



**Technische Berufsschule Zürich TBZ**  
**Höhere Fachschule**

# HF-Lehrgang

**Modul CSec**

**PodMan**



# Agenda

- PodMan, 1. Teil
  - Chapter 1, Introduction and Overview of Containers
  - Chapter 2, Podman Basics
  - Chapter 3, Container Images
  - Nachbesprechung
- PodMan, 2. Teil
  - Chapter 4, Custom Container Images
  - Chapter 5, Persisting Data
  - Nachbesprechung
- PodMan, 3. Teil
  - Chapter 6, Troubleshooting Containers
  - Chapter 7, Multi-container Applications with Compose
  - Nachbesprechung
- **Nur Fortgeschrittene (Selbststudium)**
  - Red Hat System Administration II, Kapitel 13, Container als Systemprozesse eintragen
- Nachbesprechung (vor Ort)



# Hands-on

## Sichere Container Umgebung - PodMan

Im dritten Teil wechseln wir auf eine Sichere Container Umgebung wie PodMan.

Dazu arbeitet den RedHat Kurs [Red Hat OpenShift Developer I: Introduction to Containers with Podman](#) durch.

## Chapter 1, Introduction and Overview of Containers

- Describe how containers facilitate application development.
- Describe the basics of containers and how containers differ from Virtual Machines.
- Describe container orchestration and the features of Red Hat OpenShift.

## Chapter 2, Podman Basics

- Creating Containers with Podman
- Container Networking Basics
- Accessing Containerized Network Services
- Accessing Containers
- Managing the Container Lifecycle



### Zusatzaufgabe:

- Startet Eure erstellten Microservices mit PodMan, wo sind die Unterschiede?
- Erstellt für die Datenbank ein eigenes Netzwerk, warum solltet Ihr das tun?


- Arbeitet die Kapitel in RedHat Academy durch
- Verwendet dazu das RedHat Lab Environment.
- Die Kurse sind bis zum 18. August 2024 freigeschaltet.
- **Zusatzaufgaben** ergänzen die Unterrichtsmaterialien von RedHat und stellen einen Bezug zum Modul MSVC her.


[https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2\\_Unterrichtsressourcen/C](https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2_Unterrichtsressourcen/C)


# Unterrichtsmaterialien (1)


Redhat.com Support FAQ


**Red Hat Academy** ×


 Manage Classes


 DO188-EMEA-TBZR00004

 Access Resources

 View Courses and Certifications

 Get Trained and Certified

 Give Feedback

 Get Support

 **DO180v4.12 will be retired on June 30 - 04/02/2024**  
Read the full announcement on Learning Community - click "Read more"

## Classes / DO188-EMEA-TBZR00004

### Class Details

Class name	Course title	Students	Date Range	Label
DO188-EMEA-TBZR00004	DO188 - Red Hat OpenShift Development I: Introduction to Containers with Podman 4.12	12	May 20 2024 - Aug 18 2024	ITCNE23-CSEC

## No-cost RHEL for developers subscription

This subscription includes:

- Red Hat Enterprise Linux provided via this subscription is for individual developers only. For Corporate and Enterprise subscription options, [please see this](#).
- Red Hat Enterprise Linux Server (all currently supported releases)
- Additional development tools
- Numerous add-ons such as resilient storage, scalable file systems, and high-performance networking
- Access to the Red Hat Customer Portal for software updates and thousands of knowledge-based articles

[Read the FAQ](#)

