Sichere Webanwendungen

Lerneinheit 2: Einführung in Docker

Prof. Dr. Christoph Karg

Studiengang Informatik
Hochschule Aalen

Sommersemester 2025

6.4.2025

Gliederung

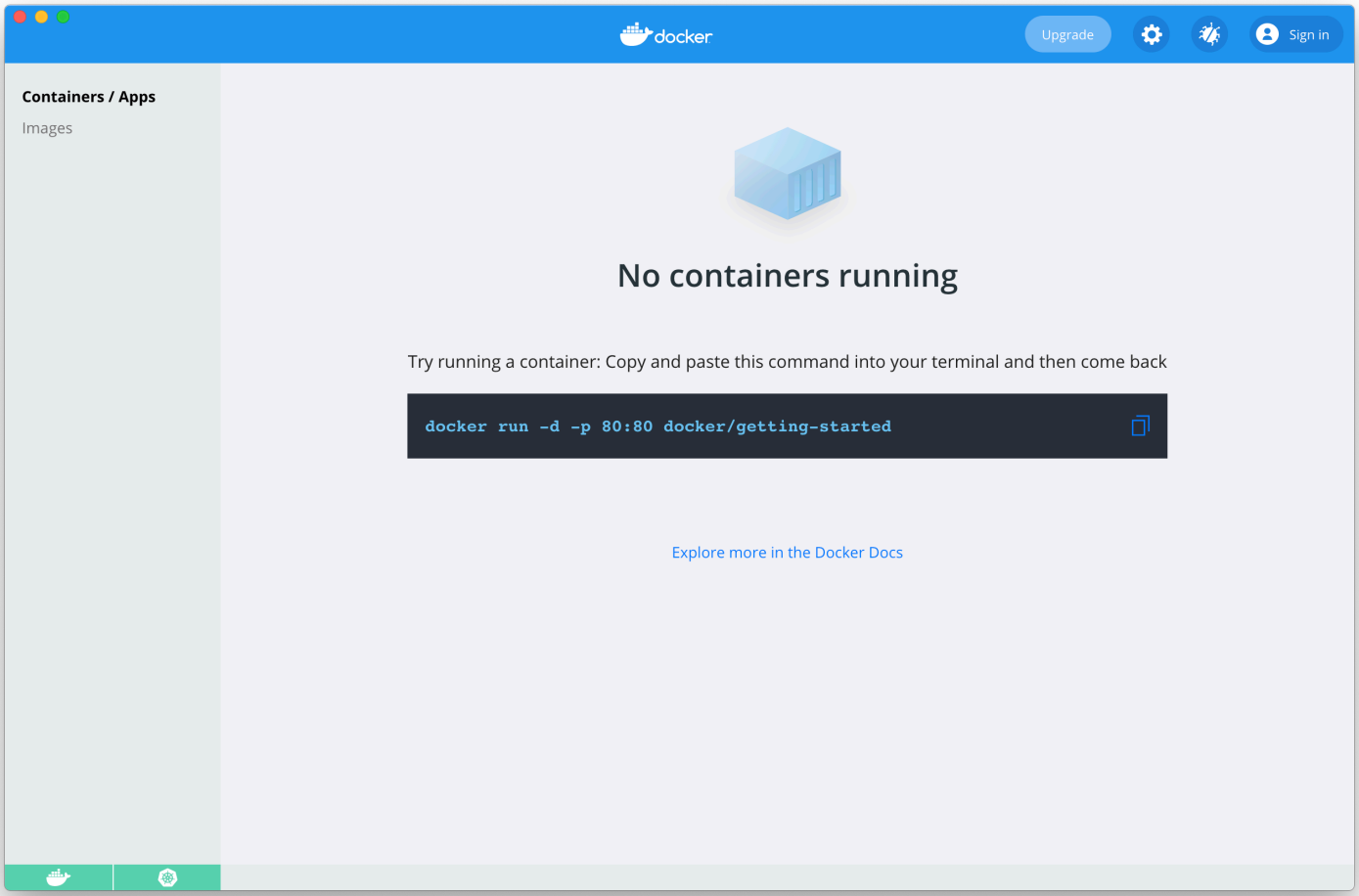
1. Grundlagen
2. Beispiele für Docker-Images
3. Docker-Compose
4. Sicherheitsaspekte

Literaturempfehlung: [Sto20; ÖK21]

Docker

- Docker ist eine Open Source Lösung für die isolierte Ausführung einer Anwendung innerhalb eines Containers.
- Das Docker-Projekt wurde 2013 gegründet und ist mittlerweile der De-facto-Standard für Containervirtualisierung.
- Docker basiert auf Linux-Technologien wie *Cgroups* und *Namespaces*.
- Docker lief zunächst nur unter Linux, wurde dann aber auch auf Windows und MacOS portiert.
- Die Software kann über die Webpage `https://www.docker.com` bezogen werden.
- Der Docker Hub ist eine Plattform zum Vertrieb vorgefertigter Container (`https://hub.docker.com`).

Docker Dashboard



Docker-Dashboard unter MacOS

Erste Schritte mit Docker

- Die Arbeit mit Docker erfolgt in der Regel über ein Terminal des Betriebssystems.
- Der Befehl `docker` ist das zentrale Werkzeug.
- Mit dem folgenden Aufruf wird eine Hilfeseite ausgegeben:

```
> docker --help
```

- Informationen über die installierte Version erhält man mit:

```
> docker version
```

- Informationen über den aktuellen Zustands des Docker-Systems frägt man ab mit:

```
> docker system info
```

Docker sagt: „Hello World!“

Befehl:

```
> docker run -it debian echo "Hello World\!"
```

Bemerkungen:

- Der Befehl lädt aus dem Docker-Hub das aktuellste Debian-Image herunter.
- Der Befehl startet das Debian-Image in eine Container und führt den echo-Befehl aus.
- Das Image wird für den zukünftigen Einsatz auf dem Computer zwischengespeichert.

