Einführung in Ansible & Docker mit praktischen Beispielen

Carina Haag
carin@haag-net.work

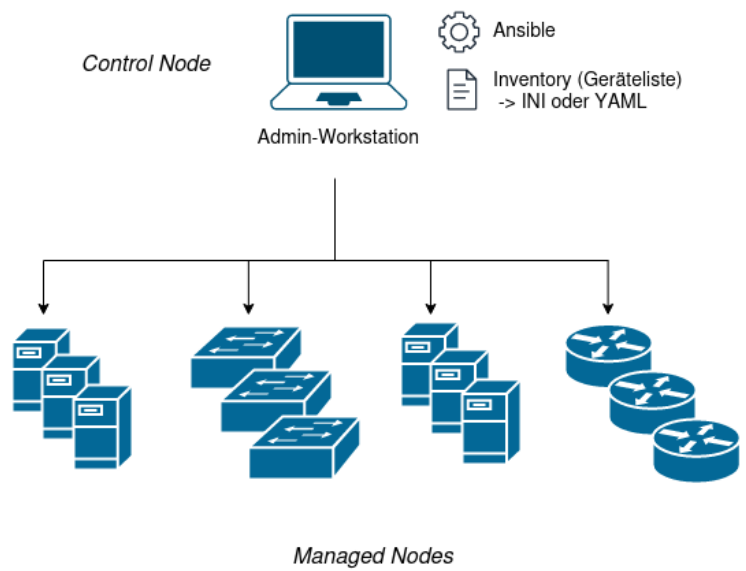
Michael Krüger
mich@elkrueger.de

Ansible

- Werkzeug zur Orchestrierung und Automatisierung
- Konfigurations-Management-System
- Control Node: Host mit Python u. Ansible
- Managed Nodes: Agent-less (SSH+Python)
- Zustand wird beschrieben, nicht der Weg zum Zustand
- Idempotent: Wiederholtes Ausführen bewirkt keine weitere Änderung

https://docs.ansible.com/

- sehr gut dokumentiert -

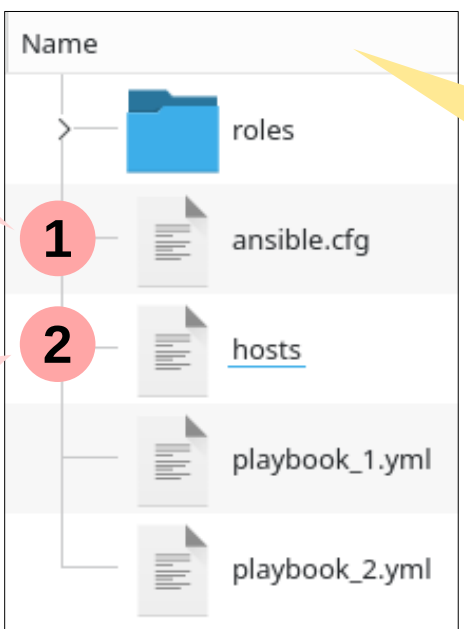


Ansible - Rahmenbedingungen

- Ansible auf dem *Control Node / Ansible-Host* installieren:
  Bsp. Debian: `apt install ansible`
- SSH-Zugriff auf dem *Managed Node* einrichten
- Verzeichnislayout anlegen – *best practice*
  (stark verkürzt für Einstieg)

https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html#directory-layout

Konfiguration von Ansible



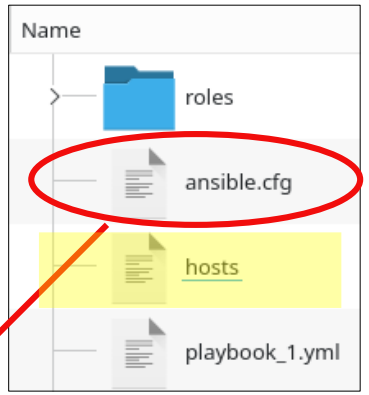
Struktur des Verzeichnisses sowie Bezeichnung der Dateien sind grundsätzlich variabel.

Inventory-Datei enthält die *Managed Nodes* quasi eine Geräteliste im INI- oder YAML-Format

roles & playbooks
→ *kommen später*

1 `ansible.cfg`

Konfigurationsdatei ist im INI-Format. In der Regel werden hier Änderungen zu den Default-Werten (`/etc/ansible/ansible.cfg`) vorgenommen.




```
1  # [...] is a section
2  [defaults]
3  »   inventory = hosts»  ;This points to the file that lists your hosts
4
```

Bsp. oben: Verweis auf die Inventory-Datei.

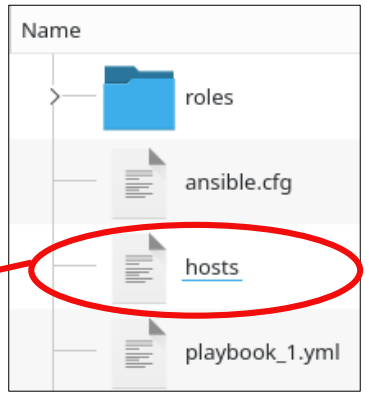
Hier direkt im Ordner, ansonsten mit angabe des Pfads.

Default Pfad lautet: `etc/ansible/hosts`

https://docs.ansible.com/ansible/latest/reference_appendices/config.html

2 hosts

- Inventory-Datei (hier mit der Bezeichnung `hosts`) wird im INI- oder im YAML-Format geführt.
- Bsp.:



Gruppierung zur Übersichtlichkeit

```
[routers]
CSR1 ansible_host=192.168.178.98
CSR2 ansible_host=192.168.56.102

[switches]
S1 ansible_host=192.168.178.103

[all:vars]
# treats the node as a network device
ansible_connection=ansible.netcommon.network_cli

# which network platform - ios, aruba, etc.
ansible_network_os=ios

# user and password
ansible_user=admin
ansible_password=geheim

# privilege mode
ansible_become=true
ansible_become_method=enable
ansible_become_password=geheim
```

Variablen können Gruppen oder einzelne Hosts zugewiesen werden, bspw. `[S1:vars]`