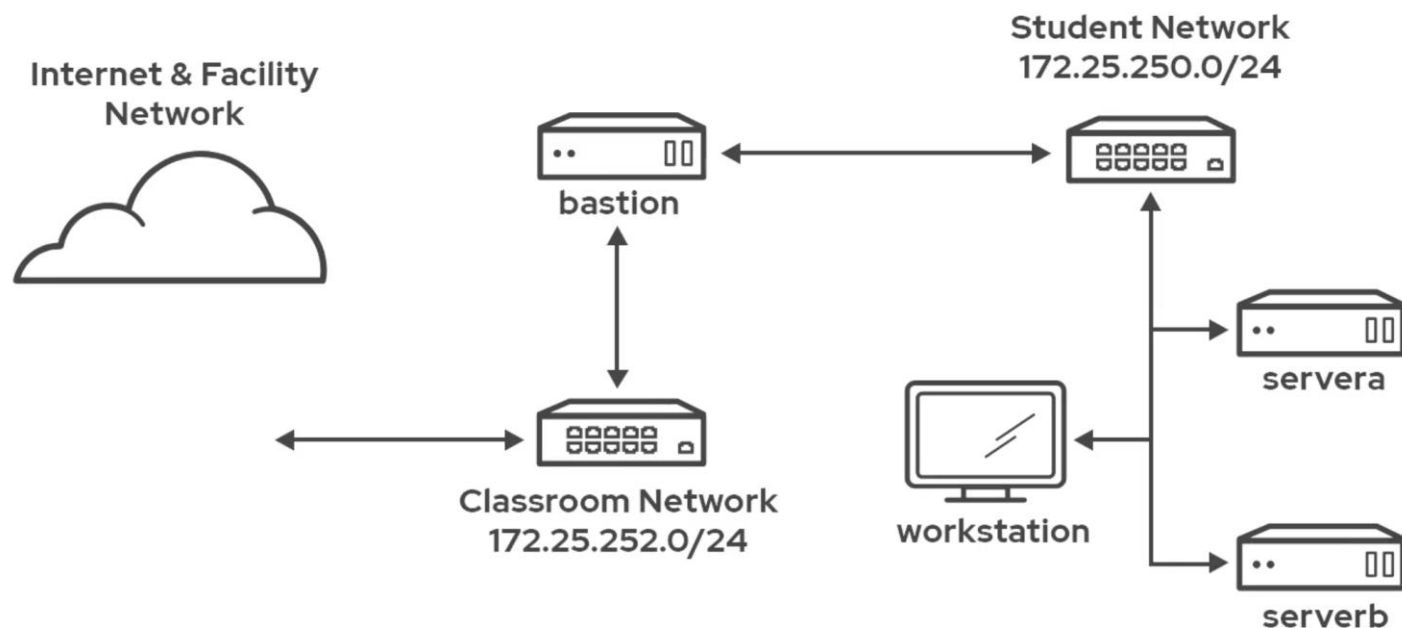
- **Red Hat Academy**

- [https://cdnapisec.kaltura.com/p/2032581/embedPlaykitJs/uiconf\\_id/52134902?iframeembed=true&entry\\_id=1\\_i1m1490c](https://cdnapisec.kaltura.com/p/2032581/embedPlaykitJs/uiconf_id/52134902?iframeembed=true&entry_id=1_i1m1490c)
- Label des Kurses: ITCN23-CSEC

- **Red Hat Developer Subscription (optional)**

- <https://developers.redhat.com/articles/getting-red-hat-developer-subscription-what-rhel-users-need-know#>

# Unterrichtsmaterialien (2)



- **RedHat Lab Environment**
  - Ihr habt 80 Stunden Lab Zugriff

Lab Hours Used: 12/80

## ► Lab Controls

Click **CREATE** to build all of the virtual machines needed for the environment. Once the environment can then be stopped and restarted.

If you **DELETE** your lab, you will remove all of the virtual machines.

<div>DELETE</div> <div>STOP <span>i</span></div>	
bastion	active
classroom	active
servera	active
serverb	active
workstation	active



Auto-stop in an hour.

+

Auto-destroy in 7 days.

+

# Nachbesprechung: PodMan, 1. Teil

- Chapter 1, Introduction and Overview of Containers
- Chapter 2, Podman Basics
- Chapter 3, Container Images



# Warum Container - Isolierung

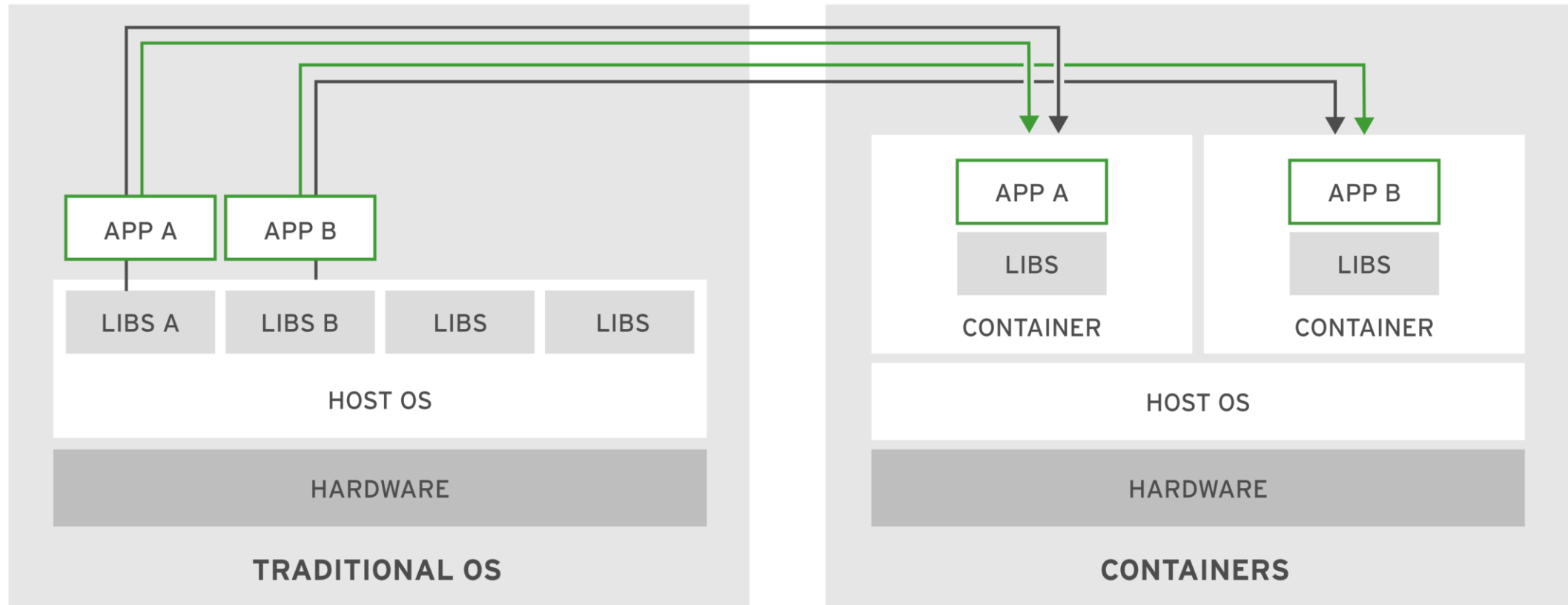
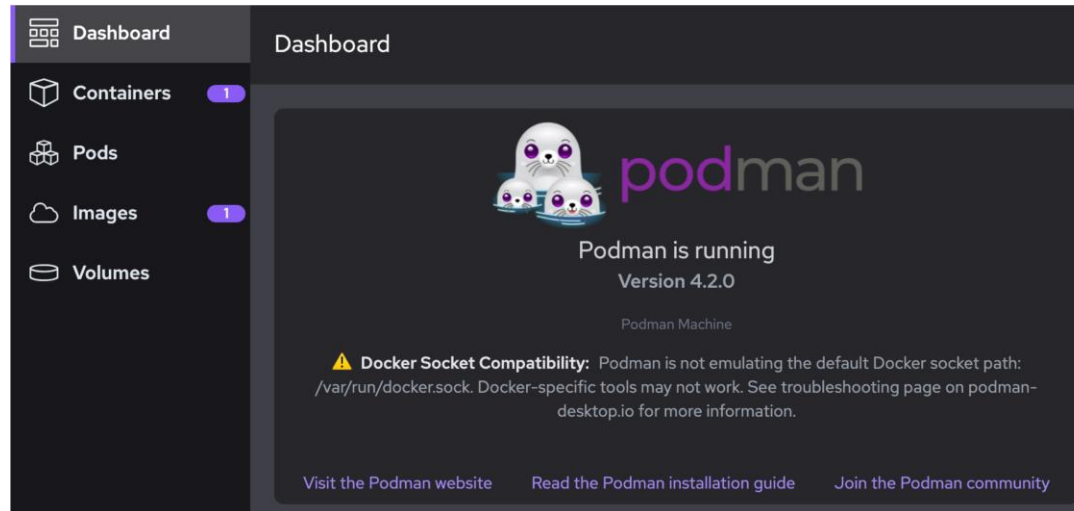


