heinlein akademie

Config-Management mit Ansible

Sven Velt - sven@velt.biz

velt.biz

Open-Source Consulting & Training

17. Juni 2016

Sven Velt

Etwas über mich:

- Dipl.-Inf. (FH), 40 Jahre
- Aus Franken! **Nicht** Bayern! ;-)
- Linux seit ca. 20 Jahren
- Consulting & Training seit 2002
- Seit 2012 “Einzelkämpfer”

Automatisierung bei Kunden notwendig/sinnvoll:

- Web-Cluster
- Monitoring von Servern (hauptsächlich Nagios & Co.)

Ansible - Inhaltsverzeichnis I

Einstieg

Ansible - Kurzbeschreibung

Zitat Wikipedia:

> *Ansible ist eine Open-Source-Plattform zur **Orchestrierung** und allgemeinen Konfiguration und Administration von Computern. Sie kombiniert **Softwareverteilung**, Ad-hoc-Kommando-Ausführung und **Konfigurationsmanagement**.*
>
> *Sie verwaltet Netzwerkcomputer über **SSH** und erfordert keinerlei zusätzliche Software auf dem zu verwaltenden System. Module nutzen zur Ausgabe JSON und können in jeder beliebigen Programmiersprache geschrieben sein. Das System nutzt **YAML** zur Formulierung wiederverwendbarer Beschreibungen von Systemen.*

Quelle: `https://de.wikipedia.org/wiki/Ansible`
Hervorhebungen von mir

Warum Ansible?

Warum **ich** Ansible verwende:

- Allgemein: niedrige Einstiegshürden
- Basiert auf Python und SSH (“überall” vorhanden)
- Ablaufbeschreibungen (“Playbooks”)
  statt Zustandsbeschreibungen
- Playbooks in YAML (gut) lesbar
- Kann auch parallel zu anderen Systemen eingesetzt werden

Immer wieder geäußerte Kritikpunkte:

- Alle Rechner müssen per SSH erreichbar sein
- Python (und ggf. weitere Module) muss auf allen Rechnern installiert sein

Übersicht: Einstieg

1. Einstieg
   - Installation
   - `ansible` - Module direkt aufrufen
   - Plays und Playbooks
   - Konfiguration

Aktuelle Version installieren

Aktuelle Versionen (mindestens 2.0) von Ansible gibt es

- Debian: In den Backports
- Ubuntu: PPA “`ansible/ansible`”
- RedHat & Co: EPEL
- SUSE: Build-Service (Repo ”`systemmanagement`“)

Installation aus dem git-Repository

Bauen eines Pakets aus dem git

Bauen eines Pakets aus dem git

```
% git clone git://github.com/ansible/ansible.git --recursive
% cd ./ansible
% git checkout remotes/origin/stable-2.X
% apt install libxml2-utils docbook-xsl docbook-xml xstlproc
% make deb
% apt install python-paramiko python-yaml python-jinja2 python-httplib2
    python-six
% dpkg -i deb-build/unstable/ansible*.deb
```

- `make rpm` wird auch unterstützt

Übersicht: Einstieg

1. Einstieg
   - Installation
   - `ansible` - Module direkt aufrufen
   - Plays und Playbooks
   - Konfiguration

Vorbereitung?

Direkt nach der Installation kann man die ersten Ansible-Aufrufe starten, weil:

- Ansible sind alle notwendigen Einstellungen (Pfade, Schalter, ...) bekannt
- Der Rechner/Host mit dem Namen "`localhost`" bzw. "`127.0.0.1`" ist automatisch bekannt
  $\Rightarrow$ Ansible verwendet dann **keine** SSH-Verbindung

Ansible Module

- Ansible ist modular aufgebaut (aktuell gut 500 Module)
- “harter Kern” von wichtigen Modulen (ca. 50)
- Viele spezialisierte Module (Cloud-Dienste, Windows, . . .)

Beispiele für Module:

- “`ping`”: Testet (SSH-)Verbindung
- “`win_ping`”: Testet Verbindung zu Windows
- “`command`”: Führt eine Kommandozeile aus
- “`package`”: (De-)Installation von Linux-Packages
- “`nagios`”: Steuert(!) Nagios-/Icinga-Monitoring
- “`s3*`”: S3-Management

Aufruf eines Modules mit “ansible”

Aufruf:

```
ansible <host-pattern> -m|--module <module>
```

Minimaler ansible-Aufruf mit ping-Modul

```
1 # Langversion:
2 # % ansible localhost --module-name=ping
3
4 % ansible localhost -m ping
5 localhost | SUCCESS => {
6     "changed": false,
7     "ping": "pong"
8 }
```

ping - Verbindung testen

Das “ping”-Modul dient zum Testen der Verbindung und ist gerade bei der Einrichtung allgemein bzw. neuer Hosts sehr praktisch.

Minimaler ansible-Aufruf mit ping-Modul

```
1 # Langversion:
2 # % ansible localhost --module-name=ping
3
4 % ansible localhost -m ping
5 localhost | SUCCESS => {
6     "changed": false,
7     "ping": "pong"
8 }
```

- Das Modul ist erfolgreich zu Ende gelaufen (“SUCCESS”)
- Es liefert zwei Variablen (“changed” und “ping”) zurück
- Weitere Parameter existieren nicht

Probleme bei der Verbindung?

Sollte es zu Problemen bei der Verbindung kommen, so kann man `ansible` mit bis zu "`-vvvv`" aufrufen:

Verbose-Ausgabe (debugging) aktiviert

```
1 % ansible localhost -m ping -vvvv
2 <localhost> REMOTE_MODULE ping
3 <localhost> EXEC ['/bin/sh', '-c', 'mkdir -p
      $HOME/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712 && chmod
      a+rx $HOME/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712 && echo
      $HOME/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712']
4 <localhost> PUT /tmp/tmp6FOUBE TO
      /home/debian/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712/ping
5 <localhost> EXEC ['/bin/sh', '-c', u'LANG=C LC_CTYPE=C /usr/bin/python
      /home/debian/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712/ping;
      rm -rf
      /home/debian/.ansible/tmp/ansible-tmp-1458811925.71-67773326054712/
      >/dev/null 2>&1']
6 localhost | success >> {
7     "changed": false,
8     "ping": "pong"
9 }
```

Was weiß Ansible vom System?

