Quellen

[1] *https://github.com/ogobrecht/plex*
[2] *https://entwickler.de/online/development/git-subversion-svn-versionskontrollsystem-579792227.html*
[3] *https://www.martinfowler.com/articles/evodb.html*
[4] *https://cleandatabase.wordpress.com/2017/09/22/there-is-no-clean-database-development-without-version-control*
[5] *https://cleandatabase.wordpress.com/2017/11/28/one-does-not-simply-update-a-database-migration-based-database-development*
[6] *https://learncodeshare.net/2018/04/30/tips-to-help-pl-sql-developers-get-started-with-ci-cd*
[7] *https://learncodeshare.net/2018/07/16/ci-cd-for-database-developers-export-data-base-objects-into-version-control*
[8] *https://www.thatjeffsmith.com/archive/2019/01/19-x-sqlcl-teaser-liquibase/*

Ottmar Gobrecht
ottmar.gobrecht@linde.com

Ansible oder Puppet: Which way to go?

Raphael Daum (DBConcepts) und Florin-Catalin Enache (T-Systems Austria)

Was verbindet DevOps, Cloud, Serverless Computing und Continous Delivery? Hinter all diesen Begriffen verbirgt sich die Notwendigkeit der Automatisierung, die Notwendigkeit, Abläufe deterministisch und reproduzierbar zu machen. Wir stellen zwei prominente Vertreter dieses Automatisierungsansatzes vor: Der Newcomer Ansible trifft auf Puppet, den Darling der letzten Dekade. Was verbindet sie, was trennt sie und welchen Nutzen bieten sie?

Alt gegen Jung, so könnte ein Vergleich zwischen den Tools Puppet und Ansible zusammengefasst werden. Während der frühere Platzhirsch scheinbar schon mit dabei war, als das Internet gerade anfing, ein Massenphänomen zu werden, betritt mit Ansible ein Werkzeug zu einem Zeitpunkt die Bühne des Geschehens, an dem nur mehr von Cloud und Terraforming die Rede ist. Somit sollte auch schon klar sein, welchem der beiden Konfigurationsmanagement-Tools die Zukunft gehört.

Werkzeugkoffer: Meiner oder deiner?

Damit würden wir es uns allerdings zu einfach machen, denn beide trennt nicht nur zeitlich die eine oder andere Dekade, sondern auch architektonisch unterscheiden sie sich in manchen Bereichen grundlegend. Und somit kann die Frage, ob sich der Einsatz von Puppet oder Ansible eher lohnt, auch nicht beantwortet werden. Obwohl eine Konfigurationsmanagement-Software im Grunde nichts anderes als ein Werkzeugkoffer ist, der einen in die Lage versetzen soll, spannende IT-Dinge zu bauen, und somit eigentlich in den Hintergrund treten soll, ist es dann doch der Werkzeugkoffer der Wahl. Und niemand will, gerade in den Zeiten von „Agile Development“ und „Continous Delivery“, seinen Werkzeugkoffer im Quartalsrhythmus austauschen, denn das verursacht unnötige Kosten und Reibungsverluste. Deshalb sollte die Entscheidung mit Bedacht getroffen und hierbei insbesondere herausgearbeitet werden, welche Probleme die Konfigurationsmanagement-Software lösen soll (und welche nicht gelöst werden können).

Ansible: Speed Transformation

Ansible ist eine Open-Source-Automatisierungsplattform. Mithilfe der einfachen Automatisierungssprache lässt sich eine IT-Anwendungsinfrastruktur in Ansible-Playbooks perfekt beschreiben. Ansible ist auch eine Automatisierungs-Engine zur Ausführung von Ansible-Playbooks *[5]*.

Die jetzt Red Hat gehörende Software kann leistungsfähige Automatisierungsaufgaben verwalten und lässt sich an viele verschiedene Workflows und Umgebungen anpassen. Gleichzeitig kann

Ansible von neuen Benutzern schnell genutzt werden, um produktiver zu werden.

