

```
PLAY [ansible-vortrag] *****

TASK [Gathering Facts] *****
ok: [ansible-vortrag]

TASK [roles/motd : Titel] *****
ok: [ansible-vortrag] => {
    "msg": "Linux Servermanagement mit Ansible"
}
```

LINUX SERVERMANAGEMENT MIT ANSIBLE

VOID – WARPZONE E.V.

ABOUT ME

- Linux Einstieg etwa 1997
- Administriert die Server der warpzone und eigene Infrastruktur
- Beteiligt am Aufbau der Freifunk Münsterland Infrastruktur

WARUM DAS GANZE ?

„Ich kann doch auch einfach die Konfigurationsdaten irgendwo sichern“

„Bei Linux liegt doch alles in etc“

„etckeeper reicht doch ..“

WARUM DAS GANZE ?

- Dokumentation
 - Servermanagement ist eine ausführbare Dokumentation
 - Idempotenz als wichtige Eigenschaft
- Tracking von Änderungen (mit git)
- Wiederverwendbarkeit
- Reproduzierbarkeit

DIE OBLIGATORISCHE MARKTÜBERSICHT

Konfiguration

- Ansible
- Salt
- Puppet
- Chef

Infrastruktur

- Terraform
- Pulumi

TOOLS: ANSIBLE

- Veröffentlicht 2012, implementiert in Python
- Konfiguration in YAML
 - Imperativ / Deklarativ
- Kein Agent
 - Lediglich SSH-Verbindung und Python auf dem Server
- Sprache: Inventar, Rollen, Tasks, Playbooks



ANSIBLE

TOOLS: SALT

- Veröffentlicht 2011, Implementiert in Python
- Konfiguration in SLS (SaLt State) – YAML Format
 - Deklarativ
- Client-Server Modell
- Sprache: Master, Minions, Grains, Pillars

'SALTSTACK®

TOOLS: PUPPET



- Veröffentlichung 2005, Implementiert in Ruby/C++/Clojure
- Konfiguration in JSON
 - Deklarativ
- Zentraler Server + lokale Agents
 - Kontinuierliche Prüfung der Konfiguration
- Sprache: Ressourcen, Manifeste

TOOLS: CHEF

- Veröffentlicht 2009, implementiert in Ruby/Erlang
- Konfiguration in Ruby-DSL
 - Deklarativ / Imperativ
- Client-Server Modell oder lokale Ausführung
- Sprache: Cookbooks, Receipes



WARUM ANSIBLE ?

- Einfacher Einstieg
 - SSH Zugriff und Python reichen aus
 - Kein Agent
 - Kein zentraler Server
- Gute Kontrolle über Änderungen
 - Check -Läufe mit Ausgabe von diffs
 - Ausführung einzelner Teile der Konfiguration über Tags
- Es ist nicht zwingend ein Zugriff auf alle Server erforderlich

INSTALLATION

- Lokaler Arbeitsplatz „Control Node“ muss Linux sein
- `pip install ansible`

```
fish @ WSL-Ubuntu
[root@DE-ADN-H42T2D3:~]
-> pip install ansible
Requirement already satisfied: ansible in /usr/local/lib/python3.10/dist-packages (7.2.0)
Requirement already satisfied: ansible-core=2.14.2 in /usr/local/lib/python3.10/dist-packages (from ansible) (2.14.2)
Requirement already satisfied: jinja2>=3.0.0 in /usr/lib/python3/dist-packages (from ansible-core=2.14.2->ansible) (3.0.3)
Requirement already satisfied: cryptography in /usr/lib/python3/dist-packages (from ansible-core=2.14.2->ansible) (3.4.8)
Requirement already satisfied: resolvelib<0.9.0, >=0.5.3 in /usr/local/lib/python3.10/dist-packages (from ansible-core=2.14.2->ansible) (0.8.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from ansible-core=2.14.2->ansible) (23.0)
Requirement already satisfied: PyYAML>=5.1 in /usr/lib/python3/dist-packages (from ansible-core=2.14.2->ansible) (5.4.1)
```

INVENTORY

- Das Inventory definiert mit welchen Servern wir arbeiten wollen
- Syntax: INI oder YAML
 - Server
 - Gruppen
 - Variablen
- Dynamisches Inventory: AWS, Azure, Netbox, etc...

