Continous Delivery with Puppet

Anders Malmborg und Michael Haslgrübler
27. Dezember 2012

1 Einleitung

Iteratives Vorgehen bei der Entwicklung von Applikationen ist mittlerweile sehr verbreitet. Jede Iteration erweitert die Applikation mit neuer Funktionalität die qualitätsgesichert werden muss. Zusätzlich zu UnitTests, welche einzelne Teile prüfen, sind Integrationstests mit einer laufenden Applikation ratsam - sei es automatisch oder manuell.

Für die Source-Code-Übersetzung, Testausführung und Paketiereung der Applikation helfen Continous Integration Systeme wie [Jenkins]. Bei der Automatisierung des nächsten Schritt - automatische Installation und Konfiguration - bietet sich [Puppet] als Configuration Management Lösung an. Mit Puppet wird den erwarteten Zustand eines Systems beschrieben. Bei Abweichungen wird das System in dem erwarteten Zustand versetzt.

Folgende Probleme werden hier addresiert:

- Automatische Installation und Konfiguration nach erfolgreichen Build im Jenkins.
  - trotz Paketierung in Web Archivs (WARs) waren einige Konfigurationen, wie Datenbank-Parametern, manuell einzutragen.
- Zentrale Definition, welche Applikationen mit welcher Konfiguration wo, auf welchen Servern, zu laufen haben.
- Neue virtuelle Servers sollten einfach von einem Basis-Image aufgesetzt werden und mit Puppet fertig konfiguriert werden, inklusive beispielsweise Apache und Tomcat.
- Gleiche Mechanismen und selbes Vorgehen bei der Test-/QA- und Produktionsumgebung.

In der Entwicklung heißt das in erster Linie Softwarecode der commited wird, der gebaut werden kann und die automatisierten Tests besteht soll auch installiert werden. Damit kann gewährleistet bzw. überprüft werden, dass die Software zu jedem Zeitpunkt einsatzbereit ist und nicht nur auf einem Entwicklungs-PC funktioniert.

2 Puppet Einführung

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

Ein Manifest ist ein Puppet "Program". Module sind für die Puppet-EntwicklerIn ähnlich wie Libraries für Programmierer. Ein Großteil der Manifeste und Module machen 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 Listing 1 wird die Ressource user *puppetdemo* und dazugehörige Home-Verzeichnis */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, da dies nur der Administrator darf. 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     user {
4         'puppetdemo' :
5             ensure => present,
6             home => '/home/puppetdemo',
7             shell => '/bin/bash',
```

```
8       }
9       file {
10          '/home/puppetdemo' :
11          ensure => 'directory',
12          owner => 'puppetdemo',
13          group => 'puppetdemo',
14      }
15  }
16  $ sudo puppet apply manifest/user.pp
17  notice: /Stage[main]//Node[default]/User[puppetdemo]/ensure: created
18  notice: /Stage[main]//Node[default]/File[/home/puppetdemo]/ensure: created
19  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

Die Ressourcen sind im Core Types Cheat Sheet `http://docs.puppetlabs.com/puppet_core_types_cheatsheet.pdf` gut beschrieben.

3 Testbox mit Vagrant

Zum kennenlernen vom Puppet stellt Puppet Labs eine virtuelle Maschine für VMware beziehungsweise VirtualBox zur Verfügung unter `http://info.puppetlabs.com/download-learning-html`.

Für die Entwicklung und Tests von Puppet Modulen und Manifeste bietet sich [Vagrant] an. 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. Vagrant benützt virtuelle Maschine die in so genannte Boxen gepackt sind. Zusätzlich zu der virtuellen Maschine beinhaltet eine Box unter anderem Chef and Puppet. Die Boxen sind portabel - sie laufen auf alle Plattformen wo Vagrant laufen. Mit Vagrant ist es relativ einfach die Puppet Module und Manifeste auf unterschiedliche Zielumgebungen zu verifizieren indem man sie auf unterschiedliche Boxen testet. Vagrant via Paketmanager für Linux installiert werden oder von den entsprechenden Downloadseiten heruntergeladen werden.