Ansible: Konzepte und Architektur [6]

In der Ansible-Architektur gibt es zwei Arten von Rechnern: Kontrollknoten und verwaltete Hosts. Ansible wird auf einem Kontrollknoten installiert und ausgeführt, und dieser Rechner enthält auch Kopien der Ansible-Projektdateien. Bei einem Kontrollknoten kann es sich um den Laptop eines Administrators, um ein von einer Reihe von Administratoren gemeinsam genutztes System oder um einen Server handeln, auf dem Ansible Tower ausgeführt wird.

Verwaltete Hosts sind in einem Inventar aufgeführt, in dem diese Systeme außerdem zur einfacheren kollektiven Verwaltung in Gruppen organisiert werden. Das Inventar kann in einer statischen Textdatei definiert oder dynamisch mithilfe von Skripten ermittelt werden, die Informationen aus externen Quellen abrufen.

Statt komplexe Skripte zu schreiben, erstellen Ansible-Benutzer Plays auf hoher Ebene, um sicherzustellen, dass ein Host oder eine Gruppe von Hosts einen bestimmten Status aufweist.

Ein Play führt eine Reihe von Aufgaben auf dem oder den Hosts in der Reihenfolge aus, die in dem Play angegeben ist. Diese Plays werden im YAML-Format in einer Textdatei angegeben. Eine Datei, die ein oder mehrere Plays enthält, wird als Playbook bezeichnet.

Mit jeder Aufgabe wird ein Modul *[7]* ausgeführt, ein kurzer Code-Abschnitt (geschrieben in Python, PowerShell oder einer anderen Sprache) mit bestimmten Argumenten. Jedes Modul ist im Grunde ein Tool in einem Toolkit. Im Lieferumfang von Ansible sind mehrere Tausend nützlicher Module enthalten, die ein breites Spektrum von Automatisierungsaufgaben ausführen können. Sie können auf Systemdateien agieren, Software installieren oder API-Aufrufe vornehmen.

Wenn ein Modul in einer Aufgabe verwendet wird, stellt es im Allgemeinen sicher, dass ein bestimmtes Element im Zusammenhang mit dem Rechner einen bestimmten Status aufweist. Eine Aufgabe mit einem Modul *[8]* kann zum Beispiel sicherstellen, dass eine Datei vorhanden ist und über bestimmte Berechtigungen und Inhalte verfügt, während eine Aufgabe mit einem anderen Modul vielleicht gewährleistet, dass ein bestimmtes Dateisystem gemountet ist. Wenn das System nicht diesen Status aufweist, sollte es von der Aufgabe auf diesen Status gesetzt werden. Wenn das System bereits diesen Status aufweist, sollte die Aufgabe nichts unternehmen. Schlägt eine Aufgabe fehl, besteht das Standardverhalten von Ansible darin, das restliche Playbook für die Hosts abzubrechen, auf denen die Auf-

```
tasks:
  - name: install httpd
    yum:
        name: httpd
        state: installed
  - name: web server is enabled
    service:
        name: httpd
        enabled: true
  - name: install mysql
    yum:
        name: mysql-server
        state: installed
  - name: mysql is enabled
    service:
        name: mysql
        enabled: true
  - name: ensure index.html file exists
    lineinfile:
    path: /var/www/html/index.html
    line: ,'Ansible is fun!'
    state: present
```

Listing 1: Ansible in Aktion

```
# install apache2 package
package { 'apache2':
  ensure => installed,
}

# ensure apache2 service is running
service { 'apache2':
  ensure => running,
}

# install mysql-server package
package { 'mysql-server':
  ensure => installed,
}

# ensure mysql service is running
service { 'mysql':
  ensure => running,
}

# ensure index.html file exists
file { '/var/www/html/index.html':
  ensure => file,
  content => 'Puppet is fun!',
  require => Package['apache2'],
}
```

Listing 2: P uppet-DSL in Aktion

gabe fehlgeschlagen ist. Aufgaben, Plays und Playbooks sollten idempotent sein. Dies bedeutet, dass man in der Lage sein sollte, ein Playbook mehrere Male sicher auf denselben Hosts auszuführen. Außerdem sollte das Playbook bei der Ausführung keine Änderungen vornehmen, wenn die Systeme den korrekten Status aufweisen *(siehe Listing 1)*.

