```
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 Aufbeu 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

TOOLS: SALT

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

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 ?

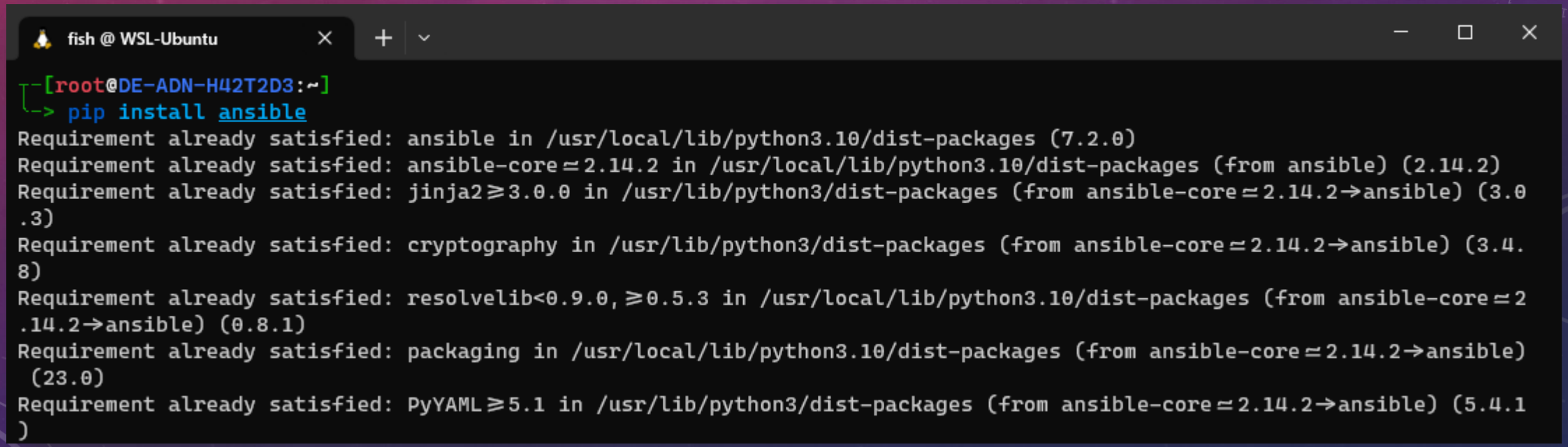
- Einfacher Einstieg
  - SSH Zugriff und Python reichen aus
  - Kein Agent
  - Kein zentraler Server

- 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



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
```

