TU Clausthal

Konfiguration von FreeRADIUS & radsecproxy mit Ansible

C. Strauf, C. Rebischke
Rechenzentrum TU Clausthal

Agenda

- Vorstellung
- Warum Konfigurationsmanagement?
- Ansible
- Notwendige Konfigurationsarbeiten
- Umsetzung in Ansible
- Demo
- Fazit

Vorstellung

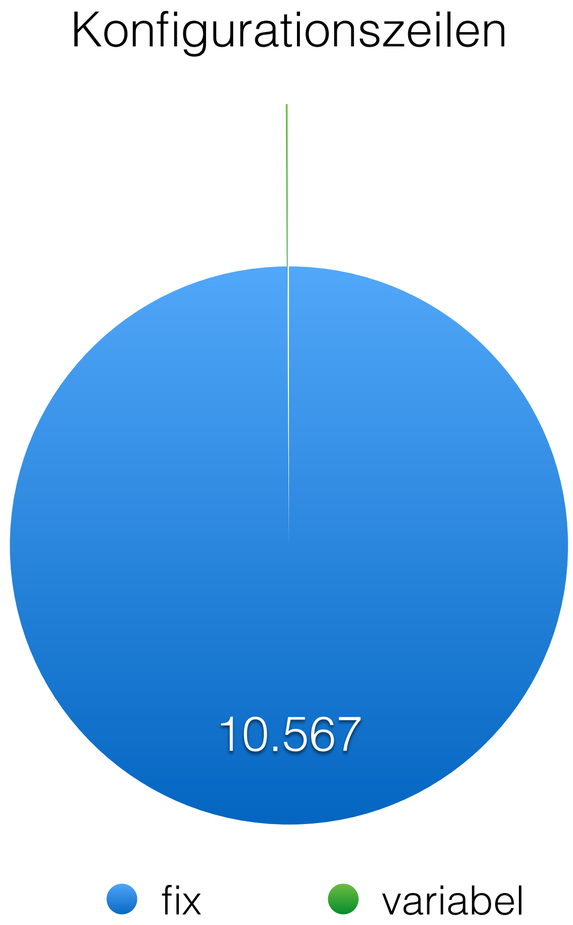
- Christian Strauf:
  - Mathematikstudium in Münster,
  - JOIN-Team des DFN im 6net-Projekt bis 2004,
  - seit 2004 Leiter der Netzabteilung des RZ TU Clausthal,
  - FreeRADIUS-Administration seit 2004.
- Christian Rebischke:
  - Informatikstudent an der TU Clausthal,
  - Hiwi am RZ seit 2016,
  - Trusted User Arch Linux,
  - Security Team Arch Linux.

Warum Konfigurationsmanagement?

- Probleme der klassischen IT:
  - Die Komplexität von Konfigurationen ist hoch und nimmt tendenziell zu.
  - **Früher:** Vertikale Skalierung = bei zu wenig Leistung wird ein Server größer dimensioniert.
  - **Heute:** Horizontale Skalierung = bei zu wenig Leistung werden zu einem Cluster mehr Server hinzugefügt.
  - Bei horizontaler Skalierung und komplexen Konfigurationen gibt es eine große Anzahl replizierter Konfiguration. Es besteht großes Potenzial von **Konfigurationsfehlern** bei **Handarbeit**.
- Lösung: Verwendung eines **systematischen, automatischen Konfigurationsmanagements**.

Warum Konfigurationsmanagement?

- TU Clausthal arbeitet an der Etablierung einer **„Single Source of Truth“ (SSOT)** für Konfigurationen und Informationen in der IT.
- Motto für Server: „Es zählt nicht, was auf dem Server **konfiguriert ist**, sondern, was laut SSOT **konfiguriert werden soll**!“
- Konfiguration von FreeRADIUS ist hochkomplex, deswegen:
  - Saubere **Revisionskontrolle** wichtig,
  - redundante RADIUS-Server müssen **konsistent** konfiguriert sein.
- FreeRADIUS-Konfiguration an der TU Clausthal:
  - **10.567 Zeilen** insgesamt (ohne Kommentare!).
  - Davon von RADIUS- zu RADIUS-Server veränderlich: **11**, das sind **0,1%**!



