Stéphan SZEREMETA, Fotolia

Konfigurationsmanagement mit Puppet

Am Schnürchen

Mit der Serverlandschaft wächst auch der damit verbundene Verwaltungsaufwand. Konfigurationsmanagement soll helfen ihn zu reduzieren. Hierzu bietet Puppet einen modernen und flexiblen Ansatz. Thomas Gelf

Welcher Admin kennt nicht das leidige Problem: Netzwerk und Serverpool wachsen immer weiter und irgendwann kommt der Punkt, an dem einem die Arbeit über den Kopf wächst. Der Admin geht im Alltagsstress unter, versinkt endgültig irgendwo zwischen Deployment und Fehlersuche. Eigentlich sollte alles schon längst automatisiert sein, aber die eigene Shellskript-Sammlung ist noch nicht ganz so weit und leider auch schon ein wenig angestaubt.

Konfigurationsverwaltung wurde in den letzten Jahren auch im Open-Source-Umfeld wieder ein großes Thema, zum Beispiel mit Programmen wie Cfengine und BCFG2. Das unter der GPL lizenzierte und hauptsächlich von Luke Kanies geschriebene Puppet [1] dürfte wohl das Tool sein, das in diesem Bereich zurzeit am meisten von sich reden macht. Das Bild des Puppenspielers, der mühelos die Fäden zieht, ist reizvoll, aber ist es denn wirklich so einfach, die eigenen Server derart tanzen zu lassen?

Grundsätzlich gilt, dass ein Tool wie Puppet nur dann eine echte Ersparnis bedeutet, wenn man wirklich mit ausufernden Administrationskosten zu kämpfen und eine ziemlich große Serverlandschaft mit zudem auch noch recht ähnlichen Servern zu betreuen hat. Geradezu prädestiniert ist eine solche Lösung dagegen für Anbieter von Cloud-Diensten oder größeren Serverinstallationen. Trotzdem kann Puppet auch für kleinere Umgebungen lohnend sein – aber vor allem auf längere Sicht.

Portabel

Im Vergleich mit anderen Tools aus dieser Familie zeichnet sich Puppet vor allem dadurch aus, dass es sich grundsätzlich unabhängig vom Betriebssystem einsetzen lässt. Puppet läuft mittlerweile unter fast allen gängigen Linux-Distributionen und fühlt sich auch unter Solaris, einigen BSDs sowie unter Mac OS X durchaus wohl. Zudem ist seit Version 2.6 erstmals auch rudimentäre Windows-Unterstützung verfügbar. So genannte Rezepte legen die gewünschte Konfiguration für das jeweilige System fest.

Puppet besteht aus einer Bibliothek, mit der ein Administrator in einer deklarativen Sprache die gewünschte Konfiguration einrichten kann, sowie Client- und Server-Komponenten zur Verteilung derselben. Die Abhängigkeiten zwischen den zu konfigurierenden Komponenten stellt Puppet in den Vordergrund – und verschiebt die Details der Konfiguration in die tieferen Schichten seines Modells. Man kann Puppet somit als Abstraktionsschicht zwischen dem Administrator und seinen Systemen betrachten.

Architektur

Der Puppet-Client verbindet sich gewöhnlich halbstündlich und SSL-gesichert per XML-RPC oder REST mit dem Puppet-Master. Dieser packt die für den jeweiligen Client vorhandene Konfiguration und

sendet ihm diese. Der Client vergleicht die empfangene Konfiguration mit dem Status quo des Rechners und korrigiert eventuelle Abweichungen.

Die Konfiguration kann bei Bedarf in LDAP-Verzeichnissen oder SQL-Datenbanken abgelegt oder in reine Textdateien geschrieben werden. Letztere sind dabei durchaus attraktiv, da sie sich einfach mit einem Versionskontrollsystem verwalten lassen – womit auch schon sämtliche Änderungen an den eigenen Systemen mit einem Male historisch nachvollziehbar umgesetzt wären.

