Verteilen, konfigurieren, machen

Aufwärmen/Warmup

Kleine Fragestunde: Konfigurationsmanagement

Kleine Fragestunde: Konfigurationsmanagement

bcfg2,

Kleine Fragestunde: Konfigurationsmanagement

bcfg2, puppet,

Kleine Fragestunde: Konfigurationsmanagement

bcfg2, puppet, chef,

Kleine Fragestunde: Konfigurationsmanagement

bcfg2, puppet, chef, cfengine,

Kleine Fragestunde: Konfigurationsmanagement

bcfg2, puppet, chef, cfengine, was anderes.

Kleine Fragestunde:
Konfigurationsmanagement

bcfg2, puppet, chef, cfengine, was anderes.

Motivation

Motivation

- Anzahl der Server wächst

Motivation

- Anzahl der Server wächst
- Anzahl der Admins stagniert

Motivation

- Anzahl der Server wächst
- Anzahl der Admins stagniert
- Wartbar

Motivation

- Anzahl der Server wächst
- Anzahl der Admins stagniert
- Wartbar
- Reproduzierbar

Motivation

- Anzahl der Server wächst
- Anzahl der Admins stagniert
- Wartbar
- Reproduzierbar
- Code als Dokumentation

*An **ansible** is a fictitious machine capable of instantaneous or superluminal communication. Typically it is depicted as a lunch-box sized object with some combination of microphone, speaker, keyboard and display. It can send and receive messages to and from a corresponding device over any distance whatsoever with no delay.*

@thomasfuchs I hear the cool kids raving about Ansible these days

@rvagg

@thomasfuchs chef is pretty bloated. Puppet is a bit more nimble. Salt and ansible is good alternatives

@uggedal

@thomasfuchs @mikey_p yeah, what do you need? http://ansible.cc is agentless, serverless, orchs. multi-tier, super short learning curve

@laserlama

- Hauptentwickler Michael DeHaan: `Cobbler`, `puppet` und `func`
- geschrieben in Python
- aktuelle Version 1.0 (02-01-2013)
- ca. 1 Jahr alt
- 17,455 SLOC (`sloccount`) (`puppet-2.6.1` 109,148 SLOC)
- Code: `http://github.com/ansible/ansible`
- Dokumentation: `http://ansible.cc/`

Voraussetzungen

1. Unix
2. Client: Python 2.4 + python-simplejson, Python 2.6
3. Server: Python 2.6, python-jinja2, python-yaml, python-crypto, python-paramiko

Wo(für) gibt es Ansible

- Linux
- OpenBSD
- NetBSD
- MacPorts/HomeBrew
- Solaris

Bausteine

- Programm `ansible`
- Module
- Programm `ansible-playbook`
- Playbooks
- Inventory
- Variablen
- Templates

Ansible

Kommandozeilen-Werkzeug für *Machen*

```
ansible group --list-hosts
ansible group -m setup --tree /tmp/fakten
ansible host -m shell -a uptime
```

Ansible II

Kommandozeilen-Werkzeug für *Verteilen*

```
ansible host -m copy -a "src=blub dest=/tmp/bla"
ansible group -m apt -a "name=foobar state=installed"
```

Ein paar Beispiele

```
ansible linuxserver --one-line -i ansible_hosts -m ping

ansible linuxserver --user operator --ask-pass\
> --one-line -i ansible_hosts -m shell\
> -a"date --rfc-3339=seconds"
```

One-Shot/On-All

```
ansible hostgroup -m shell -a uptime
clush -g all uptime
```