Warum Konfigurationsmanagement?

- FreeRADIUS-Konfiguration mit > 10 tsd. Zeilen typisches Beispiel für komplexe Konfiguration mit **Fehleranfälligkeit** bei Übertragung von Konfigurationen auf neue Server.
- Bei FreeRADIUS ist besonders wichtig, dass Änderungen an Konfiguration **konsistent** auf allen redundanten FreeRADIUS-Server sind.
- Konfiguration soll auch für Admins änderbar sein, die **keine FreeRADIUS-Experten** sind.
- Schon für FreeRADIUS und radsecproxy lohnt sich die Verwendung von Konfigurationsmanagement.

Ansible

ANSIBLE

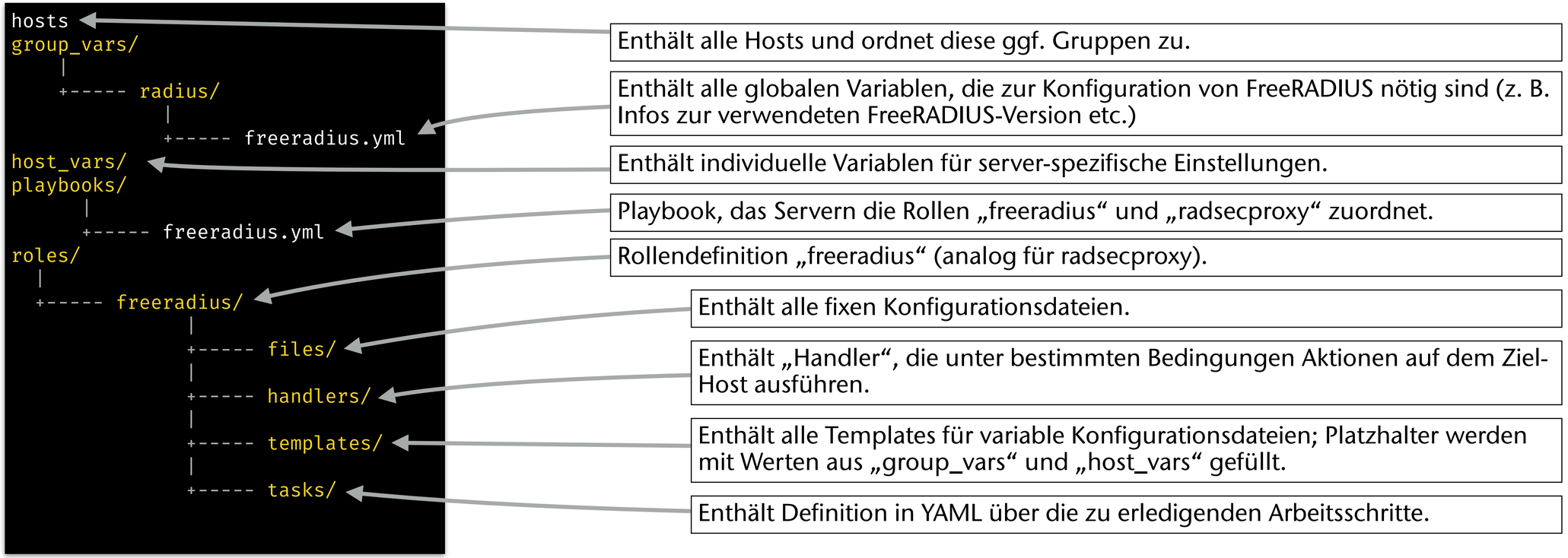
- Das RZ der TU Clausthal setzt für das Konfigurationsmanagement **Ansible** ein.
- Ansible ist ein **Open Source Orchestrierungs-Tool**.
- Es nutzt i. d. R. **SSH** für die Verbindung zu Hosts und ggf. lokal auf den Hosts **Python**.
- Ansible arbeitet **agentless**, d. h. es läuft nur bei Aufruf. I. d. R. gestaltet sich ein Aufruf wie folgt:
  - Check der Konfiguration des Hosts an Hand von Infos, die vom Host geholt werden.
  - Konfiguration von Teilen, die nicht wie in einer Rolle beschrieben konfiguriert sind.
  - Ansible arbeitet **idempotent**.
- Die Konfigurationen werden in „**Roles**“ beschrieben, die in „**Playbooks**“ Hosts zugeordnet werden.