Das Modul “setup” sammelt viele Informationen ein, welche später (z. B.in Bedingungen) verwendet werden können

Aufruf des setup-Moduls

```
% ansible localhost -m setup
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "172.22.XXX.X",
            "192.168.X.XXX"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::XXXX:XXXX:XXXX:XXXX",
            [...]
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "07/05/2011",
[...]
}
```

Argumente an Modul übergeben: Hier Filter

- Mit “-a” kann man dem Ansible-Module Argumente mitgeben
- Zum Beispiel kann das Modul “setup” die zurückgegebenen Informationen filtern

Aufruf des setup-Moduls mit Filter-Argument

```
% ansible localhost -m setup -a "filter=ansible_processor_*"
localhost | SUCCESS => {
    "ansible_facts": {
        "ansible_processor_cores": 2,
        "ansible_processor_count": 1,
        "ansible_processor_threads_per_core": 2,
        "ansible_processor_vcpus": 4
    },
    "changed": false
}
```

command - (Shell-)Befehle ausführen

Mit dem “command”-Modul kann man Befehle/Kommandos direkt ausführen lassen - später natürlich auch parallel auf mehreren Hosts.

command-Modul mit uptime-Aufruf

```
1 % ansible localhost -m command -a uptime
2 localhost | success | rc=0 >>
3  10:43:46 up 17 days, 16:12, 3 users, load average: 0.11, 0.29, 0.30
```

- “command ist der Standard für ”-m“, kann also auch weggelassen werden
- Komplexere Befehle müssen ggf. entsprechend gequotet und in (einfache) Anführungszeichen gepackt werden

command - Weitere Aufrufe

Anzeige des “ankommenden” Benutzers (insbesondere bei SSH)

Aufruf des “id”-Befehles

```
1 % ansible localhost -m command -a id
2 localhost | success | rc=0 >>
3 uid=1000(debian) gid=100(users) groups=100(users)
```

Ein etwas komplexerer Aufruf des Moduls

Anzeige der Datei /etc/issue

```
1  % ansible localhost -m command -a "cat /etc/issue"
2 localhost | success | rc=0 >>
3 Debian GNU/Linux 8 \n \l
```

Weitere Parameter für “command”

Weitere Parameter für “command”:

- `chdir`
  Wechselt vor der Ausführung in das angegebene Verzeichnis
- `creates`
  Führt das Kommando aus, wenn die angegebene Datei **nicht** existiert!
- `removes`
  Führt das Kommando aus, wenn die angegebene Datei **existiert**!

Alternatives Modul:
“shell” - Ausführung in einer Shell, mit Umleitung und anderen Shell-Features!

Das “copy”-Modul

Das “copy”-Modul ermöglicht, wie der Name schon verrät, das Kopieren von Dateien auf den Zielrechner.

Simples Kopieren einer Datei - mit Fehler

```
1 % ansible localhost -m copy -a "src=test.txt dest=/tmp/test.txt"
2 localhost | FAILED >> {
3     "failed": true,
4     "msg": "could not find src=/home/debian/localhost/test.txt"
5 }
```

Da hat wohl die Datei nicht existiert...

Simples Kopieren einer Datei

```
1 % ansible localhost -m copy -a "src=test.txt dest=/tmp/test.txt"
2 localhost | success >> {
3     "changed": true,
4     "dest": "/tmp/test.txt",
```

Alternative Form für “src” bei “copy”

- Handelt es sich nur um eine kleine Datei bzw. wenig Inhalt, kann dieser auch direkt (“content=”) angegeben werden
- Des weiteren kann man auch Backups von geänderten Dateien anlegen lassen

“copy” mit “content” und “backup”

```
1 % ansible localhost -m copy -a 'content="Sven was here" dest=/tmp/test.txt
      backup=yes'
2 localhost | success >> {
3     "backup_file": "/tmp/test.txt.2016-03-24@11:38~",
4     "changed": true,
5     "dest": "/tmp/test.txt",
```

Was passiert, wenn man diesen Befehl nochmals ausführt?!

Besonderheiten beim “src”-Parameter

Je nach Angabe beim `src`-Parameter verhält sich “copy” unterschiedlich:

- Einzelne Datei:
  Die Datei wird kopiert
- Ein Verzeichnis **ohne** Slash am Ende:
  Das **Verzeichnis** (und alles darin) wird kopiert
- Ein Verzeichnis **mit** Slash am Ende:
  Nur der **Inhalt** des Verzeichnisses wird kopiert

Ansible 2.0:

- `remote_src`=True
  “`src`” wird auf der Zielmaschine gesucht

“copy” - Weitere Parameter

- `force=no`
  Kein Überschreiben bereits existierender Dateien
- `owner` / `group` / `mode`
  Eigentümer und Rechte der (neuen) Datei
- `selevel` / `serole` / `setype` / `seuser`
  Attribute für SELinux (falls im Einsatz)
- `validate`
  Command, welches **vor** dem endgültigen Kopieren ausgeführt wird, um sicher zu stellen, dass die Datei keine Syntax-Fehler enthält

Alternatives Modul:
“template” - Kopiert Datei(en) und ersetzt Variablen darin

Übersicht: Einstieg

Begriffklärung: “Plays”, “Playbooks”, ...

Task

Ein **Task** ist ein Aufruf eines Ansible-*Moduls* mit bestimmten Parametern, ggf. in einer Schleife und/oder abhängig von Bedingungen

Play

Ein **Play** bezeichnet die Abfolge mehrere *Tasks* auf eine(n) oder mehrere (Gruppe von) Rechnern, welche im *Inventory* definiert sind

Playbook

Ein **Playbook** besteht aus einem oder mehreren *Plays*

Ein minimales Playbook

Minimalistisches Playbook

```
1  ---
2  - hosts: localhost
3    tasks:
4
5    - name: Aufruf von "id"
6      command: id
7
8    - name: Kopieren einer Datei
9      copy: src=test.txt dest=/tmp/test.txt backup=yes
10
11   - name: Aufruf von "cat /tmp/test.txt"
12     command: cat /tmp/test.txt
```

Letztendlich “nur” eine Aneinanderreihung der `ansible`-Aufrufe mit den Modulen und passenden Parametern

Playbook-Syntax

- Playbooks werden in *YAML* geschrieben
- Einrückung mit **Leerzeichen**, Einrücktiefe **wichtig**!
- Ein Abschnitt bzw. die Datei beginnt mit "–––"

Später noch etwas mehr dazu...