Passwörter im operativen Einsatz verschlüsseln → Ansible Vault

Ansible – Ad-Hoc-Befehle

Erster Verbindungstest mit einem Ad-Hoc-Befehl:

```
[redacted]:~/DevNet/devnet/ansible-examples$ ansible S1 -m ping
S1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Kommando *Host* *Modul*

IOS-Kommandos (sind auch möglich):

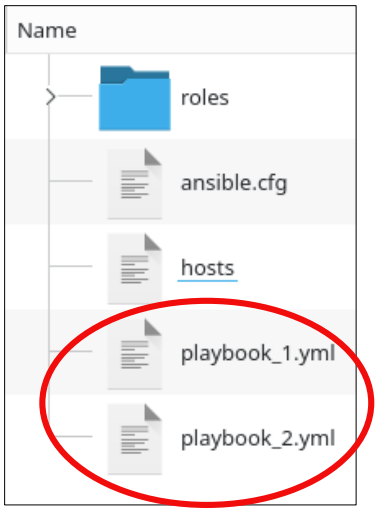
```
$ ansible all -m cli_command -a "command='show version'"
```

Modulargumente

```
[redacted]:~/DevNet/devnet/ansible-examples$ ansible all -m cli_command -a "command='show version'"
CSR1 | SUCCESS => {
    "changed": false,
    "stdout": "Cisco IOS XE Software, Version 17.03.04a\nCisco IOS Software [Amsterdam], Virtual XE So
IOSD-UNIVERSALK9-M), Version 17.3.4a, RELEASE SOFTWARE (fc3)\nTechnical Support: http://www.cisco.com
```

- `$ ansible all -m cli_command -a "command='show run'"`

3 `playbooks.yml`



- Beschreibt den Zielzustand im YAML-Format *(= YAML Ain't a Markup Language)*.
- Enthält *Tasks (Anweisungen)*, die nacheinander abgearbeitet werden.
- Wiederholende *Tasks* können auch in sogenannte *Roles* ausgelagert werden *(kommt später)*
- Definieren von Variablen ist möglich.
- Zur Nutzung der umfangreichen Module muss ggf. eine weitere Installation erfolgen:

  Bsp. Debian: `ansible-galaxy collection install cisco.ios`

Ansible – Playbooks: Beispiel

1_version_playbook.yml

```
1  ---
2  - name: Version
3    hosts: routers, switches
4    gather_facts: false
5
6
7    tasks:
8     - name: Show version
9       cisco.ios.ios_command:
10       commands:
11        - show version
12      register: version
13
14    - name: Display version
15      debug:
16       var: version["stdout_lines"]
17
```

Ausgabe:

```
PLAY [Version] ***********

TASK [Show version] ******
ok: [CSR1]
ok: [S1]

TASK [Display version] ***
ok: [CSR1] => {
    "version[\"stdout_line
        [
            "Cisco IOS XE
```

• • •

```
PLAY RECAP ************************
CSR1                       : ok=2
S1                         : ok=2
```

Aufruf des Playbooks: `$ ansible-playbook 1_version_playbook.yml`

Kommando *Pfad /Name des Playbooks*

Ansible – Playbooks: Variablen

```yaml
---
- name: Basic config vars
  hosts: routers
  gather_facts: false

  vars:
    ntp: "192.168.178.2"

  tasks:
    - name: Set banner
      cisco.ios.ios_banner:
        banner: motd
        text: |
          #Nur autorisierter Zugriff erlaubt!#
        state: present

    - name: Encrypt passwords
      cisco.ios.ios_config:
        lines:
          - service password-encryption

    - name: NTP Server
      cisco.ios.ios_ntp_global:
        config:
          servers:
            - server: "{{ ntp }} "
        state: replaced

    - name: Save running to start-config when modified
      cisco.ios.ios_config:
        save_when: modified
```

Variable definieren

Variable verwenden

Playbook wird ausgeführt

```
PLAY [Basic config vars] ********

TASK [Set banner] ***************
ok: [CSR1]

TASK [Encrypt passwords] ************************
ok: [CSR1]

TASK [NTP Server] *******************************
changed: [CSR1]

TASK [Save running to start-config when modified]
changed: [CSR1]

PLAY RECAP **************************************
CSR1                       : ok=4    changed=2
```

Überprüfung

```
CSR1#show run | section ntp
ntp server 192.168.178.2
CSR1#show run | begin banner
banner motd ^C
#Nur autorisierter Zugriff erlaubt!#
^C
```

Ansible – Playbooks: Beispiel Facts

ansible-examples > ! 0_facts.yml

```yaml
---
- name: Version
  hosts: routers, switches
  gather_facts: false


  tasks:
    # https://docs.ansible.com/ansible/latest/collections/cisco/ios/ios_facts_module.html
    - name: Retrieve ios facts
      cisco.ios.ios_facts:
        gather_subset: all


    - name: Display version
      ansible.builtin.debug:
        msg:
          - "The IOS type is {{ ansible_net_iostype }} and version is: {{ ansible_net_version }}"
          - "Filesystem: {{ ansible_net_filesystems_info }}"

    - name: Retrieve status interfaces
      cisco.ios.ios_command:
        commands: show ip interface brief
      register: interfaces

    - name: Display interfaces
      ansible.builtin.debug: msg="{{ interfaces.stdout_lines }}"

```

Ausgabe je nach Bedarf anpassen

Ansible – Playbooks: Beispiel Facts

Ausgabe:

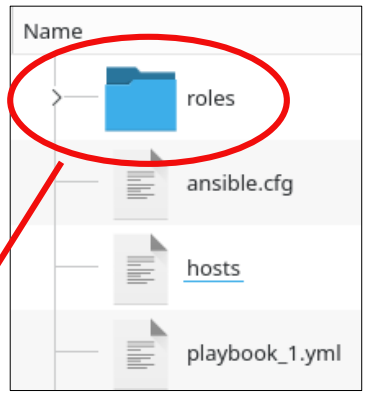
```
:~/DevNet/devnet/ansible-examples$ ansible-playbook 0_facts.yml

PLAY [Version] *****************************************************************

TASK [Retrieve ios facts] ******************************************************
ok: [CSR1]
ok: [S1]