Nachdem Vagrant und VirtualBox installiert worden sind, kann eine Box zum Testen aufsetzen. Hier wird eine 64 Bit Version von Debian Squeeze verwendet. Diese beinhaltet eine Minimalinstallation mit den für Vagrant üblichen Vorbereitungen: SSH Key Setup, VirtualBox Guest Additions, Puppet und Ruby, siehe auch `http://vagrantup.com/v1/docs/base_boxes.html`.

Zum Downloaden wird folgendes Kommando verwendet: `vagrant box add debian_squeeze_64 http://dl.dropbox.com/u/937870/VMs/squeeze64.box` Nachdem dem Download steht uns jetzt die Vagrant Box *debian_squeeze_64* zur Verfügung. Nun kann in ein beliebiges Verzeichnis gewechselt werden und eine initiale Konfiguration basierend auf der Box angelegt werden: `vagrant init debian_squeeze_64.`

Diese initiale Konfiguration beinhaltet alles was Vagrant zum Konfigurieren und Starten der Maschine braucht, es sind keine weiteren Einstellungen mehr nötig. Die virtuelle Maschine wird mit `vagrant up` gestartet.

Nachdem die virtuelle Maschine gestartet worden ist, kann man mit ssh sich anmelden, via `vagrant ssh`. Unter Windows ist dieser Befehl derzeit nicht verfügbar und man muss deshalb mit Tools wie [Putty] darauf zugreifen.

Zum Aktivieren von Puppet muss die *Vagrantfile* angepasst werden. Dieser liegt im Verzeichnis wo der Befehl *vagrant init* ausgeführt worden ist. Diese Datei beinhaltet bereits Einträge für Puppet, welche jedoch auskommentiert sind. Nach dem entfernen der Kommentarzeichen für die Sektion *config.vm.provision :puppet*, wird unter *puppet.manifests_path* und *puppet.module_path* die Verzeichnis wo Manifeste und Module liegen eingetragen. Als *puppet. manifest_file* wird *user.pp* eingetragen. Zum Testen von einer späteren Apache-Installation wird Port 6400 im Host auf Port 80 im Guest weitergeleitet: *config.vm.forward_port 80, 6400*. Das ganze sollte dann in etwa wie in Listing 3 aussehen.

```
1   config.vm.forward_port 80, 6400
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 3: Puppet Provisioning in Vagrantfile konfigurieren

Bei dieser Änderung muss die Vagrantbox neu geladen werden, da geteilte Verzeichnisse nur beim Starten des Hosts erkannt und automatisch gemounted werden. Zusätzlich dazu wird beim Starten das Provisioning, durch Puppet mit den Anweisungen aus *user.pp* durchgeführt. Mit *puppet ssh* kann verifiziert werden, dass der User 'puppetdemo' mit Home-Verzeichnis /home/puppetdemo vorhanden ist, siehe Listing 4.

```
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 4: Vagrant Box neu laden

Mit *vagrant provisioning* kann das Puppet Manifest erneut ausgeführt werden. Ohne Änderungen im Manifest oder in der virtuellen Maschine passiert nichts. Würde der Benutzer *puppetdemo* zum Beispiel entfernt, wird er wieder beim nächsten Provision-Vorgang wieder angelegt. Will man das Puppet Manifest in der virtuellen Maschine manuell ausführen, ist der *puppet.manifests_path* als */tmp/vagrant−puppet/manifest* gemounted, siehe Listing 5.

```
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 5: Puppet apply im Box

Vagrant kann wie VirtualBox die Maschine stilllegen mit *vagrant suspend* bzw mit *vagrant resume* wieder fortsetzen. Zum Starten und Stoppen kann man *vagrant up* bzw mit *vagrant halt* verwenden. Sollte man die Box nicht mehr benötigen kann Sie mit *vagrant destroy* unwiderruflich löschen. Das Vagrantfile bleibt jedoch erhalten, somit kann man wieder bei Null anfangen und mit *vagrant up* die Box inklusive Provisioning mit Puppet wieder aufsetzen.

4 Apache Webserver und eine Applikation mit HTML und JavaScript mit Puppet installieren

Nachdem die Grundlagen von Puppet und Vagrant jetzt erklärt worden sind, wird ein Apache Webserver jetzt 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 Daten. Die Struktur eines Modules ist im Detail auf `http://docs.puppetlabs.com/learning/modules1.html` beschrieben.

Auf die gleiche Ebene wie das manifests-Verzeichnis wird das Verzeichnis *modules* angelegt, darunter die Datei *apache/manifests/init.pp* und entsprechenden Verzeichnisse, siehe Listing 6.