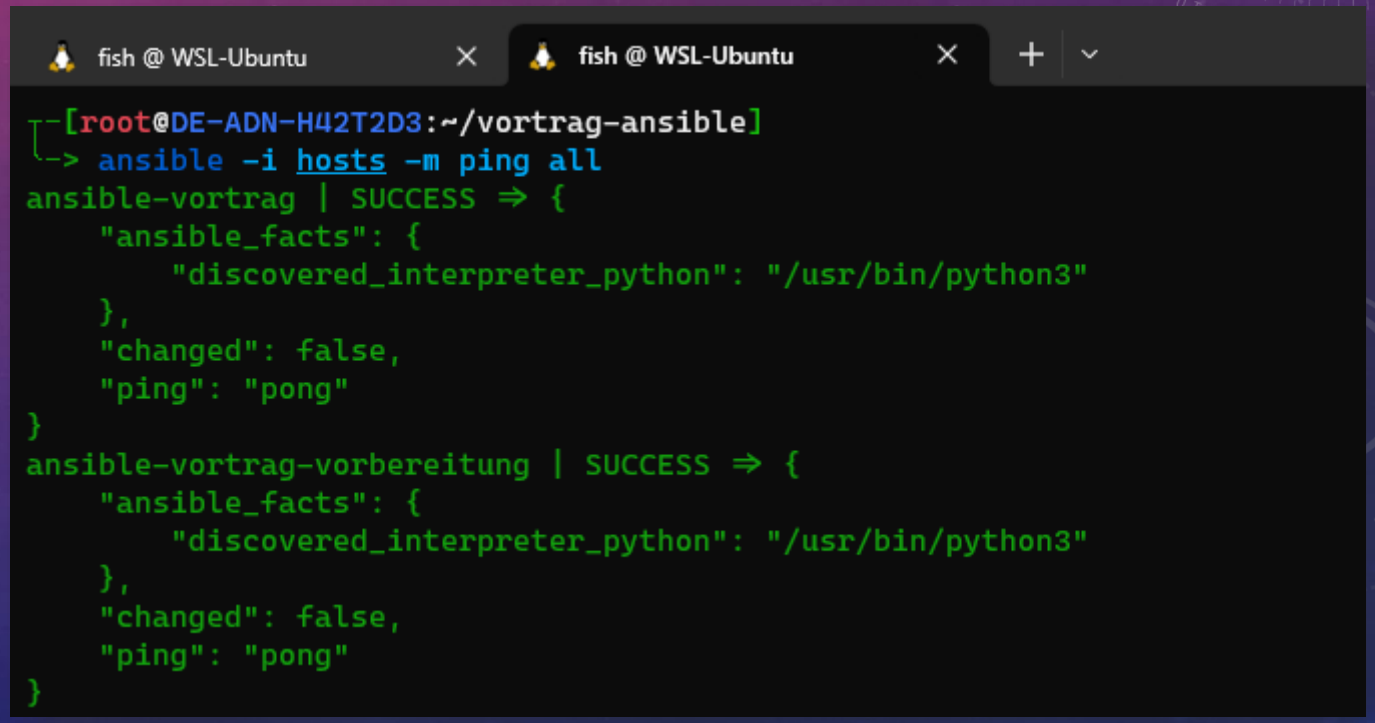
Explizite Angabe der IP

hosts

```
[test]
ansible-vortrag-vorbereitung  ansible_host=167.235.52.81

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

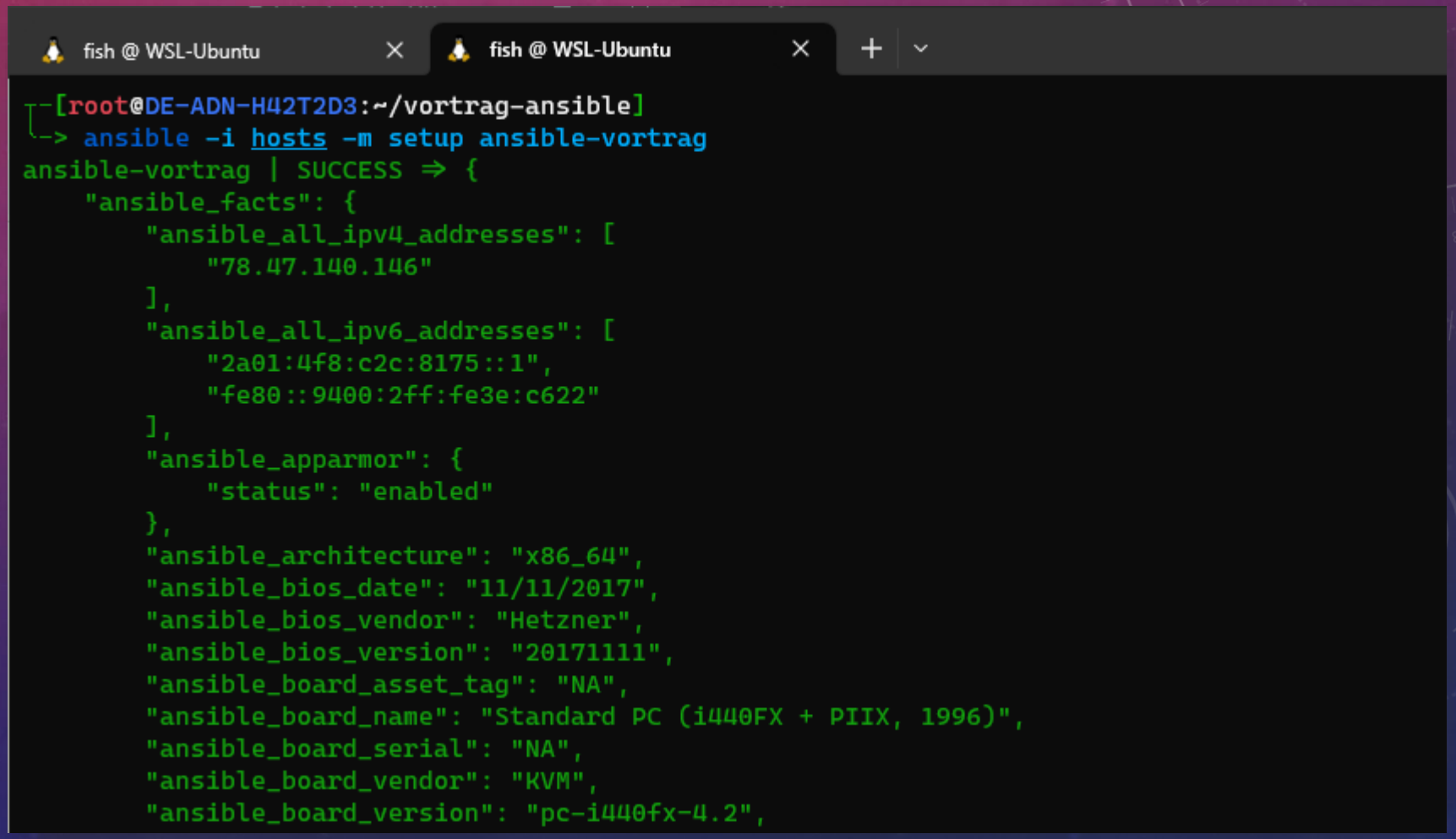
