

Die Folien sind für den persönlichen Gebrauch im Rahmen des Moduls gedacht. Eine Veröffentlichung oder Weiterverteilung an Dritte ist nicht gestattet. (G. Neugebauer)



# Development, Security and Operations (DevSecOps)

## (Bachelor Wahlfach, Modul 55803)

Wintersemester 2022/2023

### **Kapitel 4: Continuous Integration/ Delivery/Deployment & Container-Sicherheit**

Prof. Dr. Georg Neugebauer

*Lehrgebiet IT-Sicherheit*

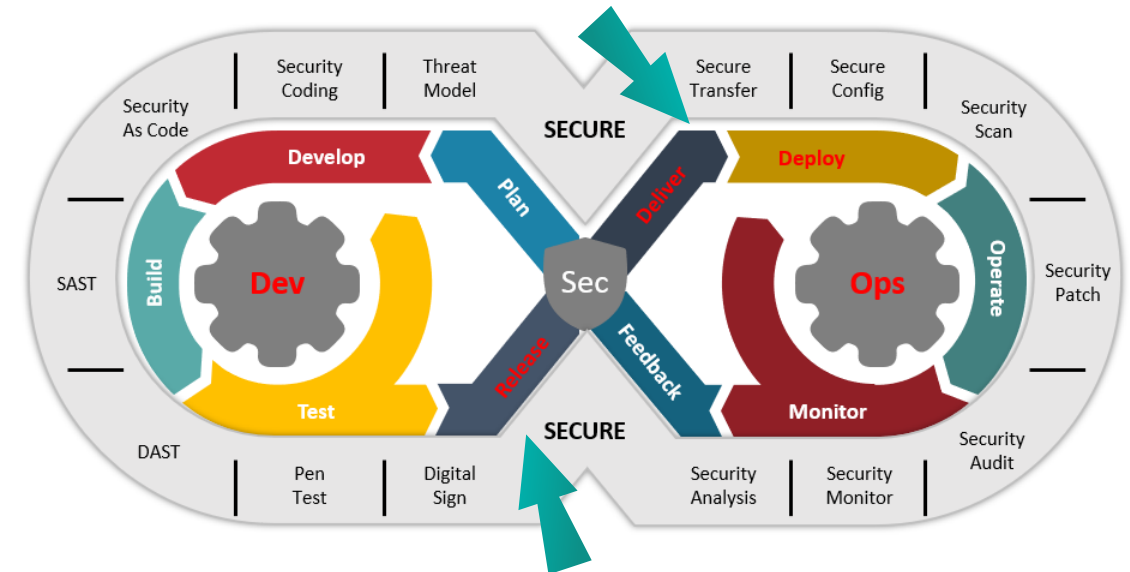
*Fachbereich 5 – Elektrotechnik und Informationstechnik*

*FH Aachen*

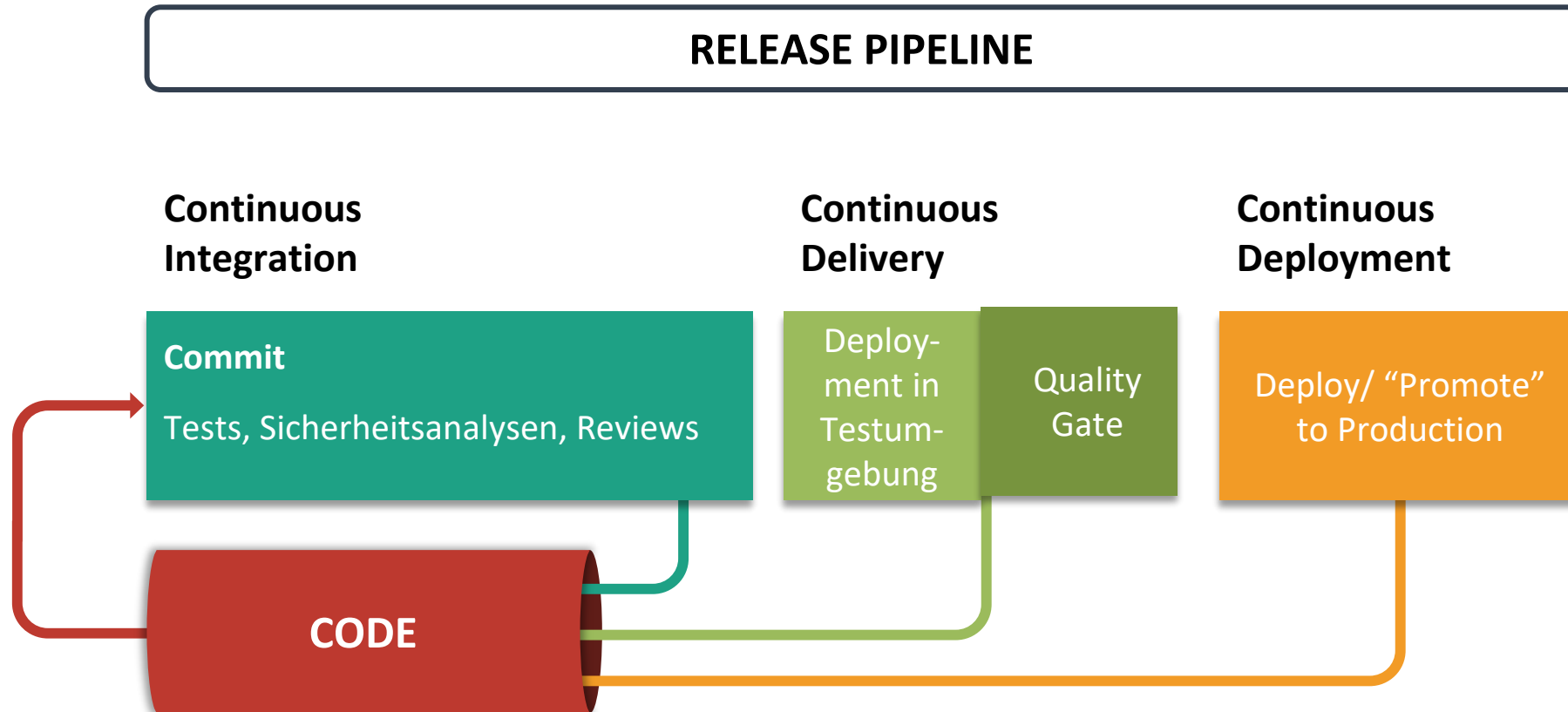
*Email: [g.neugebauer@fh-aachen.de](mailto:g.neugebauer@fh-aachen.de)*