Ansible

ANSIBLE

- Die Konfigurationen werden in Dateien im **YAML-Format** beschrieben.
- Templates für Konfigurationsdateien werden in **Jinja2** geschrieben.
- Bei Bedarf können **Custom-Filter in Python** geschrieben werden.
- RZ der TU Clausthal ist derzeit in der Phase, Ansible für diverse Aspekte einzuführen:
  - Mail-Server-Konfiguration,
  - DHCP(v6)-Server-Konfiguration,
  - Netzwerk-Switch-Konfiguration,
  - zukünftig auch Firewall-Konfiguration (Rules und Load-Balancing-Agents),
  - etc..

Umsetzung in Ansible



Struktur der Ansible-Konfiguration für FreeRADIUS

Ansible

- Hosts werden in der Datei „**hosts**“ aufgelistet und können unter **Gruppen** „[...]“ zusammengefasst werden.
- In einem Playbook werden Hosts oder Gruppen Rollen zugeordnet.

```
[radius]
radius-server1.mein.test
radius-server2.mein.test
radius-server3.mein.test

[dhcps]
dhcp1.mein.test
dhcp2.mein.test
```

```
---
- name:        Configure FreeRADIUS
  hosts:       radius
  remote_user: root
  roles:
    - { role: logrotate, tags: ['logrotate'] }
    - { role: freeradius, tags: ['freeradius'] }
    - { role: radsecproxy, tags: ['radsecproxy'] }
    - { role: sgagent, tags: ['sgagent'] }
```

Ansible

- Für jede Rolle gibt es „**tasks**“, die auszuführen sind.
- Tasks können auf Templates zurückgreifen.

```
---

- name: include install
  include_tasks: install.yml
  when: freeradius.install

- name:          Create directories
  file:
    path:        "{{ freeradius.conf_dir }}/{{ item.path }}"
    state:       directory
    mode:        "0755"
  with_filetree: files/freeradius/
  when:          item.state == "directory"
  notify:
    - freeradius-handler
- name:          Create /var/run/freeradius
  file:
    path:        /var/run/freeradius
    state:       directory
    owner:       "freerad"
    group:       "freerad"
    mode:        "0755"
  notify:
    - freeradius-handler
- name:          Copy fix templates
  copy:
    src:         "{{ item.src }}"
    dest:        "{{ freeradius.conf_dir }}/{{ item.path }}"
    mode:        "0644"
  with_filetree: files/freeradius/
  when:          item.state == "file"
  notify:
    - freeradius-handler
- name:          Create symlinks
  file:
    src:         "{{ item.src }}"
    dest:        "{{ freeradius.conf_dir }}/{{ item.path }}"
    state:       link
    force:       yes
  with_filetree: files/freeradius/
  when:          item.state == "link"
  notify:
    - freeradius-handler
- name:          Copy templates
  template:      src="{{ item }}" dest="{{ freeradius.conf_dir }}/{{ (item | splitext)[0] }}"
  with_items:
     - mods-available/eap.j2
     - mods-available/sql.j2
     - mods-config/perl/trafficcheck.pl.j2
     - clients.conf.j2
     - proxy.conf.j2
```