VERBINDUNGSTEST

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



ZUSTAND DES SERVERS

- Modul: Setup



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

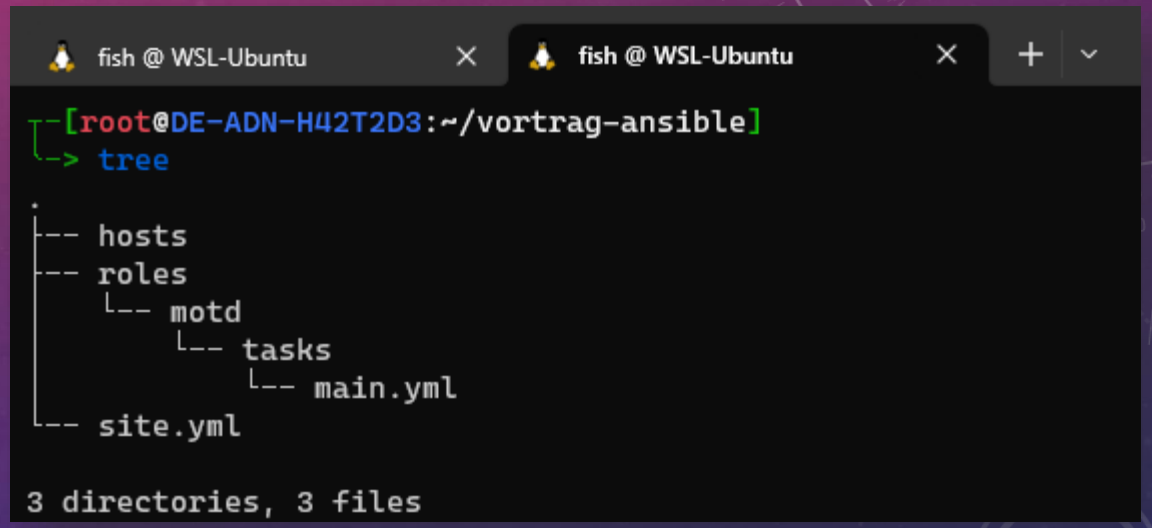
```yaml
---