Der erste `ansible-playbook`-Aufruf

Der Befehl

```
1 % ansible-playbook minimal.yml
```

- `ansible-playbook` (!) statt `ansible`
- **Keine** Limitierung
  $\Rightarrow$ alle Hosts, welche im Inventory erfasst sind
- Ein oder mehrere *Playbooks* (YAML-Dateien)

ansible-playbook-Aufruf - Ausgabe (1)

Der Anfang der Ausgabe

```
3 PLAY [localhost] ************************************************************
4
5 GATHERING FACTS *************************************************************
6 ok: [localhost]
7
8 TASK: [Aufruf von "id"] *****************************************************
```

- Der Play geht los
- Es wird ein Task names “GATHERING FACTS” gestartet
  Alternativ: “TASK [setup]”
- Erst danach kommt der Playbook-Task “Aufruf von id”

- Zu Beginn des Plays wird (wenn nicht unterdrückt) das Modul “setup” aufgerufen, welches eine Vielzahl von Variablen vorbelegt

`ansible-playbook`-Aufruf - Ausgabe (2)

Unsere Tasks

```
 8 TASK: [Aufruf von "id"] ******************************************
 9 changed: [localhost]
10
11 TASK: [Kopieren einer Datei] ************************************
12 changed: [localhost]
13
14 TASK: [Aufruf von "cat /tmp/test.txt"] ***************************
15 changed: [localhost]
```

- Jeder Task wird einzeln ausgeführt, **nacheinander**
- Alle drei Tasks melden “changed” zurück
- Es treten keine Fehler auf

`ansible-playbook`-Aufruf - Ausgabe (3)

Zusammenfassung

```
17 PLAY RECAP *********************************************************************
18 localhost              : ok=4    changed=3    unreachable=0    failed=0
```

- Auf “localhost” wurden vier Tasks ausgeführt, welche keinen Fehler zurücklieferten (“ok”)
- Drei der Tasks lieferten “changed” zurück
- Kein Host war “unreachable”
- Kein Tasks hat einen Fehler gemeldet (“failed”)

“Idempotency” - Oha!

Was passiert nun, wenn man den gleichen Befehl nochmals ausführt?

Ausgabe bei Wiederholung, Ausschnitt

```
 8 TASK: [Aufruf von "id"] ******************************************
 9 changed: [localhost]
10
11 TASK: [Kopieren einer Datei] ************************************
12 ok: [localhost]
13
14 TASK: [Aufruf von "cat /tmp/test.txt"] **************************
15 changed: [localhost]
16
17 PLAY RECAP ********************************************************
18 localhost      : ok=4   changed=2   unreachable=0   failed=0
```

“Kopieren einer Datei” meldet nun “ok”!
Siehe auch Zusammenfassung am Ende

Idempotenz

Zitat Wikipedia:

> *Idempotenz ist ein Begriff aus der Mathematik und Informatik.*
>
> *In der Mathematik bezeichnet man ein Objekt a, das mit einer Verknüpfung $\circ$ die Eigenschaft $a \circ a = a$ hat, als idempotent bezüglich dieser Verknüpfung. Ein wichtiger Spezialfall sind idempotente Funktionen bezüglich der Hintereinanderausführung.*
>
> ***Analog dazu wird in der Informatik ein Stück Programmcode, das mehrfach hintereinander ausgeführt das gleiche Ergebnis wie bei einer einzigen Ausführung liefert, als idempotent bezeichnet.***

Quelle: `https://de.wikipedia.org/wiki/Idempotenz`

Idempotenz - Was ist (nicht) passiert?

Was passiert?

- Ansible testet **vor dem Kopieren**, ob die Datei existiert bzw. sich von der neuen Version unterscheidet
- Nur wenn ein Unterschied festgestellt wird, ersetzt Ansible die Datei

Warum trifft das nicht auf die beiden “command”-Tasks zu?

- Ansible bzw. das “command”-Modul weiß schlicht und einfach nicht, was der Aufruf macht
- Daher werden die Befehle **immer** ausgeführt
- `creates` oder `removes` können dieses Verhalten verändern
  Alternativ: Ausführung nur, wenn Bedingung (nicht) erfüllt ist

Mehrere Plays in einem Playbook

Playbook mit mehreren Plays

```
1  ---
2  - hosts: gruppe1
3    tasks:
4
5    - name: Erster Task für Gruppe 1
6      command: cat /etc/issue
7
8    - [...]
9
10 - hosts: gruppe2
11   tasks:
12
13   - name: Erster Task für Gruppe 2
14     command: cat /etc/issue.net
15
16   - [...]
```

Übersicht: Einstieg

1. Einstieg
   - Installation
   - `ansible` - Module direkt aufrufen
   - Plays und Playbooks
   - Konfiguration

Notwendige Konfiguration

Um mit Ansible effektiv beginnen zu können, müssen einige Dinge konfiguriert werden:

- `ansible.cfg` - Zentrale Konfigurationsdatei
- `hosts` - Das sog. “Inventory”, definiert in `ansible.cfg`

Zentrale Konfigurationsdatei `ansible.cfg`:

- Package-Standard: `/etc/ansible/ansible.cfg`

“Inventory” `hosts`:

- In `ansible.cfg` definiert (siehe “`inventory`” bzw. “`hostfile`”)
- Package-Standard: `/etc/ansible/hosts`

`ansible.cfg` - Wo?

Ansible sucht nacheinander an folgenden Stellen bzw. Namen nach einer Konfiguratiosdatei (first wins):

- Umgebungsvariable `ANSIBLE_CONFIG`
- `ansible.cfg` im aktuellen Verzeichnis
- `.ansible.cfg` im Home-Verzeichnis des aktuellen Benutzers
- `/etc/ansible/ansible.cfg`

- Sinnvoll verwenden (kein Kauderwelsch)!
- Persönliche Meinung: Nur
  - **aktuelles Verzeichnis** und
  - `/etc/ansible/ansible.cfg`

`ansible.cfg` - Abschnitt “defaults”

Ansible 2.0: Abschnitt “defaults”

```
1  [defaults]
2  #inventory      = /etc/ansible/hosts
3  #library        = /usr/share/ansible
4  #remote_tmp     = $HOME/.ansible/tmp
5  ...
6  #sudo_user      = root
7  #ask_sudo_pass = True
8  #ask_pass       = True
9  #transport      = smart
10 #remote_port   = 22
11 #module_lang   = C
12 ...
13 #host_key_checking = False
14 #remote_user = root
```

- Nicht vollständig!
- Vor 2.0 heißt es “`hostfile`” statt “`inventory`”

Anlegen einer Konfiguration

Für die ersten Schritte bietet es sich an, ein Verzeichnis anzulegen und dort eine leicht angepasste `ansible.cfg` abzulegen

Anlegen des Verzeichnisses mit `ansible.cfg`

```
1 % mkdir ansible-testdir
2 % cd ansible-testdir
3 % cp /etc/ansible/ansible.cfg .
4 % sed -i 's/^#\?inventory.*$/inventory=.\/hosts/' ansible.cfg
```

Änderung: suche Inventory unter `./hosts`

Inventory (“`inventory`”/“`hostfile`”)

Allgemein:

- Klassische Ini-Style Konfigurationsdatei

Abschnitte (in eckigen Klammern):

- Eine Gruppe (Abschnitt) von Rechnern
- Weitere Möglichkeiten (Gruppe in Gruppen) mit besonderer Syntax (später...)