# CI/CD & Container-Sicherheit im DevSecOps-Modell

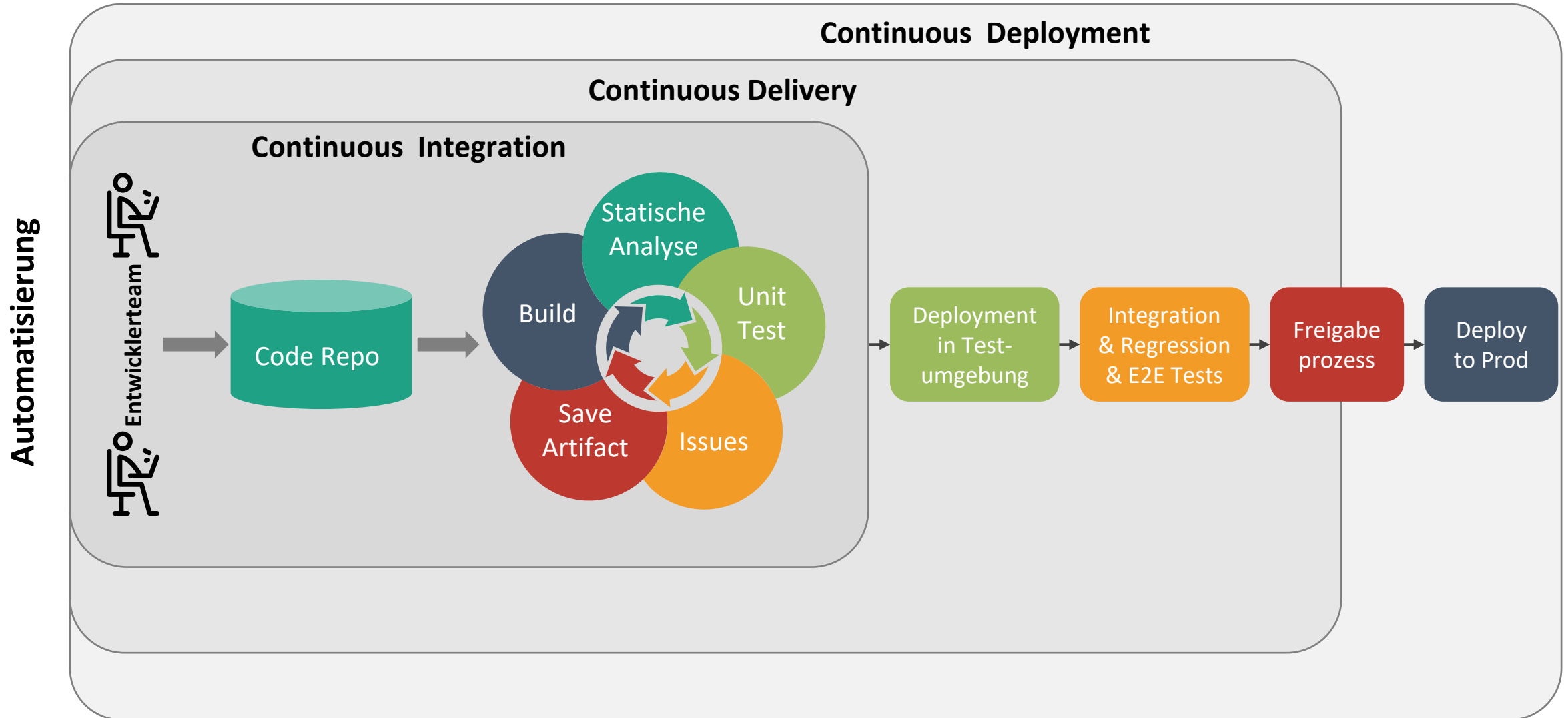
- Hauptziele des CI/CD
  - Automatisierungsframework aller DevSecOps-Komponenten
  - Wichtiges Bindeglied zwischen Entwicklung (**Dev**) und Betrieb (**Ops**)
- Hauptziele der Container-Sicherheit
  - Ausführung von Anwendungen in Container-Deployments in der Cloud sind sehr verbreitet (Skalierbarkeit, Ausfallsicherheit)
  - Sichere Konfiguration & Deployment von Container-Anwendungen ist sehr wichtig