Ansible Output

```
ansible linuxserver --user operator \
> --ask-pass --one-line -i ansible_hosts -m shell -a"date"
SSH password:
basilikum | success | rc=0 | (stdout) Sun Feb 10 18:13:34 CET 20
brainfreeze | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET
foehr | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET 2013
faser | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET 2013
control | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET 2013
icmx1 | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET 2013
import1 | success | rc=0 | (stdout) Sun Feb 10 18:13:35 CET 2013
loeserv2 | success | rc=0 | (stdout) Sun Feb 10 18:13:36 CET 201
vcs | FAILED => FAILED: [Errno 111] Connection refused
servilo | success | rc=0 | (stdout) Sun Feb 10 18:13:36 CET 2013
sp-test | success | rc=0 | (stdout) Sun Feb 10 18:13:37 CET 2013
usedom | success | rc=0 | (stdout) Sun Feb 10 18:13:37 CET 2013
swup | success | rc=0 | (stdout) Sun Feb 10 18:13:37 CET 2013
weg.campus.fu-berlin.de | success | rc=0 | (stdout) Sun Feb 10 1
```

Clush Output

```
clush -g all uptime
vcs: ssh: connect to host vcs port 22: Connection refused
clush: vcs: exited with exit code 255
basilikum:  23:01:45 up 8 days, 10:15,  0 users,  load average:
faser-neu:  23:01:45 up 8 days,  9:58,  0 users,  load average:
sp-test:  23:01:45 up 8 days, 10:11,  0 users,  load average: 0
icmx1:  23:01:45 up 127 days,  7:58,  0 users,  load average: 0
usedom:  23:01:45 up 126 days, 17:51,  0 users,  load average: 0
control:  23:01:46 up 135 days,  9:01,  2 users,  load average:
brainfreeze:  23:01:46 up 134 days,  7:28,  0 users,  load avera
swup:  23:01:46 up 135 days,  7:49,  0 users,  load average: 0.0
import1:  23:01:46 up 127 days,  8:03,  0 users,  load average:
servilo:  23:01:46 up 117 days, 10:57,  0 users,  load average:
loeserv2:  23:01:46 up 131 days, 13:21,  0 users,  load average
```

Inventory

Inventory:	Liste der Server, Servergruppen
Inventory-Datei:	ist eine INI-Datei
Externe Quellen:	mit Plugins möglich
Kontrolle:	`ansible group -i <file> --list-hosts`