INVENTORY

Auflösbare Hostnamen

```
hosts

[test]
ansible-vortrag-vorbereitung.example.org

[prod]
ansible-vortrag.example.org
```

Explizite Angabe der IP

```
hosts

[test]
ansible-vortrag-vorbereitung ansible_host=167.235.52.81

[prod]
ansible-vortrag ansible_host=78.47.140.146
```

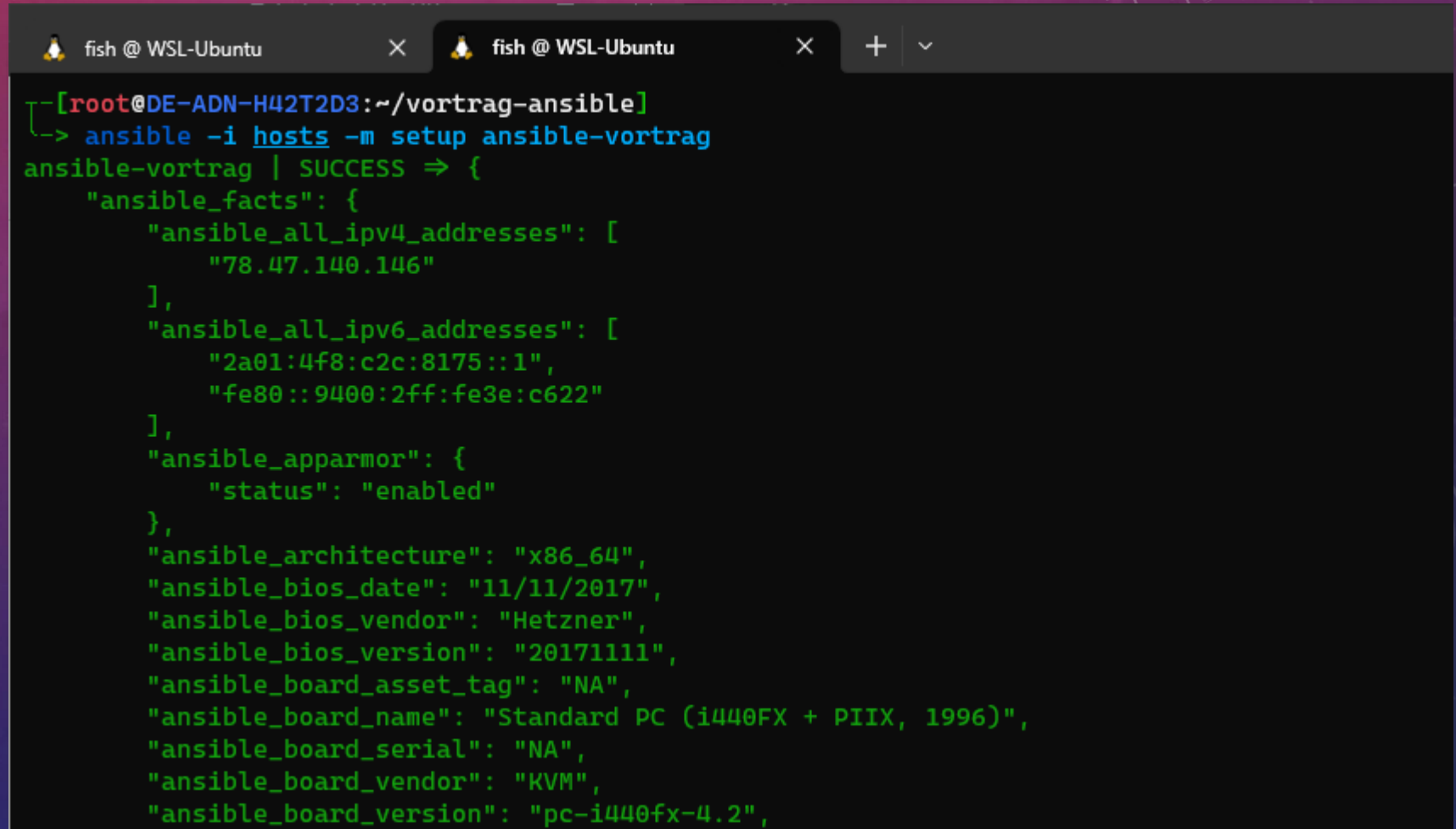
VERBINDUNGSTEST

- Mit dem Kommando „ansible“ können einzelne Module ausgeführt werden
- Ergebnisausgabe als JSON
- 1. Parameter: Inventory-Datei
- 2. Parameter: Modul
- 3. Parameter: Hostname oder Gruppe

```
fish @ WSL-Ubuntu  fish @ WSL-Ubuntu  +  v
[roo@DE-ADN-H42T2D3:~/vortrag-ansible]
-> ansible -i hosts -m ping all
ansible-vortrag | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
ansible-vortrag-vorbereitung | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

ZUSTAND DES SERVERS

- Modul: Setup

A terminal window with two tabs, both labeled 'fish @ WSL-Ubuntu'. The active tab shows a command prompt where the user has run 'ansible -i hosts -m setup ansible-vortrag'. The output shows a successful execution of the 'ansible-vortrag' module, displaying a JSON object with system facts. The facts include IPv4 and IPv6 addresses, AppArmor status, architecture, BIOS date and vendor, and board information.

```
fish @ WSL-Ubuntu X fish @ WSL-Ubuntu X + v
[ root@DE-ADN-H42T2D3:~/vortrag-ansible ]
-> ansible -i hosts -m setup ansible-vortrag
ansible-vortrag | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "78.47.140.146"
    ],
    "ansible_all_ipv6_addresses": [