Figure 1.1: Applications in containers versus on host operating system

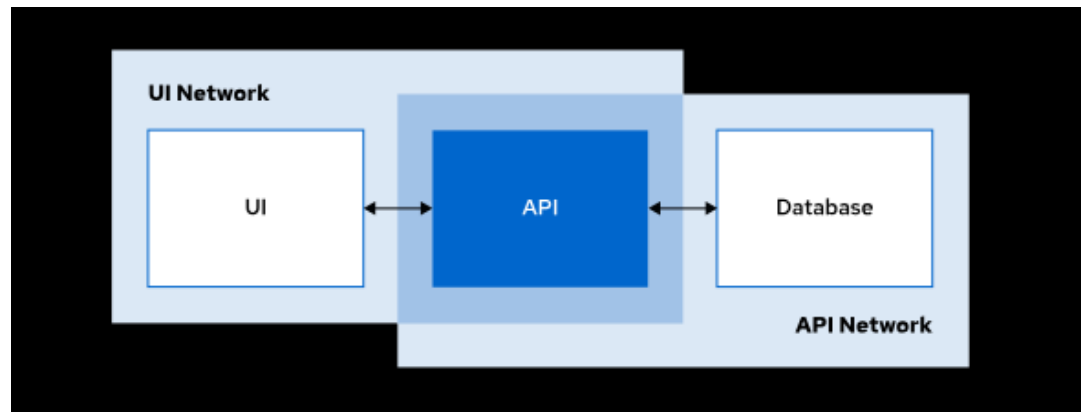
# PodMan Desktop - Verbesserungswürdig



- Verwaltung von Connection via CLI
  - podman system connection list
  - <https://docs.podman.io/en/latest/markdown/podman-system-connection.1.html>
  - Infos im MS Teams

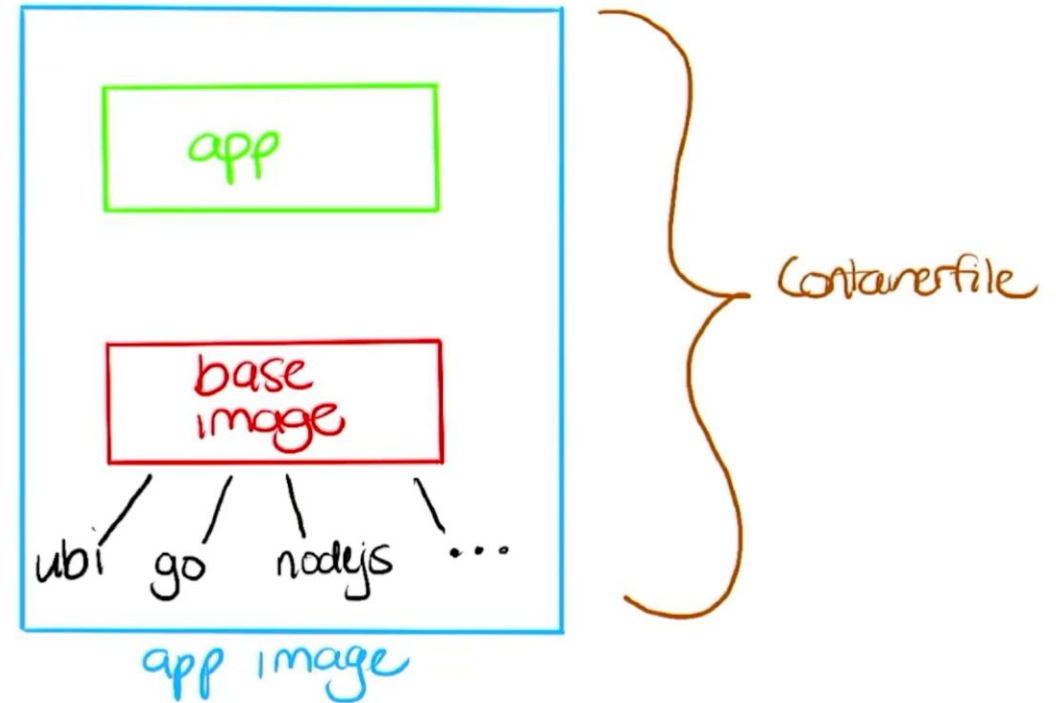
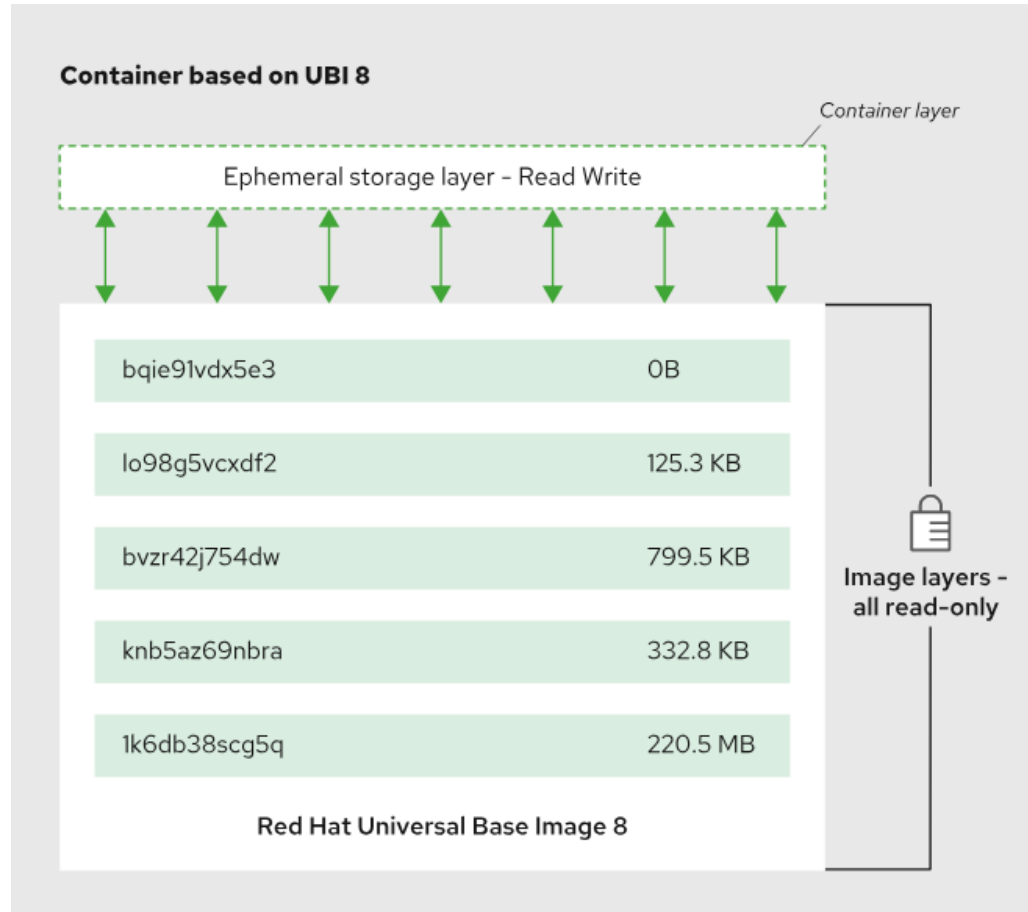