- hosts: ansible-vortrag
  roles:
    - { role: roles/motd, tags: motd }

```

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



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

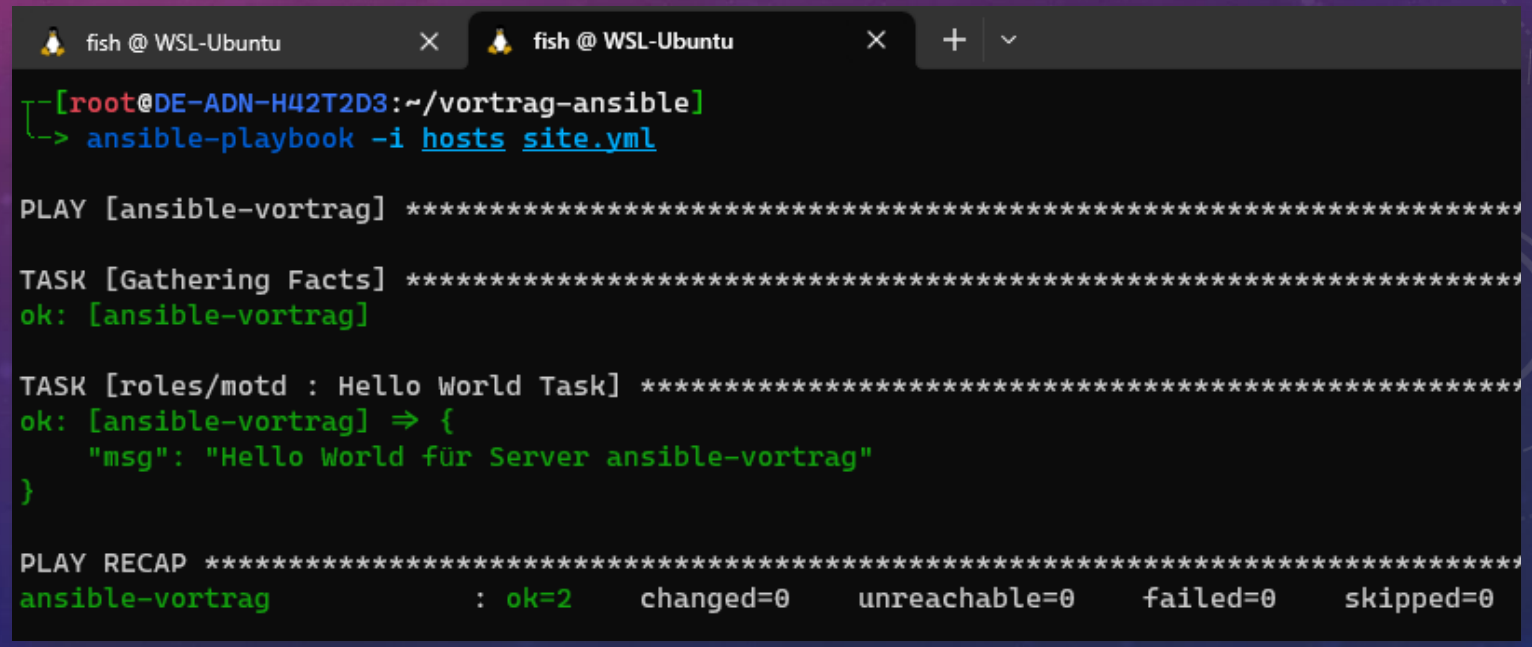
roles > motd > tasks > main.yml