Ein interaktiver Container

Befehl:

```
> docker run -it debian /bin/bash
```

Bemerkungen:

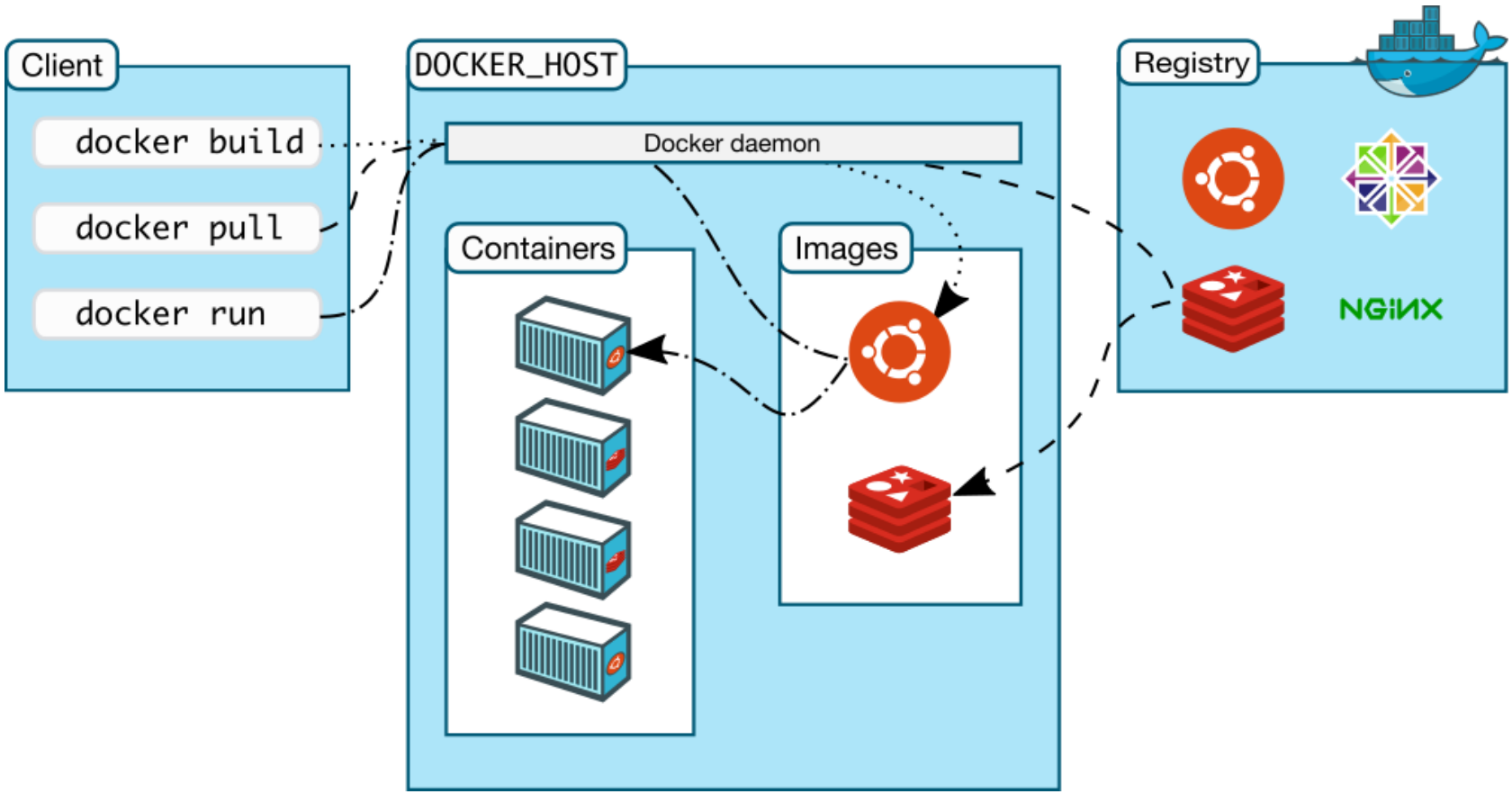
- Nach Beendigung der Session wird der Container gestoppt, aber nicht gelöscht.
- Mit dem Befehl `docker start -ai <container>` wird der Container neu gestartet.
- Der Befehl `docker rm <container>` löscht einen gestoppten Container.

Unterbefehle von `docker` (Auswahl)

- `build` ⇝ Erstellen von Docker-Images
- `run` ⇝ Ausführen von Containern
- `inspect` ⇝ Inspektion von Docker-Objekten
- `ps` ⇝ Abfrage des Zustands aktiver Container
- `info` ⇝ Abfrage von systemweiten Informationen
- `rm` ⇝ Löschen von Containern
- `rmi` ⇝ Löschen von Docker-Images

Weitere Details findet man im CLI Cheat Sheet [Doc22a].