```
puppet
|-- manifests
|   `-- user.pp
`-- modules
    `-- apache
        `-- manifests
            `-- init.pp
```

Listing 6: Verzeichnisstruktur für den apache-Module

Die Datei *modules/apache/manifests/init.pp* ist recht kompakt und zeigt auf einen Blick einen der großen Vorteile von Puppet. Die Ressourcen *package* und *service* verstecken die Komplexität und die plattformspezifische Vorgänge für Installation von Paketen und das Starten der Dienste, siehe Listing 7.

```
class apache {
        package {
                'apache2' :
                        ensure => present,
        }
        service {
                'apache2' :
                        ensure => running,
                        require => Package["apache2"]
        }
}
```

Listing 7: Inhalt von modules/apache/manifests/init.pp

Um ein Modul in einem Manifest verwenden zu können, reicht es lediglich `include apache` in unserem *manifests/setupapache.pp* zu schreiben.

In Vagrantfile wird jetzt die Zeile *puppet. manifest_file = "user.pp"* auf *puppet. manifest_file = "setupapache.pp"* geändert. Mit *vagrant provision* ein, siehe Listing 8 wird erneut die Konfiguration aktualisiert.

```
$ 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 8: vagrant provisioning für Apache

Abbildung 1: Apache aufrufen

Abbildung 2: Die Applikation im Browser

Wird in einem unseren Browser die lokale Adresse über, den weitergeleiteten, Port 6400 aufrufen, kann festgestellt werden, dass der Apache Webserver läuft, siehe Abbildung 1.

Als nächstes sollte Puppet die Dateien für die Applikation in das Apache Wurzel Verzeichnis (*/var/www/*) kopieren. Folgender Eintrag in Vagrantfile stellt das Host-Verzeichnis *~/git/puppet−demo/puppet−demoapp/src/main/webapp* als Guest-Verzeichnis, welches als Quelle dient, */demoapp* zur Verfügung, siehe Abbildung 9.

```
config.vm.share_folder "demoapp", "/home/vagrant/demoapp", "~/git/puppet-demo/puppet-
    demoapp/src/main/webapp"
```

Listing 9: Shared folders in Vagrantfile konfigurieren

Das Manifest *manifests/demoapp.pp* für die Installation der Applikation beinhaltet folgende zwei Zeilen: `include apache` und `include demoapp`. Es wird definiert dass Puppet die Dateien vom zuvor spezifizierten Verzeichnis */demoapp* nach */var/www/* kopieren soll. Das Ganze ist im Module demoapp gekapselt, siehe Abbildung 10.

```
class demoapp($demoappname='demoapp') {
    file {
        "/var/www/$demoappname" :
        ensure => directory,
        source => '/home/vagrant/demoapp',
        require => Package['apache2'],
        recurse => true,
    }
}
```

Listing 10: Puppet Module demoapp

Durch die Änderung der geteilten Verzeichnisse im Vagrantfile ist es notwendig die Vagrant Box neu zu starten um die Änderung aktiv werden zu lassen: *vagrant reload*, siehe Listing 11.

```
$ vagrant reload
[default] Attempting graceful shutdown of VM...
...
[default] Mounting shared folders...
[default] -- demoapp: /demoapp
[default] -- v-root: /vagrant
[default] -- v-pp-m0: /tmp/vagrant-puppet/modules-0
[default] -- manifests: /tmp/vagrant-puppet/manifests
[default] Running provisioner: Vagrant::Provisioners::Puppet...
[default] Running Puppet with /tmp/vagrant-puppet/manifests/demoapp.pp...
notice: /Stage[main]//File[/var/www/demoapp]/ensure: created
...
notice: /File[/var/www/demoapp/index.html]/ensure: defined content as '{md5}90
    a8d419b9c7b43b09ba73abebaf8f4c'
...
notice: Finished catalog run in 1.34 seconds
```

Listing 11: Puppet reload mit Provisioning der Applikation

Nach Abschluss des Neustarts erfolgt auch automatisch wieder der Provisioning Vorgang durch Puppet und die neue Webapplikation kann über den Browser aufgerufen werden , siehe Abbildung 2.