Zeilen in Gruppe:

- Jede Zeile definiert einen Rechner
- Format: `HOSTNAME [[Option/Variable] [Opt/Var] ...]`

Beispiel-Inventory

Ein Inventory für drei Rechner in zwei Gruppe kann wie folgt aussehen:

Inventory

```
1 [gruppe1]
2 rechner1
3 rechner2
4
5 [gruppe2]
6 rechner3
```

Weitere Möglichkeiten:

- Gruppe in Gruppen
- Variablen je Host definieieren
- Variaben für Gruppe definieren

Inventory mit verschiedenen Distributionen

Müssen verschiedene System gepflegt werden, kann man Ansible-Einstellungen im Inventory definieren

Inventory

```
1 [debian]
2 debian7 ansible_user=root
3 debian8 ansible_user=debian ansible_become=True
4
5 [ubuntu]
6 ubuntu  ansible_user=ubuntu ansible_become=True
7
8 [redhat]
9 rhel7   ansible_user=root
```

Defaults können in der `ansible.cfg` eingestellt werden

Ein simples Playbook: NTP

Übersicht: Ein simples Playbook: NTP

2. Ein simples Playbook: NTP
   - Statisches Playbook
   - Playbook mit Variablen

Vorgabe: NTP (Zeitserver) installieren

- **NTP**: “Network Time Protocol”
- Typische, verwendete Software:
  - **NTP**
  - **Chrony**
- Kontaktiert (definierte) Server(-Pools) und setzt lokale Zeit
- **NTP** kann auch externe (USB, serielle) Zeit-Empfänger, z. B. DCF-77 oder GPS, ansprechen

Empfehlung für’s LAN:

- 1-3 zentrale NTP-Server
- Alle anderen kontaktieren lokale Server

NTP: Vorgehen / notwendige Schritte

Vorgehen:

- Zuerst: notwendige Schritte überlegen
- Dann: Playbook schreiben

Notwendige Schritte:

1. NTP-Paket installieren
2. Konfigurationsdatei anpassen/schreiben
3. Dienst neu starten
4. Sicherstellen, dass Dienst auch bei Neustart läuft (“registrieren”)

NTP: Benötigte Module

Benötigte Module in Ansible:

- **Paket installieren**:
  - “package” (ab Ansible 2.0)
    verwendet intern passendes Modul
  - sonst: “apt”, “yum”, “zypper”, . . .
- **Konfiguration anpassen/schreiben**:
  - “copy”
  - Später “template” (dynamisches Anpassen)
- **Dienst (neu)starten, registrieren**:
  - “service”

NTP-Installation per Playbook

Playbook “ntp.yml”

```
1  ---
2  - hosts: all
3    tasks:
4
5      - name: Install NTP
6        package: name=ntp state=latest
7
8      - name: Copy config file
9        copy: src=ntp.conf dest=/etc/ntp.conf mode=0644 owner=root group=root
             backup=yes
10
11     - name: Restart NTP with new config
12       service: name=ntp state=restarted
13
14     - name: Ensure NTP is running
15       service: name=ntp state=started enabled=yes
```

Paket und Dienst heißen “ntp”

Konfigurationsdatei(en) für NTP

Natürlich braucht NPT auch eine Konfigurationsdatei:

Debian-Standard "`/etc/ntp.conf`", erleichtert um Kommentare

```
1  driftfile /var/lib/ntp/ntp.drift
2  statistics loopstats peerstats clockstats
3  filegen loopstats file loopstats type day enable
4  filegen peerstats file peerstats type day enable
5  filegen clockstats file clockstats type day enable
6  pool 0.debian.pool.ntp.org iburst
7  pool 1.debian.pool.ntp.org iburst
8  pool 2.debian.pool.ntp.org iburst
9  pool 3.debian.pool.ntp.org iburst
10 restrict -4 default kod notrap nomodify nopeer noquery limited
11 restrict -6 default kod notrap nomodify nopeer noquery limited
12 restrict 127.0.0.1
13 restrict ::1
14 restrict source notrap nomodify noquery
```

- "`pool`": Definition der Gegenstelle(n)

Ausführen von Playbooks

- Zum Ausführen von Playbooks wird `ansible-playbook` verwendet
- Wesentlicher Unterschied zu `ansible`:
  - Wird auf alle Hosts im Inventory (“all” bei `ansible`) ausgeführt
  - Kann per `--limit`/`-l` eingeschränkt werden
- Weitere Parameter (wie `-k`, `-K`, `--ask-become-pass`, `-u`) werden ebenfalls akzeptiert

Aufruf von `ansible-playbook`

```
1 % ansible-playbook ntp.yml
2 PLAY [all] ***************************************************************
```

Wichtige Schalter für `ansible`/`ansible-playbook`

Sollte kein Login ohne Passwort (SSH-Key, `sudo` ohne Passwort) möglich sein, kann `ansible`/`ansible-playbook` über einige Schalter gesteuert werden:

- `-u USER` / `--user=USER`
  Login-User (Default aktueller User)
- `-k` / `--ask-pass`
  Nach Login-Passwort fragen
- `-b` / `--become`
  Benutzt z. B. `sudo`, um anderer Benutzer zu werden
- `--become-method=METHOD`
  Stellt Method zum Benutzer-Wechsel ein (Default “sudo”)
- `--become-user=USER`
  Ziel-Benutzer für “become” (Default “root”)
- `-K` / `--ask-become-pass`
  Fragt nach entsprechendem Passwort

Ausgabe

Die Ausgabe am Beispeil eines einzelnen Rechners

```
% ansible-playbook ntp.yml
PLAY [all] ********************************************************************

GATHERING FACTS ***************************************************************
ok: [localhost]

TASK: [Install NTP] ***********************************************************
changed: [localhost]

TASK: [Copy config file] ******************************************************
changed: [localhost]

TASK: [Restart NTP with new config] *******************************************
changed: [localhost]

TASK: [Ensure NTP is running] *************************************************
ok: [localhost]
...
```