```yaml
---

- name: Hello World Task
  ansible.builtin.debug:
    msg: Hello World für Server {{ inventory_hostname }}

```

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



TEMPLATES

- Schreiben von Konfigurationsdateien auf den Server
- Ansible verwendet hierbei Jinja2 Templates

- Verzeichnis „templates“ in der Rolle

main.yml | motd

roles > motd > templates > motd

```
1
2   Willkommen auf dem Server {{ inventory_hostname }} !!
3
```

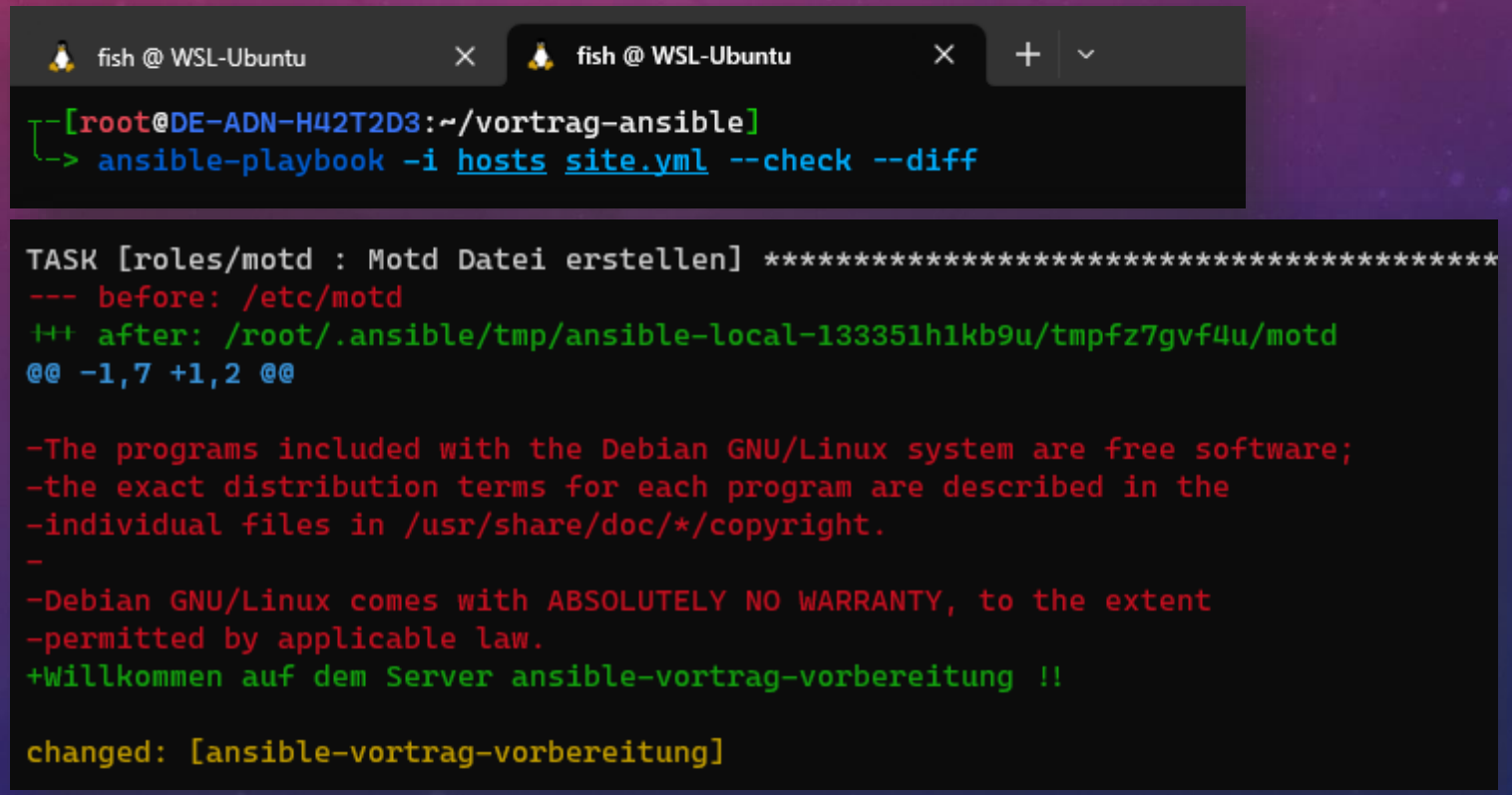
main.yml | motd

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



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 | 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 einehr „höheren“ Ebene überschreiben bereits definierte Variablen

- 1 - Standardwerte innerhalb der Rolen
- 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
```

all

group_vars > all

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

prod

group_vars > prod

```
1
2   motd_zusatztext: Das ist ein Zusatztext für alle Server in der Gruppe Prod
3
4
```

ansible-vortrag

host_vars > ansible-vortrag

```
1
2   motd_zusatztext: Das ist ein Zusatztext speziell für den Server
3
```

VARIABLEN DEFINIEREN

site.yml

```yaml
---

- hosts: ansible-vortrag-vorbereitung
  roles:
    - {
        role: roles/motd,
        tags: motd,
        motd_zusatztext: Der Text wurde beim Anwenden der Rolle überschrieben
      }

```

VARIABLEN DEFINIEREN

main.yml

roles > motd > tasks > main.yml

```yaml
---

- name: Hello World Task
  ansible.builtin.debug:
    msg: Hello World für Server {{ inventory_hostname }}

- name: Variablen hinzufügen
  ansible.builtin.set_fact:
    motd_zusatztext2: Der Text wurde in der Rolle gesetzt

- name: Motd Datei erstellen
  ansible.builtin.template:
    src: motd
    dest: /etc/motd

```

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

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

roles > motd > tasks > main.yml

```yaml
- name: Motd Datei erstellen
  ansible.builtin.template:
    src: "{{ item }}"
    dest: "/etc/{{ item }}"
  with_items:
    - motd
  when: ansible_os_family == "Debian"
```

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

```yaml
- name: Motd Datei erstellen
  ansible.builtin.template:
    src: "{{ item }}"
    dest: "/etc/{{ item }}"
  with_items:
    - motd
  when: ansible_os_family == "Debian"
  notify: restart_sshd

```

main.yml

roles > motd > handlers > main.yml

```yaml
---

- name: restart_sshd
  service:
    name: ssh
    state: restarted