Eine Änderung in der Datei *index.html* wird bei Puppet im Provisioning-Vorgang (*vagrant provision*) erkannt und Puppet aktualisiert die Datei auch in */var/www/*, siehe Listing 12.

```
$ vagrant provision
[default] Running provisioner: Vagrant::Provisioners::Puppet...
[default] Running Puppet with /tmp/vagrant-puppet/manifests/demoapp.pp...

notice: /File[/var/www/demoapp/index.html]/content: content changed '{md5}90
    a8d419b9c7b43b09ba73abebaf8f4c' to '{md5}0a4ee5bb63c3e5c29cc54cf36a4be23c'

notice: Finished catalog run in 0.71 seconds
```

Listing 12: Puppet Provisioning nach Änderung von index.html

5 Puppet am Server

Nachdem dem erfolgreichen erstellen erster Manifeste, geht es nun darum wie man die Manifeste auf einem Server verlagert und eine ganze Serverfarm damit betreibt. Grundsätzlich unterscheidet Puppet zwischen zwei Typen: Puppet Master und Puppet Agent. Der Puppet Agent ist auf einem x-belieben Server installiert und kontaktiert eine zentrale Einheit, den Puppet Master, um von ihn gesteuert zu werden.

5.1 Puppet Server-Agent-Workflow

Der Puppet Server-Agent-Workflow funktioniert grundsätzlich nach dem Pull Prinzip, d.h. der Puppet Agent fragt aktiv beim Puppet Master nach was zu tun ist, er fordert einem *Catalog* an in dem er dem Master seinem Namen und seine Informationen über sich selbst, sogenannte *facts* mitteilt. Ein *fact* wäre zum Beispiel das Betriebssystem des Agent. Der Puppet Master identifiziert und sucht nach Arbeitsanweisungen für den Agent. Der Master compiliert aus allen anwendbaren Manifesten einen *Catalog* und sendet ihn zurück an den Agent. Dieser wendet den *Catalog* an d.h. es wird versucht den durch den *Catalog* definierten Zustand herzustellen. Das Herstellen dieses Zustandes wird protokolliert und dann an den Puppet Master gesendet. Dieser Report kann zu einem späteren Zeitpunkt ausgewertet werden.

5.2 Arbeitsanweisungen für den Agent suchen

Wenn der Puppet Master vom Puppet Agent kontaktiert wird, wird er mithilfe seines Hostnamen identifiziert und der Puppet Master liest das Manifest */etc/puppet/manifests /site.pp* ein und sucht nach einer passenden *node* Definition. Wenn der Master nun vom server1.example.org kontaktiert wird wird dieser die Ausgabe *hello server1!* erzeugen, analog für server2. Statt alle Server einzeln zu definieren kann man auch das Schlüsselword default verwenden damit werden die beinhaltende Anweisungen auf allen Servern die den Master kontaktieren ausgeführt werden.

```
node server1.example.org {
    notice("hello server1!)
}

node server2.example.org {
    include apache
    notice("hello server2!)
}

node default {
```

```
11      notice("hello server")
12  }
```

Listing 13: Node Definitionen in /etc/puppet/manifests/site.pp

5.3 Alternative: External Node Classifier

```
1  [master]
2     node_terminus = exec
3     external_nodes = /usr/local/bin/my_node_classifier
```

Listing 14: External Node Classifier Konfiguration des Puppet Master

Alternativ zu obigen deklarativen Ansatz kann Puppet zusätzlich auch einen External Node Classifier verwenden. Ein External Node Classifier ist ein Script dem als Argument der Puppet Agent Name übergeben wird und der eine *YAML* Datei ausgibt. Die YAML Datei enthält eine Liste von Klassen und Parametern welche dem Knoten zugewiesen werden.

Im vorigen Beispiel hat die Knotendeklaration, siehe Listing 13 für den server2 das *apache* Modul enthalten, analog dazu müsste der ENC das YAML Dokument, siehe Listing 15, ohne die letzte Zeile zurückliefern.