Versioniert

Jede einzelne Korrektur lässt sich im Nachhinein untersuchen und bei Bedarf wieder rückgängig machen. Ob CVS, SVN, Git oder anderes bleibt jedem selbst überlassen. Puppet stellt lediglich die entsprechende Plattform bereit, was von wem in den oder die jeweiligen Konfigurationsordner ausgecheckt wird, ist ihm ziemlich egal. Wichtig ist bloß, dass die Syntax stimmt.

Die Codeschnipsel, mit denen ein Administrator Puppet beibringt, was zu tun ist, heißen Manifeste, was im Sinne der Ladungsliste eines Schiffes zu verstehen ist. Es geht hier nämlich nicht bloß um das sequenzielle Abarbeiten eines Skripts. Deshalb wurde dieser Begriff bewusst vermieden. Stattdessen arbeitet Puppet eine Art Checkliste ab, die festlegt, was mit welchen Inhalten auf dem jeweiligen Rechner vorhanden sein soll.

Puppet lässt dem Anwender große Freiheiten beim Aufbau der Manifeste. Es ist aber eine gute Idee, sich dabei an den aktuellen Best Practices [2] zu orientieren. Auf diese Weise und gepaart mit dem ebenfalls vorhandenen obligatorischen Reporting kommt Puppet dem aktuellen Idealbild einer zentralen Konfigurationsverwaltung schon ziemlich nahe.

In gängigen Linux-Distributionen sind meist noch die Versionen 0.24 und 0.25 anzutreffen, aktuell ist die 2.6. Wer heute mit Puppet loslegt, sollte gleich mit der 2.6 beginnen, während von 0.24 mittlerweile eher abzuraten ist. Grundsätzlich gilt: Alte Clients laufen meist auch mit einem neuen Server, umgekehrt gilt dies nicht immer. Mit dieser Feststellung wäre auch gleich die Vorgehensweise bei einem Upgrade geklärt: erst den Server und dann die Clients aktualisieren.

Eine der auffälligsten Änderungen in Version 2.6 betrifft die vorhandenen Binaries. Anstelle von mehreren unterschiedlichen gibt es jetzt nur noch »puppet«. Dies ist nun mit den entsprechenden Parametern aufzurufen. Tabelle 1 zeigt die Befehle in Puppet alt und neu. Diese Änderung hat auch Einfluss auf die Konfigurationsdateien, wobei dort vor allem zwei Änderung wichtig sind: aus »[puppetd]« wurde »[agent]«, aus »[puppetmaster]« einfach »[master]«.

Ubuntu 10.10 bringt ebenso wie das noch nicht erschienene Debian Squeeze bereits die Version 2.6 mit, für Ubuntu 10.04 ist immerhin Version 0.25 verfügbar. Wer Debian Lenny benutzt, sollte für Puppet 0.25 auf die mittlerweile offiziell unterstützten Backports [3] zurückgreifen. Falls die eigene Distribution keine entsprechend aktuellen Pakete mitliefert, empfiehlt sich die Installation aus den Quellen oder als Ruby-Gem.

Tabelle 1: Neue Aufrufe in Puppet 2.6

Puppet bis 0.25	Puppet 2.6
puppetmasterd	puppet master
puppetd	puppet agent
puppet	puppet apply
puppetca	puppet cert
ralsh	puppet resource
puppetrun	puppet kick
puppetqd	puppet queue
filebucket	puppet filebucket
puppetdoc	puppet doc
pi	puppet describe

Namen und andere Fakten

Wer seine Hostnamen bereits fein säuberlich im DNS pflegt, erspart sich zeitraubendes Troubleshooting und manuelle Konfiguration. Wer das nicht will oder kann, sollte sicherstellen, dass seine jeweils lokale Konfiguration dem Beispiel in Listing 1 entspricht. Abgesehen von einer funktionierenden Namensauflösung benötigt Puppet ansonsten lediglich Ruby sowie die ebenfalls vom Puppet-Projekt gepflegte Bibliothek Facter. Auch diese ist in gängigen Linux-Distributionen bereits über die jeweiligen Repositories leicht verfügbar.