Docker Architektur



Quelle: Docker

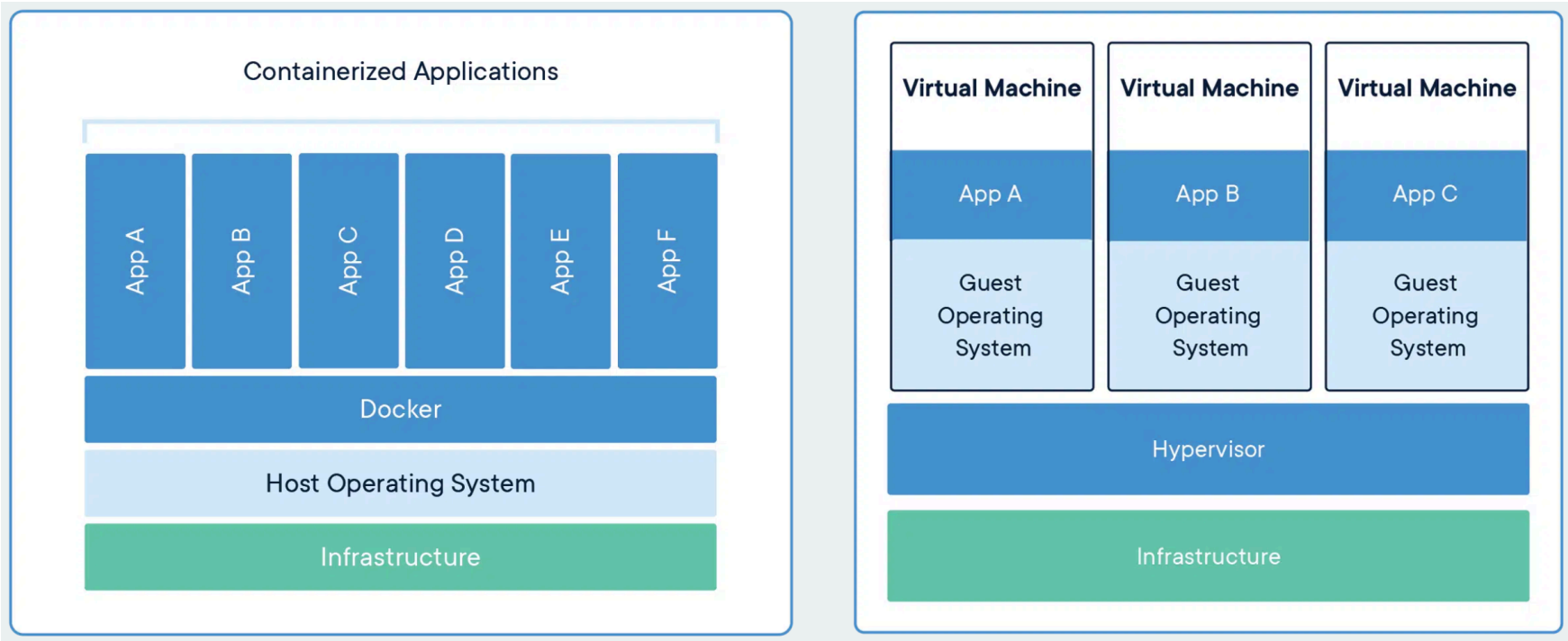
Komponenten der Docker Architektur

- Docker Daemon:
  - bildet das Backend des Docker-Systems
  - verwaltet Docker Objekte wie Container und Images
  - wird über API-Zugriffe gesteuert
- Docker Client:
  - Werkzeug zur Arbeit mit dem Docker-System
  - Bereitstellung als Konsolenprogramm (`docker`)
- Docker Desktop:
  - Anwendung für Linux, MacOS oder Windows
  - beinhaltet den Docker Daemon und Client sowie diverse andere Werkzeuge wie Docker Compose

Komponenten der Docker Architektur (Forts.)

- Image:
  - Vorlage zur Erstellung eines Containers
  - Grundlage für weitere Images (z.B. Debian oder Ubuntu Image)
- Container:
  - Ausführbare Instanz eines Images
  - Verwaltung über die Docker API
  - keine persistente Speicherung der Daten
- Registry:
  - Ort zur Speicherung von Images
  - Beispiel: Docker Hub

Container versus Virtuelle Maschine



Container versus Virtuelle Maschine (Forts.)

Container:

- Objekt zur Bereitstellung einer Anwendung
- Nutzung des Betriebssystems des Host-Computers
- Ein Host kann mehrere Container ausführen
- Ressourcenschonender im Vergleich zu Virtuellen Maschinen

Virtuelle Maschine:

- Objekt zur Virtualisierung einer physikalischen Hardware
- Betrieb über einen Hypervisor
- Ein Server kann mehrere virtuelle Maschinen ausführen
- Administrationsaufwand einer VM vergleichbar mit der eines „echten“ Servers

Für weitere Details siehe [Doc22b]

Erstellung eines Docker-Images

- Der Befehl `docker build` dient zur Erstellung von Docker-Images.
- Die Datei `Dockerfile` enthält die Befehle zur Erstellung des Images.
- Unter [Doc24b] findet man eine detaillierte Dokumentation zur Erstellung von Dockerfiles.
- Docker gibt zahlreiche Empfehlungen zur Erstellung von Dockerfiles [Doc24c].

Wichtige Schlüsselwörter

- `FROM` ⇝ Auswahl des Basis-Images
- `ADD` ⇝ Kopieren von Dateien in das Image
- `COPY` ⇝ Kopieren von Dateien in das Image
- `RUN` ⇝ Ausführen eines Befehls innerhalb des Images
- `CMD` ⇝ Ausführen eines Befehls beim Starten des Containers
- `ENTRYPOINT` ⇝ Ausführen eines Befehls beim Starten des Containers
- `USER` ⇝ Festlegung des aktuellen Benutzers
- `WORKDIR` ⇝ Festlegung des aktuellen Arbeitsverzeichnisses
- `EXPOSE` ⇝ Freigabe eines Netzwerk-Ports
- `VOLUME` ⇝ Freigabe von Verzeichnissen

Unterschied zwischen `ADD` und `COPY`

- Mit `COPY` kann man ausschließlich lokale Dateien in das Image kopieren.
- `ADD` ist eine Erweiterung von `COPY`.
- Erhält `ADD` ein Verzeichnis, dann wird der komplette Inhalt des Verzeichnisses in das Image kopiert.
- `ADD` ist in der Lage, Archive innerhalb des Images zu extrahieren.
- `ADD` kann Dateien aus dem Internet herunterladen und anschließend in das Image kopieren.
- Empfehlung: Der Befehl `ADD` sollte nur benutzt werden, wenn dessen erweiterte Funktionalität benötigt wird.