Puppet: Neuer Wein in alten Schläuchen

Puppet *[1]* gibt es seit dem Jahr 2005 und erfreut sich immer noch großer Beliebtheit. Ein Blick auf Puppetforge *[2]* macht schnell deutlich, dass Puppet schon zum Lösen verschiedenster Probleme Anwendung fand und es hier kaum etwas gibt, das noch nicht als fertiges Modul zum Download zur Verfügung steht. Das ist sicherlich die Stärke von Puppet, zumal viele der Module von Puppetlabs – das ist die Firma hinter Puppet – selbst gewartet werden und somit den hohen Qualitätsansprüchen entsprechen.

DSL: Schnörkellose Abstraktion

Puppet unterscheidet sich in zwei wesentlichen Punkten von Ansible. Erstens verfügt Puppet über eine eigene Sprache (DSL, domain specific language) – das darf auch nicht weiter verwundern, denn Puppet wurde vor fast 15 Jahren ersonnen und zu dieser Zeit waren DSLs sehr modern. Auch wenn die Lernkurve dadurch etwas steiler ist und das erste Deployment mit Puppet vielleicht nicht in einem Nachmittag zu bewerkstelligen ist, wird man mit einer sehr ausgereiften Sprache belohnt. Ein Vorteil der Puppet DSL ist sicherlich, dass der Fokus vom imperativen und sequenziellen Programmieren (zuerst das, dann jenes, ...) auf ein deklaratives Arbeiten verlegt wird. Nicht die Reihenfolge und die spezifische Ausformulierung der Schritte sollen im Vordergrund stehen, sondern die Beschreibung des gewünschten Endzustands. Ich deklariere, dass ein Paket x auf Host y installiert sein soll, und Puppet kümmert sich um die Umsetzung meines Wunsches, egal ob implizit vorrangige Schritte notwendig sind oder „yum/dnf/apt-get/pacman“ aufgerufen werden muss. Puppet weiß, was zu tun ist; die Puppet DSL steht sozusagen über den Niederungen der OS-Spezifika. Der „Visual Index“ *[3]* bietet einen schnellen Einstieg in die Sprache. In *Listing 2* wird die Sprache kurz anschaulich gemacht.

Master and Agent

Der zweite Unterschied besteht darin, dass Puppet von Beginn an eine Mas-

ter-Agent-Architektur verfolgt hat, was Ad-hoc-Deployments etwas schwieriger macht. Die Agents können nur vom Master aus gesteuert werden und akzeptieren nicht beliebige Befehle. Die Kommunikation zwischen Agent und Master ist verschlüsselt und die Konfiguration der Agents erfolgt zentral vom Master aus. Das bringt initial einen etwas höheren Installations- und Konfigurationsaufwand mit sich, zahlt sich dann jedoch recht schnell aus, weil sichergestellt ist, dass die Agents auf den zu verwaltenden Hosts nur Code ausführen, der auf dem Master hinterlegt worden ist. Und das ist gerade ab einer gewissen Teamgröße nur mit Vorteilen verbunden, weil über einfache Mechanismen sichergestellt werden kann, wer was wann am Puppet-Master verändert hat. In Kombination mit Framework r10k *[4]* können Git-Repositories dazu verwendet werden, Puppet-Manifeste (Puppet DSL mit der Datei-Endung .pp) wie Source Code zu verwalten („Configuration as Code"). Das Branchen innerhalb von Git findet seine Entsprechung im Konfigurationsmanagement, wenn der Puppet-Code im Branch „dev" nur auf jenen Hosts ausgeführt wird, die in der Gruppe „dev" stecken. Nichts steht hier dem Mantra „Release early, release often" im Wege. Sobald ein Push im Branch „dev" registriert wird, kann dadurch ein Puppet-Run auf allen Hosts der Gruppe „dev" getriggert werden, um zu sehen, ob der „Change" richtig „deployed" wurde.