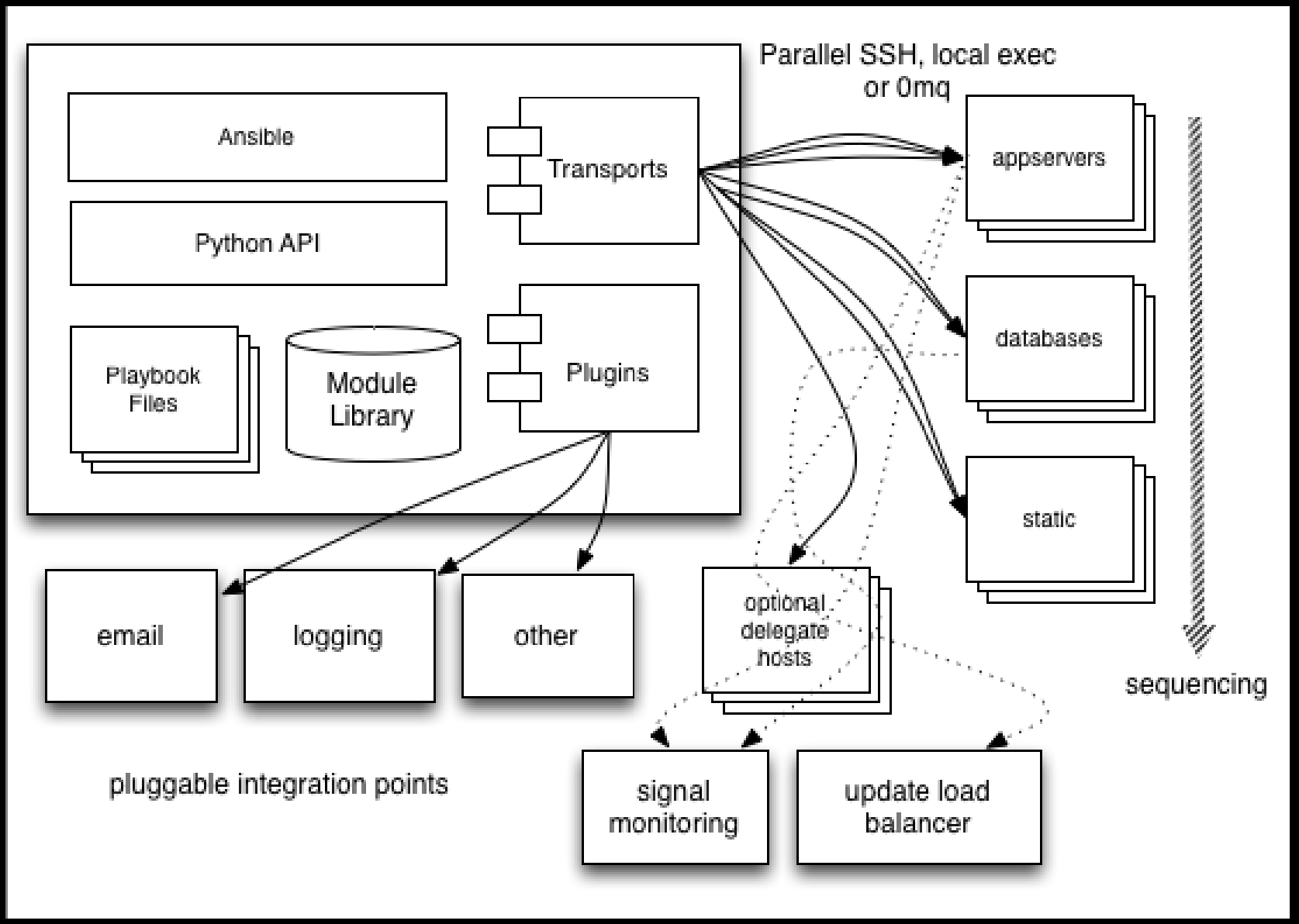
Inventory II

```
[web]
willi port=8080
walter
werner

[web:vars]
timeserver=time.ptb.de

[db]
heiner
hans

[linux:children]
db
web
```

Ablauf

1. pro Host (aus Liste)
2. transportiere Modul
3. führe es aus
4. sammle Output, Returnwert

Architektur



Ansible-playbook

- Playbooks
- YAML
- Tasks
- Variablen
- Handler

Playbooks

- Regelwerk für Konfiguration
- sollten idempotent sein
- können Tasks und weitere Playbooks einbinden

YAML: YAML Ain't Markup Language

```
1  ---
2  - name: Web server setup
3    hosts: web
4    user: root
5
6    vars:
7      http_proxy: http://example.com:80/
8
9    tasks:
10     - name: install Apache
11       actions: apt package=apache2 state=installed
```

YAML Beispiel

```
1  tasks:
2    - name: Validating setup
3      action: fail msg="owner, group and webroot must be set!"
4      with_items: [ $owner, $group, $webroot ]
5      when_unset: $item
```

Playbook Beispiel I

```
1  ---
2  # serversetup.yml
3  - name: Basic configration setup
4    hosts: wls
5    user: root
6    connection: ssh
7    vars:
8      template_dir: /home/phgrau/work/cm/ansible/templates
9      file_dir: /home/phgrau/work/cm/ansible/files
10     task_dir: /home/phgrau/work/cm/ansible/tasks
11
12   tasks:
13   - name: Show Info
14     local_action: debug msg="System $inventory_hostname has uuid
15                   $ansible_product_uuid $ansible_hostname"
```