Die Befehle `CMD` und `ENTRYPOINT`

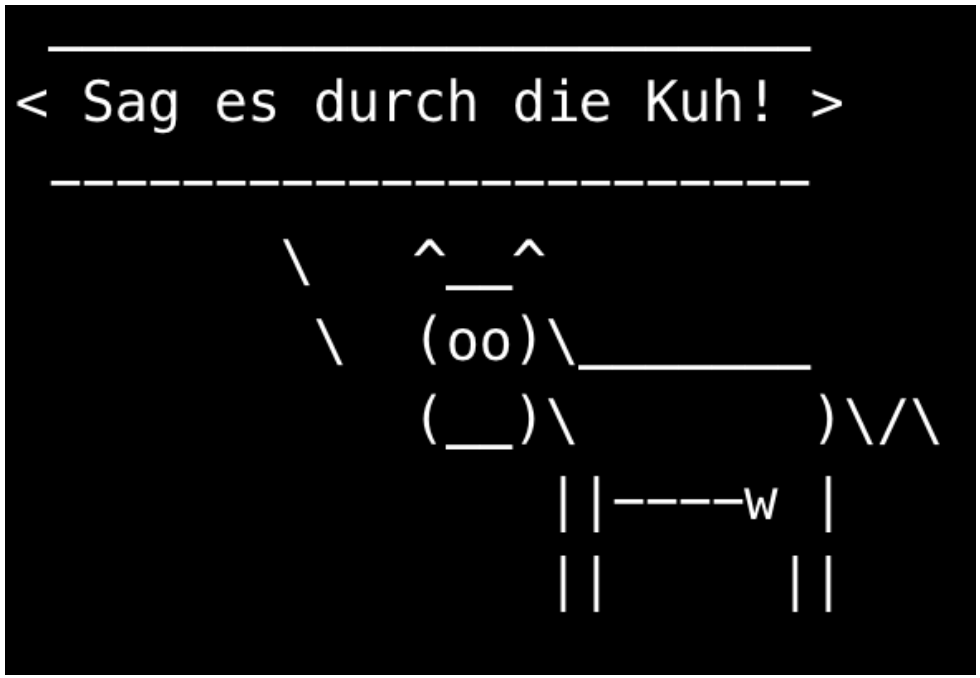
- Mit `CMD` und `ENTRYPOINT` wird der Befehl festgelegt, der beim Start des Containers ausgeführt wird.
- Der mit `CMD` festgelegte Befehl kann beim Start des Containers über die Kommandozeile ersetzt werden.
- Kommandozeilenparameter werden beim Start des Containers an den `ENTRYPOINT` übergeben.
- Will man einen mittels `ENTRYPOINT` festgelegten Befehl ersetzen, dann ist die Option `--entrypoint` beim Start des Containers erforderlich.
- Mit `docker exec` kann man in einem bereits laufenden Container einen weiteren Befehl starten.

Beispiele für den Einsatz von Docker

- Cowsay (ein einfaches Image)
- Git-Server mit Gitolite3

Beispiel: Cowsay

Aufgabe: Erstellen eines Docker-Images für cowsay



Beispiel: Cowsay (Forts.)

Dockerfile:

```
1 FROM debian
2 RUN apt-get update
3 RUN apt-get install -y fortune cowsay
4 COPY entrypoint.sh /
5 RUN chmod +x entrypoint.sh
6 ENTRYPOINT [ "/entrypoint.sh" ]
```

Warnung: Dieses Dockerfile ist nicht optimal!

Beispiel: Cowsay (Forts.)

Skript entrypoint.sh:

```
1 #! /bin/bash
2 if [ $# -eq 0 ]
3 then
4     /usr/games/fortune | /usr/games/cowsay
5 else
6     /usr/games/cowsay "$@"
7 fi
```

Bemerkungen:

- Das Skript erzeugt bei fehlender Nachricht mittels `fortune` einen zufälligen Spruch.
- Das Skript muss im selben Verzeichnis wie das Dockerfile gespeichert werden.

Beispiel: Cowsay (Forts.)

Erstellen des Images:

```
> docker build -t cowsay .
```

Ausführen des Containers:

```
> docker run --rm cowsaw "Sag es durch die Kuh\!"
```

Analyse des Images:

```
> docker image inspect cowsay
> docker history cowsay
```

Beispiel: Cowsay (Forts.)

Bemerkungen

- Jeder Befehl im Dockerfile erzeugt im Image eine Schicht (Layer).
- Docker nutzt einen Cache zur Zwischenspeicherung der einzelnen Schichten.
- Eine Schicht wird nur neu erstellt, wenn dies erforderlich ist.
- Mit dem Befehl `docker system prune` werden alle Images, Container und der Inhalt des Cache gelöscht.

Beispiel: Cowsay (Forts.)

Überarbeitetes Dockerfile:

```
1 FROM debian
2 RUN apt-get update && \
3     apt-get install -y fortune cowsay
4 COPY entrypoint.sh /
5 RUN chmod +x entrypoint.sh
6 ENTRYPOINT [ "/entrypoint.sh" ]
```

Bemerkungen:

- Das Update des Debian-Basissystems sowie die Installation der zusätzlichen Pakete wird mit einem RUN-Befehl erzeugt.
- Vorteile:
  - Es wird nur eine Schicht erzeugt.
  - Bei Änderung der zu installierenden Pakete wird zwingend vorher ein Update der Paketliste durchgeführt.

Beispiel: Git-Server mit Gitolite3

Aufgabe: Erstellen eines Git-Servers auf Basis von Gitolite3

Anforderungen:

- Als Betriebssystem kommt Debian zum Einsatz.
- Das Verzeichnis für gitolite3 soll über ein Volume bereitgestellt werden.
- Der Zugriff auf den Git-Server erfolgt über die Secure Shell SSH.

Beispiel: Git-Server mit Gitolite3 (Forts.)