Facter ist eine eigenständige und plattformübergreifende Ruby-Bibliothek, die eine ganze Reihe nützlicher Informationen über das zugrunde liegende System liefern kann. Listing 2 zeigt einen gekürzten Beispiel-Aufruf, mit entsprechenden Parametern lässt sich dieser

Listing 1: Saubere Namensauflösung ohne DNS

```
01 ## Auf dem Server:
02 # hostname
03 server
04 # hostname --fqdn
05 server.example.com
06 # cat /etc/hosts
07 127.0.0.1 localhost
08 127.0.1.1 server.example.com server puppet
09
10 ## Auf den Clients:
11 # hostname
12 client1
13 # hostname --fqdn
14 client1.example.com
15 # cat /etc/hosts
16 127.0.0.1 localhost
17 127.0.1.1 client1.example.com client1
18 192.168.0.1 server.example.com server puppet
```

Listing 2: »facter« liefert Fakten

```
01 # facter
02 architecture => amd64
03 domain => example.com
04 facterversion => 1.5.1
05 fqdn => srv10.example.com
06 hardwareisa => unknown
07 hardwaremodel => x86_64
08 hostname => srv10
09 id => root
10 interfaces => eth1
11 ipaddress => 192.168.14.101
12 ipaddress_eth1 => 192.168.14.101
13 kernel => Linux
14 kernelrelease => 2.6.26-2-amd64
15 kernelversion => 2.6.26
16 lsbdistcodename => lenny
17 lsbdistdescription => Debian GNU/Linux 5.0.6 (lenny)
18 lsbdistid => Debian
19 lsbdistrelease => 5.0.6
20 lsbmajdistrelease => 5
21 ...
```

auch auf einzelne Werte beschränken. Nicht gezeigt, aber ebenfalls per Default verfügbar sind die SSH-Schlüssel des jeweiligen Systems. Sollten diese Informationen nicht ausreichen und auch online keine passenden Module auffindbar sein, dann kann der Admin natürlich jederzeit selbst eigene Facts [4] entwickeln. Wer außerdem Wert auf IPv6 legt, wird bisher enttäuscht: Derlei Fakten sind leider noch nicht zu sehen, es wird aber bereits daran gearbeitet [5].

Learning by doing

Puppet ist gut dokumentiert, im Netz finden sich viele Beispiele und Anleitungen. Doch am besten lernt den Umgang mit Puppet, wer es selber ausprobiert. Ein nützliches Tool, um einen ersten Eindruck zu gewinnen, ist »ralsh«, die Resource Abstraction Layer Shell. Sie erlaubt es, Puppet sofort auszuprobieren, auch ohne eine Client/Server-Architektur parat zu haben. Ein einfacher Aufruf von

```
ralsh package
```

liefert eine Liste der aktuell installierten Pakete (Listing 3).

Wer zum Beispiel sichergehen will, dass auf dem System ein Benutzer »testuser« bereits vorhanden ist, um ihn andernfalls anlegen zu lassen, erreicht dies mit einem Aufruf, wie in Listing 4 dargestellt.

Den Benutzer wieder loszuwerden respektive sicherzustellen, dass er nicht existiert, geht genauso einfach von der Hand:

```
$ ralsh user "testuser" ensure=absent
notice: /User[testuser]/ensure: removed
```

In Version 2.6 erledigt dieselbe Aufgabe ein Aufruf von »puppet«:

```
puppet resource User "testuser" ensure=
absent
```

Wer statt solcher Eingriffe in seine Benutzerdatenbank zum Testen ein klassisches „Hello World“ bevorzugt, kann dies auch haben: Listing 5 legt dazu extra die Datei »/tmp/hello-world« an.