# PodMan - Network



- Isolierung von Containern, z.B.:
  - Mandant A, network a
  - Mandant B, network b
  - Etc.
- Scheint Podman Compose automatisch zu erstellen.
- Beispielapplikation (im Aufbau)
  - <https://gitlab.com/ch-mc-b/autoshop-ms/app/shop#mehrere-instancen>
- Erstellungsprozess via ChatGPT
  - <https://gitlab.com/ch-mc-b/autoshop-ms/edu/chatgpt>

# Container Layer / Images



[Unterrichtsressource A](#): Baut für jede eingesetzte Programmiersprachen ein Base Image: Java, .NET, NodeJS etc.



# Container Namespaces inkl. Tag :xxx

```
[student@workstation ~]$ podman tag registry.ocp4.example.com:8443/developer/images-lab registry.ocp4.example.com:8443/developer/images-lab:grue
[student@workstation ~]$ podman push registry.ocp4.example.com:8443/developer/images-lab
Getting image source signatures
Copying blob f95ee31bf3b7 skipped: already exists
Copying blob 65839b7bf496 skipped: already exists
Copying blob 71b5f690a294 skipped: already exists
Copying blob 335d2a0e4ca8 skipped: already exists
Copying blob 2c9b1d3d1a0a skipped: already exists
Copying config 33be255250 done
Writing manifest to image destination
Storing signatures
[student@workstation ~]$ podman push registry.ocp4.example.com:8443/developer/images-lab:grue
Getting image source signatures
Copying blob 71b5f690a294 skipped: already exists
Copying blob 65839b7bf496 skipped: already exists
Copying blob 2c9b1d3d1a0a skipped: already exists
Copying blob 335d2a0e4ca8 skipped: already exists
Copying blob f95ee31bf3b7 skipped: already exists
Copying config 33be255250 done
Writing manifest to image destination
Storing signatures
```

Ohne Tags funktioniert, Rolling-Update in Kubernetes nicht. D.h. im laufenden Betrieb neue Container ausrollen.



# Zusatzaufgaben - erweitert

## Chapter 2, Podman Basics

---

- Creating Containers with Podman
- Container Networking Basics
- Accessing Containerized Network Services
- Accessing Containers
- Managing the Container Lifecycle

### Zusatzaufgabe:

- Startet Eure erstellten Microservices mit PodMan, wo sind die Unterschiede?
- Erstellt für die Datenbank ein eigenes Netzwerk, warum solltet Ihr das tun?

## Chapter 3, Container Images

---

- Navigate container registries to find and manage container images.
- Navigate container registries.
- Pull and manage container images.

### Zusatzaufgabe:

- Analysiert Eure Container Images mit `podman inspect`. Sind keine sensitiven Daten wie Passwörter ersichtlich?
- Was ist an diesem [Beispiel](#) alles schlecht?
- Welches System verwendet Ihr um Container Images zu taggen.
- Versucht Eure Container Images statt mit `docker build` mit `buildah` zu erstellen.