      "2a01:4f8:c2c:8175::1",
      "fe80::9400:2ff:fe3e:c622"
    ],
    "ansible_apparmor": {
      "status": "enabled"
    },
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "11/11/2017",
    "ansible_bios_vendor": "Hetzner",
    "ansible_bios_version": "20171111",
    "ansible_board_asset_tag": "NA",
    "ansible_board_name": "Standard PC (i440FX + PIIX, 1996)",
    "ansible_board_serial": "NA",
    "ansible_board_vendor": "KVM",
    "ansible_board_version": "pc-i440fx-4.2",
```

PLAYBOOKS

- Ein Playbook verbindet Server und Rollen
 - Einzelne Server
 - Gruppen
 - „all“ für alle Server
- Ausführung in der Reihenfolge in der Datei
- Optional: Tags
 - Ausführung einzelner Teile / Rollen

! site.yml X

! site.yml

1 ---

2

3 - hosts: ansible-vortrag

4 | roles:

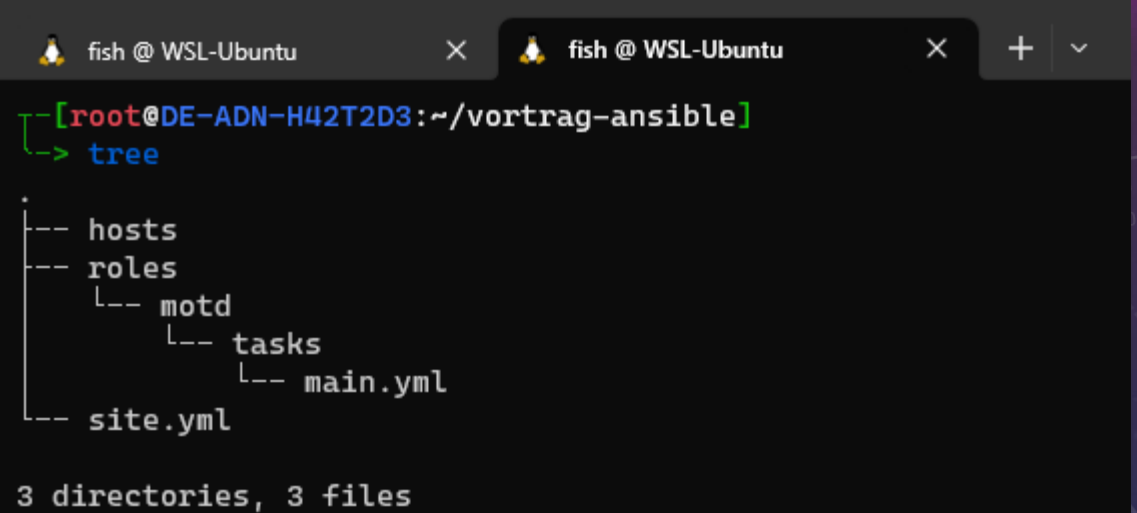
5 | | - { role: roles/motd, tags: motd }

6

7

ROLLEN

- Rollen enthalten die eigentliche Konfiguration
- Eine Rolle in immer ein gleichnamiges Unterverzeichnis
- Feste Verzeichnisstruktur innerhalb der Rolle
 - Einstiegspunkt: tasks/main.yml
- Best-Practice:
 - Unterverzeichnis „roles“ für Rollen
 - Alternativ mehrere Unterverzeichnisse für Gruppen von Rollen

A terminal window titled 'fish @ WSL-Ubuntu' showing the output of the 'tree' command in the directory ~/vortrag-ansible. The output shows a directory structure with 'hosts', 'roles', and 'site.yml'. The 'roles' directory contains 'motd', which in turn contains 'tasks', which contains 'main.yml'. The terminal also reports '3 directories, 3 files'.

