SaltStack

Alexander Elbs, 05.12.2022

Warum?

Administration einer Linux-Maschine

- Am Anfang alles manuell: Alleine klappt das ganz gut
- In einem Team: Oh, wer hat das da geändert?
- Mit vielen Servern: Alle Server brauchen diese oder jene Konfiguration / Software
- Re-installation: Hmm, wie hatten wir den genau konfiguriert?

Verbesserung: Wiki

- Wir dokumentieren jetzt alles im Wiki!
- Jede Installation, jede Config
- Wenn sich was ändert ziehen wir das im Wiki nach
- Dann aber: oh ich muss eben mal schnell das anpassen, Wiki vergessen
- Bald: Keiner weiß mehr so genau, ob Wiki stimmt: Soll das so sein? Oder ist das veraltet?

Verbesserung: Config Management

- Zustand einer Maschine wird in einer Computersprache beschrieben
- Dieser Zustand wird von einer Software angewendet
- Gemeinsamkeiten von mehreren Rechner: nur einmal hinschreiben
- Änderungen unter Versionsverwaltung
- Am besten nichts mehr manuell anpassen

Versionsverwaltung

- Versionsverwaltung:
  - Wer hat wann warum etwas geändert?
    - Commits enthalten automatisch Datum und Autor
    - Warum: Immer dazu schreiben, warum etwas getan wurde.
      - Gut:
        - „Ticket 123: Neuer Mitarbeiter Hans Wurst, E-Mail angelegt“
      - Schlecht:
        - „mail“ (ja, da wurde eine E-Mail geändert, aber warum?)
        - „elbs“ (im Commit steht, wer du bist, braucht man nicht in die Nachricht schreiben. Und wArUm?!?)
- Beschwerde von Chef: „bla ist falsch!!! Warum machst du sowas?!“
  - Antwort: „Am 24.12.1912 hab ich das commited mit „Chef will bla““

git

Kurzer Ausflug: Versionsverwaltung mit git

- „git clone …“: Das git repo initial holen
- „git pull“: Neueste Änderungen von Kollegen holen
- „git commit DATEI“: Eine eigene Änderung an DATEI im git speichern. Editor startet für Commit Nachricht: WARUM
- „git push“: Einen oder mehrere Commits veröffentlichen
- Es gibt für alles Youtube-Videos: https://www.youtube.com/results?search_query=git

Reproduzierbar

- Mit CM ist der Zustand eines Rechners „reproduzierbar“
- Nach Desaster kann er re-installiert werden.
  - Was fehlt: die Nutzdaten. Backup!
- Neue Rechner können analog zu anderen Rechnern installiert werden. Noch ein imap-Server, …
- Alle Rechner bekommen die immer-gleichen Dinge („Basisinstallation“):
  - Mailanbindung (für admin mails), logcheck, ...
  - AD-Integration, ssh-Zugang der Admins, ...
  - Auto-Update, …
  - …
- Dann werden sie spezialisiert, z.B. Webserver, SMTP-Server, ...

Was fehlt?

- CM ist nur ein Baustein
- Monitoring: Nagios, Prometheus
- Ungewöhnliche Events: logcheck, ELK
- Ticketsystem
- ...

SaltStack

- Ein Config Management Werkzeug; primär für Linux
- Alternativen: Puppet, Chef, Ansible, CFEngine, Terraform, …
- Zielzustand wird deklarativ beschrieben
  - Softwarepaket x soll installiert sein
  - Die Configdatei y soll genau so aussehen
  - Der Service z soll gestoppt sein
- Abhängigkeiten
  - Falls Configdatei a sich ändert, dann Service b neu starten

Code vs. Config vs. Geheimnisse

- Ein git-Repo für Code, z.B. "salt"
- Ein weiteres git-Repo für Config und Geheimnisse, z.B. "salt-pillar"