TASK [Display version] *********************************************************
ok: [S1] => {
    "msg": [
        "The IOS type is IOS and version is: 12.2(44)SE5",
        "Filesystem: {'flash:': {'spacetotal_kb': 27342.0, 'spacefree_kb': 20367.5}}"
    ]
}
ok: [CSR1] => {
    "msg": [
        "The IOS type is IOS-XE and version is: 17.03.04a",
        "Filesystem: {'bootflash:': {'spacetotal_kb': 6139200.0, 'spacefree_kb': 5309112.0}}"
    ]
```

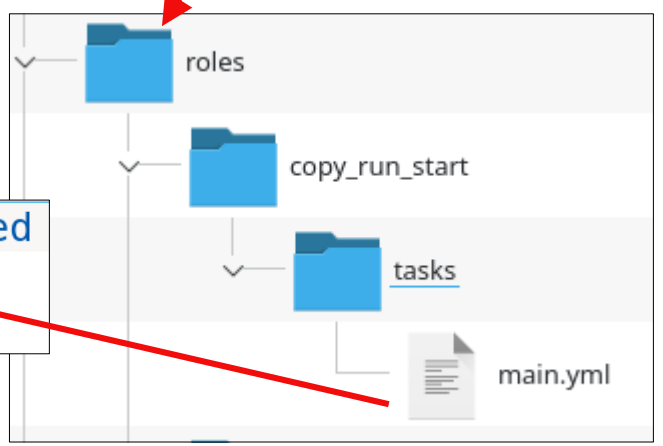
4 roles

- In *Roles* können alle Ansible-Artifakte ausgelagert und in *Playbooks* aufgerufen werden.
- *Tasks* werden als *Role* definiert, die häufig verwendet werden – eine Art („Unterprogramm“)
- Einfaches Beispiel:

`main.yaml`

```
1  - name: Save running to start-config when modified
2    cisco.ios.ios_config:
3     save_when: modified
```



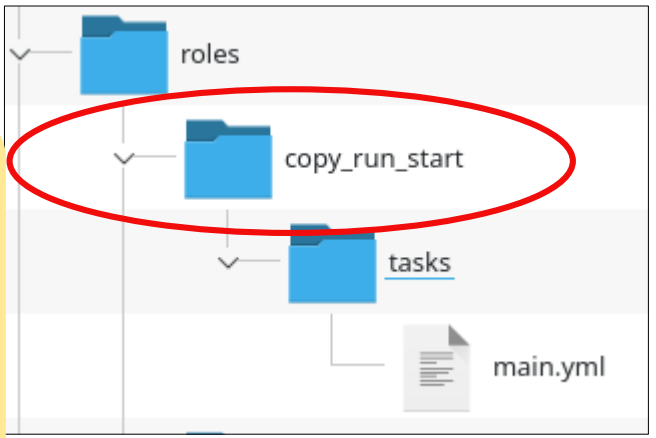


Ansible – Roles: Einfaches Beispiel

ansible-examples > ! 3c_basic_config_role_playbook.yml

```yaml
---
- name: Basic config role
  hosts: routers, switches
  gather_facts: false

  vars:
    ntp: "192.168.178.2"


  tasks:
    - name: set banner
      cisco.ios.ios_banner:
        banner: motd
        text: |
          #Nur autorisierter Zugri
        state: present

    - name: Encrypt passwords
      cisco.ios.ios_config:
        lines:
          - service password-encryption

    - name: NTP Server
      cisco.ios.ios_ntp_global:
        config:
          servers:
            - server: "{{ ntp }}"
        state: replaced

    - name: Save config
      ansible.builtin.include_role:
        name: copy_run_start
```

Role wird innerhalb des *Playbooks* mit dem entsprechenden Modul und Namen aufgerufen.



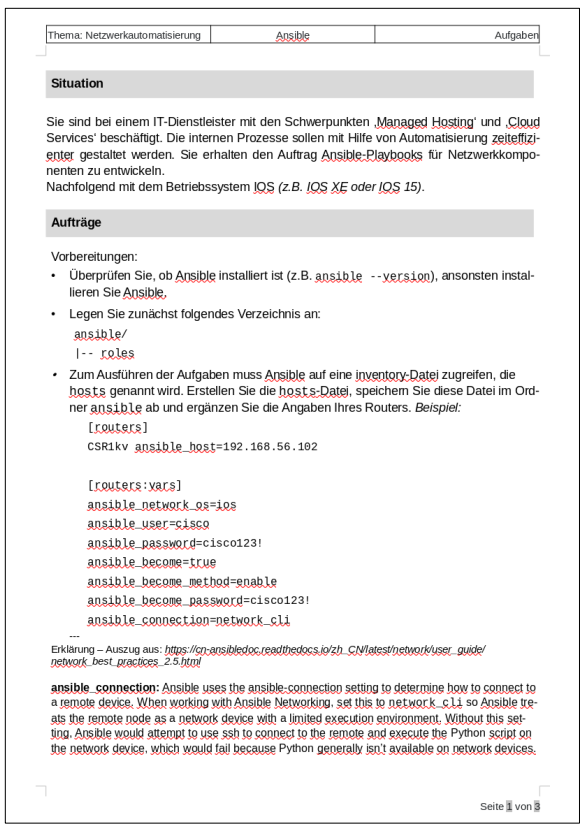

```yaml
- name: Save config
  ansible.builtin.include_role:
    name: copy_run_start
```

Ansible – Weitere Beispiele / Aufgaben

- Informations- und Aufgabenblatt im Verzeichnis
- Zugehörige Ansible-Playbooks:

  https://codeberg.org/cahaag/devnet/src/branch/main/ansible-examples
- Tipp:


```
WARNING  Listing 3 violation(s) that are fatal
fqcn[action-core]: Use FQCN for builtin module actions (debug)
6_staticRoute_playbook.yml:23 Use `ansible.builtin.debug` or

name[casing]: All names should start with an uppercase letter.
roles/show_ip_route/tasks/main.yaml:1 Task/Handler: run show i
```



Thema: Netzwerkautomatisierung	Ansible	Aufgaben

Situation

Sie sind bei einem IT-Dienstleister mit den Schwerpunkten ‚Managed Hosting' und ‚Cloud Services' beschäftigt. Die internen Prozesse sollen mit Hilfe von Automatisierung zeiteffizienter gestaltet werden. Sie erhalten den Auftrag Ansible-Playbooks für Netzwerkkomponenten zu entwickeln.
Nachfolgend mit dem Betriebssystem IOS *(z.B. IOS XE oder IOS 15)*.

Aufträge

Vorbereitungen:

- Überprüfen Sie, ob Ansible installiert ist (z.B. `ansible --version`), ansonsten installieren Sie Ansible.
- Legen Sie zunächst folgendes Verzeichnis an:

```
ansible/
|-- roles
```

- Zum Ausführen der Aufgaben muss Ansible auf eine inventory-Datei zugreifen, die `hosts` genannt wird. Erstellen Sie die `hosts`-Datei, speichern Sie diese Datei im Ordner `ansible` ab und ergänzen Sie die Angaben Ihres Routers. *Beispiel:*

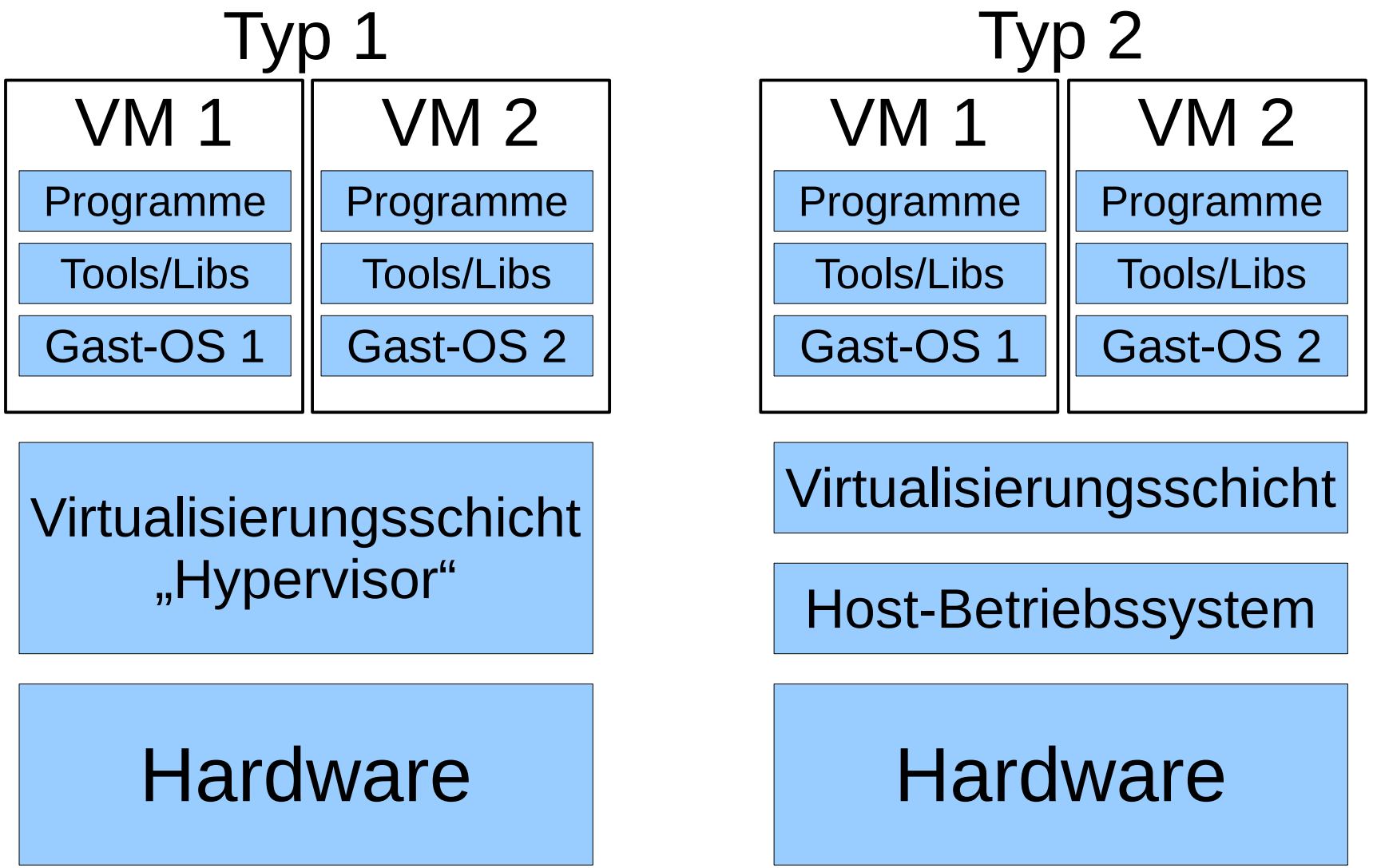
```
[routers]
CSR1kv ansible_host=192.168.56.102