Übersicht: Ein simples Playbook: NTP

2 Ein simples Playbook: NTP
- Statisches Playbook
- Playbook mit Variablen

Potentielle Probleme mit dem NTP-Playbook

Es existieren verschiedene Probleme:

- Ggf. unterschiedliche Paket-Namen
- Ggf. unterschiedliche Servicen-Namen
- Je nach Setup unterschiedliche Konfigurationsdateien und/oder NTP-Server
- Vor Ansible 2.0: verschiedene Module zur Paket-Installation

Debian-Konfiguration als Basis

Als Basis soll die Debian-Konfiguration dienen:

Debian-Konfiguration ohne “pool”

```
1  driftfile /var/lib/ntp/ntp.drift
2  statistics loopstats peerstats clockstats
3  filegen loopstats file loopstats type day enable
4  filegen peerstats file peerstats type day enable
5  filegen clockstats file clockstats type day enable
6  # Hier standen vorher die "pool"-Einträge
7  restrict -4 default kod notrap nomodify nopeer noquery limited
8  restrict -6 default kod notrap nomodify nopeer noquery limited
9  restrict 127.0.0.1
10 restrict ::1
11 restrict source notrap nomodify noquery
```

Aufgabe: **Dynamisches** Hinzufügen der Server (“pool”)

Vorgehen dynamische NTP-Server

Im Wesentlichen folgende Schritte:

- Definition der Server in Variable(n)
- Ausrollen der (jeweils angepassten) Konfiguration
  Ersetzen des “copy”-Modul durch “template”

Vorgabe:
Defintion der Variable als Liste von Server-Namen/IP-Adressen

Variablen-Definition 1: Im Playbook

Im Playbook können Variablen im Abschnitt “`vars`” übergeben werden:

Anfang des Playbooks mit Variablen

```
1  ---
2  - hosts: all
3
4    vars:
5      ntp_server: [0.de.pool.ntp.org, 1.de.pool.ntp.org, 2.de.pool.ntp.org]
6
7    tasks:
8
9      - name: Install NTP
10       package: name=ntp state=latest
```

“template” statt “copy”

- Konfiguratiosdatei soll nicht mehr “nur” kopiert werden
- “Platzhalter” müssen durch Variablen befüllt werden
  $\Rightarrow$ “Ersetzungssprache” erforderlich
  Ansible verwendet “**Jinja2**” als Template-Sprache

Playbook: “copy” durch “template” ersetzen

- Ersetzung “copy” durch “template” im Playbook
- Anpassung des Dateinamens nach “ntp.conf.j2”

Der “template”-Task

```
12      - name: Copy config file
13        template: src=ntp.conf.j2 dest=/etc/ntp.conf mode=0644 owner=root
              group=root backup=yes
```

Zum Vergleich: der “copy”-Task

```
 8      - name: Copy config file
 9        copy: src=ntp.conf dest=/etc/ntp.conf mode=0644 owner=root group=root
              backup=yes
```

Das Template `ntp.conf.j2`

`ntp.conf.j2` auf dem Ansible-Master

```
1  # {{ ansible_managed }}
2  driftfile /var/lib/ntp/ntp.drift
3  statistics loopstats peerstats clockstats
4  filegen loopstats file loopstats type day enable
5  filegen peerstats file peerstats type day enable
6  filegen clockstats file clockstats type day enable
7  {% for server in ntp_server %}pool {{ server }} iburst
8  {% endfor %}
9  restrict -4 default kod notrap nomodify nopeer noquery limited
10 restrict -6 default kod notrap nomodify nopeer noquery limited
11 restrict 127.0.0.1
12 restrict ::1
13 restrict source notrap nomodify noquery
```

Ergebnis: `ntp.conf`

Das Ergebnis sieht wie folgt aus:

`ntp.conf` auf dem Server

```
1  # Ansible managed: .../ntp.conf.j2 modified on 2016-04-01 15:07:25 by svelt
       on ansible
2  driftfile /var/lib/ntp/ntp.drift
3  statistics loopstats peerstats clockstats
4  filegen loopstats file loopstats type day enable
5  filegen peerstats file peerstats type day enable
6  filegen clockstats file clockstats type day enable
7  pool 0.de.pool.ntp.org iburst
8  pool 1.de.pool.ntp.org iburst
9  pool 2.de.pool.ntp.org iburst
10 restrict -4 default kod notrap nomodify nopeer noquery limited
11 restrict -6 default kod notrap nomodify nopeer noquery limited
12 restrict 127.0.0.1
13 restrict ::1
14 restrict source notrap nomodify noquery
```

Variablen - wo kommen sie her?

Grobe Unterscheidung:

- Automatisch von Ansible befüllte Variablen
  Modul “setup”, welches automatisch beim Playbook-Start ausgeführt wird
- Vom Benutzer definierte Variablen

- Beide lassen sich z. B. in Jinja2-Templates verwenden

Variablen definieren: Wo?

- Aktuell: Variablen (hier: NTP-Server) in Playbook definiert
- Kein/Wenig Vorteil gegenüber Eintrag in Konfigurationsdatei direkt
- **Aber:** Variablen können an mehreren Stellen definiert werden!

Mögliche Variablen-Quellen:

- Playbook (später auch Rollen)
- Inventory (siehe "`ansible_...`")
  - Pro Host
  - Pro Gruppe
- Variablen-Dateien
  - Pro Host
  - Pro Gruppe

Gruppen-Variablen für Ubuntu

- Im Inventory existiert eine Gruppe “ubuntu”
- Ansible liest “`group_vars/ubuntu.yml`” relativ zum Playbook (wenn sie existiert)

./group_vars/ubuntu.yml

```
1 ---
2 ntp_servers:
3   - 0.ubuntu.pool.ntp.org
4   - 1.ubuntu.pool.ntp.org
5   - 2.ubuntu.pool.ntp.org
6   - 3.ubuntu.pool.ntp.org
```

Host-Variablen für “rhel7”

- Im Inventory existiert ein Host “rhel7”
- Ansible liest “`host_vars/rhel7.yml`” relativ zum Playbook (wenn sie existiert)

./host_vars/rhel7.yml

```
1 ---
2 ntp_servers:
3   - 0.de.pool.ntp.org
4   - 1.de.pool.ntp.org
5   - 0.pool.ntp.org
```

Konkretes Problem mit NTP-Playbook

- Bisher fest im Playbook:
  - Package-Name: “ntp”
  - Service-Name: “ntp”
- Funktioniert nicht, weil
  - in Redhat der Service “ntp**d**” heißt
  - in alten SUSE teilweise beides “ntp**d**”