# Nachbesprechung: PodMan, 2. Teil

- Chapter 4, Custom Container Images
- Chapter 5, Persisting Data



# Linux Signale und Container

```
import signal
import time
import os
import setproctitle

# Setzen des Prozessnamens
setproctitle.setproctitle("non_terminating_script")

# Signalhandler, der nichts tut
def ignore_signals(signum, frame):
    print(f"Erhielt Signal {signum}, aber werde es ignorieren!")

# Registriere den Signalhandler für alle Signale, die der Prozess ignorieren kann
for sig in dir(signal):
    if sig.startswith("SIG") and not sig.startswith("SIG_"):
        try:
            signum = getattr(signal, sig)
            signal.signal(signum, ignore_signals)
        except (OSError, RuntimeError, ValueError):
            # Einige Signale können nicht abgefangen werden, daher diese ignorieren
            pass

print("Starte Endlosschleife. Senden Sie Signale, um zu testen.")
try:
    while True:
        time.sleep(1)
except KeyboardInterrupt:
    print("Erhielt SIGINT (Strg+C), beende Prozess.")
```

- <https://github.com/Yelp/dumb-init>
- Scheint podman und python sauber zu händeln.
- Docker?
- Beispiel: non\_termination\_script
- [https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2\\_Unterrichtsressourcen/C/microservices](https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2_Unterrichtsressourcen/C/microservices)

# Multibuild

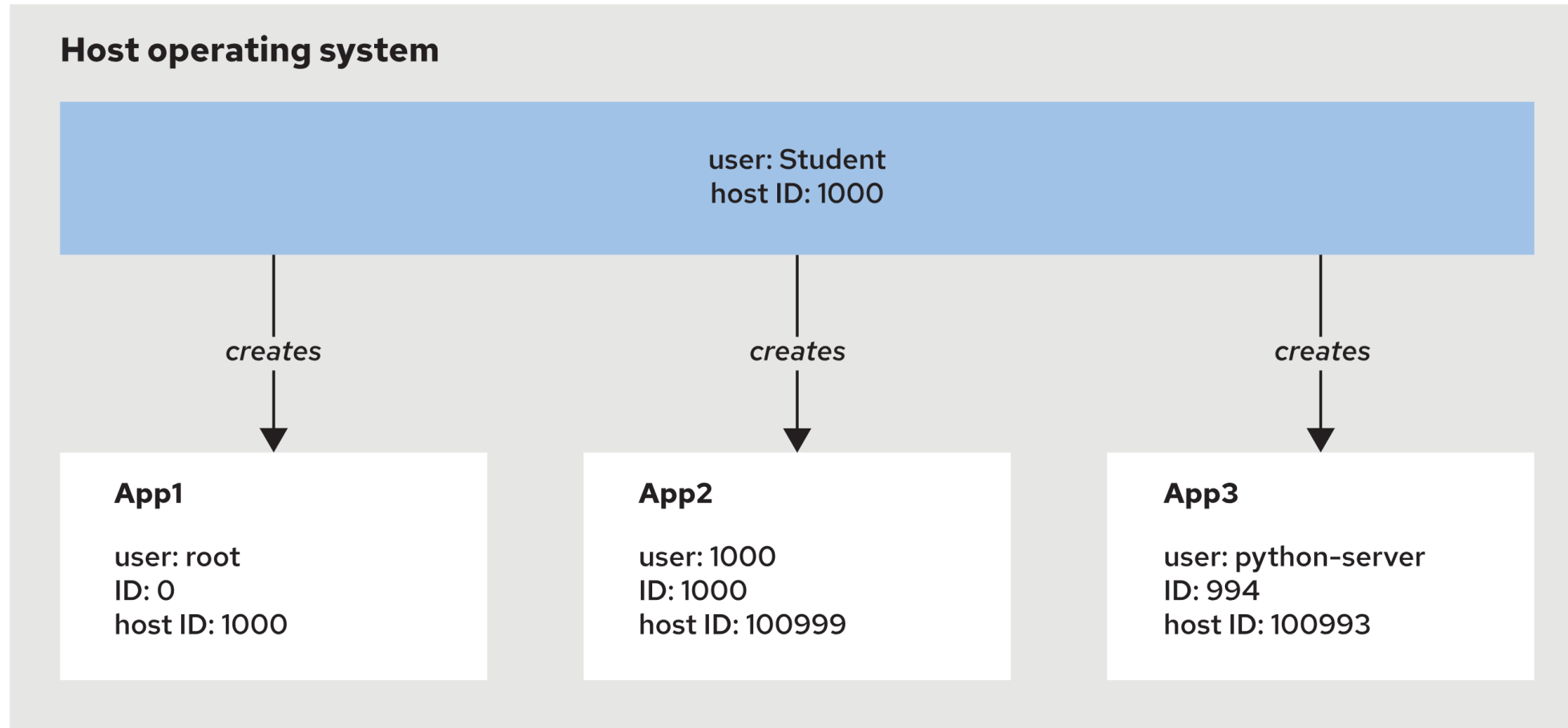
```
# First stage
FROM registry.access.redhat.com/ubi8/nodejs-14:1 as builder ❶
COPY ./ /opt/app-root/src/
RUN npm install
RUN npm run build ❷

# Second stage
FROM registry.access.redhat.com/ubi8/nginx-120 ❸
COPY --from=builder /opt/app-root/src/ /usr/share/nginx/html ❹
```

- ❶ Define the first stage with an alias. The second stage uses the `builder` alias to reference this stage.
- ❷ Build the application.
- ❸ Define the second stage without an alias. It uses the `ubi8/nginx-120` base image to serve the production-ready version of the application.
- ❹ Copy the application files to a directory in the final image. The `--from` flag indicates that Podman copies the files from the `builder` stage.