Der jeweils gezeigte Output entspricht jenem Format, das zur Konfiguration verwendet wird. Puppet kann es auch ohne Server direkt verarbeiten, und so ähnlich sieht nachher auch ein fertiges Manifest aus:

```
puppet -e 'package { "screen":
  ensure => installed
}'
```

Mit ihm wurde, falls nicht sowieso bereits vorhanden, das Paket »screen« installiert. Als letzte Übung zum Kennenlernen lässt sich eine solche Anweisung auch in eine Datei schreiben und dann mit Puppet verarbeiten:

```
echo 'package { "vim":
  ensure => installed
}' > /tmp/test-vim.pp
puppet -v /tmp/test-vim.pp
```

```
class hosting {
  case $operatingsystem {
    fedora: { $httpd_packages = ["httpd", "php"] }
    debian: { $httpd_packages = ["dapache2", "libapache2-mod-php5"] }
    ubuntu: { $httpd_packages = ["dapache2", "libapache2-mod-php5"] }
    default: { $httpd_packages = ["apache", "php"] }
  }
  package { $httpd_packages: ensure => installed }
  service { "webserver":
    name => $operatingsystem ? {
      debian => "apache2",
      ubuntu => "apache2",
      redhat => "httpd",
      default => "apache",
    },
    ensure => running,
  }
}

node "client2.example.com" {
  include hosting
}
```

Abbildung 1: Puppet-Manifest zur Webserver-Konfiguration mit Syntax Highlighting im Vim-Editor.

In Ubuntu 10.10 ist bereits das Paket »vim-puppet «enthalten, das Vi-Freunden das Editieren des Manifests noch bequemer macht. Wie das aussehen kann, zeigt Abbildung 1.

Die von Puppet verwalteten Objekte nennen sich Ressourcen, sie sind in unterschiedlichen Gruppen organisiert. So nutzt der Admin, um sicherzustellen dass ein bestimmter Dienst läuft, die Gruppe »service« und setzt als Bedingung »ensure = > running«. Stoppt er jetzt manuell den Dienst, wird er beim nächsten Durchlauf von Puppet automatisch wieder gestartet.

Logik

Was zur vollständigen Puppet-Umgebung noch fehlt, sind nur noch das Protokoll und die Logik, um Dateien in diesem Format an die jeweiligen Rechner zu verteilen. Die zentrale Konfigurationsdatei ist jeweils »puppet.conf«, die nach der Installation gewöhnlich nur einen Abschnitt »[main]« mit den wichtigsten Pfadangaben enthält. Um die passenden Startskripte müssen sich Debian/Ubuntu-Anwender keine Gedanken machen. Der Client startet aber erst, wenn er in »/etc/default/puppet« aktiviert wurde.

Auf dem Master schreibt man jetzt ein erstes kleines Manifest in die Datei »/etc/puppet/manifests/site.pp«:

```
node default {
  notice("It's working!")
}
```

Damit der Client es erhalten kann, fehlt noch eine Kleinigkeit: Die beiden kennen sich noch gar nicht. Ist der »master« (ehemals »puppetmasterd«) auf dem Ser-

Listing 3: Installierte Pakete

```
01 package { 'acpi-support-base':
02     ensure => '0.109-11'
03 }
04 package { 'acpid':
05     ensure => '1.0.8-11enny2'
06 }
07 package { 'adduser':
08     ensure => '3.110'
09 }
10 ...
```

Listing 4: Test-User anlegen

```
01 $ ralsh user "testuser" ensure=present\
02  shell=/bin/bash
03
04 notice: /User[testuser]/ensure: created
05 user { 'testuser':
06     password => '!',
07     shell => '/bin/bash',
08     uid => '1001',
09     home => '/home/testuser',
10     gid => '1001',
11     ensure => 'present'
12 }
```

Listing 5: Datei anlegen

```
01 $ puppet resource file "/tmp/hello-world"
    ensure=present content="Hallo Welt!\n"
    mode=640
02
03 file { "/tmp/hello-world":
04   ensure => present
05   content => "Hallo Welt!\n",
06   mode => 640
07 }
```

ver und der »agent« (ehemals »puppetd«) auf dem Client gestartet, stellt der Client zuerst eine Zertifikatsanfrage. Noch nicht beantwortete Anfragen sieht und bestätigt der Administrator auf dem Master wie folgt:

```
# puppet cert --list
client1.example.com
# puppet cert --sign client1.example.com
```