```
[root@DE-ADN-H42T2D3:~/vortrag-ansible]  
tree  
  
.-- hosts  
.-- roles  
    |-- motd  
        |-- tasks  
            |-- main.yml  
.-- site.yml  
  
3 directories, 3 files
```

TASKS

- Innerhalb einer Rolle werden Tasks ausgeführt
- Eine Rolle in immer ein gleichnamiges Unterverzeichnis
- Feste Verzeichnisstruktur innerhalb der Rolle
 - Einstiegspunkt: tasks/main.yml

```
! main.yml X
roles > motd > tasks > ! main.yml
1  ---
2
3  - name: Hello World Task
4    ansible.builtin.debug:
5      msg: Hello World für Server {{ inventory_hostname }}
6
```

AUSFÜHREN VON PLAYBOOKS

- Ausführung des Playbooks mit dem Kommando „ansible-playbook“
- Inventory und Playbook-Datei müssen immer angegeben werden
- Eingrenzung der Ausführung:
 - Server
 - Gruppen
 - Tags

```
fish @ WSL-Ubuntu x fish @ WSL-Ubuntu x + v
[root@DE-ADN-H42T2D3:~/vortrag-ansible]
-> ansible-playbook -i hosts site.yml

PLAY [ansible-vortrag] *****

TASK [Gathering Facts] *****
ok: [ansible-vortrag]

TASK [roles/motd : Hello World Task] *****
ok: [ansible-vortrag] => {
  "msg": "Hello World für Server ansible-vortrag"
}

PLAY RECAP *****
ansible-vortrag : ok=2 changed=0 unreachable=0 failed=0 skipped=0
```

TEMPLATES

- Schreiben von Konfigurationsdateien auf den Server
- Ansible verwendet hierbei Jinja2 Templates
- Verzeichnis „templates“ in der Rolle

```
! main.yml  ! motd  X
roles > motd > templates > ! motd
1
2  Willkommen auf dem Server {{ inventory_hostname }} !!
3
```

```
! main.yml  X  ! motd
roles > motd > tasks > ! main.yml
1  ---
2
3  - name: Hello World Task
4    ansible.builtin.debug:
5      msg: Hello World für Server {{ inventory_hostname }}
6
7  - name: Motd Datei erstellen
8    ansible.builtin.template:
9      src: motd
10     dest: /etc/motd
11
```


TESTLAUF UND ANZEIGE VON ÄNDERUNGEN

- Der Parameter `--check` ermöglicht es die Änderungen zu prüfen
- Der Parameter `--diff` zeigt Änderungen an Dateien an