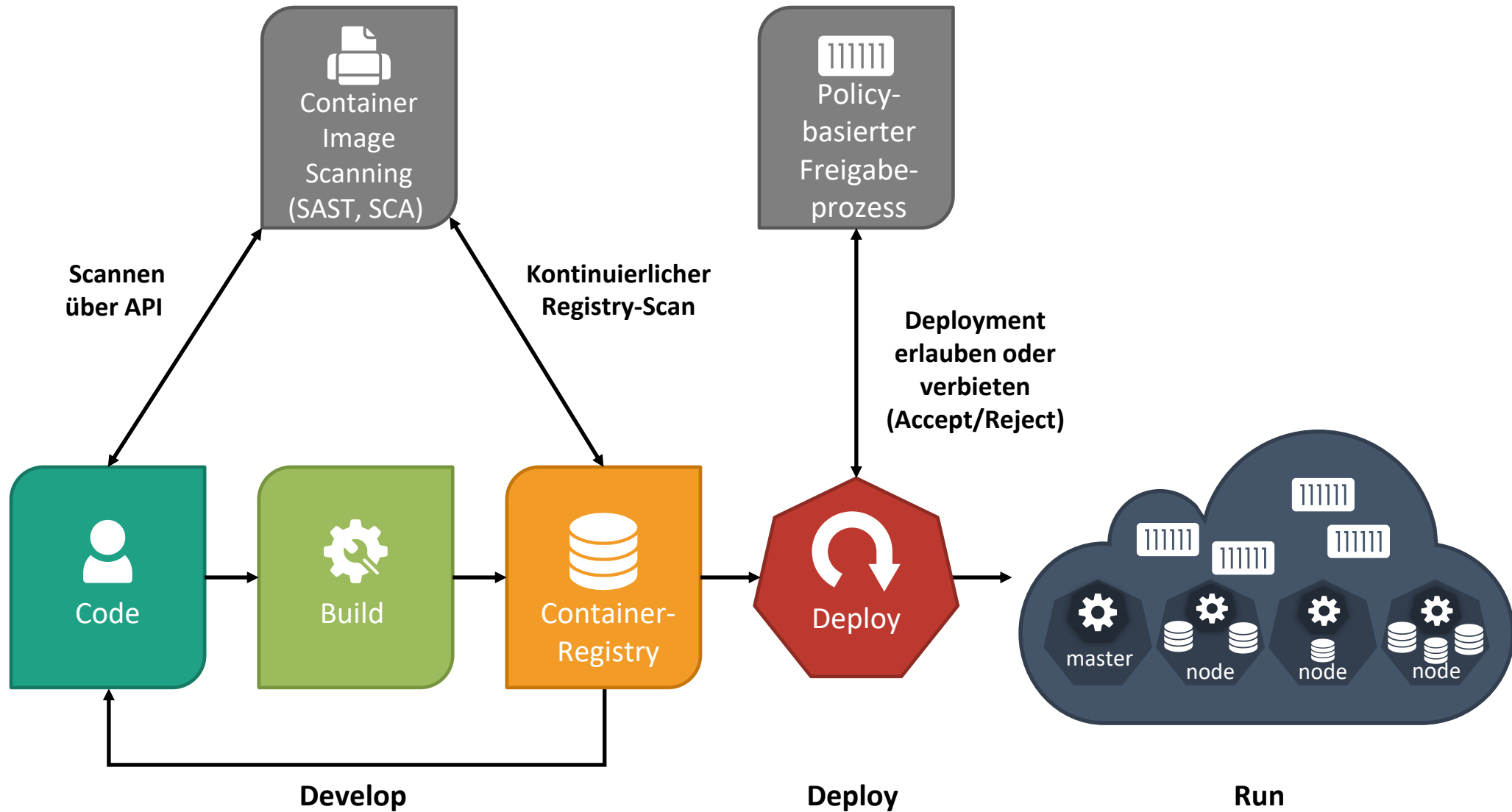
# CI/CD – High-Level View



# Continuous Integration vs Continuous Delivery vs Continuous Deployment

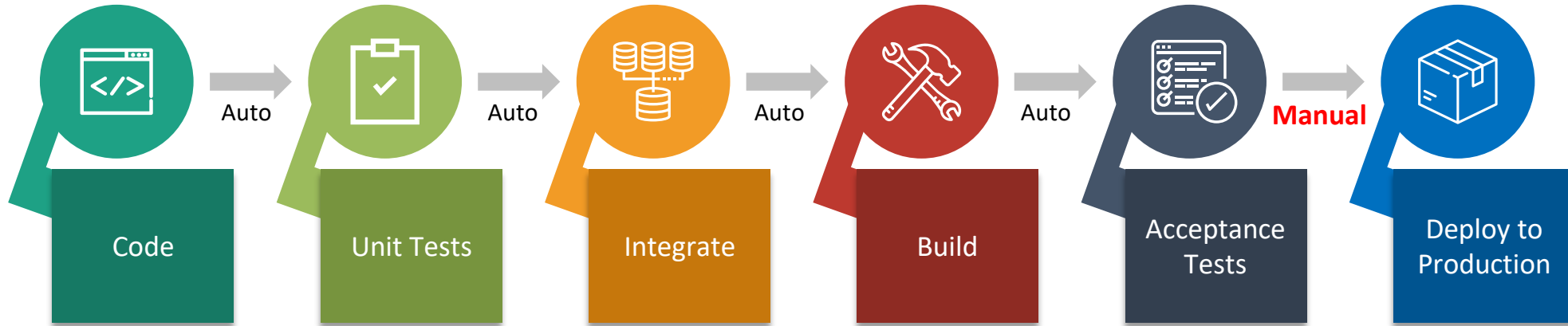


# Container-Deployment

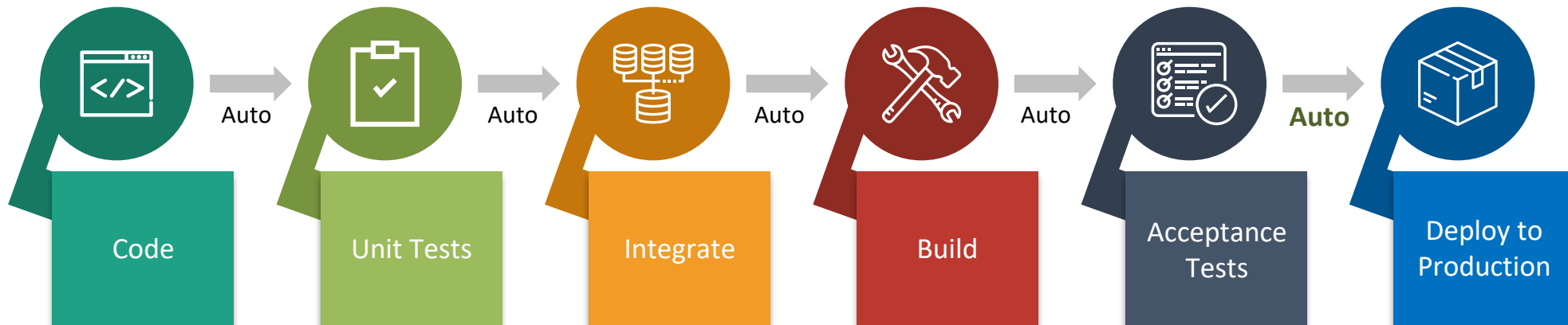


# CI/CD – Automatisierung

## CONTINUOUS DELIVERY



## CONTINUOUS DEPLOYMENT



# Application Deployment - Herausforderungen

01

## Verlorene Einnahmen

Fehler im Produktivsystem bei der Bereitstellung führen zu Roll-Backs und ungeplanten Ausfallzeiten von geschäftskritischen Anwendungen.

02

## Kundenunzufriedenheit

Probleme beim Sicherstellen, dass die Testumgebung mit der Produktivumgebung übereinstimmt, führt zu spürbaren Qualitätsproblemen beim Kunden.

03

## Hohe wiederkehrende Arbeitslast

Die manuelle Konfiguration komplexer Anwendungen beim Deployment führt zu einer ineffizienten Nutzung von DevSecOps-Ressourcen.

04

## Sanktionen, Firmenwertverlust

Das Fehlen eines überprüfbaren Deploymentprozesses erschwert die Einhaltung von externer Vorschriften und Gesetzen.

# Application Deployment - Strategien



## Recreate

Wenn Softwareversion A beendet ist, wird Version B ausgerollt.



## Ramped

Die Softwareversion B wird langsam ausgerollt und ersetzt Softwareversion A.



## Blue/Green

Die Softwareversion B wird parallel zu Softwareversion A ausgerollt. Im Anschluss wird der Netzwerkverkehr auf B umgeleitet.



## Shadow

Softwareversion B erhält ebenfalls den Produktiv-Netzwerkverkehr neben Softwareversion A, aber Antworten kommen von A.



## A/B Testing

Softwareversion B wird für eine Untergruppe von Benutzern unter bestimmten Bedingungen freigegeben.



## Canary

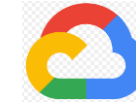
Ähnlich wie Blue/Green, aber Softwareversion B wird für eine Untergruppe von Benutzern freigegeben, danach erfolgt das vollständige Rollout.



# Motivation – Cloud-Deployments und IT-Sicherheit

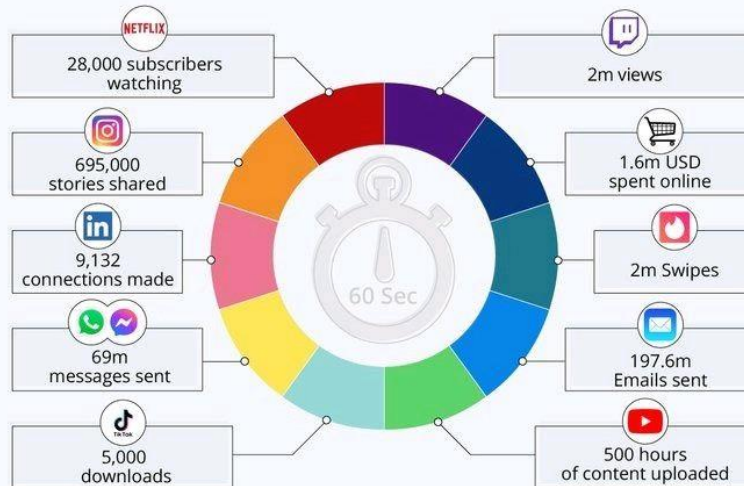
There is no cloud – It's just someone else's computer

**und noch schlimmer** mit unbekannten laufenden Diensten und einer großen Anzahl von Verbindungen und Schnittstellen zu anderen Cloud-Systemen.