Playbook Beispiel I

```
1     - include: ${task_dir}/check-etc-hosts.yml
2
3     - include: ${task_dir}/exim-setup.yml
4
5     - name: Disable IPv6
6       sysctl: name=net.ipv6.conf.all.disable_ipv6
7                 state=present value=1
8
9     - name: Get a descent resolv.conf
10      copy: src=${file_dir}/resolv.conf dest=/etc/resolv.conf
11              backup=yes
12
13    - name: Ensure python-software-properties is installed
14      apt: name=python-software-properties state=installed
```

Playbook Beispiel I

```
1  ---
2  # Exim setup
3
4    - name: Exim | First we need an exim
5      apt: package=exim4-daemon-light state=installed
6
7    - name: Exim | create exim config
8      action: template
9      src=${template_dir}/etc-exim4-update-exim4.conf.conf.j2
10                dest=/etc/exim4/update-exim4.conf.conf backup=yes
11     register: last_result
12
13   - name: Exim | rebuild exim config only if
14     command: /usr/sbin/update-exim4.conf
15     only_if: '${last_result.changed}'
16
17   - name: Exim | sendtest mail
18     shell: 'cat /etc/mailname | mail -s test phgrau@zedat.fu-ber
19     only_if: '${last_result.changed}'
```

Playbook Beispiel I

```
1     - name: Copy WLS-apache2-site configuration
2       copy: src=${file_dir}/weg-site-wls
3             dest=/etc/apache2/sites-available/weg backup=yes
4       notify: restart apache2
5
6     - name: Verify that WLS-Site is on
7       command: /usr/sbin/a2ensite weg
8                creates=/etc/apache2/sites-enabled/weg
9       notify: restart apache2
10
11    handlers:
12    - name: restart chrony
13      action: service name=chrony state=restarted
14    - name: restart apache2
15      action: service name=apache2 state=restarted
```

Playbook Beispiel II

```
1  ---
2  - name: HIS | Debian Package stuff
3    hosts: his-servers
4    user: vagrant
5    sudo: True
6    gather_facts: False
7
8    vars:
9     pgv: 9.1
10
11   tasks:
12   - name: HIS | Postgres packages $items I
13     apt: pkg=$item install_recommends=yes
14                    default_release=squeeze-pgdg
15                    state=latest
16     with_items:
17       - libpq5
18       - postgresql-${pgv}
19       - postgresql-doc-${pgv}
20       - postgresql-client-${pgv}
```

Verbindungstypen

Paramiko: SSH2-Implementation in Python

SSH: „Normale“ SSH-Verbindungen

Fireball: Library `0mq` Socket-System (10 * Schneller als Paramiko)

Rückgabe-String aus Ansible

- JSON JavaScript Object Notation
- Untermenge von YAML
- Mehr Syntax: " und , und Klammerung
- Serialisierung von Objekten

```
vagrant | success >> {
    "changed": true,
    "dest": "/tmp/motd-backup",
    "group": "vagrant",
    "md5sum": "99f4718b7cb986930041a2090c8ef9b6",
    "mode": "0664",
    "owner": "vagrant",
    "size": 106,
    "src": "/home/vagrant/.ansible/tmp/ansible-1362048596.16-179
    "state": "file"
}
```

Beispiel für Bladecenter-Konfigurations-Abfragen

```
---
- hosts: bladecenter
  user: bcuser
  sudo: no
  gather_facts: false

  tasks:
    - name: BC | Collect server names
      action: raw executable= show server names
      register: bc_conf

    - name: BC | put Information in file
      local_action: template src=bc_conf.j2 dest=/tmp/bc_conf
```

```
ansible-playbook -K bc-inventory.yml
```

API

```
#!/usr/bin/env python

import ansible.runner
import pprint

pp = pprint.PrettyPrinter()

runner = ansible.runner.Runner(
    module_name='command',
    module_args='/bin/date',
    pattern="linuxserver"
)
results = runner.run()

pp.pprint(results)
```