Lösung: Variablen je Gruppe/Host

Playbook mit Variablen

```
1  ---
2  - hosts: all
3    vars:
4      ntp_server: [0.de.pool.ntp.org, 1.de.pool.ntp.org, 2.de.pool.ntp.org]
5    tasks:
6      - name: Install NTP
7        package: "name={{ ntp_package }} state=latest"
8      - name: Copy config file
9        template: src=ntp.conf.j2 dest=/etc/ntp.conf mode=0644 owner=root
             group=root backup=yes
10     - name: Restart NTP with new config
11       service: "name={{ ntp_service }} state=restarted"
12     - name: Ensure NTP is running
13       service: "name={{ ntp_service }} state=started enabled=yes"
```

Ersetzen der definierten Werte durch Variablen

Variablen definieren

Die Variablen **müssen** nun definiert werden

- Für alle Hosts (auch via Gruppen)
- Im Inventory oder in Variablen-Dateien unter `group_vars/` oder `host_vars/`

./group_vars/redhat.yml

```
1 ---
2 ntp_package: ntp
3 ntp_service: ntpd
```

Später (in einer Rolle): Default-Werte!

Rollen

3 Rollen

Aktuelle Situation

- Playbook erledigt einen Aufgabe (Installation eines NTP-Servers)
- In der Praxis: Viele (kleine) Aufgaben
- Komplexität entsteht durch Kombination
- Kombination aktuell: Mehrere bis viele Playbooks
- Alle Playbooks für alle (betreffenden) Rechner durchlaufen lassen

$\Rightarrow$ “Kapselung” der Aufgaben (jetzt noch Inhalt des Playbook) wäre sinnvoll
$\Rightarrow$ In Ansible: **Rollen** / **Rolls**

Übersicht: Rollen

3 Rollen
- Aufbau einer Rolle
- Mehrere Rollen

Speicherort der Rollen

Ansible sucht Rollen in

- `/etc/ansible/roles`
- Weitere Pfade, welche in `ansible.cfg` unter `roles_path` aufgelistet sind
  Mehrere Pfade mit Komma getrennt

Anpassen der `ansible.cfg`

```
1 % sed -i 's|# *roles_path.*$|&\nroles_path = /etc/ansible/roles,./roles|'
      ansible.cfg
```

...Hinzufügen von "`./roles`" zu "`roles_path`"

Aufbau des Playbooks in Rolle überführen

Das Playbook `ntp.yml` besteht aus 3 “Abschnitten”:

1. `hosts`:
   Für welche Hosts dieses Playbook abgearbeitet wird
   → Bleibt im Playbook (nicht in der Rolle)
2. `vars`:
   Eine Variable mit den gewünschten NTP-Servern
   → Wird zu einem Standardwert (“Default”) in der Rolle
3. `tasks`:
   Die abzuarbeitenden Schritte
   → Sind die Tasks der Rolle

Anlegen der Verzeichnisstruktur

Rollen werden strukturiert abgelegt

Verzeichnisstruktur anlegen

```
% mkdir -p roles/ntp/{tasks,templates,defaults}

% tree roles/ntp
roles/ntp
+-- defaults
+-- tasks
+-- templates

3 directories, 0 files
```

Übernahme in Rolle: Tasks

Im Verzeichnis `roles/ntp/tasks` wird die Datei `main.yml` mit den Tasks aus dem Playbook angelegt

roles/ntp/tasks/main.yml

```
1  ---
2  - name: Install NTP
3    package: name={{ ntp_package }} state=latest
4
5  - name: Copy config file
6    template: src=ntp.conf.j2 dest=/etc/ntp.conf mode=0644 owner=root
         group=root backup=yes
7
8  - name: Restart NTP with new config
9    service: name={{ ntp_service }} state=restarted
10
11 - name: Ensure NTP is running
12   service: name={{ ntp_service }} state=started enabled=yes
```

Übernahme in Rolle: Templates

- **Achtung!** “template” sucht nun seine Templates im Unterverzeichnis “`templates`” der Rolle!
- Es bietet sich daher an, in “`template`” eine (mehr oder minder) komplette Verzeichnisstruktur abzubilden
  Zum Beispiel: `./roles/ntp/templates/etc/ntp.conf.j2`
- Angabe der “`src`” **ohne** führenden “/” (Slash)

Übernahme in Rolle: Variable(n) als Default

Großer Vorteil von Defaults: Können durch Host- und/oder Gruppen-Variablen “überschrieben” werden

roles/ntp/defaults/main.yml

```
1 ---
2 ntp_servers:
3   - 0.de.pool.ntp.org
4   - 1.de.pool.ntp.org
5   - 2.de.pool.ntp.org
6   - 3.de.pool.ntp.org
7
8 ntp_package: ntp
9 ntp_service: ntp
```

Rolle in Playbook verwenden

Das Playbook kann nun die Rolle verwenden:

ntp.yml

```
1 ---
2 - hosts: all
3
4   roles:
5     - ntp
```

- Es können auch mehrere Rollen angegeben bzw. abgearbeitet werden
- Der Rolle können Variablen übergeben werden
- Rollen können von anderen Rollen abhängen

Übersicht: Rollen

3 Rollen
- Aufbau einer Rolle
- Mehrere Rollen

Mehrere Rollen in einem Playbook

In einem Playbook können auch mehrere Rollen (nacheinander) verwendet werden

Mehrere Rollen

```
1 ---
2 - hosts: all
3
4   roles:
5     - hosts
6     - ntp
```

Die Rollen werden in der angegebenen Reihenfolge abgearbeitet

Rollen-Abhängigkeit

Ist eine Rolle **zwingend** von einer anderen Rolle abhängig, so definiert man dies in "`meta/main.yml`" der abhängigen Rolle:

`roles/ntp/meta/main.yml`

```
1 ---
2 dependencies:
3   - { role: hosts }
```

- Es können beliebig viele Rollen angegeben werden
- Sie werden in der genannten Reihenfolge abgearbeitet
- Eine Rolle, welche mehrfach genannt wird, wird nur einmal abgearbeitet
- Übergabe von Variablen möglich

Fragestunde

4 Fragestunde

Übersicht: Fragestunde

4. Fragestunde
   - Monitoring-Plugins installieren
   - Ansible-Vault
   - Ansible-Galaxy
   - Module schreiben

Plugins installieren

Zwei Pakete:

- “Monitoring-Plugins”:
  Aktuell, gepflegt, wenn möglich verwenden
- “Nagios-Plugins:
  Nicht so gut gepflegt, nur bei alten Distributionen verwenden

⇒ Für Ansible: Erst MP probieren, dann NP installieren

Abschnitt aus “`roles/monitored/tasks/main.yml`”

```
19 - include: packages.yml
20   when: monitored_packages_install != False
```

Drei Wege zur Installation

Drei Wege, um Pakete für verschiedene Distributionen zu installieren:

- Ansible 2.0: Modul “package”
  Das einfachste, sauberste
- Simulation des Ansible-2-Verhaltens
  “hack’ish”, aber geht
- Unterscheidung nach Paket-Manager
  Tipp- und Fleißarbeit

Weitere “Probleme”:

- “Monitoring-Plugins” bevorzugt vor “Nagios-Plugins”
- Pakete heißen unterschiedlich

Installation: “package”

Variante für Ansible 2.0, sauber

Installation mit Hilfe von “package”

```
1  ---
2  - name: Install Monitoring-Plugins
3    package:
4      name: "{{ item }}"
5      state: latest
6    register: monitoringplugins
7    ignore_errors: True
8    with_items: "{{ monitored_packages_mp }}"
9
10
11 - name: Install Nagios-Plugins
12   package:
13     name: "{{ item }}"
14     state: latest
15   with_items: "{{ monitored_packages_np }}"
16   when: monitoringplugins|failed
```

Installation: “hack’ish”

Funktioniert mit Ansible 1.x, aber nicht schön

Installation Distributions-unabhängig

```
1  ---
2  - name: Install Monitoring-Plugins
3    action: "{{ ansible_pkg_mgr }} name={{ item }} state=latest"
4    register: monitoringplugins
5    ignore_errors: True
6    with_items: "{{ monitored_packages_mp }}"
7
8
9  - name: Install Nagios-Plugins
10   action: "{{ ansible_pkg_mgr }} name={{ item }} state=latest"
11   with_items: "{{ monitored_packages_np }}"
12   when: monitoringplugins|failed
```

Installation: Nach Paket-Manager

Tipp- und Fleißarbeit: unterschiedliche Module (in unterschiedlichen Dateien)

`tasks/packages.yml` mit Includes

```
1 ---
2 - include: packages_apt.yml
3   when: ansible_pkg_mgr == "apt"
4
5 - include: packages_yum.yml
6   when: ansible_pkg_mgr == "yum"
7
8 - include: packages_zypper.yml
9   when: ansible_pkg_mgr == "zypper"
```

Installation: Nach Paket-Manager, YUM

Das Beispiel anhand von YUM

tasks/packages_yum.yml

```
1  ---
2  - name: Install Monitoring-Plugins
3    yum:
4      name: "{{ item }}"
5      state: latest
6    register: monitoringplugins
7    ignore_errors: True
8    with_items: "{{ monitored_packages_mp }}"
9
10 - name: Install Nagios-Plugins
11   yum:
12     name: "{{ item }}"
13     state: latest
14   with_items: "{{ monitored_packages_np }}"
15   when: monitoringplugins|failed
```

Packages

Die Variablen für die Packages müssen noch definiert werden:

Debian/Ubuntu:

vars/debian.yml

```
2 monitored_packages_mp:
3   - monitoring-plugins
4 monitored_packages_np:
5   - nagios-plugins
```

RedHat/CentOS:

vars/redhat.yml, gekürzt

```
6  monitored_packages_mp:
7    - monitoring-plugins-disk
8    - monitoring-plugins-http
9    - monitoring-plugins-load
10   - monitoring-plugins-procs
11   - monitoring-plugins-smtp
12   - monitoring-plugins-ssh
13   - monitoring-plugins-swap
14   - monitoring-plugins-tcp
15   - monitoring-plugins-users
```

Übersicht: Fragestunde

4. Fragestunde
   - Monitoring-Plugins installieren
   - Ansible-Vault
   - Ansible-Galaxy
   - Module schreiben

Ansible-Vault - Sinn und Zweck

“Heikle” Informationen auf dem Kontroll-Host:

- Passwörter
- Private Schlüssel
- VPN-Keys
- ...

⇒ `ansible-vault` zum Verschlüsseln

Variante 1: Anlegen der Variablen-Datei mit Vault

Anlegen der Datei, Editor wird geöffnet:

Anlegen einer verschlüsselten Datei

```
% ansible-vault create secrets.yml
New Vault password:
Confirm New Vault password:
```

Anzeigen des Inhalts:

Anzeigen des Inhalts

```
% ansible-vault view secrets.yml
Vault password:
---
password: S4cur3
```

Variante 2: Verschlüsseln einer Datei

Verschlüsseln einer bestehenden Datei:

Verschlüsseln einer Datei

```
% ansible-vault encrypt secrets.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

- Meist bei der Entwicklung der einfachere Weg...
- Funktioniert auch mit `group_vars/` und `host_vars/`

Editieren und Entschlüsseln

Editieren einer verschlüsselten Datei:

Editieren der Datei

```
% ansible-vault edit secrets.yml
Vault password:
```

- Die verschlüsselte Datei wird erst bei **Beenden** wieder geschrieben!

Entschlüsseln einer Datei:

Entschlüsseln einer Datei

```
% ansible-vault decrypt secrets.yml
Vault password:
Decryption successful
```

Playbook mit Vault

`ansible-playbook` kann direkt das Passwort für die verwendeten Vault-Files mitgegeben werden:

Mit Vault-Passwort

```
1 % ansible-playbook --ask-vault-pass site.yml
```

Alternativ: Passwort in Datei oder durch Skript (ausführbar) ausgegeben:

Mit Vault-Passwort-Datei/-Skript

```
1 % ansible-playbook --vault-password-file ... site.yml
```

Übersicht: Fragestunde

4. Fragestunde
   - Monitoring-Plugins installieren
   - Ansible-Vault
   - Ansible-Galaxy
   - Module schreiben

Was ist “Ansible-Galaxy”

“Ansible Galaxy is your hub for finding, reusing and sharing the best Ansible content.”
Quelle: `http://galaxy.ansible.com`

- Basierend auf Rollen
- Name immer “`username.role`”

- Funktioniert teilweise nicht mit Ansible 2.0
- Neuen (Beta-)Server angeben

Vor- und Nachteile

Vorteile:

- Sehr viele Rollen vorhanden
- Viel “Grips”, viel Code zum Lernen
- Abhängigkeits-Management (Datei “`requirements.yml`”)

Nachteile:

- Suche nach “collectd” bringt rund
  - 20 Rollen zum Installieren von “collectd” in verschiedenen Varianten
  - 20 Rollen, in denen ebenfalls “collectd” installiert wird
- Oft auf bestimmte Szenarien/Umgebungen zugeschnitten
- Oder: irrsinnig komplex (Defintion von 50+ Variablen)
- Selten Distributions-übergreifend

Suchen und weitere Informationen

Aufruf - Suchen