## A Minute on the Internet in 2021

Estimated amount of data created on the internet in one minute

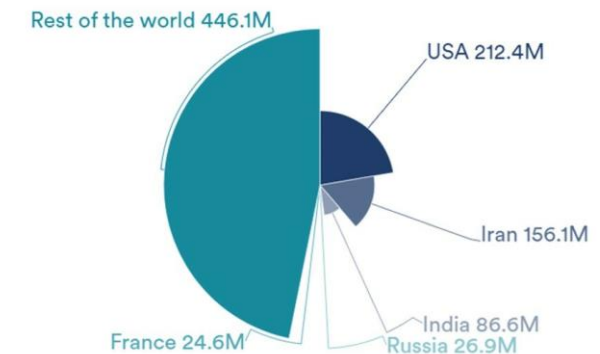


Source: Lori Lewis via AllAccess



statista

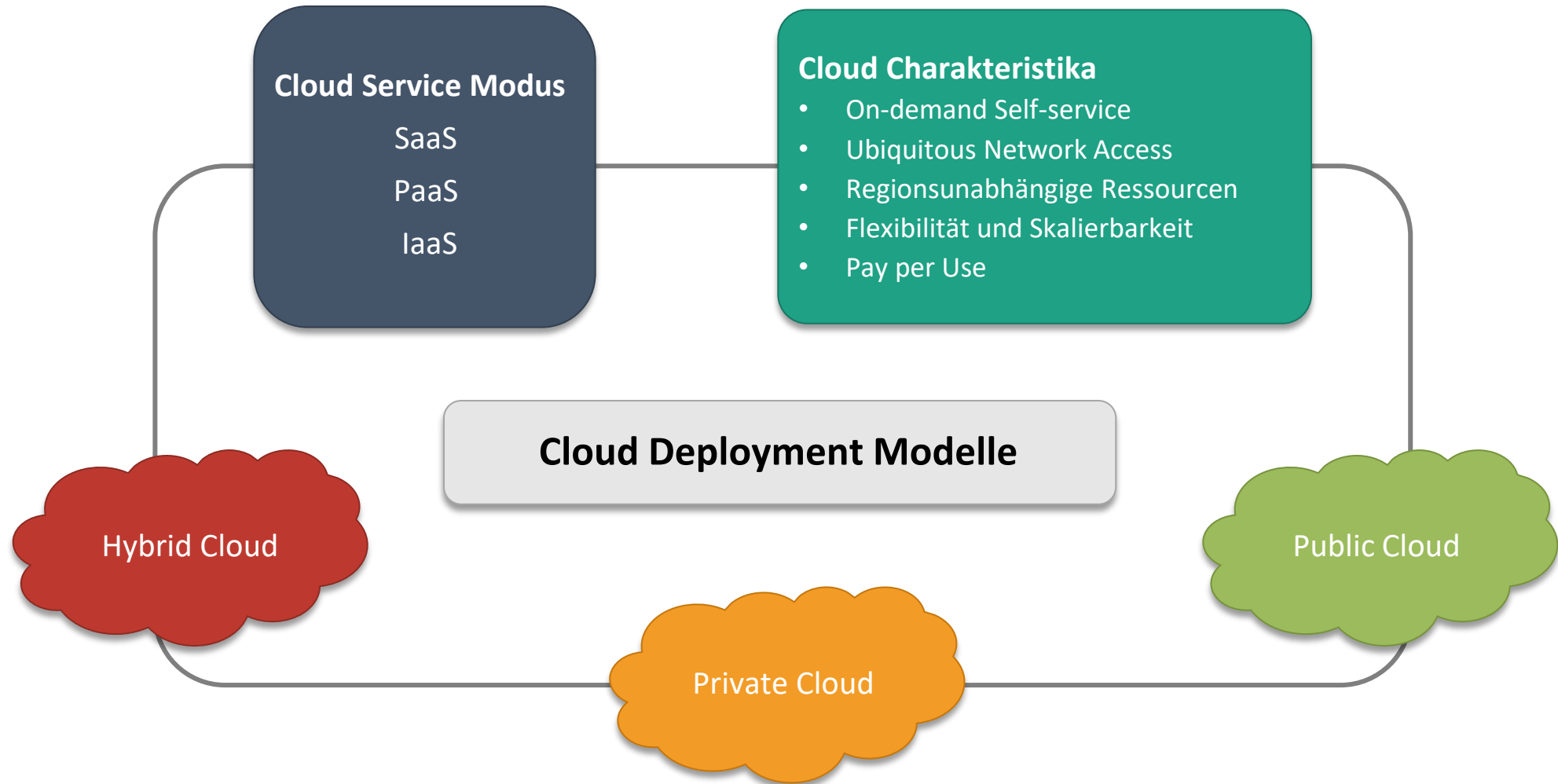
## Breached users in 2021



Surfshark

**Cloud-Anwendungen werden jedoch die dominierende Anwendungsart der Zukunft sein!**

# Cloud-Deploymentmodelle



# IaaS vs PaaS vs SaaS

On-site	IaaS	PaaS	SaaS
Applications	Applications	Applications	Applications
Data	Data	Data	Data
Runtime	Runtime	Runtime	Runtime
Middleware	Middleware	Middleware	Middleware
O/S	O/S	O/S	O/S
Virtualization	Virtualization	Virtualization	Virtualization
Servers	Servers	Servers	Servers
Storage	Storage	Storage	Storage
Networking	Networking	Networking	Networking

■ You manage  
■ Service provider manages

RedHat, 08/2022<sup>1</sup>

<sup>1</sup> <https://www.redhat.com/de/topics/cloud-computing/iaas-vs-paas-vs-saas>

# Auf dem Weg zu SaaS-Only Lösungen



Software for PC



Client/Server







SaaS

SaaS provider

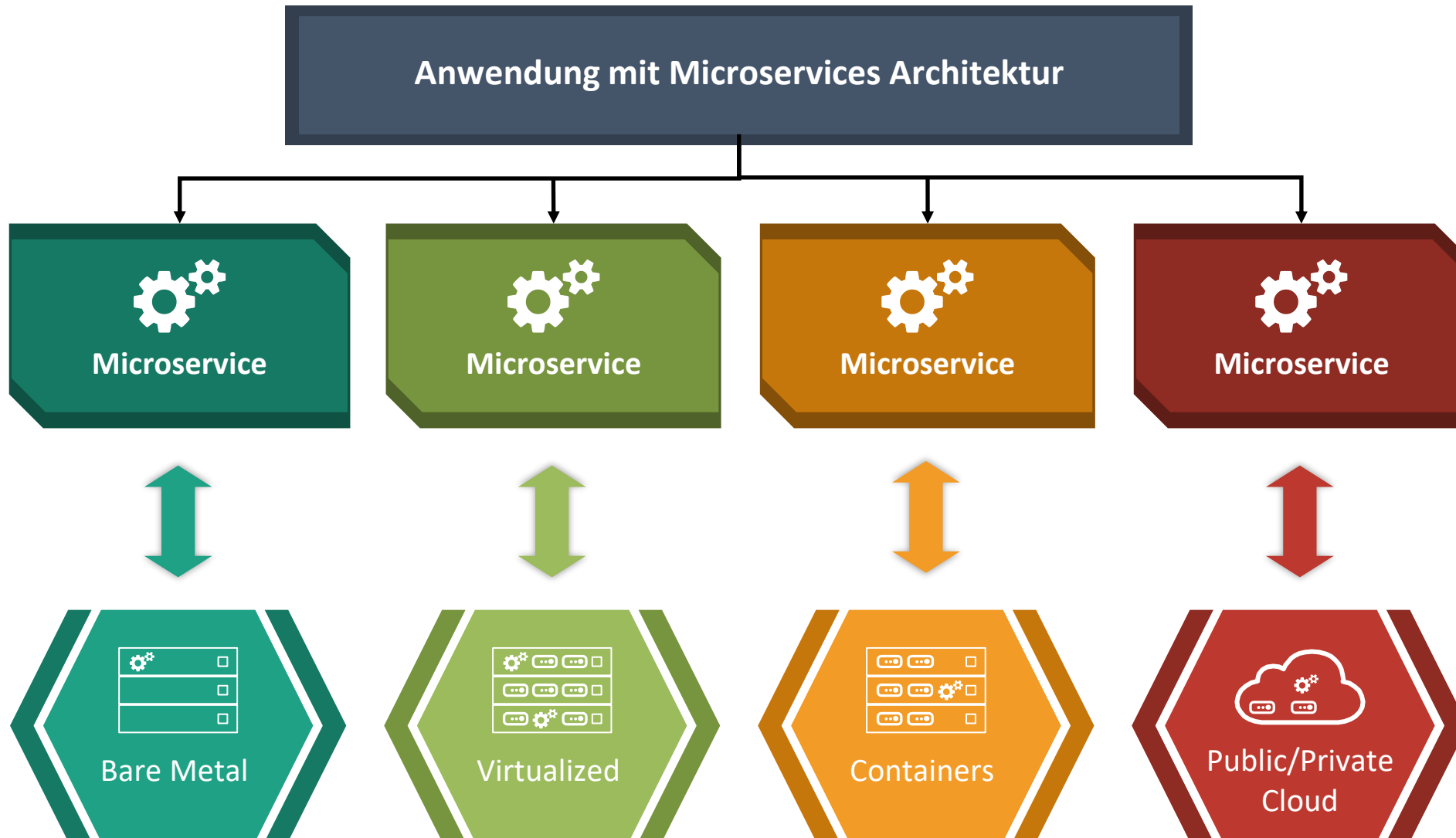


<sup>1</sup> <https://carnegieendowment.org/2020/08/31/cloud-security-primer-for-policymakers-pub-82597>