Dockerfile:

```
1  FROM debian:latest
2
3  RUN apt update && \
4      apt -y upgrade && \
5      DEBIAN_FRONTEND='noninteractive' \
6      apt -y install openssh-server sudo vim locales gitolite3
7
8  RUN echo 'de_DE.UTF-8 UTF-8' >> /etc/locale.gen
9  RUN locale-gen
10
11 RUN groupadd -g 110 gitolite3
12 RUN useradd -rm -d /var/lib/gitolite3 -u 110 -g gitolite3 \
13     -c "Gitolite Admin" -s /bin/bash gitolite3
14 COPY admin.pub /gitolite3.pub
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

```
15 RUN service ssh start
16 EXPOSE 22
17
18 VOLUME /var/lib/gitolite3
19 VOLUME /etc/gitolite3
20
21 COPY entrypoint.sh /
22 RUN chmod +x /entrypoint.sh
23 CMD [ "/entrypoint.sh" ]
24 #ENTRYPOINT [ "/entrypoint.sh" ]
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

Bemerkungen:

- In den Zeilen 8 bis 9 wird die Unterstützung für das deutsche Gebietsschema konfiguriert.
- In den Zeilen 11 bis 14 wird ein Nutzer für Gitolite3 angelegt und ein SSH-Schlüssel für die Administration hinterlegt.
- In Zeilen 15 bis 16 wird SSH Service gestartet und der SSH-Port für den externen Zugriff freigegeben.
- In Zeilen 18 bis 19 werden zwei Volumes für die Git-Daten freigegeben.
- Beim Start des Containers wird ein Skript ausgeführt, welches den SSH-Dämon startet (Zeilen 21 bis 24).

Beispiel: Git-Server mit Gitolite3 (Forts.)

Inhalt von `entrypoint.sh`:

```
1  #! /bin/bash
2
3  if [ -d "/etc/gitolite3/" ];
4  then
5      chown -R gitolite3.gitolite3 /etc/gitolite3
6  fi
7
8  if [ -d "/var/lib/gitolite3/" ];
9  then
10     chown -R gitolite3.gitolite3 /var/lib/gitolite3
11 fi
12
13 /usr/sbin/sshd -D
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

Skript zum Starten des Containers:

```
#! /bin/bash

GIT_DIR=./data
GIT_DIR_VAR=${GIT_DIR}/var
GIT_DIR_ETC=${GIT_DIR}/etc

docker run --rm -d -p 6000:22 \
  --volume ${GIT_DIR_VAR}:/var/lib/gitolite3 \
  --volume $GIT_DIR_ETC}:/etc/gitolite3 \
  gitolite
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

Skript zum Starten einer Bash im Container:

```
1  #! /bin/bash
2
3  GIT_DIR=./data
4  GIT_DIR_VAR=${GIT_DIR}/var
5  GIT_DIR_ETC=${GIT_DIR}/etc
6
7  docker run --rm -it -p 6000:22 \
8    --mount type=bind,source=${GIT_DIR_ETC},target=/etc/gitolite3 \
9    --mount type=bind,source=${GIT_DIR_VAR},target=/var/lib/gitolite3 \
10   gitolite /bin/bash
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

Achtung: Das Image ist prinzipiell einsatzfähig, aber vor dem ersten Zugriff auf Gitolite3 muss das Verzeichnis initialisiert werden.

Inbetriebnahme:

1. Starten des Containers: $\rightsquigarrow$ Skript `gitolite.sh`
2. Starten einer Shell auf dem Container:

```
local> docker exec -it <container_id> /bin/bash
```

3. Rekonfiguration von Gitolite3:

```
container> dpkg-reconfigure gitolite3
```

   Parameter:

   - User: `gitolite3`
   - Pfad: `/var/lib/gitolite3`
   - Public Key: `/gitolite3.pub`

Zugriff auf Gitolite

1. Eintrag in der SSH-Config Datei:

```
Host dockergit
    Hostname localhost
    Port 6000
    User gitolite3
    IdentityFile <homedir>/.ssh/id_rsa
```

2. Test des Zugriffs:

```
> ssh dockergit
```

3. Klonen des Admin-Repositories:

```
> git clone dockergit:gitolite-admin ./Gitolite
```

Beispiel: Git-Server mit Gitolite3 (Forts.)

Bemerkungen

- Je nach Anwendung muss die UID und GID des Nutzers im Image mit den Berechtigungen des eingebundenen Verzeichnisses übereinstimmen.
- Beim Kopieren der Dateien in ein Image muss darauf geachtet werden, dass die Zugriffsberechtigungen passen.
- Will man das Verhalten eines Containers untersuchen, muss gegebenenfalls eine Logging-Funktionalität installiert werden.
- Zu Diagnosezwecken kann ein Dienst im Debug-Modus gestartet werden.

Docker Compose

- Docker Compose ist ein Werkzeug zur Arbeit mit mehreren Containern, gemeinsam eine verteilte Applikation bilden [Doc24d].
- Der Befehl `docker-compose` dient zur Erstellung der Images und dem Betrieb der Applikation.
- Der Aufbau der Images sowie deren Vernetzung wird in einer Konfiguration in YAML-Syntax [BEN09] beschrieben.
- Docker Compose Projekte können auch als Docker Stack betrieben werden.

Beispiel: Wordpress

Aufgabe: Erstellen einer Webanwendung bestehend aus:

- Wordpress Image (Docker Hub Link)
- MariaDB Image (Docker Hub Link)

Umsetzung in zwei Schritten:

- Erstellung eines einfachen Docker Compose Projekts
- Erweiterung durch die Einbindung von externen Laufwerken

Beispiel: Wordpress (Forts.)

Datei `wordpress-v1.yml`:

```
1  services:
2
3    wordpress:
4      image: wordpress
5      restart: always
6      ports:
7        - 8080:80
8      environment:
9        WORDPRESS_DB_HOST: db
10       WORDPRESS_DB_USER: exampleuser
11       WORDPRESS_DB_PASSWORD: examplepass
12       WORDPRESS_DB_NAME: exampledb
13     volumes:
14       - wordpress:/var/www/html
```

Beispiel: Wordpress (Forts.)