[routers:vars]
ansible_network_os=ios
ansible_user=cisco
ansible_password=cisco123!
ansible_become=true
ansible_become_method=enable
ansible_become_password=cisco123!
ansible_connection=network_cli
```

Erklärung – Auszug aus: *https://cn-ansibledoc.readthedocs.io/zh_CN/latest/network/user_guide/network_best_practices_2.5.html*

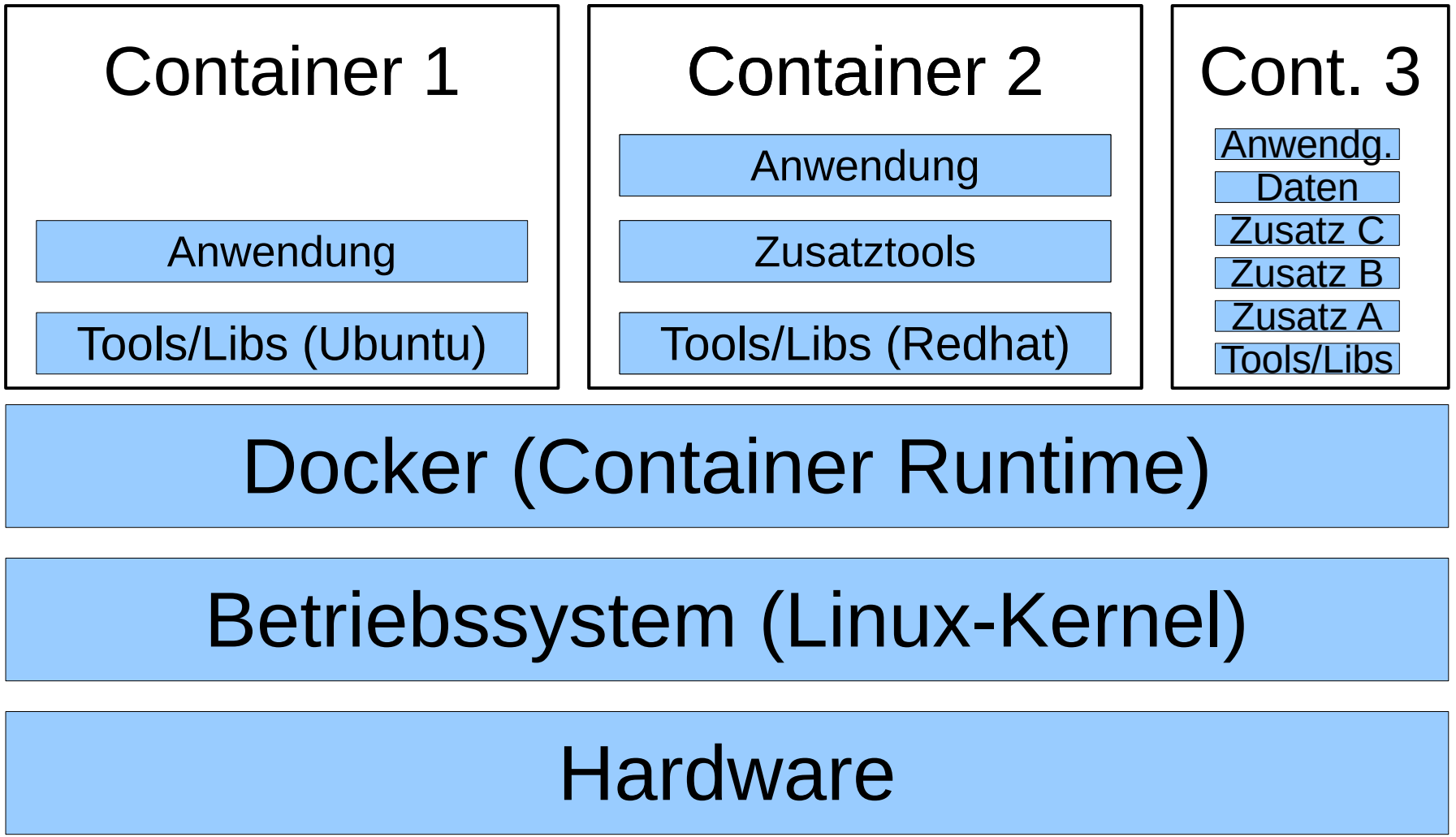
ansible_connection: Ansible uses the ansible-connection setting to determine how to connect to a remote device. When working with Ansible Networking, set this to `network_cli` so Ansible treats the remote node as a network device with a limited execution environment. Without this setting, Ansible would attempt to use ssh to connect to the remote and execute the Python script on the network device, which would fail because Python generally isn't available on network devices.

Seite 1 von 3

Vor Docker: Klassische Virtualisierung

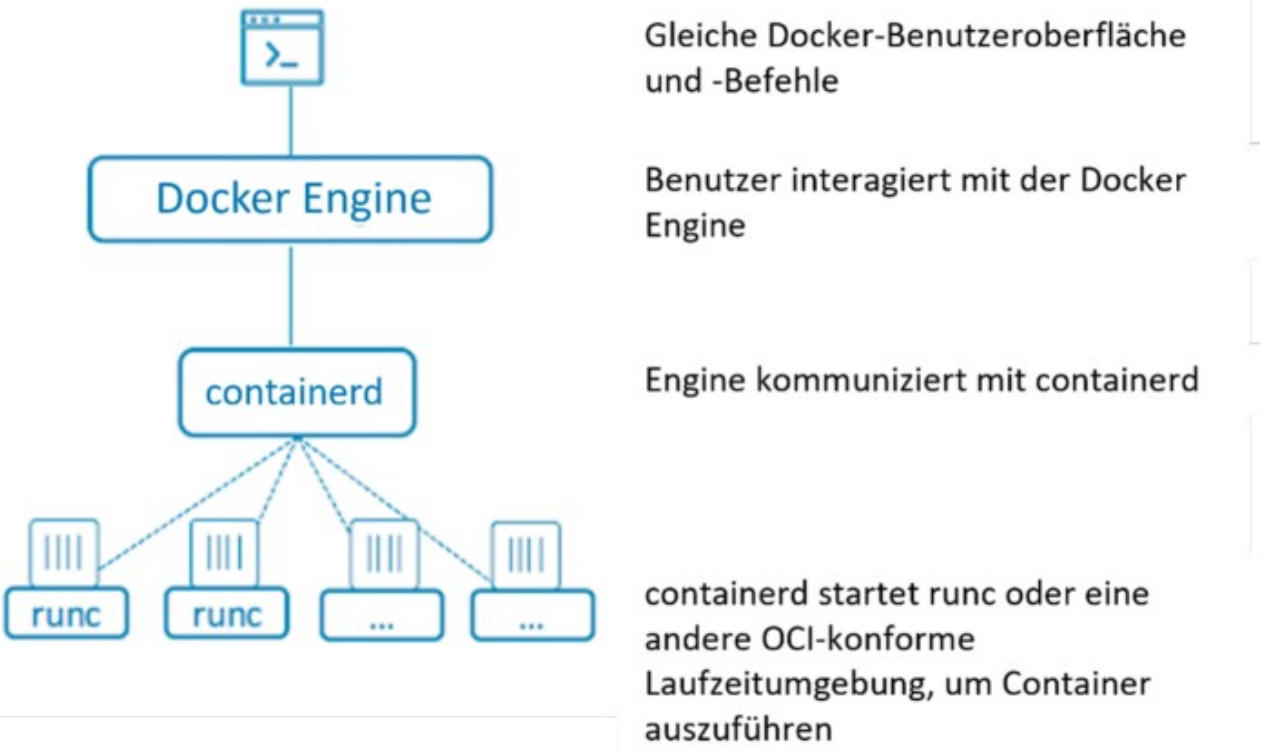


Docker ist keine echte Virtualisierung, sondern Container-Engine



Docker-Übersicht

- Docker unterstützt als Hostsystem Windows, MacOS, Linux
  - Container „sehen“ allerdings immer einen Linux-Kernel
  - Konkurrenz: LXC, Podman, Kubernetes (k8s), Hyper-V Container
  - Details: https://optimizedbyotto.com/post/linux-containers-docker/



Wie starte ich mit Docker?

- Docker Desktop installieren
  - GUI für bequeme Verwaltung der Container
- Alternativ: Container-Engine eines vorhandenen Systems
  - Viele NAS beinhalten Docker-Engine
- Für den Zugriff von außen einen Reverse-Proxy einrichten
  - Dieser ist selbst ein Container innerhalb von Docker
  - jwilder/nginx-proxy
  - Traefik
  - Caddy
  - Envoy
  - Nginx Proxy Manager

Images

- Zentrales Element von Docker: Images
  - „A container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.“
- Images bauen aufeinander auf. Die Erstellung eines Images (aus einem anderen) geschieht mit einem **`Dockerfile`**
  - Selbst erstelltes Image **ddorf** baut auf **nginx**
  - öffentliches Image **nginx** baut auf **debian**
  - öffentliches Image **debian** ist ein Basisimage („FROM scratch“), wird direkt mit Dateien eines Debian-Systems befüllt
  - **scratch** ist ein Schlüsselwort für „komplett leeres Image“
- Jedes Image hat einen **Entrypoint** → das Programm, welches im Standardfall ausgeführt wird.

Drei Dockerfiles im Vergleich

debian (komplett)