# Cloud Deployment Models - Anwendungsszenarios

Szenario	Service Modus	Deployment Modell	Vorteile
<b>Lohn-abrechnung</b>	SaaS, Daten in der Cloud	 Public Cloud	<ul style="list-style-type: none"> <li>• Verarbeitungszeit reduziert</li> <li>• Hardwareanforderungen reduziert</li> <li>• Flexibel für Unternehmensexpansion</li> </ul>
<b>Astronomie-datenverarbeitungs-service</b>	IaaS (VMs), Daten in der Cloud	 Public Cloud	<ul style="list-style-type: none"> <li>• Hardwarekosten erheblich reduziert</li> <li>• Energiekosten für CPU- und Speicherintensive Berechnungen erheblich reduziert</li> <li>• Vereinfachte Administration</li> </ul>
<b>Bundesregie-rungsdienste</b>	IaaS, PaaS	 Private Cloud, On-Site	<ul style="list-style-type: none"> <li>• Hoher Vertraulichkeitsanforderungen</li> <li>• Spezielle Anwendungen für Bundesbehörden</li> </ul>
<b>Kommunale Regierungs-dienste</b>	IaaS, PaaS	 Hybrid Cloud	<ul style="list-style-type: none"> <li>• Vertraulichkeitsanforderungen aber auch eingeschränkte Ressourcen</li> <li>• Reduzierter Aufwand &amp; flexibel durch Hybrides Modell</li> </ul>