API

```
if results is None:
    print "No hosts found"
    sys.exit(1)

print "UP ***********"
for (hostname, result) in results['contacted'].items():
    if not 'failed' in result:
        print "%s >>> %s" % (hostname, result['stdout'])

print "FAILED *******"
for (hostname, result) in results['contacted'].items():
    if 'failed' in result:
        print "%s >>> %s" % (hostname, result['msg'])

print "DOWN *********"
for (hostname, result) in results['dark'].items():
    print "%s >>> %s" % (hostname, result)
```

Vagrant

- Einbindung in Vagrant (`vagrant-ansible`)
- virtuelle Maschinen mit VirtualBox
- einfaches Interface: `vagrant init ubuntu`,
- `vagrant up`
- Einbindung eines Playbooks
- `vagrant provision` (natürlich mehrfach möglich, Idempotenz)

Module

- Herz von Ansible (Workhorses)
- sollten idempotent sein
- in Python

Shell

raw: Gut für Router, Bladecenter, wirft einen Befehl ab

script: Kopiert Script von Server zum Client und startet es (Kein Python, aber Shell)

shell: Shellscript in der Umgebung des Remote-Users (kann idempotent gemacht werden)

command: Braucht Pfade

Datei

fetch:	Datei vom Client zu Server
file:	Mach alles mögliche mit einer Datei
assemble:	Bau eine Datei aus Stücken zusammen
ini_file:	INI-Dateien bearbeiten
lineinfile:	Eine Zeile bearbeiten (Für den Cfengine-Liebhaber)
template:	Der wahre™ Weg: Jinja2-Templates
slurp:	Datei vom Client zum Server base64-encodiert
get_url:	Datei per HTTPS/HTTP/FTP runterladen

Datenbank

mysql_db:	MySQL-Datenbanken
mysql_user:	MySQL-User
postgresql_db:	Postgres-Datenbanken
postgresql_user:	Postgres-User

DEB und RPM

apt:	DEB-Paket-Verwaltung
apt_key:	Repository-Key-Verwaltung
apt_repository:	Debian/Ubuntu Repository-URLs
yum:	RPM-Pakete

Unix-Grundbedürfnisse

cron:	cron-Einträge
group:	Gruppen verwalten
user:	User verwalten
authorized_key:	ssh key bei Client-Accounts eintragen und austragen
sysctl:	Pflegt Einträge für sysctl
mount:	`/etc/fstab`
service:	startet/stoppt Dienste

Versionsverwaltungen

subversion:	SVN
git:	GIT
hg:	Mercurial

Python

easy_install:

pip:

Wolke

ec2:

ec2_facts:

SELinux

seboolean:

selinux:

Ansible intern

setup:	Fakten sammeln
add_host:	Neuen Host oder Gruppe bauen, für weitere Benutzung
wait_for:	Tomcat
pause:	Press Any Key
debug:	Debug Statement
facter:	Puppets `facter`
ohai:	`ohai` von Chef
fail:	Ende mit Meldung
group_by:	Gruppe von Servern in Abhängigkeit von Variablen
mail:	Mail verschicken

Verschiedenes

nagios:	Downtimes und Alerts
pacman:	Archlinux Packages
ping:	Test-Modul
django:	Django-Management
pkgin:	SmartOS Packages
supervisorctl:	Programme die mit `supervisord` verwaltet werden
svr4pkg:	Solaris 10/11 Pakete (altes Format)
update_alternatives:	Selbst gebaut (nicht schön, aber es ist nicht so schwer...)
virt:	Verwaltet `libvirt` verwaltete Systeme

Fakten: Facts, Facter, ohai

- Das Modul `setup` sammelt Informationen
- Werden als Variablen bereitgestellt
- Auswertung in Templates und der Ablaufsteuerung möglich
- `gather_facts: false` schaltet das Sammeln aus

```
control | success >>
{
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "130.133.175.100"
        ],
        "ansible_all_ipv6_addresses": [],
        "ansible_architecture": "i386",
        "ansible_bios_date": "03/03/2005",
        "ansible_processor_cores": 2,
        "ansible_processor_count": 2,
        "ansible_product_name": "ProLiant DL360 G3",
        "ansible_product_serial": "NA",
        "ansible_system": "Linux",
        "ansible_system_vendor": "HP"
    },
    "changed": false,
    "verbose_override": true
}
```