```
16   db:
17     image: mariadb
18     restart: always
19     environment:
20       MYSQL_DATABASE: exampledb
21       MYSQL_USER: exampleuser
22       MYSQL_PASSWORD: examplepass
23       MYSQL_RANDOM_ROOT_PASSWORD: '1'
24     volumes:
25       - db:/var/lib/mysql
```

Beispiel: Wordpress (Forts.)

```
26 volumes:
27   wordpress:
28     driver: local
29     driver_opts:
30       type: none
31       device: "data/www"
32       o: bind
33
34   db:
35     driver: local
36     driver_opts:
37       type: none
38       device: "data/db"
39       o: bind
```

Beispiel: Wordpress (Forts.)

Bemerkungen:

- Die Konfiguration enthält mit `services`, `secrets` und `volumes` zwei Abschnitte.
- Im Abschnitt `services` werden die Images der in der Anwendung enthaltenen Services definiert.
- Im Abschnitt `secrets` werden vertrauliche Informationen (Passwörter, Zertifikate, ...) spezifiziert.
- Der Abschnitt `volumes` legt die Datenspeicher fest. Die Konfiguration erfolgt später.
- Die Konfiguration der in einem Image enthaltenen Software erfolgt über Umgebungsvariablen.

Beispiel: Wordpress (Forts.)

Inbetriebnahme:

```
> docker compose -f wordpress-v1.yml up -d
```

Bemerkungen:

- Mit dem Parameter `-f` wird die Konfigurationsdatei übergeben. Per Default wird `docker-compose.yml` ausgelesen.
- Die Volumes werden innerhalb der virtuellen Docker-Umgebung erstellt.

Beispiel: Wordpress (Forts.)

Anzeige der Prozesse:

```
> docker compose -f wordpress-v1.yml ps
```

Anzeige der Prozesse:

```
> docker compose -f wordpress-v1.yml ps
```

Anzeige der Logging-Ausgaben:

```
> docker compose -f wordpress-v1.yml logs
```

Zugriff auf den Wordpress Container:

```
> docker compose -f wordpress-v1.yml exec wordpress /bin/bash
```

Beispiel: Wordpress (Forts.)

Anzeige der vorhandenen Volumes:

```
> docker volume ls
```

Inspektion eines Volumes:

```
> docker volume inspect wordpress_db
```

Anzeige der Auslastung der Anwendung:

```
> docker compose -f wordpress-v1.yml top
```

Beenden der Anwendung:

```
> docker compose -f wordpress-v1.yml down
```

Beispiel: Wordpress (Forts.)

Datei `wordpress-v2.yml`:

```
1  services:
2
3    wordpress:
4      image: wordpress
5      restart: always
6      ports:
7        - 8080:80
8      environment:
9        WORDPRESS_DB_HOST: db
10       WORDPRESS_DB_NAME: wordpressdb
11       WORDPRESS_DB_USER: wordpress
12       WORDPRESS_DB_PASSWORD: /run/secrets/db_password
13     volumes:
14       - wordpress:/var/www/html
15     secrets:
16       - db_password
```

Beispiel: Wordpress (Forts.)

```
17  db:
18    image: mariadb
19    restart: always
20    environment:
21      MYSQL_DATABASE: wordpressdb
22      MYSQL_USER: wordpress
23      MYSQL_PASSWORD: /run/secrets/db_password
24      MYSQL_RANDOM_ROOT_PASSWORD: '1'
25    volumes:
26      - db:/var/lib/mysql
27    secrets:
28      - db_password
```

Beispiel: Wordpress (Forts.)