Fazit: Puppet & Ansible

Es gibt sicherlich Platz für beide und die Überlappungen von Puppet und Ansible sind beträchtlich, beackern sie doch das gleiche Problemfeld. Tendenziell hat Puppet dort sein Stärken, wenn es darum geht, IT-Infrastruktur mittel- und langfristig über den gesamten Lifecycle hinweg zu betreuen. Das hierarchische Prinzip, die solide Codebasis und die zahlreichen zusätzlichen Management-Features in der Puppet Enterprise Edition *[9]* machen Puppet immer noch zu einem sehr attraktiven Produkt, auch wenn der Hype gerade anderswo stattfindet.

Mit seinem Designziel ist Ansible sehr leicht erlernbar und zuverlässig. Ansible ist sehr leicht auch mit Puppet integrierbar, aber auch mit zahlreichen anderen Tools, die seine Funktionalitäten erweitern, zum Beispiel Ansible Tower (AWX), Jenkins etc.

Ob jetzt Ansible oder Puppet oder ein anderes Automatisierungstool zum Einsatz kommt, ist letztlich nicht so wichtig. Wichtig ist, dass automatisiert wird. Über das „Wie", also das Tool, darf dann diskutiert werden. Die Notwendigkeit ist jedoch nicht bestreitbar...

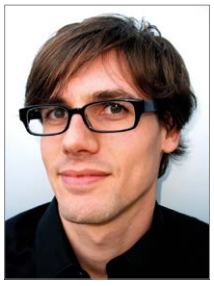
Quellen

[1] *https://puppet.com*
[2] *https://forge.puppet.com*
[3] *https://puppet.com/docs/puppet/5.3/lang_visual_index.html*
[4] *https://puppet.com/blog/git-workflows-puppet-and-r10k*
[5] *https://www.ansible.com*
[6] *https://www.ansible.com/how-ansible-works*
[7] Manpage (1) ansible
[8] *http://docs.ansible.com/ansible/intro_installation.html*
[9] *https://puppet.com/products/puppet-enterprise*

Über die Autoren

Raphael Daum ist DBA, Automatisierungsgezeichneter und Anhänger des relationalen Datenmodells sowie endloser Rekursion sowie endloser ...

Florin-Catalin Enache ist DBA und System Administrator, der auf Oracle und Red-Hat-Technologien fokussiert ist. Er ist Oracle Certified Master und Red Hat Certified Engineer. In seiner Freizeit fährt er gerne Rad und hört Hörbücher.



Raphael Daum
raphael.daum@dbconcepts.at

Florin-Catalin Enache
catalin@enache.at

Petition der DOAG zur Modernisierung von Forms an Oracle

Oracle Forms wird dieses Jahr 40 Jahre alt. In den letzten Jahren sind bei vielen Oracle-Kunden erhebliche Investitionen in Ihre Forms-Entwicklungen geflossen. Mit den zukünftig geplanten Forms-Versionen ist ein langfristiger Support gesichert und Oracle bietet Forms weiterhin als mögliches PL/SQL-4GL-Werkzeug an.

Frank Hoffmann und die DOAG starteten als Initiative zur Modernisierung der Oracle-Forms-Technologie eine Petition. Die Befürworter dieser Petition fordern von Oracle eine zeitgemäße Modernisierung der Laufzeitumgebung und die Möglichkeit einer Cloud- und mobilfreundlichen Deploymentoption (JavaScript).

Weiterführende Informationen

Zum Hintergrund: *https://backoffice.doag.org/formes/pubfiles/11309466/docs/Presse/2019/2019-05-22_DOAG_Oracle_Forms_Petition_Info.pdf*

Die Petition: *https://www.doag.org/go/petition*