Ansible

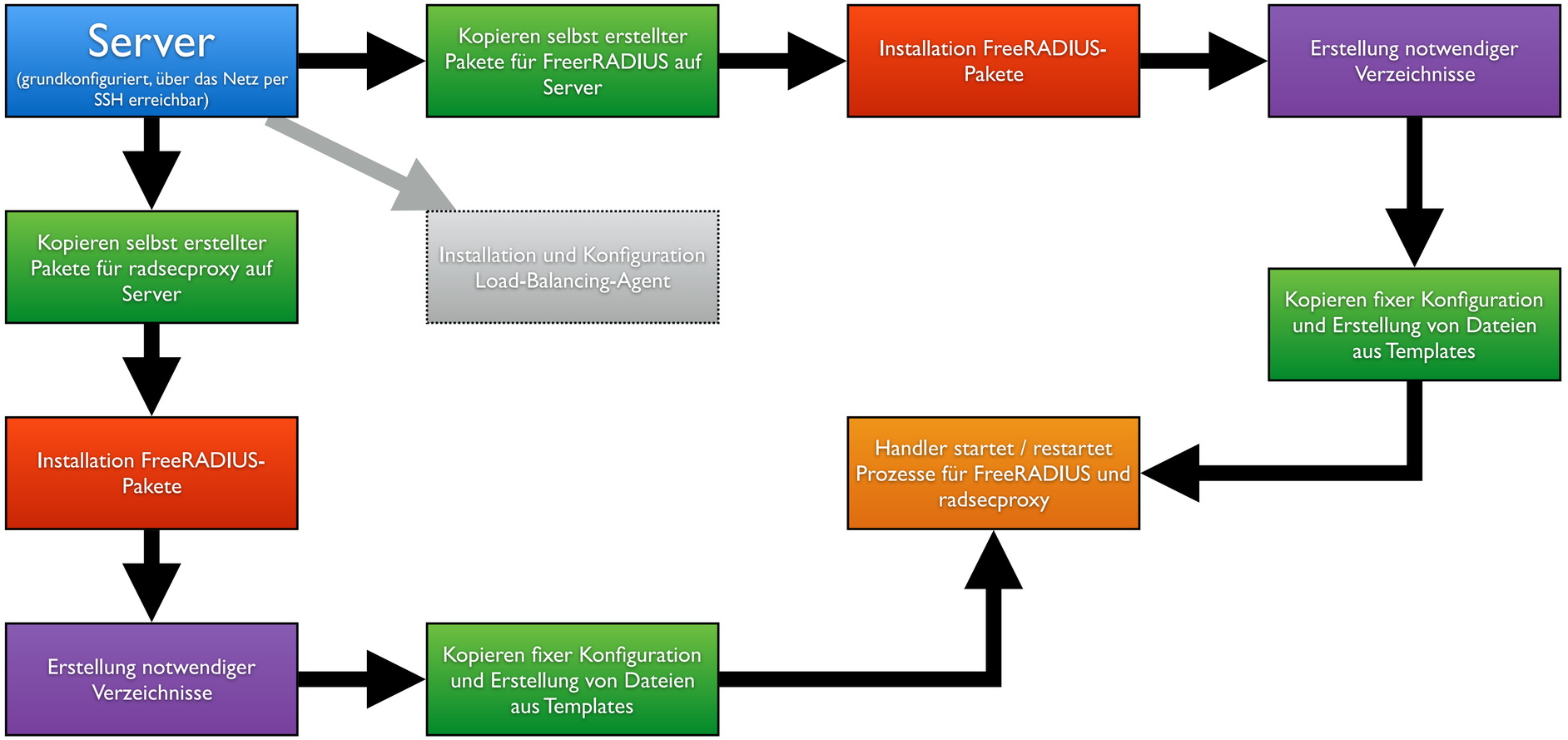
- **Templates** beschreiben, an welchen Stellen **Platzhalter** durch in YAML festgelegte Daten ersetzt werden.

```
---
freeradius_individual:
  global_tls_cert:          "radius-public-key.pem"
  global_tls_key:           "radius-private-key.pem"
  old_individual_tls_cert: "alt-radius-cert.pem"
  old_individual_tls_key:   „alt-radius-key.pem"
  radsecproxy_backup:
    name:                   "radius-server2.mein.test"
    ip:                     "X.X.X.X"
radsecproxy:
  listenTLS:                "X.X.X.X"
  peer:                     "X.X.X.X"
  version:                  "1.7.2-1"
```