Alle Informationen sammeln

```
ansible server -i productions_hosts  --user operator \
--ask-pass -m setup --tree /tmp/facts
```

Templates mit jinja2

```
{# ... #} {# Kommentar #}
{% ... %} {# Statements, Loop, If #}
{{ ... }} {# Variablen-Ausgabe #}
```

Templates: Whitespace

```
{% for item in seq -%}
    {{ item }}
{%- endfor %}
```

```
 123456789
```

Templates: Variablen

```
# {{ ansible_managed }}

server time.fu-berlin.de offline minpoll 8
server zeit.fu-berlin.de offline minpoll 8

keyfile /etc/chrony/chrony.keys
[...]

{% if ansible_hostname == "control" %}
allow 192.168/16
{% else %}
allow 130.133/16
{% endif %}
```

Protokolle, Logging

Client: Gibt immer syslog-Einträge

Es gibt verschiedene Callback-Plugins

- mail.py: Fehlermeldungen per Mail
- log_play: Eigene Logfiles (ohne Syslog)
- sql.py: Results to database for better result checking

log_play–Callback

- Pro Host eine Datei (Ort ist im Moment hartcodiert)

```
Feb 11 2013 15:04:30 - OK - {"module_name": "shell", "module_arg
{"changed": true, "end": "2013-02-11 15:04:30.385562",
 "stdout": " 15:04:30 up 6 days,  2:18,  1 user,  load average:
 "cmd": "uptime ", "rc": 0,
 "start": "2013-02-11 15:04:30.383776",
 "stderr": "", "delta": "0:00:00.001786"}
```

Problembehandlung

```
export ANSIBLE_KEEP_REMOTE_FILES=1
ansible-playbook --syntax-check
ansible-playbook -v playbook.yml
ansible-playbook -vvv playbook.yml
```

Was hat es für das *perfekte* Werkzeug

- keine PKI
- One-Shot und On-All werden einfach in ein Playbook überführt
- Wiederverwendbarkeit
- einfache von Oben nach Unten Abarbeitung der Tasks
- YAML, einfach zu lesen, keine DSL

Was fehlt für das *perfekte* Werkzeug

- `--dry-run` (`--check` wird noch eingebaut, `--diff` vorhanden)
- Logging per Syslog auf dem Server
- Klicki-Bunti-Oberfläche mit „Runden Ecken“
- Dashboard
- Verbesserter Einsatz im Cron, Automatisierung (ssh-agent, SSH-Key ohne Passphrase?)
- Noch nicht alles „ausgereift“ (es gibt noch Ecken und Kanten)

Bemerkungen

- Schnelle Entwicklung, aber keine Brüche
- Netter Chef-Entwickler
- Guter Support im IRC-Channel
- Gute Dokumentation

Auslassungen

- `ansible-pull`
- Details zum Zugriff auf Facts und Variablen
- Facter und Ohai
- Tags
- `cowsay`
- `group_vars` und `host_vars`
- Fireball-Mode
- `ignore_errors`

Vergleich: Die „Big-Player“

- cfengine
- Chef
- Puppet
- bcfg2

Die anderen Marktteilnehmer

- clush (Cluster-Shell)
- Sprinkle (Python)
- (R)?ex rexify.org (Perl)
- slaughter (Perl)
- Salt Stack (Python)

Wo geht man schauen?

- http://ansible.cc
- http://github.com/ansible/ansible
- `#ansible` im Freenode-Netz
- jpmens Blog http://jpmens.net
- https://coderwall.com/p/t/ansible

Danke.

Danke.

Fragen?