```
FROM scratch
ADD rootfs.tar.xz /
CMD ["bash"]
```

nginx (stark gekürzt)

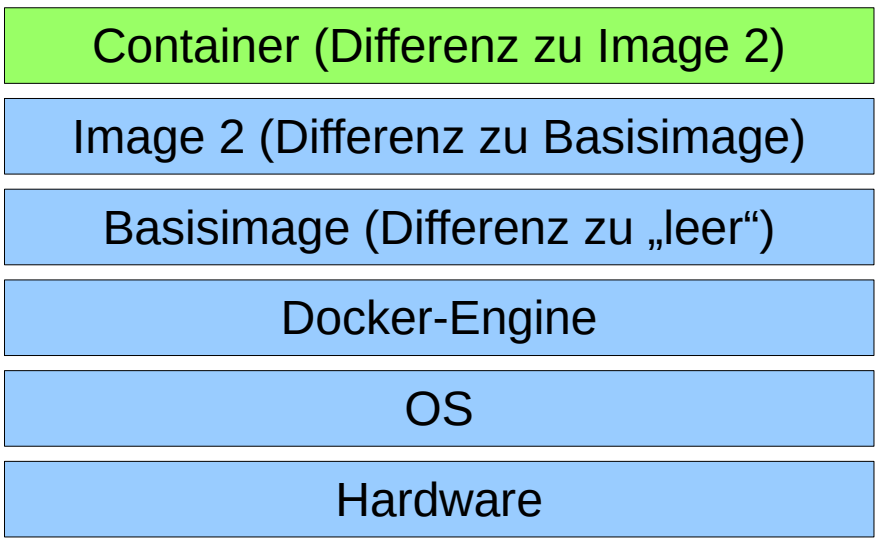
```
FROM debian
RUN apt-get update
 && apt-get install nginx
 && rm -rf /var/apt/*
EXPOSE 80
CMD ["nginx", "-g",
"daemon off;"]
```

ddorf (fiktiv)

```
FROM nginx
RUN apt-get install php
ADD ddorf.tgz
/var/www/html/
COPY ddorf.conf
/etc/nginx/nginx.conf
```

Container sind laufende, beschreibbare Images

- Jedes Mal, wenn ein Image gestartet wird, entsteht ein (meist kurzlebiger) Container
  - „A container is a runtime instance of an image—what the image becomes in memory when actually executed. It runs completely isolated from the host environment by default, only accessing host files and ports if configured to do so.“
- Ein Container ist eine beschreibbare Schicht über einem Image (technisch dem Image ähnlich, aber veränderbar)
- Aus einem Image können viele Container sowie abgeleitete Images erstellt werden



Container starten: docker run

- docker run *[optionen] imagename[:tag] [befehl]*
  - Docker lädt automatisch alle noch nicht lokal vorhandenen Images und deren Basisimages vom „Docker Hub“ herunter
  - befehl kann weggelassen werden, dann wird der Entrypoint (CMD) aus dem Dockerfile genommen
  - Nützliche Optionen
    - --rm : Remove – Lösche Container nach Ende des Programms
    - -it : Interaktives Terminal – In Verbindung mit einer Shell nützlich
    - -d : Detach – im Hintergrund laufen lassen
    - -n : Name – Container benennen (statt ID)
    - -p : Publish – Öffentliche Ports vom Host in den Container weiterleiten
    - -v : Volume – Verzeichnis vom Host auf dem Container einbinden
    - -e : Environment – Umgebungsvariablen setzen

Beispiele für docker run

- docker run hello-world:latest
  - Lädt (falls nötig) das Image hello-world (Tag „latest“) vom Docker Hub und startet den Entrypoint (CMD-Standardbefehl) des Images
- docker run -it ubuntu bash
  - Lädt (falls nötig) …. und startet in einen ubuntu-Container mit einem interkativen Terminal die bash-Shell
- docker run -n webserver -v /website:/usr/share/nginx/html:ro -p 4000:80 -d nginx
- docker run -n datenbank -v /dbdaten:/var/lib/mysql:rw -e MYSQL_ROOT_PASSWORD=supergeheim -d mysql

Docker Compose

- Container einzeln über Kommandozeile zu starten ist mühsam
- „Docker Compose is a tool for defining and running multi-container applications. […] Compose simplifies the control of your entire application stack, making it easy to manage services, networks, and volumes in a single, comprehensible YAML configuration file. Then, with a single command, you create and start all the services from your configuration file.“
- Die Konfigdatei heißt i.d.R. **docker-compose.yml**
- War früher ein Zusatztool, ist aber inzwischen in die Docker-Engine integriert
  - Alt: docker-compose ← Einzelbefehl
  - Neu: docker compose ← Unterkommando von docker

Compose-Beispiel

- Rechts ist eine docker-compose.yml, die zwei Container definiert
  - MariaDB-Datenbank
  - Webserver mit Nextcloud
- In den Volumes werden die Nutzerdaten unabhängig vom Container hinterlegt
- Die beiden Container kommunizieren intern („links“)
- Nur der Port 80 des Webservers wird (als 8080) nach außen geöffnet