```
fish @ WSL-Ubuntu x fish @ WSL-Ubuntu x + v
[root@DE-ADN-H42T2D3:~/vortrag-ansible]
> ansible-playbook -i hosts site.yml --check --diff

TASK [roles/motd : Motd Datei erstellen] *****
--- before: /etc/motd
+++ after: /root/.ansible/tmp/ansible-local-133351h1kb9u/tmpfz7gvf4u/motd
@@ -1,7 +1,2 @@

-The programs included with the Debian GNU/Linux system are free software;
-the exact distribution terms for each program are described in the
-individual files in /usr/share/doc/*/copyright.
-
-Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
-permitted by applicable law.
+Willkommen auf dem Server ansible-vortrag-vorbereitung !!

changed: [ansible-vortrag-vorbereitung]
```

TEMPLATES - VARIABLEN

- Variablen aus dem Setup-Modul sind „einfach da“
- Ausgabe im Template mit {{ }}

! motd ×

roles > motd > templates > ! motd

1

2 Willkommen auf dem Server {{ inventory_hostname }} !!

3 Dieser Server hat {{ ansible_processor_count }} CPUs

4

5

TEMPLATES – SCHLEIFEN UND BEDINGUNGEN

```
fish @ WSL-Ubuntu
[root@DE-ADN-H42T2D3:~/vortrag-ansible]
-> ansible -i hosts -m setup ansible-vortrag
ansible-vortrag | SUCCESS => {
  "ansible_facts": {
    "ansible_all_ipv4_addresses": [
      "78.47.140.146"
    ],
    "ansible_all_ipv6_addresses": [
      "2a01:4f8:c2c:8175::1",
      "fe80::9400:2ff:fe3e:c622"
    ],
  },
}
```

```
! motd
roles > motd > templates > ! motd
1
2   Willkommen auf dem Server {{ inventory_hostname }} !!
3   Dieser Server hat {{ ansible_processor_count }} CPUs
4
5   {% if ansible_virtualization_role == 'guest' %}
6   |   Dieser Server ist virtualisiert
7   {% endif %}
8
9   {% for ip in ansible_all_ipv4_addresses %}
10  |   IPv4: {{ ip }}
11  {% endfor %}
12
13  {% for ip in ansible_all_ipv6_addresses %}
14  |   IPv6: {{ ip }}
15  {% endfor %}
16
```

VARIABLEN DEFINIEREN

- Eigene Variablen können an mehreren Ebenen definiert werden definiert werden
- Variablen einer „höheren“ Ebene überschreiben bereits definierte Variablen
- 1 - Standardwerte innerhalb der Rollen
- 2 - Inventory Datei
- 3 - group_vars Dateien
- 4 - host_vars Dateien
- 5 - Definition in site.yml
- 6 (nur hinzufügen) - Task set_facts innerhalb der Rolle

VARIABLEN DEFINIEREN

- group_vars/all
- group_vars/<Gruppenname>
- host_vars/<Servername>
- YAML Datei
- Alles, was hier definiert wird, ist als Variable verfügbar

VARIABLEN DEFINIEREN

```

fish @ WSL-Ubuntu
fish @ WSL-Ubuntu
[root@DE-ADN-H42T2D3:~/vortrag-ansible]
tree
.
|-- group_vars
|   |-- all
|   |-- prod
|   |-- test
|-- host_vars
|   |-- ansible-vortrag
|   |-- ansible-vortrag-vorbereitung
|-- hosts
|-- roles
|   |-- motd
|       |-- tasks
|           |-- main.yml
|       |-- templates
|           |-- motd
|-- site.yml

```

```
group_vars > all
1
2  motd_zusatztext: Das ist ein Zusatztext für alle Server
3
```