```
% ansible-galaxy search naemon

Found 1 roles matching your search:

 Name           Description
  ----           -----------
   deimosfr.naemon Ansible playbook to install Naemon
```

Aufruf - Informationen, Anfang

```
% ansible-galaxy info deimosfr.naemon

Role: deimosfr.naemon
        description: Ansible playbook to install Naemon
        active: True
        commit: 43ac2b61972c561bed6e3491f36d2a28b89cd657
        commit_message: Adding default params
```

Installieren mit `ansible-galaxy`

Installation einer Role:

- Direkter Aufruf:
  `ansible-galaxy install username.role`
- Mit Datei:
  - Liste der Abhängigkeiten:
    `ansible-galaxy -r requirements.txt`
  - `requirements.yml` einer Rolle (ab 1.8):
    `ansible-galaxy -r requirements.yml`

In beiden Fällen können in der Datei bestimmte Versionsnummern mitgegeben werden

Installieren, Liste der vorh. Rollen, Entfernen

Aufruf - Installation

```
% ansible-galaxy -p ./roles install deimosfr.naemon
- downloading role 'naemon', owned by deimosfr
- downloading role from
    https://github.com/deimosfr/ansible-naemon/archive/v1.1.tar.gz
- extracting deimosfr.naemon to ./roles/deimosfr.naemon
- deimosfr.naemon was installed successfully
```

Aufruf - Liste

```
% ansible-galaxy -p ./roles list
- deimosfr.naemon, v1.1
```

Aufruf - Entfernen

```
% ansible-galaxy -p ./roles remove deimosfr.naemon
- successfully removed deimosfr.naemon
```

Vorlage für neue Rolle erstellen

Aufruf von `ansible-galaxy init`

```
% ansible-galaxy init newrole
- newrole was created successfully
% tree newrole
newrole
|-- README.md
|-- defaults
|   `-- main.yml
|-- files
|-- handlers
|   `-- main.yml
|-- meta
|   `-- main.yml
|-- tasks
|   `-- main.yml
|-- templates
`-- vars
    `-- main.yml

7 directories, 6 files
```

Übersicht: Fragestunde

4. Fragestunde
   - Monitoring-Plugins installieren
   - Ansible-Vault
   - Ansible-Galaxy
   - Module schreiben

Eigene Module schreiben

- Python bietet sich an
- Jede andere Sprache (auch Shell-Skripten) funktioniert
- Rückgabe (Standard-Ausgabe) muss passen
- Ablage im Verzeichnis "`./library/`", relativ zum Playbook bzw. in der Rolle

Minimalistisches Shell-Modul

```
1 #!/bin/bash
2
3 # Ansible übergibt Dateiname - GEFÄHRLICH!
4 source ${1}
5
6 # Hier könnte nun was passieren
7 # echo "changed=True msg=OK"
8
9 echo "changed=False"
```

Konkrete Idee: Debian-Apache-style-Config-Dirs

- Debian-Apache unterscheidet …
  - verfügbare (“available”)
  - aktivierte (“enabled”)

  Config-Schnippsel
- Verzeichnisse für …
  - Module (“mods-*”)
  - VHosts (“sites-*”)
  - seit Jessie auch allgemein (“conf-*”)
- Befehle
  - `a2enmod` & `a2dismod`
  - `a2ensite` & `a2dissite`
  - `a2enconf` & `a2disconf` (ab Jessie)

Parameter im Modul

Für das Modul:

- Basis-Verzeichnis “path” (z. B. “`/etc/apache2/`” [1])
- Basis-Name der Config-Verzeichnisse “base” (z. B. “`conf-`” [2])
- Endungen der Verzeichnisse (Standard: “p_available”: “`available`” und “p_enabled”: “`enabled`”)
- Endung “suffix” der Schnippsel-Dateien (Standard: “`.conf`”)
- Name “name” der zu managenden Datei (z. B. “`foo`” [1])
- “state” des Links (Standard: “present”, möglich “absent”)

[1] Muss angegeben werden
[2] Kann auch in “path” enthalten sein

Modul in Python

Für Python vieles fertig:

Parameter-Übergabe aus Ansible

```
7     module = AnsibleModule(
8         argument_spec = dict(
9             path = dict(required=True),
10            base = dict(default=""),
11            p_enabled = dict(default='enabled'),
12            p_available = dict(default='available'),
13            name = dict(required=True),
14            suffix= dict(default='.conf'),
15            state = dict(default='present', choices=['present', 'absent']),
16        )
17    )
```

Modul beenden

Verschiede Fälle

```
57      if state:
58          # try to create symlink
59          try:
60              os.symlink(os.path.relpath(src, p_enabled), dest)
61          except:
62              module.fail_json(msg="Could not create symlink")
63      else:
64          # test, if symlink
65          if not os.path.islink(dest):
66              module.fail_json(msg="Destination is not a symlink")
67
68          # try to remove symlink
69          try:
70              os.unlink(dest)
71          except:
72              module.fail_json(msg="Could not remove symlink")
73
74      module.exit_json(changed=True)
```

Aufrufen des Moduls

Voraussetzungen

```
% cd /tmp/apache2
% ls -ldR **/*
drwxrwxr-x 2 ... mods-available/
-rw-rw-r-- 1 ... mods-available/foo.conf
drwxrwxr-x 2 ... mods-enabled/
```

Link anlegen

```
% ansible -i hosts.localhost all -m availenabled -a "path=/tmp/apache2
    base=mods- name=foo"
localhost | SUCCESS => {
    "changed": true
}
% ls -ldR **/*
drwxrwxr-x 2 ... mods-available/
-rw-rw-r-- 1 ... mods-available/foo.conf
drwxrwxr-x 2 ... mods-enabled/
lrwxrwxrwx 1 ... mods-enabled/foo.conf -> ../mods-available/foo.conf
```

Noch Fragen?

Ansonsten:

- Danke!
- Guten Heimweg!
- Ich freue mich auf ein Wiedersehen!

Config-Management mit Ansible

Sven Velt - sven@velt.biz

velt.biz

Open-Source **Consulting**
& Training

17. Juni 2016