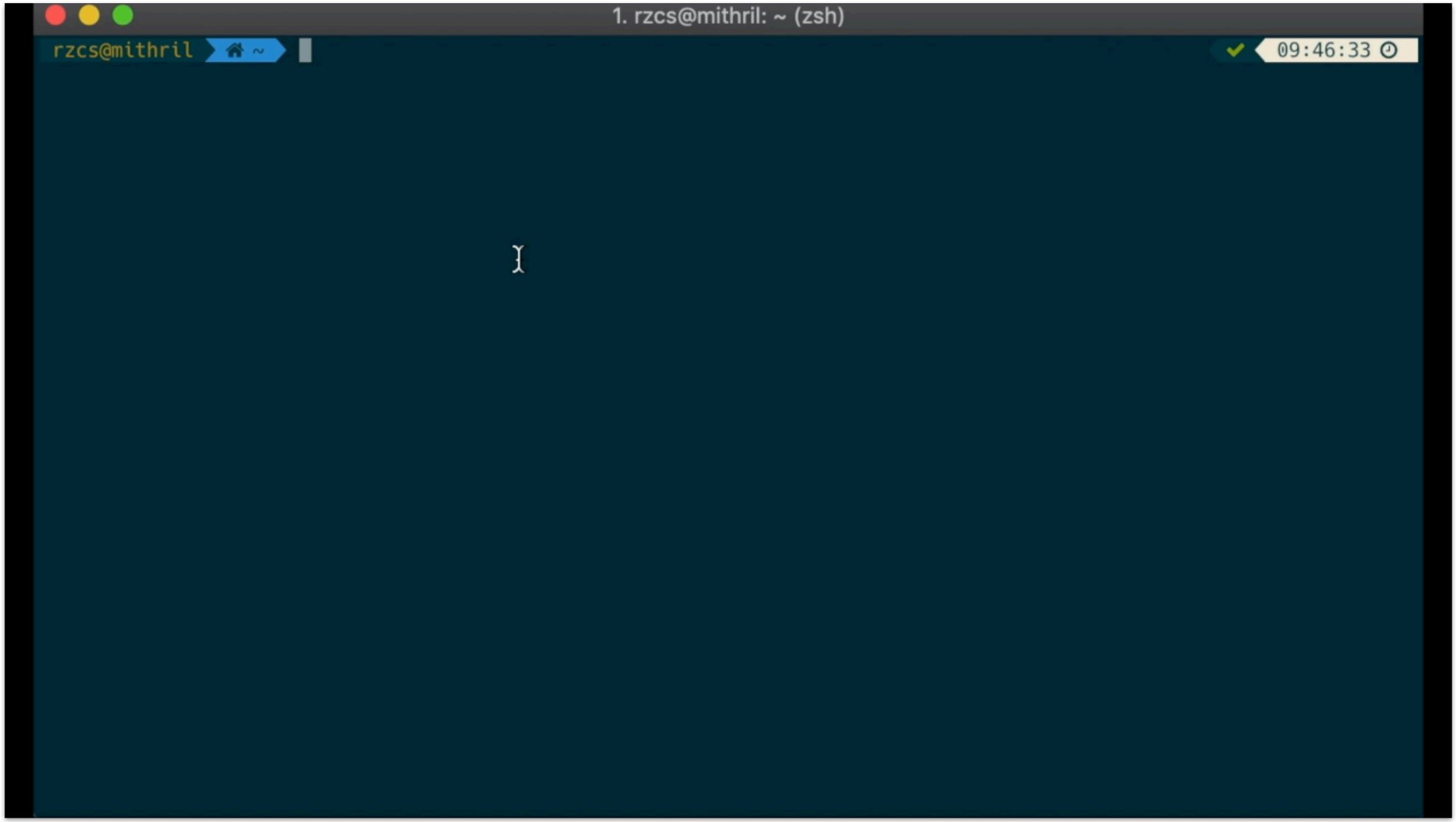
```
home_server radsecproxy-backup {
      type = auth+acct
      ipaddr = {{ freeradius_individual.radsecproxy_backup.ip }}
      port = 2084
      secret = XXXXXXXXX
        response_window = 60
        response_timeouts = 5
        zombie_period = 10
      status_check = status-server
}
```