Wer noch eine Puppet-Version vor 2.6 nutzt, ersetzt »puppet cert« durch »puppetca«. Sobald der Client dieses Manifest erhalten hat, erscheint im Syslog des Masters die entsprechende Rückmeldung:

```
(Scope(Node[default])) It's working!
```

Jede Deklaration eines der Objekte, die bisher vorgestellt wurden, bezeichnet man als Ressource. In Listing 5 bestimmt »file« den Typ der Ressource, »/tmp/hello-world« ist deren Titel. Zusätzlich gibt es die Attribute »owner«, »group« und »mode«. Jeder Titel darf nur einmal vergeben werden, sonst verweigert Puppet mit einer entsprechenden Fehlermeldung den Dienst.

Ressourcen

Auf andere Ressourcen kann mit deren Titel Bezug genommen werden. Eine weitere Datei ist mit den folgenden Zeilen von der Existenz der oben erzeugten »hello-world«-Datei abhängig:

```
file { "/tmp/test2":
  require => File["/tmp/hello-world"]
}
```

Mehrere Ressourcen bündelt Puppet in den so genannten Resource Collections. Hiervon gibt es zwei Varianten, Klassen (»classes«) und Definitionen (»definitions«).
Eine Klasse ist hier üblicherweise eine Sammlung von Ressourcen, die ein bestimmtes Konfigurationselement beschreibt. Dies kann ein Web- oder Mailserver oder auch eine Cluster-Mitgliedschaft sein. Eine Definition funktioniert ähnlich, steht aber für eine Serie an Konfigurationselementen, die mehrmals auf einem Host vorkommen können. Dies könnten Virtual-Hosts oder gleich ganze virtuelle Maschinen sein.
In Listing 6 ist zu erkennen, was eine Definition ausmacht. Das Beispiel erstellt eine Definition namens »newip«, die als Parameter die entsprechende IP-Adresse erwartet. Das zu konfigurierende Interface wird dem Titel entnommen. Wie Listing 7 demonstriert, kann man Definitionen auch in Klassen packen.
Variablen lassen sich nach Belieben definieren, und alle »facts«, welche Facter zusammengetragen hat, liegen ebenfalls als solche vor. Listing 8 zeigt, wie sich solche Informationen in so genannten Conditionals nutzen lassen. Die folgenden Zeilen

```
node 'client1.example.com' {
  include hosting::webserver
}
```

wenden die Konfiguration schließlich auf einen bestimmten Knoten (»node«) an:

Stored Configurations

Um Daten in einer Datenbank ablegen zu können, nutzt Puppet eine Technologie, die sich Stored Configurations nennt. Hierfür benötigt Puppet Rails, das unter Debian der Aufruf

```
apt-get install rails
```

installiert. Zudem ist je nach gewünschter Datenbank noch »libsqlite3-ruby« für SQLite (Default), »libmysql-ruby« für MySQL oder »libpgsql-ruby« für PostgreSQL erforderlich. Rails sollte mindestens in Version 2.2.2 vorhanden sein, unter Debian Lenny findet sich eine entsprechende Version wiederum in den offiziellen Backports.
Selbst wenn MySQL nach der Installation von Rails auch ohne »libmysql-ruby« bereits funktioniert, ist dringend anzuraten, Letzteres zu installieren. Der mitgelieferte, in Ruby implementierte Connector ist nicht für den Produktivbetrieb bestimmt. Wer es dennoch versucht, wird im »rails.log« entsprechend gewarnt.
Um jetzt zum Beispiel mit MySQL loszulegen, müssen zuerst eine entsprechende Datenbank und ein passender Benutzer angelegt werden:

```
# mysql -u root -p
mysql> CREATE DATABASE puppet;
mysql> GRANT ALL PRIVILEGES ON puppet.*\
to puppet@localhost IDENTIFIED BY \
'Passwort';
```