```
Version: '2'
volumes:
  nextclouddata:
  dbdata:
services:
  db:
    image: mariadb:10.6
    restart: always
    volumes:
      - dbdata:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=lsdk$f(gnkdsfjgnkldfsj
      - MYSQL_PASSWORD=5n45ni$un5i4u5ngu5oiz6
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
  app:
    image: nextcloud
    restart: always
    ports:
      - 8080:80
    links:
      - db
    volumes:
      - nextclouddata:/var/www/html
    environment:
      - MYSQL_PASSWORD=5n45ni$un5i4u5ngu5oiz6
      - MYSQL_DATABASE=nextcloud
      - MYSQL_USER=nextcloud
      - MYSQL_HOST=db
```

Compose: Skalierung mit replicas, Ressourcenlimits

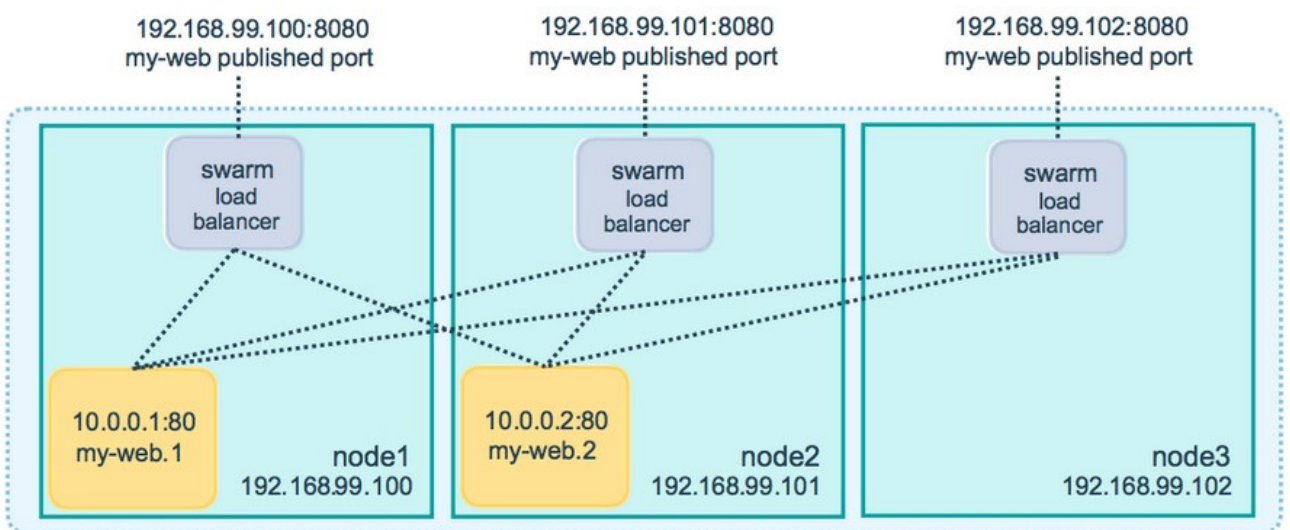
```
services:
  webserver:
    image: mmattes/ddorf:latest
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
```

Starte 5 identische Container parallel

Max 10% Rechenzeit eines Kerns

Skalierung: Ein Schwarm von Docker-Hosts

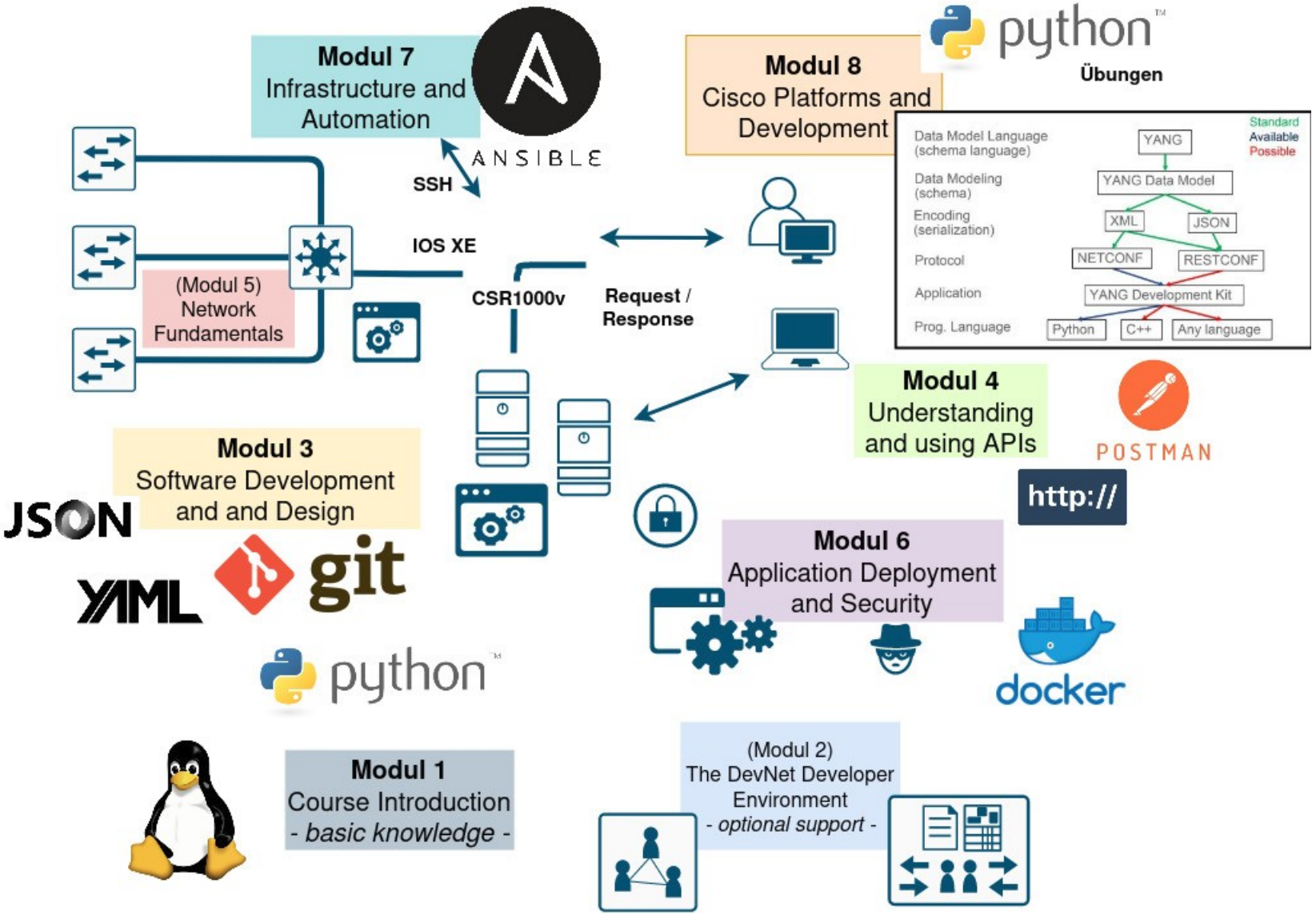
- Mit Stacks kann man Gruppen von Containern anlegen, die zusammenarbeiten sollen → laufen in der Docker Cloud
- Ein Swarm ist eine Gruppe von Docker-Hosts, die sich die Last der Container teilen → eigene Hardware bzw. Maschinen von Azure, AWS, GCP, Hetzner, Oracle Cloud, ...
  - docker swarm init auf dem Master
  - docker swarm join --token … *ip:port* auf den Workern
  - docker swarm leave



Übung

- Vor dem Verzweifeln lieber abbrechen und in Ruhe später nochmal anschauen ;-)
  - Richten Sie Docker ein (auf VM oder echtem Host)
  - Starten Sie das Image hello-world
  - Starten Sie das ubuntu-Image interaktiv
  - Erstellen Sie ein Webserver-Image auf Basis von nginx
  - Fassen Sie Webserver+MySQL in einer docker-compose.yml zusammen
  - Starten Sie die docker-compose.yml
- Viel Spaß!

Networking Academy Program – DevNet Associate



Verankerung im Unterricht

- Fachinformatiker:in (AO 2020), insbesondere Systemintegration
  - LF10b: Kompetenz, Serverdienste bereitzustellen, zu administrieren
  - „[SuS] informieren sich über Serverdienste sowie Plattformen […] wählen diese gemäß den Kundenanforderungen aus […] planen [und implementieren] die Konfiguration der ausgewählten Dienste […] automatisieren Administrationsprozesse“
  - LF11b: Kompetenz, [...] Schutzmaßnahmen umzusetzen
  - LF12b: […] Systemintegration vollständig durchzuführen
- RLP: Berufliches Gymnasium Technik → Informationstechnik
  - Lernbereich 8: „Virtuelle Maschinen erzeugen und im Netzwerk bereitstellen“
  - Lernbereich 9: „Geeignete [Netzwerk]Architektur […] auswählen“

Danke

Herzlichen Dank für Ihre Aufmerksamkeit!

Anhang: Images bearbeiten und selbst erstellen

- Das einzelne Image ist nach der Erstellung **unveränderlich**
  - Änderung 1: Image löschen und neu unter gleichem Namen anlegen → vernichtet alle alten Container mit
  - Änderung 2: Neue Version eines Images mit höherer Versionsnummer erstellen → alte Container bleiben
- Image neu erstellen: Zuerst **`Dockerfile`** anlegen
  - Am besten alle Dateien inkl. Dockerfile zu jedem Image in einem eigenen Unterverzeichnis sammeln
- Ins UV wechseln, docker build -t name-des-neuen-images
  - -t heißt „tag“ - ansonsten bekommt Image eine Seriennummer

Beispiel (1) - Dockerfile anlegen