# Microservice Architecture

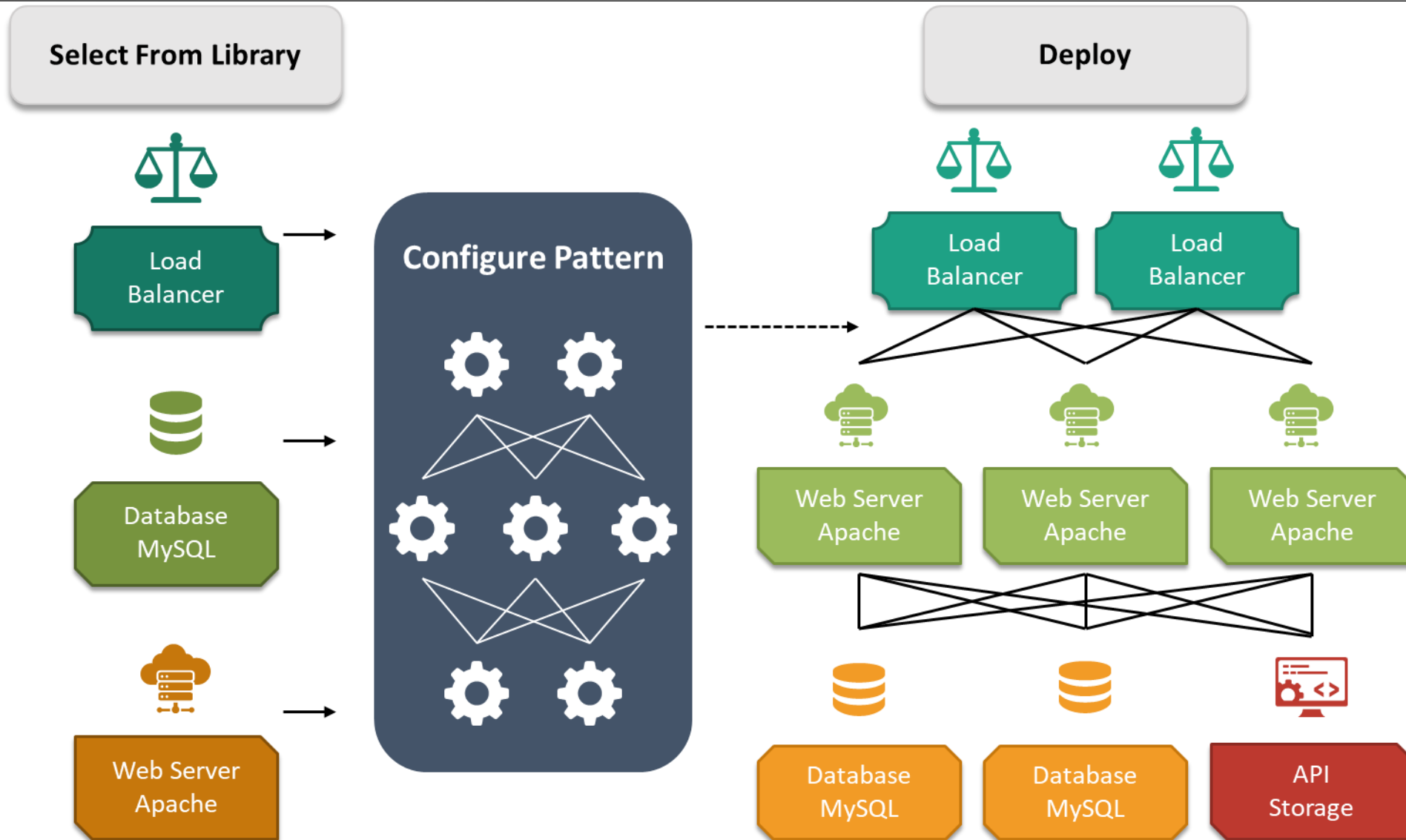


# Microservice Architecture – Prinzipien



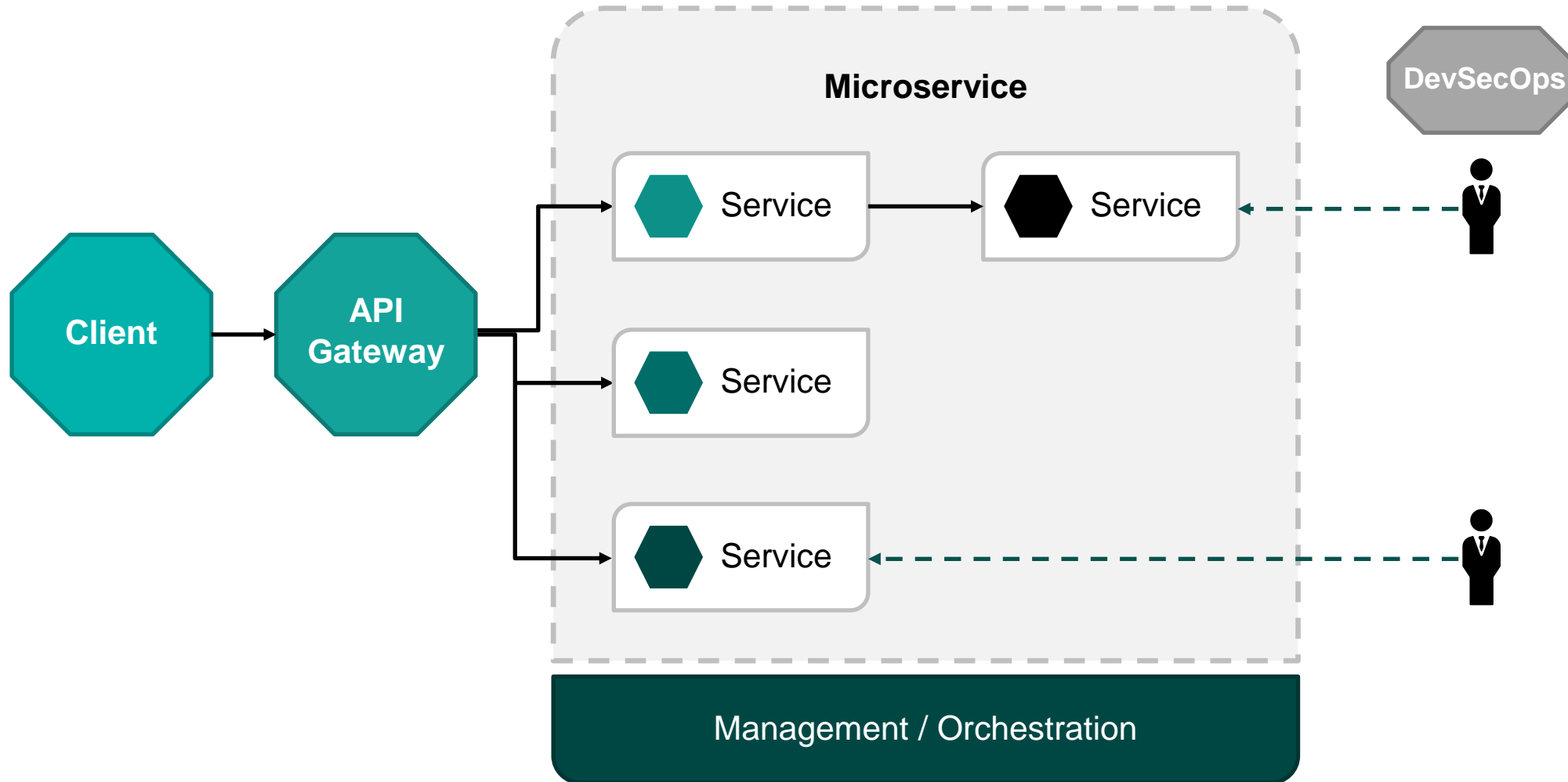


# Application Deployment - Konfiguration

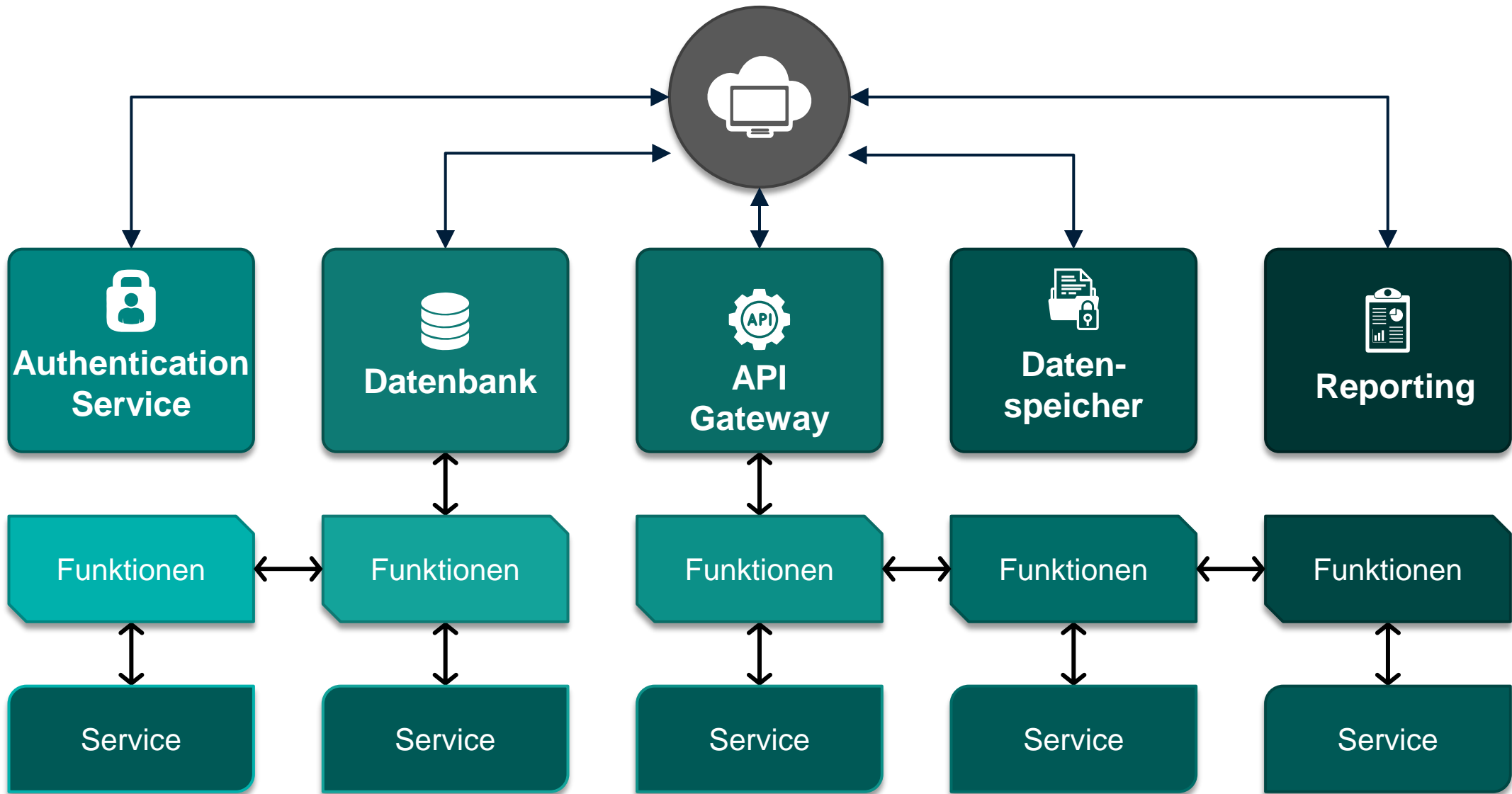