Kommunikation

- salt-master: Daemon auf zentraler Maschine
  - Etwas Config nötig
  - Braucht Zugang zu den git-Repos
- salt-minion: Daemon auf jedem Rechner
  - Verbindet sich zum salt-master
  - Etwas Zusatzconfig nötig für z.B. mysql-Zugang

Benutzung

- Code, Config oder Geheimnisse in git-repo anpassen, commiten, pushen
- Auf master: Auf allen Rechnern alles anpassen:
  `sudo salt '*' state.highstate`
- Eigenes Vereinfachungstool:
  `mail-control`
  um E-Mail Nutzer (per salt + git) zu verwalten.
- Web-UI-Interfaces (keine probiert) z.B: Uyuni, Alcali, SaltGUI
  - (Anbindung an Jenkins: Jeder State ist ein Testcase
    Macht aber nur Sinn, wenn man sowieso viel Jenkins einsetzt)

Top.sls

- Startpunkt ist die /top.sls im git-Repo
- Rechner oder Gruppen von Rechner bekommen States zugewiesen

```
'server1':
  - webserver

'*imap*':
  - imapserver

'G@virtual:VirtualPC':
  - match: compound
  - hyperv
```

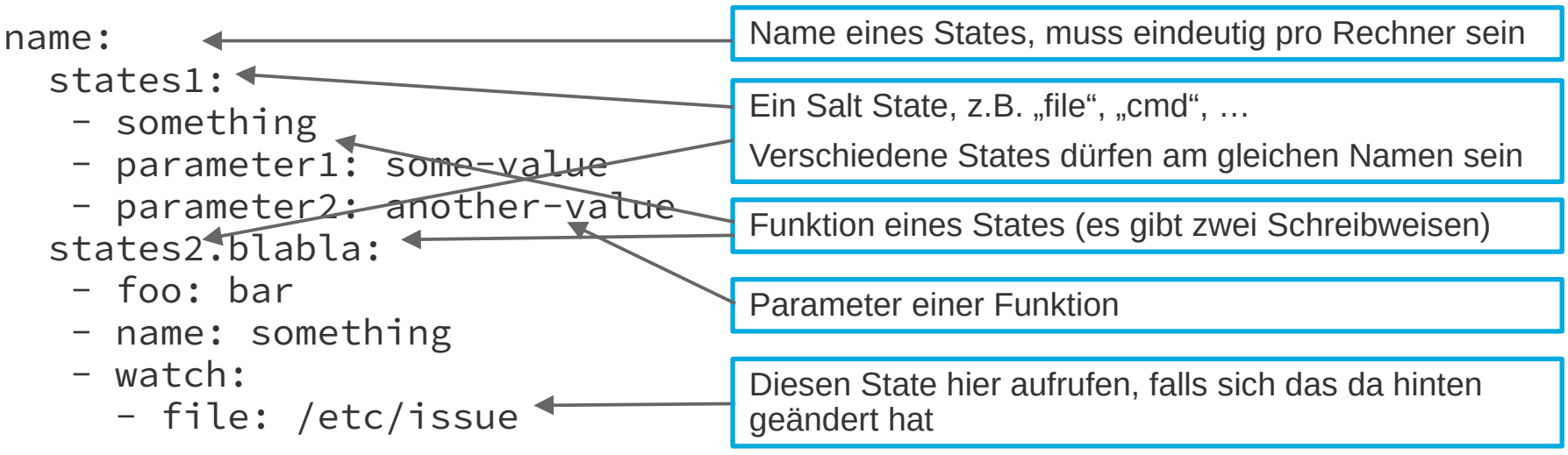
- Ebenso im salt-pillar-Repo

State-Dateien

- State-Dateien haben die Endung .sls
  - Einfacher State „foo“: `foo.sls`
- State mit Extradateien: `foo/init.sls`
- "include" anderer States möglich (benutze ich manchmal)
- Unterzustände möglich (benutze ich selten)

States

- Ein State definiert, wie etwas sein soll
- Struktur:



States

- Siehe https://docs.saltproject.io/en/latest/ref/states/all/index.html
- Ein paar wichtige States
  - file: Dateien platzieren, löschen, …
  - cmd: Einen Befehl (shell) ausführen (normalerweise mit Bedingung)
  - pkg: Pakete installieren, entfernen
  - service: Einen Service aktivieren, deaktivieren, starten, stoppen
  - user, group: Benutzer und Gruppen verwalten
- Seltener:
  - mysql: Mysql Benutzer, Datenbanken, Rechte verwalten
  - pip_state: Python-Pakete (ohne apt) installieren
  - git: Ein git-Repo auschecken

States: file

```
blacklist-floppy:
  file:
    - managed
    - name: /etc/modprobe.d/floppy-blacklist.conf
    - user: root
    - group: root
    - mode: 644
    - contents: |
        blacklist floppy
```

State: cmd

```
update-initramfs:
  cmd:
    - wait
    - name: update-initramfs -u
    - runas: root
    - watch:
      - file: /etc/modprobe.d/floppy-blacklist.conf
```

State: pkg

```
hyperv-daemons:
  pkg:
    - installed
    - pkgs:
      - linux-image-4.9.0-0.bpo.5-amd64
      - hyperv-daemons
      - linux-base
```

State: service

```
hyperv-daemons.hv-kvp-daemon.service:
  service:
    - dead
    - enable: False
```

Grains

- Jeder Rechner hat Eigenschaften
- Per Kommandozeile:

```
$ sudo salt 'amor*' grains.items | less
    ipv4:
        - 127.0.0.1
        - 192.168.1.2
    nodename:
        amor
    num_cpus:
        4
    os:
        Debian
    os_family:
        Debian
    osarch:
        amd64
    virtual:
        VirtualPC
```

Diese Liste ist sehr gekürzt

Grains

- Grains können in States verwendet werden
- Z.B. in „top.sls“

```
'G@productname:*ProLiant* and G@os:Debian’:
   - match: compound
   - hp
```

- In Templates:

```
{% if grains['os'] == 'Debian' and grains['oscodename'] == 'stretch’ %}
…
{% endif %}
```

Templates: jinja

- Die deklarative Sprache von SaltState kann mit Templates (jinja) verändert werden
- Manche States unterstützen Templates (z.B. Dateien platzieren)
- Jinja ist eine python-artige Sprache
- Jinja Code wird innerhalb von speziellen Markern verwendet:
  - `{% … %}`: für Code
  - `{{ x }}`: für Variablen

Templates: Beispiele

- Unterschiede zwischen Linux Version oder Varianten

```
{% if grains['os_family'] == 'RedHat' and grains['osmajorrelease'] == 7 %}
    - ntp
{% elif grains['os_family'] == 'RedHat' and grains['osmajorrelease'] == 8 %}
    - chrony
{% endif %}
```

- Verschiedene Dateien je nach Rechner

```
file:
    - managed
[…]
    - source: salt://test/bla.txt-{{ grains['nodename'] }}
```

Templates: Beispiele

- Schleifen

```
{% for i in pillar['mediawiki']['wikis'] %}
... {{ i }} ...
{% endfor %}
```

- Variablen setzen

```
{% set dist = salt['pillar.get']('node:' + grains.host + ':repo', '') %}
```

Templates: Beispiele

- Macros

```
{% macro apache_enable_mod(name) %}
apache_mod_{{ name }}_enable:
  cmd:
    - run
    - name: a2enmod {{ name }}
    - unless: test -L /etc/apache2/mods-enabled/{{ name }}.conf -o -L
/etc/apache2/mods-enabled/{{ name }}.load
    - watch_in:
      - service: apache2
{%- endmacro %}

{{ apache_enable_mod('ssl') }}
```

Templates: Dateien

- apache2/default-ssl.conf

```
[…]
                ServerName {{ grains['fqdn'] }}
[…]
```

- mail-server/amavis/50-user

```
@local_domains_acl = (
{%- for i in pillar['mail']['domains'] -%}
 ".{{ i }}",
{%- endfor -%}
);
```

Ende

Fragen?