Deliver with Puppet

Anders Malmborg und Michael Haslgrübler
13. 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, 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 [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 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.

2 Entwicklung mit Puppet

Puppet kann unter für Unix mit einem Paketmanager installiert werden oder für Windows von [Puppetlabs] runtergeladen werden. Für die Entwicklung von Puppet Module und Manifeste, mehr dazu später, bietet sich [Geppeto] an, welches als Standaloneapplikation installiert werden kann oder in eine bestehende Eclipseinstallation als Plugin hinzugefügt werden kann.

Für die Entwicklung und Tests von Puppet Modulen und Manifeste wird hier [Vagrant] verwendet. 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 Puppet können Vagrant und Virutalbox via Paketmanager für Linux installiert werden oder von den entsprechenden Downloadseiten runtergeladen werden.

2.1 Testbox mit 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 1: 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 2.

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 2: 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 3.

```
1 vagrant up
```

Listing 3: Starten der Vagrant Maschine

Nachdem die virtuelle Machine gestartet worden ist, können wir mit ssh einsteigen, siehe. Listing 4.

```
1 vagrant ssh
```

Listing 4: Mit ssh in der Vagrant Maschine einsteigen

2.2 Puppet Einführung

Puppet benutzt eine Domain-Specific-Language um den Zustand eines System zu beschreiben. Diese Beschreibung wird in einem Manifest festgehalten. Ein Großteil dieser Manifeste

user.pp

```
1  node default {
2
3      user {
4          'cross' :
5              ensure => present,
6              home => '/home/cross',
7      }
8      file {
9          '/home/cross' :
10             ensure => directory,
11     }
12     notice("Created user 'cross'")
13 }
```

Abbildung 1: User mit Puppet anlegen

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. Im Beispiel, siehe. Abbildung 1, sehen wir wie wir einen Benutzer und eine Datei mit der Puppet DSL anlegen.

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.

[Puppetlabs] Puppetlabs. Puppet open source.

[Vagrant] Vagrant. Virtualized development for the masses.

[VirtualBox] VirtualBox. Virtualbox is a powerful x86 and amd64/intel64 virtualization product.