# User mapping - wichtig beim Schreiben von Dateien auf Filesystem der VM



# Volume

```
import time
from datetime import datetime

# Pfad zur Datei
file_path = "/data/db.txt"

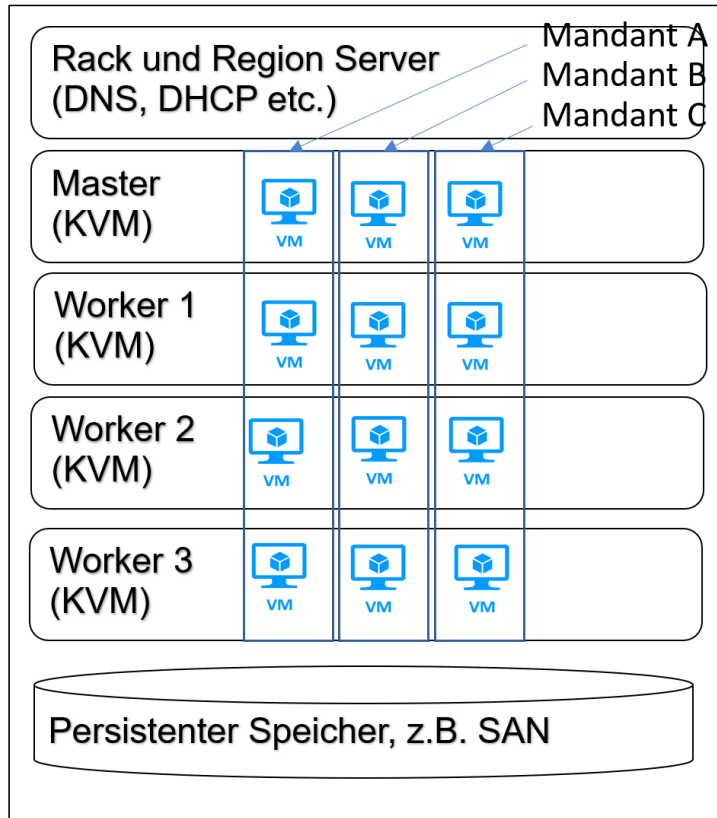
print("Starte das Schreiben in die Datei.")

try:
    while True:
        # Öffne die Datei im Append-Modus und schreibe die aktuelle Zeit hinein
        with open(file_path, "a") as f:
            f.write(f"{datetime.now()}: Eine neue Zeile\n")
        # Warte eine Minute
        time.sleep(60)
except KeyboardInterrupt:
    print("Erhielt SIGINT (Strg+C), beende Prozess.")

writer:
  container_name: writer${WEBSHOP_PREFIX}
  image: registry.gitlab.com/ch-tbz-wb/stud/csec/writer:V2.0
  build:
    context: ./writer
  volumes:
    - ./data:/data
```

- Container läuft in WSL
- Daten werden im Windows Dateisystem gespeichert
- Probleme u.a.
  - Unterschiedliche Handhabung Klein-/Grossbuchstaben
  - Geschwindigkeit (bei Datenbanken)
  - Zugriffsrechte
- Beispiel Writer
  - [https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2\\_Unterrichtsressourcen/C/microservices](https://gitlab.com/ch-tbz-wb/Stud/csec/-/tree/main/2_Unterrichtsressourcen/C/microservices)

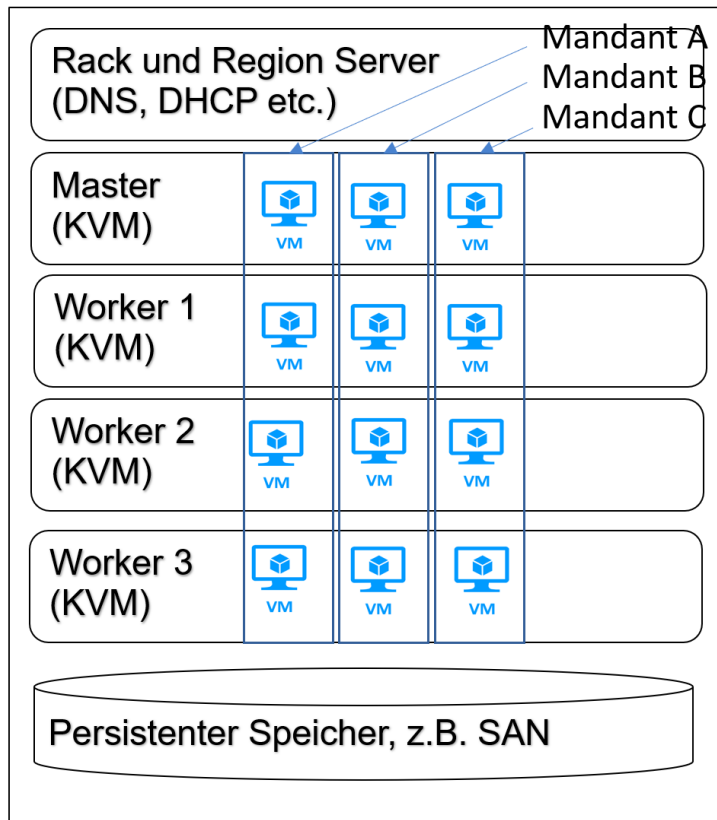
# Volume



17" Rack (Serverschrank)

- Nie Dateien in
  - Containern
  - VMs
- speichern!
- Separate Speicherlösungen, z.B. der Cloud-Anbieter nutzen.

# Volume - Datenbanken



17" Rack (Serverschrank)

- Datenbanken
  - MariaDB kein MySQL (wegen Root Password in Dockerfile)
- Container verwenden ein Copy-on-Write-Dateisystem (COW).
- Bei einigen Workloads, z. B. Datenbanken, kann es mit dem COW-Dateisystem zu Leistungsproblemen kommen.
- Sie können Podman-Volumes verwenden, um die Verwendung des COW-Dateisystems zu vermeiden.



# Zusatzaufgaben - erweitert

## Chapter 4, Custom Container Images

---

- Build custom container images to containerize applications.
- Create a containerfile using basic commands.
- Create a containerfile using best practices.
- Run rootless containers with Podman.

### Zusatzaufgabe:

- Überprüft Eure `Dockerfile`, entsprechen sie den best practices
- Entpackt Eurer Container-Image Layer für Layer und analysiert diese, siehe [hier](#)
- Was macht dieses `Dockerfile` besser als das Beispiel von RedHat? Ab Zeile 80.
- **Selbststudium:** Problem mit Process-ID 1 und Lösung [dumb-init](#).