In der Konfiguration ist dann noch der Abschnitt des Puppetmasters entsprechend anzupassen:

```
[master]
storeconfigs = true
dbadapter = mysql
dbuser = puppet
dbpassword = Passwort
dbserver = localhost
dbsocket = /var/run/mysqld/mysqld.sock
```

Puppet will in erster Linie ein Framework fürs Konfigurationsmanagement sein, ein GUI gehörte nicht zu den Prioritäten. Mittlerweile sind zwei nennenswerte

Listing 6: Definition

```
01 define newip ( $ip ) {
02   exec { "/sbin/ip address add $ip dev $title": }
03 }
04
05 newip { eth0:
06   ip => "10.0.0.30",
07 }
```

Listing 7: Klasse

```
01 class network {
02   define newip ( $ip ) {
03     exec { "/sbin/ip address add $ip dev $title": }
04 }
05
06 network::newip { eth0:
07   ip => "10.0.0.30",
08 }
```

Listing 8: Conditionals

```
01 class hosting {
02 class hosting {
03   case $operatingsystem {
04     fedora: { $httpd_packages = ["httpd",
   "php"] }
05     debian: { $httpd_packages = ["apache2",
   "libapache2-mod-php5"] }
06     ubuntu: { $httpd_packages = ["apache2",
   "libapache2-mod-php5"] }
07     default: { $httpd_packages = ["apache",
   "php"] }
08   }
09   package { $httpd_packages: ensure =>
   installed }
10   service { "webserver":
11     name => $operatingsystem ? {
12       debian => "apache2",
13       ubuntu => "apache2",
14       redhat => "httpd",
15       default => "apache",
16     },
17     ensure => running,
18   }
19 }
```

Webfrontends für Puppet verfügbar, es sind dies das Puppet Dashboard [6] und Foreman [7]. Während das Dashboard (Abbildung 2) in erster Linie eine Übersicht über die von Puppet gesteuerten Rechner sowie Reports über erfolgreiche und fehlgeschlagene Aktionen gibt, hat sich Foreman einiges mehr vorgenommen. Neben dem Dashboard ähnlichen Reports und einem auf Facter basierenden Inventar will Foreman alle manuellen Arbeitsschritte vom Provisioning bis zur Konfiguration des Rechners automatisieren. Der „Vorarbeiter" macht sich Gedanken über DNS, DHCP, TFTP und PXE, die Verwaltung der CA sowie eine rudimentäre CMDB oder Schnittstellen zu einer solchen.

Listing 9: Installation des Puppet Dashboard

```
01 apt-get install build-essential irb libmysql-ruby
   libmysqlclient-dev libopenssl-ruby libreadline-ruby
   mysql-server rake rdoc ri ruby ruby-dev
02 cp config/database.yml.example config/database.yml
03 # Datenbank-Konfiguration eventuell anpassen
04 rake RAILS_ENV=production db:create
05 rake RAILS_ENV=production db:migrate
06 ./script/server -e production
07 mv puppet-dashboard-1.0.3 /usr/share/puppet-dashboard
08 chown -R www-data. /usr/share/puppet-dashboard
09 cp -a /usr/share/puppet-dashboard/ext/puppet/puppet_
   dashboard.rb /var/lib/puppet/reports/
```

Listing 10: Virtualhost für das Puppet Dashboard

```
01 Listen 3000
02
03 <VirtualHost *:3000>
04     PassengerHighPerformance on
05     PassengerMaxPoolSize 12
06     PassengerPoolIdleTime 1500
07     # PassengerMaxRequests 1000
08     PassengerStatThrottleRate 120
09     RailsAutoDetect On
10     ServerName localhost
11     DocumentRoot /usr/share/puppet-dashboard/public/
12     <Directory /usr/share/puppet-dashboard/public/>
13         Options None
14         AllowOverride AuthConfig
15         Order allow,deny
16         allow from all
17     </Directory>
18     ErrorLog /var/log/apache2/dashboard.example.
   com_error.log
19     LogLevel warn
20     CustomLog /var/log/apache2/dashboard.example.
   com_access.log combined
21     ServerSignature On
22 </VirtualHost>
```

