Einführung in Ansible

Work in Progress!

Dokumentation

Das Ansible-Projekt betreibt eine hervorragende aber leider englischsprachige Doku-Website inclusive einer detaillierten Übersicht der verfügbaren Module
http://docs.ansible.com/ansible/modules_by_category.html

Kurzeinführung in YAML

Ansible benutzt „Yet Another Markup Language“, die wie Python Indentierung (Einzug) zur Strukturierung benutzt. Funktionell ähnlich zu JSON (→ JavaScript) oder XML; dabei aber leichter zu lesen. Wird z.B. bei Ruby on Rails oder PHP Symfony als Konfigurationssyntax benutzt.

Aufbau und Datentypen

YAML kennt als Datentypen geordnete und assoziative Listen, auf neudeutsch lists und dictionaries.

list.yml

```
---
Werkzeuge:
  - Hammer
  - Schere
  - Messer
...
```

dict.yml

```
---
Maschinenwart:
```

```
  Schneidplotter: Ralph
  Drucker: Philip
  Etikettendrucker: Sarah
...
```

Die beiden können ineinander geschachtelt und zu komplexen Datentypen montiert werden:

complex.yml

```
---
Mitglieder:
  - thb:
    name: Thomas Bätzler
    job: Sysadmin
    skills:
      - apache
      - perl
      - routing
  - fred:
    name: Fred Feuerstein
    job: Kranführer
    skills:
      - jabba
      - dabba
      - doo
...
```

Für beide Listentypen gibt es auch eine kompaktere Schreibweise, die mehrere Werte auf einer Zeile zusammenfasst:

shortlists.yml

```
---
Werkzeuge: [ 'Hammer', 'Schere', 'Messer' ]
Maschinenwart: { Schneidplotter: Ralph, Drucker: Philip,
Etikettendrucker: Sarah }

...
```

Werkzeuge wie ansible-lint können helfen, Fehler in YAML-Files zu finden.

Installation von Ansible

Beides sehr gut beschrieben in der Ansible Dokumentation, deshalb hier nur in Stichwörtern.

Software-Anforderungen

Clients

- ssh mit sftp (oder scp) als Datentransport; PublicKeyAuthentication=yes und sudo empfohlen.
- Python 2.x mit dem Paket python-simplejson (python3-Support derzeit nur als Tech Preview verfügbar).

Ansible-Server

- Python 2.7 sowie diverse Module (werden vom Paketmanagement mitinstalliert).

Installation per Paket-Manager

- Passende Paketquellen im System eintragen bzw. bekanntmachen
- Zur Installation jeweils den „üblichen“ Paket-Manager benutzen; also z.B. apt, yast, yum, ...

Installation per Quellcode am Beispiel Debian 8

Da ein einfaches „`make install`“ langfristig Leichen im System hinterläßt, bauen wir uns „einfach“ selbst ein Debian-Paket:

```
  # apt-get install git build-essential fakeroot cdbs debhelper dpkg-dev
        git-core reprepro asciidoc libalgorithm-merge-perl quilt
        xsltproc devscripts pbuilder --no-install-recommends
  # ... pbuilder config goes here ...
  # apt-get install python-2.7 python-yaml python-paramiko python-jinja2
        python-httplib2 python-sphinx python-setuptools sshpass
  # git clone https://github.com/ansible/ansible.git --recursive
  # cd ansible
  # git checkout -b v2.3.0.0-1 tags/v2.3.0.0-1
  # git submodule update
  # make DEB_DIST=jessie debian
  # mk-build-deps
      --root-cmd sudo
      --install
      --build-dep deb-build/jessie/ansible-2.3.0.0/debian/control
  # make DEB_DIST=jessie deb
  # dpkg -i deb-
build/stable/ansible_2.3.0.0-100.git201704121356.d56ba09.headsv23001~stable_
all.deb
```

Erste Tests

Vorbereitungen

Auf dem zukünftigen Ansible-Master für den eigenen User einen ssh-Keypaar generieren, mit dem man sich bei den verwalteten Systemen anmeldet.

```
$ ssh-keygen -trsa -b2048
$ cp .ssh/id_rsa.pub .ssh/authorized_keys
$ ssh <remote> "mkdir .ssh; chmod 750 .ssh"
$ scp ~/.ssh/id_rsa.pub <remote>:.ssh/authorized_keys
```

Wer sich auskennt und agent-forwarding benutzt, kann darauf verzichten und deployt statt dessen seinen Public Key direkt auf die zu verwaltenden Systeme.