Interessant wird das Ganze erst wenn der ENC mit einem Konfigurationswerkzeug gekoppelt wird, in dem dynamisch Konfigurationen erstellt werden können. Ein solches Werkzeug wäre das [Puppet Dashboard]. Nachteil hierbei ist es, das parametrisierte Klassen derzeit nicht verwaltet werden können. In unserem obigen Beispiel könnte die Demoapplikation unter einer anderen URL verfügbar sein in dem der Klassenparameter demoappname überschreiben wird, siehe Listing 15.

```
1  name: server2.example.org
2  ---
3  classes:
4      apache:
5      demoapp:
6          demoappname: mydemoapp
```

Listing 15: ENC's YAML mit Klassenparameter

5.3.1 Dashboard

Das Puppet Dashboard kann nicht nur als ENC verwendet werden sondern ist primär eigentlich ein Reporting Tool. In Abbildung 3(a) sieht man eine Übersicht über den aktuellen Knoten. Im oberen Teil sieht man welche Gruppen, Klassen und Parameter dem Knoten zugeordnet sind. In unserem Fall sind das die Klassen *apache* und *demoapp*. Die Klassen müssen im Modulpfad (*/etc/puppet/modules/*) des Puppet Master liegen. Desweiteren sieht man eine Übersicht über die Änderungen am Knoten, als sich der Puppet Agent mit dem Master verbunden hat und das Provisioning durchgeführt wurde. Der Provisioning-Vorgang wird im Dashboard, siehe Abbildung 3(b) und in der Ausgabe des Puppet Agent festgehalten, siehe Abbildung 16.

```
1  $ puppet agent --test --server vagrant-debian-squeeze-64.vagrantup.com
2  info: Caching catalog for vagrant-debian-squeeze-64.vagrantup.com
3  info: Applying configuration version '1349723897'
4  notice: /Stage[main]/Apache/Package[apache2]/ensure: ensure changed 'purged' to 'present'
```

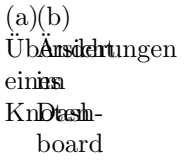

Abbildung 3: Puppet Dashboard

```
5 notice: /Stage[main]/Demoapp/File[/var/www/demoapp]/ensure: created
6 ...
7 notice: /File[/var/www/demoapp/js/knockout-2.1.0.js]/ensure: defined content as '{md5
      }235475c7c3dc43c7cb7f6125be536c32'
8 notice: Finished catalog run in 35.07 seconds
```

Listing 16: Puppet Agent Run durchführen

6 Resumeé

Ziele einer Lösung mit Puppet und Vagrant:

- Automatische Installation und Konfiguration nach erfolgreichen Build im Jenkins
- Zentrale Definition, welche Applikationen mit welcher Konfiguration wo, auf welchen Servern, zu laufen haben.
- Neue virtuelle Servers sollten einfach von einem Basis-Image aufgesetzt werden und mit Puppet fertig konfiguriert werden, inklusive beispielsweise Apache und Tomcat.
- Gleiche Mechanismen und Vorgehen bei der Test-/QA- und Produktionsumgebung.

Beim Jenkins Build wird nicht nur die Software compiliert sondern es erfolgt auch eine Paketierung. Die Konfiguration kann aber nicht Teil der Paketierung sein sondern muss zu einem späteren Zeitpunkt bestimmt werden. Weiters muss definiert werden, wenn möglich zentral, wo diese Applikationen mit welcher Konfiguration zu laufen haben. Puppet in Verknüpfung mit einem ENC bietet uns genau diese Funktionalität. Im ENC wird definiert, welcher Server mit welcher Software von Puppet ausgestattet werden soll und über Parameter der dazugehörigen Puppet Module wird festgelegt, wie diese Software konfiguriert wird.

Mithilfe von Puppet kannauf Basis eines Standard-Image, wie in obigen bei Vagrant eingesetzt, einen Server fertig konfigurieren und auf die Bedürfnisse anpassen und das mit dem selben Mechanismus – wiederholbar – in jeder Umgebung, sei es QA oder Produktion.

Autoren

Anders Malmborg

hat jahrezehntelange Erfahrung in der Applikations- und Produktentwicklung im C++ und JavaEE Umfeld 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

[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.

[Puppet Dashboard] Puppet Dashboard. Web interface und reporting tool für puppet.

[Putty] Putty. Ein ssh client für windows.

[Vagrant] Vagrant. Virtualized development for the masses.