Noch ist nicht ganz klar, ob und wie die beiden Projekte in Zukunft zusammenarbeiten wollen. Zwei sich überschneidende Oberflächen sind zu viel des Guten, aber Konkurrenz belebt ja bekanntlich das Geschäft. Gegenwärtig scheint der Status quo ihnen jedenfalls nicht zu schaden – abgesehen von der Tatsache, dass beide den Port 3000 für sich beanspruchen. Um sie nebeneinander zu betreiben, kommt der Admin aber um einen kleinen Eingriff in den Code der jeweiligen Report-Skripte nicht herum.

Dashboard

Das Dashboard ist noch nicht als Debian-Paket verfügbar, weshalb die Installation händisch entsprechend Listing 9 vorzunehmen ist. Vorher wurde bereits das Dashboard von der Puppet-Website heruntergeladen und in den Ordner »puppet-dashboard-1.0.3« entpackt.

Ein wenig unschön ist die Tatsache, dass seit Version 1.0.3 die Empfehlung gilt, dem Benutzer des Webservers Schreibrechte auf sämtliche Ordner des Dashboard zu geben [8]. Aufgrund eines weiteren kleinen Bugs rund um den Konfigurationsschalter »libdir« muss zudem derzeit »puppet_dashboard.rb« noch von Hand verschoben werden.

Die in Listing 10 gezeigte Virtualhost-Konfiguration legt der Admin als neue Datei »puppet-dashboard« in dem Ordner »/etc/apache2/sites-available« ab, aktiviert sie mit »a2ensite puppet-dashboard« und startet anschließend den Apache neu. Unter »http://*Puppet-Server*:3000/« ist jetzt das neue Puppet-Dashboard zu bewundern.

Installation Foreman

Von der Website des Foreman-Projekts [7] lassen sich fertige Debian-Pakete herunterladen, vorher sollte der Admin aber noch Rails und die SQLite-Bibliothek für Ruby installieren:

```
apt-get install rails libsqlite3-ruby
```

Auch hier muss die Schnittstelle für die Live-Reports leider noch händisch kopiert werden:

```
cp /usr/share/foreman/extras/puppet/\
foreman/files/foreman-report.rb \
/var/lib/puppet/reports/foreman.rb
```

Da auch Foreman unbedingt auf der Nutzung des Ports 3000 besteht, muss jetzt einer der beiden weichen. Am einfachsten ist es, in der zuvor für das Dashboard erstellten Virtual-Host-Konfiguration den Port zu wechseln und in dem entsprechenden Skript unter »/var/lib/puppet/reports« anzupassen. Auch das Skript von Foreman muss der Admin eventuell noch anpassen, wenn er sich nicht »foreman« als zusätzlichen Host-Alias einrichten will.

Skalierbarkeit

Puppet nutzt gewöhnlich Webrick als Webserver, jenseits von Testumgebungen mit einigen wenigen Rechnern ist hiervon aber strikt abzuraten. Jede Puppet-Version kann alternativ auch mit Mongrel arbeiten, seit Version 0.24.6 unterstützt es auch Passenger (»mod_rails«).

Passenger (auch bekannt als »mod_rails« oder »mod_rack«) ist eine Apache-Erweiterung, die das Ausführen von Rails- oder Rack-Anwendungen innerhalb von Apache erlaubt. Ubuntu 10.10 liefert eine entsprechende Konfiguration bereits mit, und ein einfaches

```
apt-get install puppetmaster-passenger
```

nimmt dem Administrator viel lästige Konfigurationsarbeit ab. Damit sollte er eine Weile auskommen, ansonsten lässt sich Puppet bei Bedarf – ganz ähnlich einer gewöhnlichen Webanwendung – mit Reverse-Proxys auch auf mehrere Rechner verteilen.