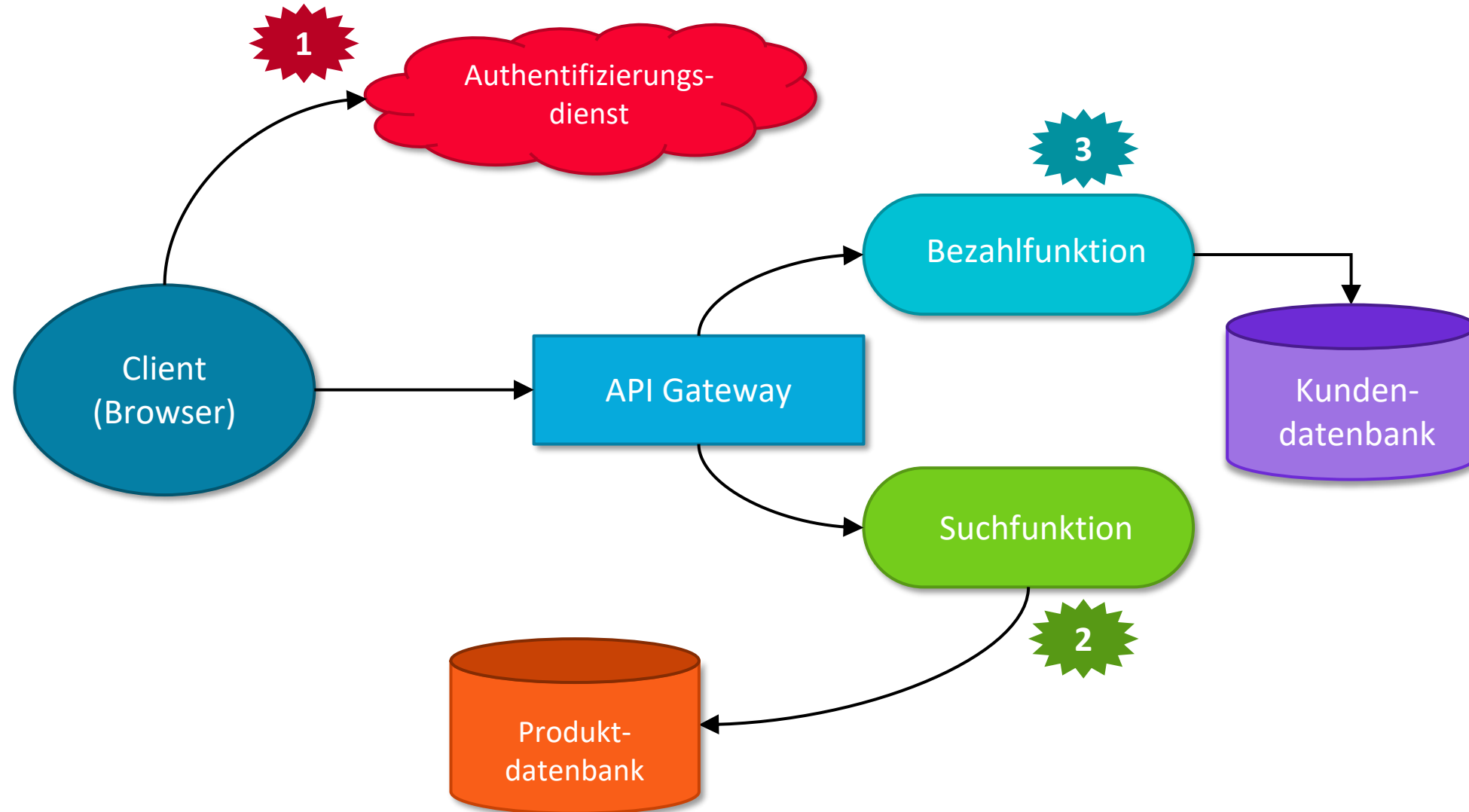
# Microservice Architecture – Management



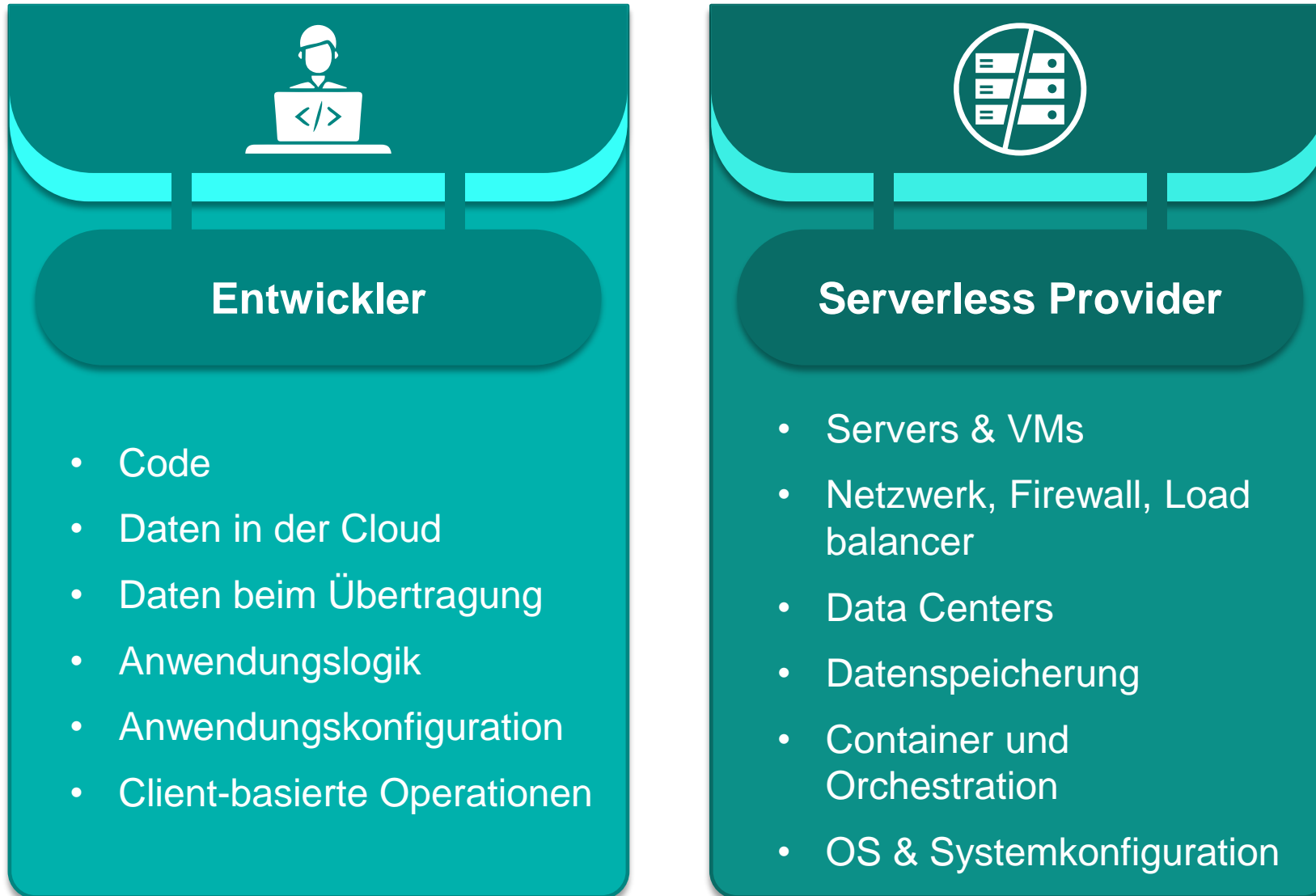
## Noch ein Schritt weiter: Serverless Architekturen