```
nutzer@dockerhost:~$ docker images
REPOSITORY   TAG      IMAGE ID       CREATED         SIZE
ubuntu       latest   7b9b13f7b9c0   6 weeks ago     118MB
hello-world  latest   48b5124b2768   6 months ago    1.84kB

nutzer@dockerhost:~$ cd ddorf

nutzer@dockerhost:~/ddorf$ cat Dockerfile
FROM ubuntu
RUN echo "Willkommen im Image ddorf!" >/etc/motd
RUN echo '[ ! -z "$TERM" -a -r /etc/motd ] && cat
/etc/motd' >> /etc/bash.bashrc

nutzer@dockerhost:~/ddorf$
```

Beispiel (2) - Image erstellen

```
nutzer@dockerhost:~/ddorf$ docker build .
Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu
 ---> 7b9b13f7b9c0
Step 2/3 : RUN echo "Willkommen im Image ddorf!"
>/etc/motd
 ---> Running in cdac85df6e75
 ---> f4e34a4b221c
Removing intermediate container cdac85df6e75
Step 3/3 : RUN echo '[ ! -z "$TERM" -a -r /etc/motd ]
&& cat /etc/motd' >> /etc/bash.bashrc
 ---> Running in 16adbdd96349
 ---> 64a0f31f5f64
Removing intermediate container 16adbdd96349
Successfully built 64a0f31f5f64
```

Beispiel (3) - Image als Container ausführen

```
Successfully built 64a0f31f5f64


nutzer@dockerhost:~$ docker images
REPOSITORY   TAG      IMAGE ID      CREATED         SIZE
<none>       <none>   64a0f31f5f64  5 minutes ago 118MB
ubuntu       latest   7b9b13f7b9c0  6 weeks ago   118MB
hello-world  latest   48b5124b2768  6 months ago  1.84kB


nutzer@dockerhost:~$ docker run -it 64a0f31f5f64
Willkommen im Image ddorf!
root@4a8a161a259b:/# exit
exit


nutzer@dockerhost:~$
```

Beispiel (4) - Image wieder entfernen

```
nutzer@dockerhost:~$ docker image rm 64a0f31f5f64
Error response from daemon: conflict: unable to delete
64a0f31f5f64 (must be forced) - image is being used by
stopped container 4a8a161a259b
nutzer@dockerhost:~$ docker container rm 4a8a161a259b
4a8a161a259b
nutzer@dockerhost:~$ docker image rm 64a0f31f5f64
Deleted: sha256:64a0f31f5f647841db205484d2325eb6540bc234cbbfaf9dead18e56c7bf8f70
Deleted: sha256:21616aa59e1d5138659239a8981978fb6d313a9e7d32259a4132336cbfea1d98
Deleted: sha256:f4e34a4b221c644ccc02e866267e8e86ace94bbb7724947dd2f4436582c3ee77
Deleted: sha256:8485f3b67d19a3143dbc5b64aff6dd36c78f8a83bc8d5783dfab15a7687e743b
```

Beispiel (5) - Mit Tags arbeiten

```
nutzer@dockerhost:~/ddorf$ docker build -t ddorf .

Sending build context to Docker daemon  2.048kB
Step 1/3 : FROM ubuntu
[… gekürzt …]
Removing intermediate container e60029dd9b57
Successfully built 0fa604fd2939
Successfully tagged ddorf:latest

nutzer@dockerhost:~/ddorf$ docker images

REPOSITORY   TAG      IMAGE ID       CREATED         SIZE
ddorf        latest   0fa604fd2939   8 seconds ago   118MB
ubuntu       latest   7b9b13f7b9c0   6 weeks ago     118MB
hello-world  latest   48b5124b2768   6 months ago    1.84kB
```

Images im Docker Hub teilen

- Auf hub.docker.com registrieren
- docker login
- docker tag ddorf mmattes/ddorf:april24
- docker push mmattes/ddorf:april24
  - Das neueste hochgeladene Image ist automatisch :latest
  - Standardmäßig sind alle Images öffentlich
- Fremdes Image: docker run username/imagename:tag
- Wenn ein öffentliches Image (vom Hub) keinen Usernamen hat, ist es ein offizielles Image der Firma Docker Inc.