```
29 secrets:
30   db_password:
31     file: db_password.txt
32
33 volumes:
34   wordpress:
35     driver: local
36     driver_opts:
37       type: none
38       device: ./data/www
39       o: bind
40
41   db:
42     driver: local
43     driver_opts:
44       type: none
45       device: ./data/db
46       o: bind
```

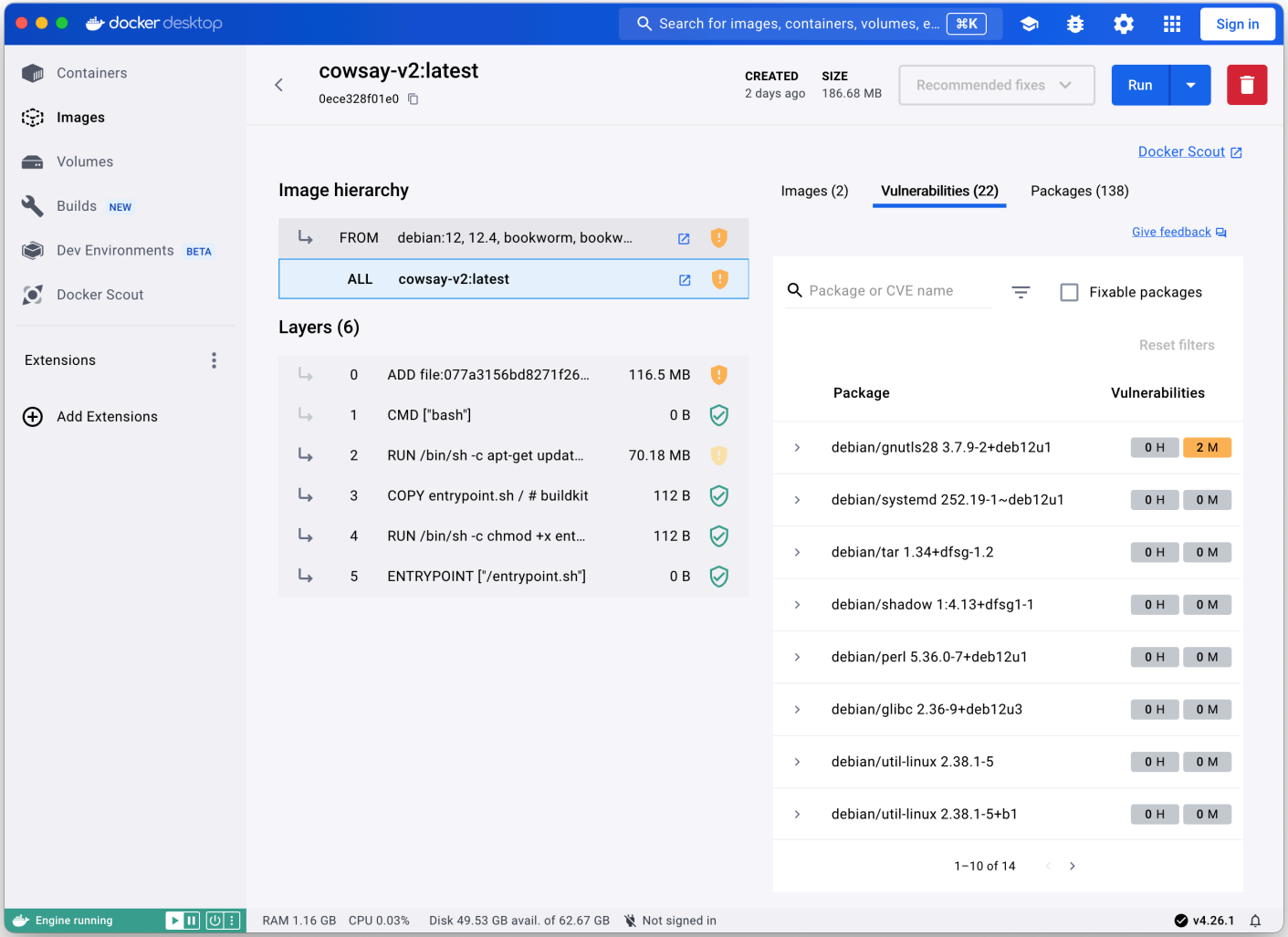
Sicherheitsaspekte

- Docker Security Dokumentation [Doc24a]
- Docker Security Best Practice [Doc24e]
- OWASP Docker Top 10 [OWA21]
- OWASP Docker Cheat Sheet [OWA24]

Docker Security Dokumentation

- Docker liefert in seiner Online-Dokumentation zahlreiche Tipps zum Thema Sicherheit.
- Unterscheidung:
  - Empfehlungen für Administratoren
  - Empfehlungen für Entwickler
- Im Docker Desktop ist mit Scout ein Tool zum Schwachstellen-Scan von Images integriert.

Docker Scout



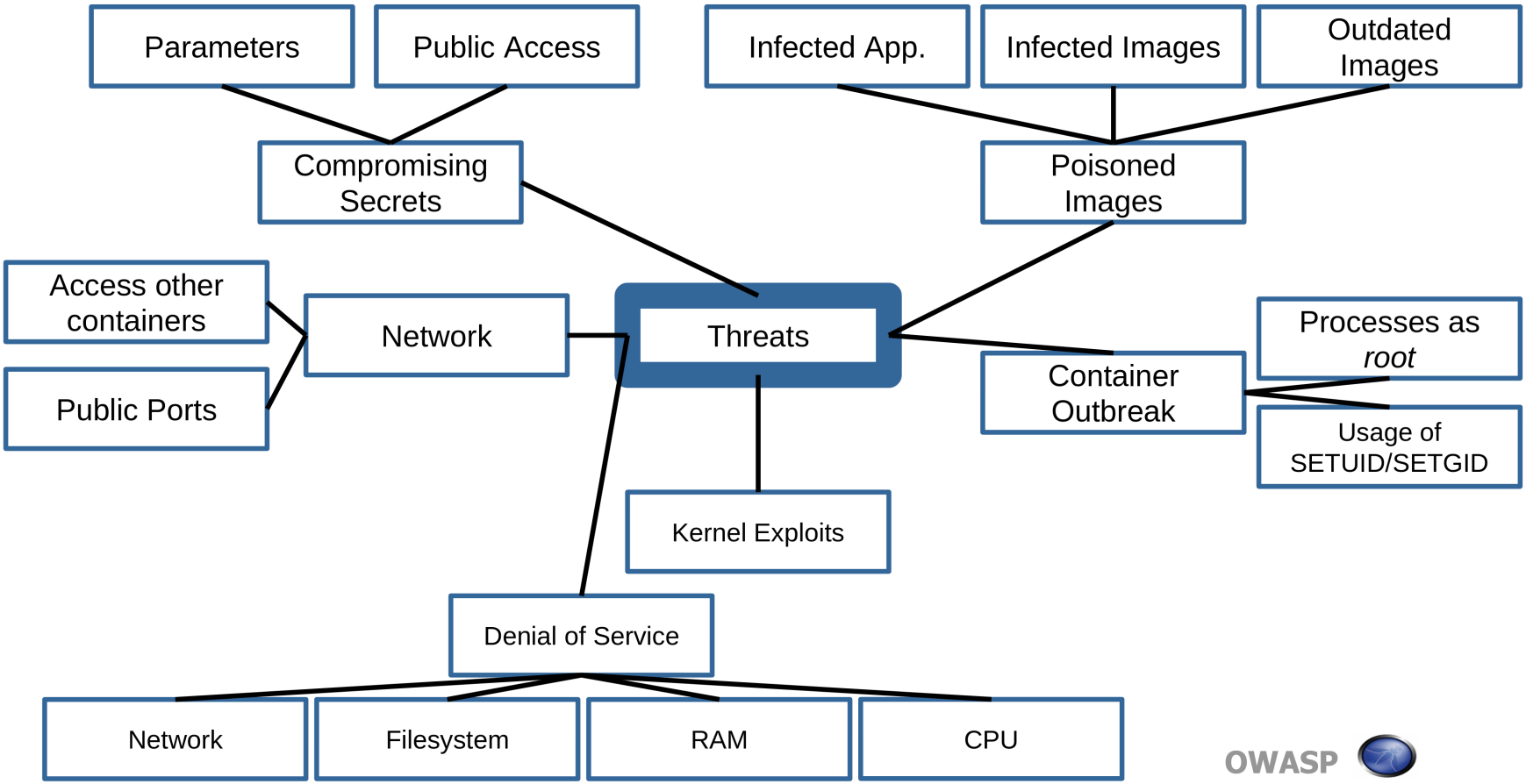
Docker Security Best Practices

1. Auswahl des richtigen Basis-Images
   - Image aus den offiziellen oder von verifizierten Quellen verwenden
   - Je „schlanker“ das Image, desto besser!
2. Einsatz von Multi-Stage-Builds
   - Ziel: Erstellung eines einfach zu wartenden Dockerfiles
3. Regelmäßiger Rebuild von Images
   - Ziel: Aktualisierung der Pakete im Image, um Schwachstellen zu beseitigen
   - Wichtig: Cache vor dem Rebuild leeren
4. Überprüfung der Images auf Schwachstellen

OWASP Docker Top 10

- Eine Arbeitsgruppe der OWASP hat gängige Bedrohungen im Zusammenhang mit Docker analysiert.
- Die Dokumente werden über GITHUB bereit gestellt.
- Die Top 10 zeigt typische Bedrohungen und entsprechende Gegenmaßnahmen auf.
- Die Dokumente sind seit längerer Zeit nicht aktualisiert worden.

Threat Modelling



Bedrohungen

1. Ausbruch aus dem Container und Zugriff auf den Host
2. Netzwerk-Zugriff auf andere Container
3. Angriff auf das Orchestrierungstool über das Netzwerk
4. Angriff auf den Host über das Netzwerk
5. Angriff auf andere Ressourcen über das Netzwerk
6. Hoher Verbrauch von Ressourcen (z.B. CPU, RAM, ...)
7. Kompromittierung des Hosts
8. Beeinträchtigung der Integrität der Images

Docker Top 10

- D01: Secure User Mapping
- D02: Patch Management Strategy
- D03: Network Segmentation and Firewalling
- D04: Secure Defaults and Hardening
- D05: Maintain Security Contexts
- D06: Protect Secrets
- D07: Resource Protection
- D08: Container Image Integrity and Origin
- D09: Follow Immutable Paradigm
- D10: Logging

Docker Cheat Sheet Rules

0. Keep Host and Docker up to date
1. Do not expose the Docker daemon socket (even to the containers)
2. Set a user
3. Limit capabilities (Grant only specific capabilities, needed by a container)
4. Add `-no-new-privileges` flag