Anschließend testen wir, ob man sich paßwortlos am entfernten System anmelden kann. Damit laden wir auch gleich die Hosts-Keys in .ssh/known_hosts, so daß uns das später nicht stört.

Auf den verwalteten Systemen sollte der Einfachheit halber `sudo` installiert sein; unser Verwaltungsbenutzer sollte root-Rechte haben, z.B. in dem man in per `visudo` in /etc/sudoers einträgt:

```
<user>     ALL=NOPASSWD:ALL
```

Außerdem brauchen wir ein Verzeichnis der zu verwaltenden Systeme, das Inventory. Das wird normalerweise in /etc gesucht; per Ergänzung der .bashrc verweisen wir auf eine in unserem $HOME abgelegte Variante:

```
$ cat >> ~/.bashrc
export ANSIBLE_INVENTORY=~/.ansible_hosts
^D
$ cat > ~/.ansible_hosts
localhost
[slaves]
slave1
slave2
^D
```

Danach einmal aus- und wieder einloggen!

Damit haben wir drei Hosts definiert: localhost und zwei slave-Rechner, die zusammen eine Gruppe namens „slaves“ bilden.

Anschließend können wir verifizieren, daß wir ein Inventory haben:

```
$ ansible --list-hosts all
  hosts (3):
    localhost
    slave1
    slave2
```

Und mit dem Modul „ping“ prüfen, ob wir soweit alles richtig installiert haben und ob die

Kommunikation klappt:

```
$ ansible -m ping all
slave2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
slave1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
localhost | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Der Ad-Hoc-Modus

Prinzipiell können wir mit Ansible auch beliebige Systembefehle auf den gemanagten Systemen ausführen:

```
$ ansible all -a "/usr/bin/free" -f 2 --become
localhost | SUCCESS | rc=0>>
             total       used       free     shared    buffers     cached
Mem:      16459700    1366844   15092856     336860       2676    1073664
-/  buffers/cache:     290504   16169196
Swap:      3904508          0    3904508

slave1 | SUCCESS | rc=0>>
             total       used       free     shared    buffers     cached
Mem:      16459700     690784   15768916     320476       2676     468496
-/  buffers/cache:     219612   16240088
Swap:      3904508          0    3904508

slave2 | SUCCESS | rc=0>>
             total       used       free     shared    buffers     cached
Mem:       4054320    1448256    2606064          0      11756    1135848
-/  buffers/cache:     300652    3753668
Swap:      3903676          0    3903676
```

Wichtige mögliche Parameter sind hier `-u <user>` um den Anmeldenamen auf den gemanagten Systemen anzugeben, sofern das nicht der Username des Users ist, der das Kommando ausführt; `–become` um automatisch Rootrechte zu erlangen und `-f <anzahl>` um die Anzahl der parallel ausgeführten Kommandos zu beschränken.

Statt `all` könnten wir auch den Namen eines Hosts oder einer Hostgruppe angeben.

Unser erstes Ansible-Skript

Meistens möchte man jedoch mehr machen, als nur ein Kommando auf einem Host auszuführen. Ansible fasst Befehlsgruppen in den eingangs erwähnten YAML-Dateien zu sogenannten „Playbooks“ zusammen, die jeweils ein oder mehrere „Plays“ enthalten können.

playbook1.yml

```
---
- hosts: localhost
  become: yes

  tasks:

  - name: install proper editor
    apt: name=vim-scripts install_recommends=yes state=latest

  - name: configure default editor
    alternatives: name=editor path=/usr/bin/vim.basic
...
```

```
$ ansible-playbook playbook1.yml

PLAY [localhost]
*********************************************************************

TASK [setup]
*********************************************************************
ok: [localhost]

TASK [install proper editor]
*************************************************
ok: [localhost]

TASK [configure default editor]
*************************************************
ok: [localhost]

PLAY RECAP
*********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0
```

Durch hinzufügen weiterer „hosts“-Elemente auf der obersten Ebene könnten wir die Datei um weitere Plays erweitern.

Weitere Elemente von Playbooks

Variablen

Handler

Schleifen

Benutzen von Rollen