Dipl.-Math. Christian Strauf, Christian Rebischke
Rechenzentrum TU Clausthal

Notwendige Konfigurationsarbeiten



Demo


1. rzcs@mithril: ~ (zsh)
rzcs@mithril ~
09:46:33

Anekdote

- Beim Umzug der FreeRADIUS-Server auf neues Ubuntu LTS:
  - FreeRADIUS lief sofort (Debian-Pakete wurde lediglich auf der neuen Plattform kompiliert).
  - **Aber:** Telefon-Authentisierung per EAP-TLS hörte auf zu funktionieren!
- Was war passiert?
  - Auf Ubuntu 16 LTS: FreeRADIUS linkt gegen OpenSSL 1.0.x.
  - Auf Ubuntu 18 LTS: FreeRADIUS linkt gegen OpenSSL 1.1.x.

```
Cipher Suite: TLS_RSA_WITH_SEED_CBC_SHA (0x0096)
Cipher Suite: TLS_RSA_WITH_IDEA_CBC_SHA (0x0007)
Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
Cipher Suite: TLS_ECDH_anon_WITH_RC4_128_SHA (0xc016)
Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0008)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x0006)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
Cipher Suite: TLS_EMPTY_RENEGOTIATION_INFO_SCSV (0x00ff)
```

Huch?! TLS 1.0 mit „Super-Ciphers“

Anekdote

- OpenSSL 1.1.x bietet die schlechten Ciphers per Default nicht mehr an. (Gut so!)
- Ehrenrettung des Telefonherstellers: Auch TLS 1.2 mit sehr guten Ciphers kann aktiviert werden (bei alten Telefonen nicht Default).
- Vorgehen: FreeRADIUS in Übergangszeit passenden Cipher-String für Telefone (nur für Telefone!) vorgeben, der maximal einen der unsicheren Ciphers und alle sicheren Ciphers unterstützt.

Anekdote

- Dank Ansible ist die Konfigurationsänderung unproblematisch:
  - Branch in git,
  - Zeile im Repo ändern,
  - auf Test-Server testen,
  - in Produktiv-Branch mergen,
  - Ansible-Playbook ablaufen lassen.
- Das **Debugging** des Problems hat **3,5 Stunden** gedauert. Die **Installation** von drei frischen Servern inkl. **Testprogramm** und **Migration** nur **1 Stunde**. :)

```
[…]
cipher_list = „DEFAULT“
[…]
```

```
[…]
cipher_list = „ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:\
               ECDHE-ECDSA-AES128-SHA:ECDHE-ECDSA-AES256-SHA:\
               ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA384:\
               ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:\
               ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES256-SHA:\
               ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA384:\
               DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:\
               DHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA:\
               DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA256:SEED-SHA“
[…]
```

eap.conf in Ansible-Repo

Fazit

- Ansible erleichtert die Arbeit mit Servern ungemein.
- Bei Migrationsarbeiten oder Desaster-Recovery ist die Komplettinstallation einer funktionierenden FreeRADIUS-/radsecproxy-Infrastruktur auf grundinstallierten Servern in unter 1h möglich.
- Die Nutzung eines git-Repos für Ansible hilft bei der Revisionskontrolle und ggf. beim Vier-Augen-Prinzip.
- Änderungen von Konfigurationen in YAML-Dateien ist auch für Nicht-FreeRADIUS-Experten möglich.

Vielen Dank für Ihre Aufmerksamkeit!