    - `init.pp` — Contains a class definition. **This class's name must ma**
      name.
    - `other_class.pp` — Contains a class named **`my_module::other_clas`**
    - `my_defined_type.pp` — Contains a defined type named **`my_module:`**
    - `implementation/` — This directory's name affects the class names b
      - `foo.pp` — Contains a class named **`my_module::implementation`**
      - `bar.pp` — Contains a class named **`my_module::implementation`**
  - `files/` — Contains static files, which managed nodes can download.
  - `lib/` — Contains plugins, like custom facts and custom resource types.
  - `templates/` — Contains templates, which can be referenced from the mo
  - `tests/` — Contains examples showing how to declare the module's class

Abbildung 1: Puppet Module Struktur

```
$ cat modules/apache/manifests/init.pp
class apache {
        package {
                'apache2' :
                        ensure => present,
        }
        service {
                'apache2' :
                        ensure => running,
                        require => Package["apache2"]
        }
}
```

Listing 10: modules/apache/manifests/init.pp

Um der Module in einem Manifest zu verwenden reicht lediglich include apache, siehe. Listing 11.

```
$ cat manifests/setupapache.pp
include apache
```

Listing 11: 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 12.

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

Listing 12: vagrant provisioning 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.