```
host_vars > ansible-vortrag
1
2  motd_zusatztext: Das ist ein Zusatztext speziell für den Server
3
```

VARIABLEN DEFINIEREN

! site.yml ×

! site.yml

```
1  ---
2
3  - hosts: ansible-vortrag-vorbereitung
4    roles:
5      - {
6          role: roles/motd,
7          tags: motd,
8          motd_zusatztext: Der Text wurde beim Anwenden der Rolle überschrieben
9        }
10
```

VARIABLEN DEFINIEREN

! main.yml ×

roles > motd > tasks > ! main.yml

```
1  ---
2
3  - name: Hello World Task
4    ansible.builtin.debug:
5      msg: Hello World für Server {{ inventory_hostname }}
6
7  - name: Variablen hinzufügen
8    ansible.builtin.set_fact:
9      motd_zusatztext2: Der Text wurde in der Rolle gesetzt
10
11 - name: Motd Datei erstellen
12   ansible.builtin.template:
13     src: motd
14     dest: /etc/motd
15
```


WIEDERHOLUNGEN FÜR TASKS

- Ein Task kann mit eine Liste von Optionen mehrfach ausgeführt werden
- `{{ item }}` als Schleifenvariable
- Achtung: Anführungszeichen verwenden
- Anmerkung: Einige Tasks haben hierfür eigene Funktionen, z.B. `apt`

```
! main.yml X
roles > motd > tasks > ! main.yml
11  - name: Motd Datei erstellen
12    ansible.builtin.template:
13      src: "{{ item }}"
14      dest: "/etc/{{ item }}"
15    with_items:
16      - motd
17
```

BEDINGUNGEN FÜR TASKS

- Die Ausführung eines Tasks kann von einer Bedingung abhängig gemacht werden

```
! main.yml X
roles > motd > tasks > ! main.yml
11 - name: Motd Datei erstellen
12   ansible.builtin.template:
13     src: "{{ item }}"
14     dest: "/etc/{{ item }}"
15   with_items:
16     - motd
17   when: ansible_os_family == "Debian"
18
```

HANDLER

- Handler werden angestoßen, wenn durch Tasks Änderungen vorgenommen wurden
- Handler werden immer am Ende der Rolle ausgeführt
- Handler werden immer nur einmal ausgeführt
- Definition in der Rolle in handlers/main.yml

```
! main.yml ×
roles > motd > tasks > ! main.yml
11 - name: Motd Datei erstellen
12   ansible.builtin.template:
13     src: "{{ item }}"
14     dest: "/etc/{{ item }}"
15   with_items:
16     - motd
17   when: ansible_os_family == "Debian"
18   notify: restart_sshd
19
```

```
! main.yml ×
roles > motd > handlers > ! main.yml
1 ---
2
3 - name: restart_sshd
4   service:
5     name: ssh
6     state: restarted
7
```

TYPISCHE TASKS

- Verzeichnisse erstellen: `ansible.builtin.file`
- Dateien kopieren / bearbeiten: `ansible.builtin.file`, `ansible.builtin.lineinfile`
- Pakete Installieren (apt, yum, etc.): `ansible.builtin.package`, `ansible.builtin.apt`, (dnf, yum)
- Dienste starten / stoppen / konfigurieren: `ansible.builtin.service`, `ansible.builtin.systemd`
- Cron-Dienste einrichten: `ansible.builtin.cron`
- Shell-Commands ausführen: `ansible.builtin.shell`
- Docker Container starten / stoppen / einrichten: `community.docker.docker_compose`

PASSWÖRTER UND GEHEIMNISSE

- Wer hat eigentlich Zugriff auf meine Konfiguration ?
- Liegt die Konfiguration in einem öffentlichen Git Repository ?
- Ansible-Vault als Standard-Verfahren
 - Speichert einzelne Variablen in einer verschlüsselten Datei
 - Entsperren beim Ausführen erforderlich
- Plugins, z.B. für Keepass

DAS SETUP DER WARPZONE

- Ein externer Server
 - tiffany, Webserver, Verwaltung
- Ein interner Server
 - weatherwax, ogg, carrort
- Ein Remote-Server für Veranstaltungen
 - hex, hix

DAS SETUP DER WARPZONE

- Öffentliches Git-Repository: <https://gitlab.warpzone.ms/infrastruktur/ansible-warpzone>
- Rollen in den Verzeichnissen:
 - all
 - common
 - intern
 - remote
 - verwaltung
 - webserver

▼ ANSIBLE-WARPZONE [WSL: UBUNTU]

- > all
- > common
- > functions
- > group_vars
- > host_vars
- > intern
- > keyfiles
- > remote
- > test
- > verwaltung
- > webserver

DAS SETUP DER WARPZONE

- Semi-einheitliche Variablen für Rollen
 - servicename
 - basedir
 - domain
 - Freigegebene Ports