## Chapter 5, Persisting Data

---

- Run database containers with persistence.
- Describe the process for mounting volumes and common use cases.
- Build containerized databases.

### Zusatzaufgabe:

- Sind Eure Daten nach Zerstören des Containers noch vorhanden?

# Nachbesprechung: PodMan, 3. Teil



- Chapter 6, Troubleshooting Containers
- Chapter 7, Multi-container Applications with Compose
- **Nur Fortgeschrittene (Selbststudium)**
  - Red Hat System Administration II, Kapitel 13, Container als [Systemprozesse](#) eintragen

# inspect (JSON-Notation) und nsenter

To get the container PID you can use the following `podman inspect` command:

```
[user@host ~]$ podman inspect CONTAINER --format '{{.State.Pid}}'  
CONTAINER_PID
```

After getting the container PID, you can run the `nsenter` command. Run the command with elevated privileges by using `sudo`, as follows:

```
[user@host ~]$ sudo nsenter -n -t CONTAINER_PID ss -pant  
Netid State ... Local Address:Port Peer Address:Port Process  
tcp LISTEN ... 0.0.0.0:9091 0.0.0.0:* ...  
...output omitted...
```

# Network

## Container Network Connectivity Issues

Developers commonly use Podman networks to isolate container network traffic from the host. If a container does not attach to a Podman network, then it cannot communicate with other containers on that network.

You can use the `podman inspect` command to verify that every container is using a specific network.

```
[user@host ~]$ podman inspect CONTAINER --format='{{.NetworkSettings.Networks}}'  
map[network_name:0xc000a825a0]
```

When containers communicate by using Podman networks, there is no port mapping involved.



# Debugging, via separaten TCP/IP Port

## Remote Debugging Containers

### Objectives

- Configure a remote debugger during application develop

## Remote Debugging Containers

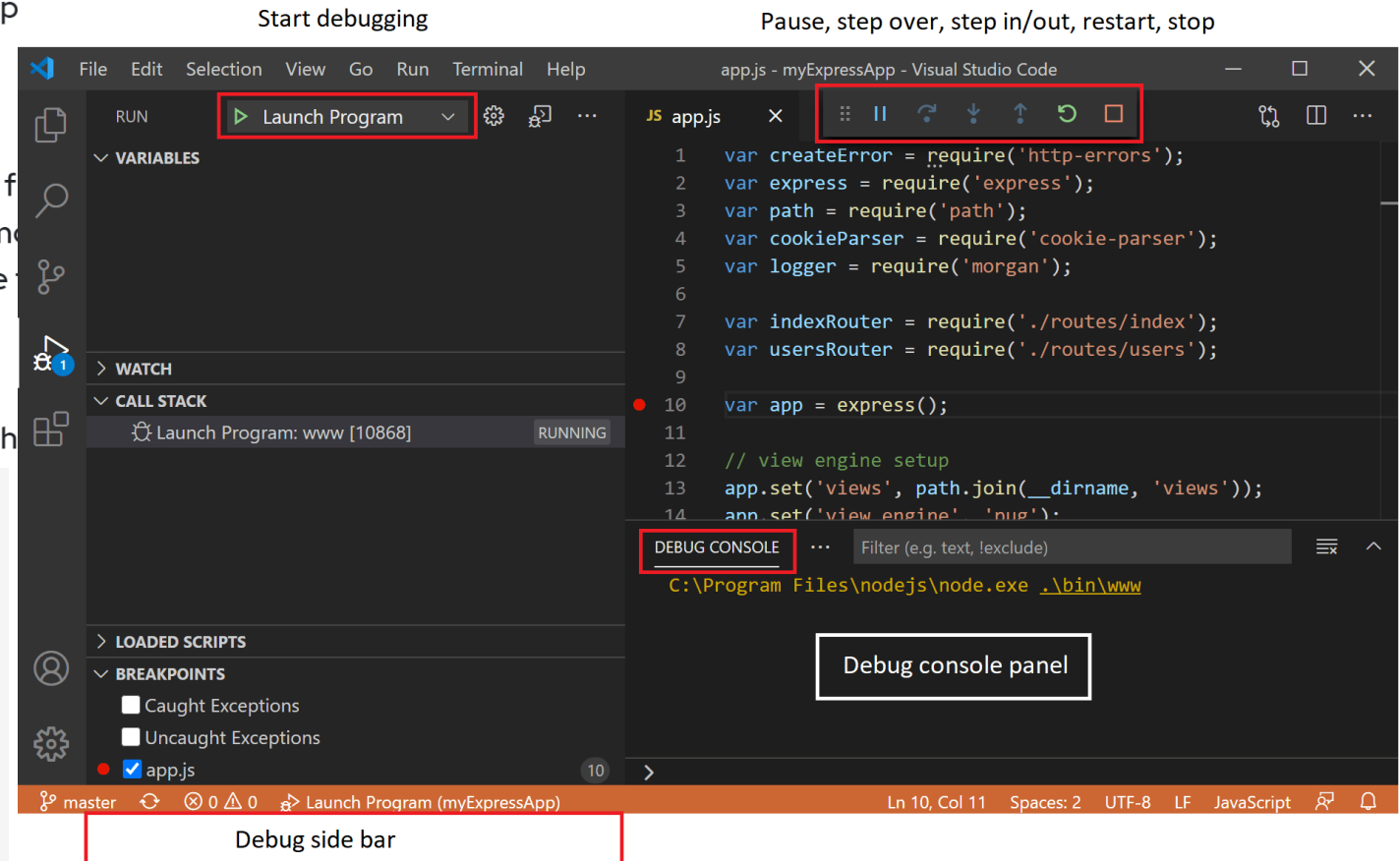
Debugging applications is an important technique to find and fix bugs. To observe runtime variables and behavior, using a debugger is more effective than observing complex interactions, which are more likely to have bugs in the code.