Um einen großen Verbund von Puppen und Marionetten macht sich Marionette Collective [9] oder kurz Mcollective Gedanken. Mcollective setzt auf einen Broadcast-basierten Ansatz, um möglichst effizient große Cluster ansteuern zu können. Indem auf bewährte Middleware und Message Broker wie Active MQ gesetzt wird, muss hier niemand das Rad neu erfinden. Auch für die jeweiligen Metadaten setzt man auf Bewährtes: Mcollective kommt sowohl mit Puppet, Chef, Facter und Ohai als auch mit Eigenentwicklungen zurecht.

Der Ansatz scheint zu überzeugen, die jüngst erfolgte Übernahme des Projekts durch Puppet Labs lässt auf eine integrierte Lösungen dieser beiden Frameworks in der Zukunft schließen. Die

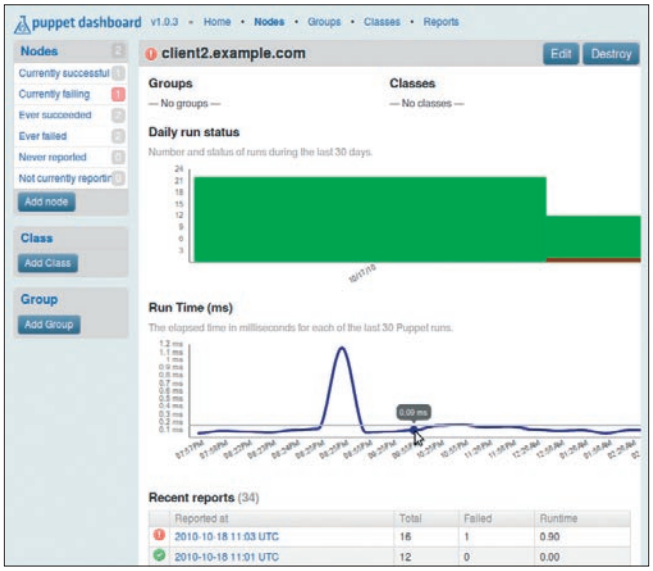

Abbildung 2: Das Dashboard zeigt grafisch den Verlauf der Puppet-Durchläufe für das Beispielszenario an.

Übernahme wurde anlässlich des Puppet-Camps im Oktober 2010 bekanntgegeben, eine kleine Roadmap zur künftigen Zusammenarbeit steht auch schon fest.

Stay tuned

Puppet und sein Umfeld entwickeln sich beeindruckend schnell weiter und gerade die jüngsten Entwicklungen lassen aufhorchen. Die immer größer werdende Community stellt auf den Plattformen des Herstellers eine wachsende Zahl an Erweiterungen und Modulen zur Verfügung. So sind die meisten Standardaufgaben bereits implementiert und lassen sich einfach auf die eigene Umgebung übertragen. Auf das, was kommt, darf man also auf jeden Fall gespannt sein. *(ofr)* ■

Infos

[1] Puppet: [http://www.puppetlabs.com/puppet/]
[2] Puppet best Practice: [http://reductivelabs.com/trac/puppet/wiki/PuppetBestPractice]
[3] Debian-Backports: [http://backports.debian.org/]
[4] Eigene Facts entwickeln: [http://projects.puppetlabs.com/projects/1/wiki/Adding_Facts]
[5] Facter und IPv6: [http://projects.puppetlabs.com/issues/3502]
[6] Puppet Dashboard: [http://www.puppetlabs.com/puppet/related-projects/dashboard/]
[7] Foreman: [http://theforeman.org/]
[8] Bug #4455: [http://projects.puppetlabs.com/issues/4455]
[9] Marionette Collective: [http://marionette-collective.org/]

Der Autor

Thomas Gelf ist Senior Consultant bei der Netways GmbH [http://www.netways.de], die seit über zehn Jahren im Bereich Open Source Systems Management und Datacenter-Solutions tätig ist, und hat langjährige Erfahrung im Bereich großer Open-Source-Infrastrukturen, Systems-Management-Lösungen und Datenbanken.