5. Disable inter-container communication

Docker Cheat Sheet Rules (Forts.)

6. Use Linux Security Module (seccomp, AppArmor, or SELinux)
7. Limit resources (memory, CPU, file descriptors, processes, restarts)
8. Set filesystem and volumes to read-only
9. Use static analysis tools
10. Set the logging level to at least INFO
11. Lint the Dockerfile at build time
12. Run Docker in root-less mode

Zusammenfassung

- Docker ist eine leistungsfähige Plattform für Container-Virtualisierung.
- Die Virtualisierung erfolgt auf Basis von Linux.
- Mittels Docker kann man maßgeschneiderte Container für viele Anwendungsszenarien erstellen.
- Docker Compose ermöglicht die Erstellung von Applikationen, die sich aus mehreren Containern zusammensetzen.
- Es gibt zahlreiche Empfehlungen, um die Sicherheit von Docker zu verbessern.

Literatur I

[BEN09] Oren Ben-Kiki, Clark Evans und Ingy döt Net. *YAML Ain't Markup Language (YAML) Version 1.2*. 3. Aufl. 2009. URL: https://yaml.org/spec/1.2/spec.html.

[Doc22a] Docker, Hrsg. *Dockerfile CLI cheat sheet*. 2022. URL: https://docs.docker.com/get-started/docker_cheatsheet.pdf (besucht am 16. 01. 2024).

[Doc22b] Docker, Hrsg. *Use containers to Build, Share and Run your applications*. 2022. URL: https://www.docker.com/resources/what-container/ (besucht am 16. 01. 2024).

Literatur II

[Doc24a] Docker, Hrsg. *Docker Security*. 2024. URL: https://docs.docker.com/security/ (besucht am 19. 01. 2024).

[Doc24b] Docker, Hrsg. *Dockerfile reference*. 2024. URL: https://docs.docker.com/engine/reference/builder/ (besucht am 16. 01. 2024).

[Doc24c] Docker, Hrsg. *Overview of best practices for writing Dockerfiles*. 2024. URL: https://docs.docker.com/develop/develop-images/dockerfile_best-practices/ (besucht am 16. 01. 2024).

Literatur III

[Doc24d] Docker, Hrsg. *Overview of Docker Compose*. 2024. URL: https://docs.docker.com/compose/ (besucht am 16. 01. 2024).

[Doc24e] Docker, Hrsg. *Security best practices*. 2024. URL: https://docs.docker.com/develop/security-best-practices/ (besucht am 16. 01. 2024).

[ÖK21] Bernd Öggl und Michael Kofler. *Docker. Das Praxisbuch für Entwickler und DevOps-Teams*. Rheinwerk Computing, 2021.

[OWA21] OWASP, Hrsg. *OWASP Docker Top 10*. Open Worldwide Application Security Project (OWASP). 7. Juni 2021. URL: https://owasp.org/www-project-docker-top-10/ (besucht am 19. 01. 2024).

Literatur IV

[OWA24] OWASP, Hrsg. *OWASP Docker Security Cheat Sheet*. Open Worldwide Application Security Project (OWASP). 9. Jan. 2024. URL: https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html (besucht am 19. 01. 2024).

[Sto20] Elton Stoneman. *Learn Docker in a Month of Lunches*. Manning, 2020.