```
! site.yml x
! site.yml
240 - {
241     role: webserver/docker_warpapi, tags: warpapi,
242     servicename: "warpapi",
243     basedir: /srv/warpapi,
244     domain: "api.warpzone.ms"
245 }
246 - {
247     role: webserver/docker_wordpress, tags: wordpress,
248     servicename: "wordpress",
249     basedir: /srv/wordpress,
250     domain: "www.warpzone.ms"
251 }
```


DAS SETUP DER WARPZONE

- Services als Docker
- docker-compose.yml
- Start über Docker Modul

```
! main.yml x
webserver > docker_vpnserver > tasks > ! main.yml
3 - include_tasks: ../functions/get_secret.yml
4   with_items:
5     - { path: "{{ basedir }}/wg_admin_pass", length: 32 }
6     - { path: "{{ basedir }}/wg_private_key", length: -1 } # 'wg genkey'
7
8
9 - name: create folder struct for keycloak
10  file:
11    path: "{{ item }}"
12    state: "directory"
13  with_items:
14    - "{{ basedir }}"
15    - "{{ basedir }}/data"
16
17
18 - name: "copy {{ servicename }} config files"
19  template:
20    src: "{{ item }}"
21    dest: "{{ basedir }}/{{ item }}"
22  with_items:
23    - docker-compose.yml
24  register: config
25
26
27 - name: "stop {{ servicename }} docker"
28  docker_compose:
29    project_src: "{{ basedir }}"
30    state: absent
31  when: config.changed
32
33
34 - name: "start {{ servicename }} docker"
35
```

DAS SETUP DER WARPZONE

- Alternativer Ansatz für Passwörter / Geheimnisse
 - Speicherung auf dem Server selbst (Klartext) => Stehen meist auch im Klartext in Konfigurationen
 - Zugriff für alle, die auch Zugriff auf den Server haben
 - Sicherung als Teil der Backups
 - Dynamische Erzeugung beim Einrichten von Diensten

```
! main.yml ×
roles_webserver > docker_grafana > tasks > ! main.yml > {} 1 > [ ] with_items
2
3   - include_tasks: ../functions/get_secret.yml
4     with_items:
5       - { path: "{{ basedir }}/grafana_admin_pass", length: 12 }
6       - { path: "{{ basedir }}/oauth_client_secret", length: 32 }
7
```

STÄRKEN VON ANSIBLE

- Viele fertige Module, z.B. für Docker
- Mittlerweile sind viele Module unabhängig von Distribution und Betriebssystem
- Deklarative / Imperative Mischung erlaubt auch „exotische“ Konfigurationen
- Anwenden von Änderungen ist ein kontrollierter Prozess
- Es ist nicht zwingend ein Zugriff auf alle Server erforderlich
- Es gibt keinen Zentralen Server mit Zugriff auf alles

SCHWÄCHEN VON ANSIBLE

- Zugriff zu den Servern muss vom Client aus möglich sein
- Keine vorgegebene, einheitliche Projektstruktur
- Prüfung der Konfiguration nur bei Ausführung
- Git-Commits sind ein manueller Prozess



VIELEN DANK!

VOID@MEMBER.WARPZONE.MS