## Debugging Without Containers

Many programming language runtimes provide a way to attach a debugger. The Visual Studio Code debugger provides the following features:

- Attach breakpoints
- Step through individual lines of code
- Inspect and modify variables
- Evaluate custom expressions

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "attach",
      "name": "My Config",
      "port": 9229,
      "address": "localhost",
      "localRoot": "${workspaceFolder}",
      "remoteRoot": "/"
    }
  ]
}
```



# Podman Compose

Consider the following Compose file that declares three containers: a front-end application, a back-end application, and a database.

```
services:
  frontend:
    image: quay.io/example/frontend
    networks: ❶
    - app-net
    ports:
    - "8082:8080"
  backend:
    image: quay.io/example/backend
    networks: ❷
    - app-net
    - db-net
  db:
    image: registry.redhat.io/rhel8/postgresql-13
    environment:
      POSTGRESQL_ADMIN_PASSWORD: redhat
    networks: ❸
    - db-net

networks: ❹
  app-net: {}
  db-net: {}
```

- ❶ The frontend service is part of the app-net network.
- ❷ The backend service is part of the app-net and db-net networks.
- ❸ The db service is part of the db-net.
- ❹ Definition of the networks.

# Container als Systemprozesse

## Anforderungen für systemd-Benutzerservices

Als regulärer Benutzer können Sie einen Service mit dem Befehl `systemctl` aktivieren. Der Service wird gestartet, wenn Sie eine Sitzung (grafische Oberfläche, Textkonsole oder SSH) öffnen, und wird beendet, wenn Sie die letzte Sitzung schließen. Dieses Verhalten unterscheidet sich von dem eines Systemservices, der beim Starten des System gestartet und beim Herunterfahren gestoppt wird.

## Erstellen von systemd-Benutzerdateien für Container

Sie können `systemd`-Services im Verzeichnis `~/.config/systemd/user/` manuell definieren. Die Dateisyntax für Benutzerservices ist dieselbe wie für die Systemservices-Dateien. Weitere Informationen finden Sie in den Manpages `systemd.unit(5)` und `systemd.service(5)`.

Verwenden Sie den Befehl `podman generate systemd`, um `systemd`-Service-Dateien für einen vorhandenen Container zu generieren. Der Befehl `podman generate systemd` verwendet einen Container als Modell zum Erstellen der Konfigurationsdatei.

Sie verwenden den Befehl `podman generate systemd` mit der Option `--name`, um die Servicedatei `systemd` anzulegen, die für den Container `webserver1` modelliert wurde.

```
[appdev-adm@host ~]$ podman generate systemd --name webserver1
...output omitted...
ExecStart=/usr/bin/podman start webserver1 ❶
ExecStop=/usr/bin/podman stop -t 10 webserver1 ❷
ExecStopPost=/usr/bin/podman stop -t 10 webserver1
...output omitted...
```

- ❶ Beim Start führt der `systemd`-Daemon den Befehl `podman start` aus, um einen vorhandenen Container zu starten.
- ❷ Beim Anhalten führt der `systemd`-Daemon den Befehl `podman stop` aus, um den Container anzuhalten. Der Daemon den Container hierbei nicht löscht.

```
ubuntu@m169-20-test:~$ podman generate systemd --name webserver1
# container-webserver1.service
# autogenerated by Podman 4.6.2
# Sat Jun 15 15:33:28 UTC 2024

[Unit]
Description=Podman container-webserver1.service
Documentation=man:podman-generate-systemd(1)
Wants=network-online.target
After=network-online.target
RequiresMountsFor=/run/user/1000/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman start webserver1
ExecStop=/usr/bin/podman stop \
-t 10 webserver1
ExecStopPost=/usr/bin/podman stop \
-t 10 webserver1
PIDFile=/run/user/1000/containers/overlay-containers/2cbd7740ff709
Type=forking

[Install]
WantedBy=default.target
```

# Zusammenfassung Sicherheit

- User und Gruppe in Dockerfile setzen, Kapitel B, Folie 8
- Ressourcen beschränken (Memory, CPU), Kapitel B, Folie 3
- Schwachstellen Scanner, Kapitel B, Folie 6 verwenden, z.B. [trivy](#)
- Bei Verwendung CI/CD: [SaST](#) aktivieren
- Netzwerksicherheit, Eigene Netzwerke verwenden, Folie 9
- Container Tags verwenden, Folie 12
- Signalhandling beachten, <https://github.com/Yelp/dumb-init>
- Logfiles weiterleiten <https://github.com/kubernetes/ingress-nginx/blob/main/rootfs/Dockerfile>
- [setproctitle](#) - verwenden für korrekten Prozess-Namen in Linux
- Für Monitoring (Prometheus), [/metrics](#) implementieren.
- Klein-/Grossschreibung berücksichtigen. Problem Windows/Linux siehe [writer](#)
- TCP/IP Port 5000 wird auch von der Docker Registry verwendet, 8080 verwenden
- [.dockerignore](#) verwenden
- ADD berücksichtigt auch gepackte Dateien und entpackt diese zuerst. Wo möglich COPY verwenden.
- **Fortgeschritten:** verwende Security Profile [seccomp](#).
- Und es ist nicht falsch, den Code ChatGPT zum Überprüfen zu geben.

# Nachbesprechung: Generell



- Wie findet Ihr RedHat Academy
  - Inhaltlich
  - Lerneffekt
  - Level der Übungen
  - Sollen wir es mehr verwenden?
- *Mein Fazit*
  - *Gute Repetition von technischer Seite*
  - *Vertiefung des Verständnisses von Containern*

# Praktische Arbeit

- Verbessert die Security Eurer Container Images für die Semesterarbeit