```

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

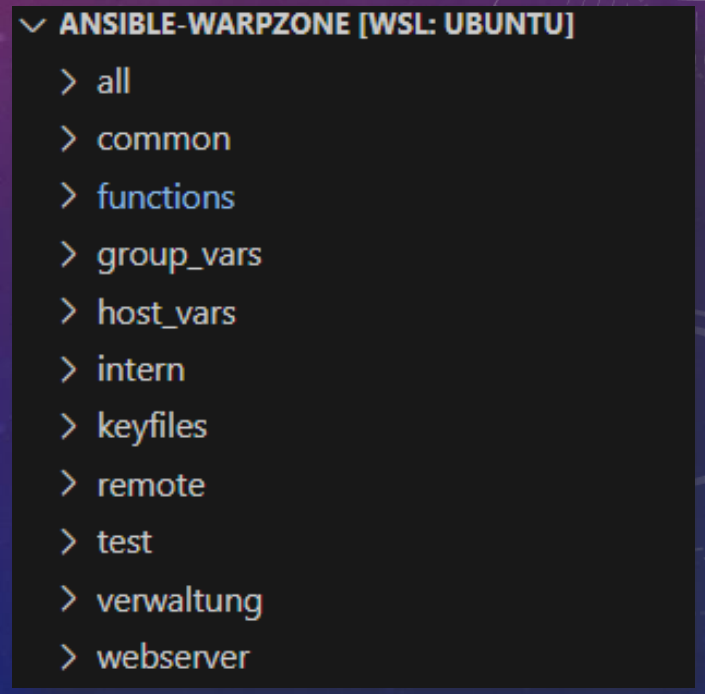
DAS SETUP DER WARPZONE

- 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



DAS SETUP DER WARPZONE

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

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

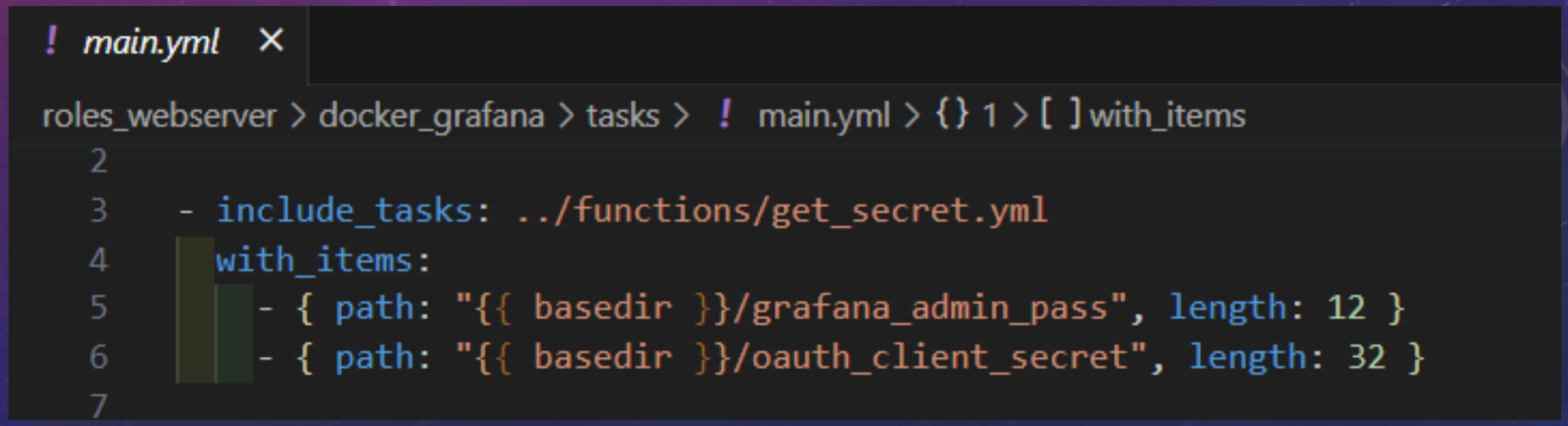
main.yml

webserver > docker_vpnserver > tasks > main.yml

```yaml
- include_tasks: ../functions/get_secret.yml
  with_items:
    - { path: "{{ basedir }}/wg_admin_pass",  length: 32 }
    - { path: "{{ basedir }}/wg_private_key",  length: -1 } # 'wg genkey'


- name: create folder struct for keycloak
  file:
    path: "{{ item }}"
    state: "directory"
  with_items:
    - "{{ basedir }}"
    - "{{ basedir }}/data"


- name: "copy {{ servicename }} config files"
  template:
    src: "{{ item }}"
    dest: "{{ basedir }}/{{ item }}"
  with_items:
    - docker-compose.yml
  register: config


- name: "stop {{ servicename }} docker"
  docker_compose:
    project_src: "{{ basedir }}"
    state: absent
  when: config.changed


- name: "start {{ servicename }} docker"
```

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



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