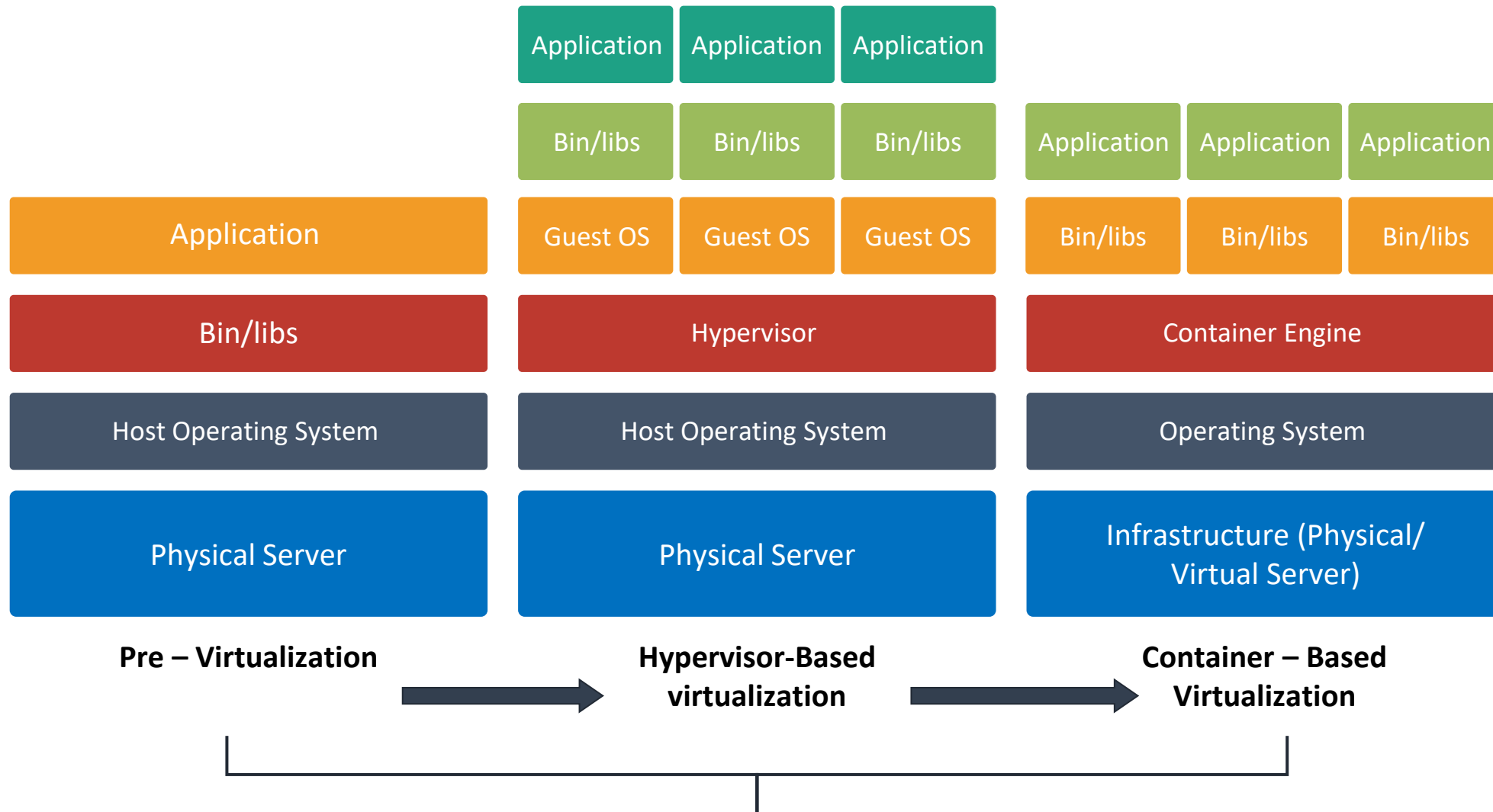
# Beispielablauf Webshop – Service-orientierte Darstellung



# Serverless Architekturen– Geteilte Verantwortung



# Entwicklung zur Container-basierten Virtualisierung



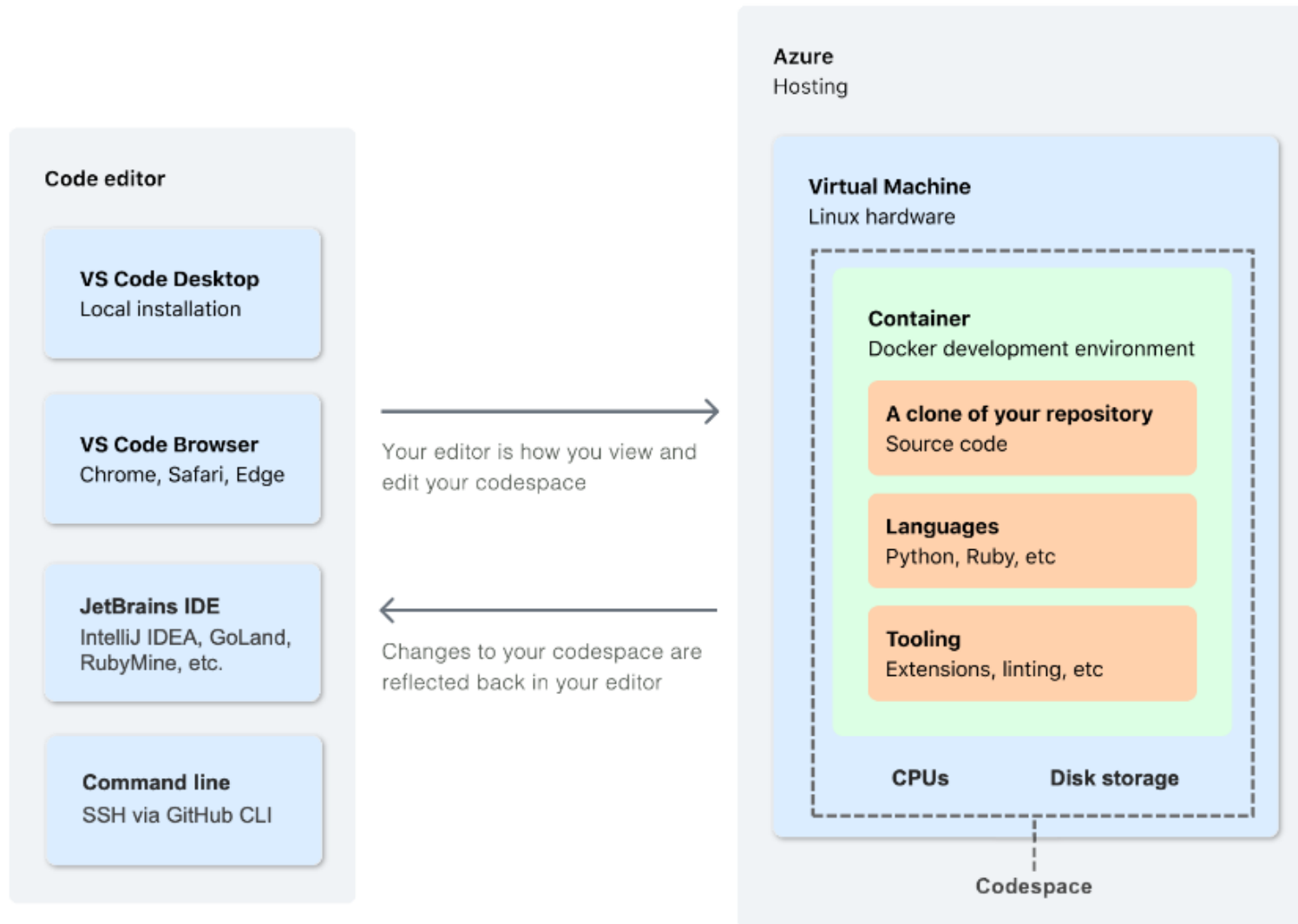
## Evolution der Virtualisierung

# GitHub Codespaces



- GitHub Codespaces ist eine Cloud-IDE gehostet auf Microsoft Azure mit allen GitHub-Funktionen.
  - Ermöglicht Programmierung in der Cloud in einer SaaS-Umgebung
  - Schafft reproduzierbare Entwicklungsumgebungen
- GitHub hat seine interne Entwicklung komplett auf Codespaces umgestellt.
- Codespaces ist verfügbar für Kunden in kostenpflichtigen Team- und Enterprise-Angeboten, aber auch eine freie Version ist verfügbar.
  - Monatlich 60 Stunden Codespaces in der VM-Basiskonfiguration mit 2 CPU-Kernen, 4 GByte RAM und 32 GByte Speicher
  - GitHub zählt inzwischen 94 Millionen Nutzer

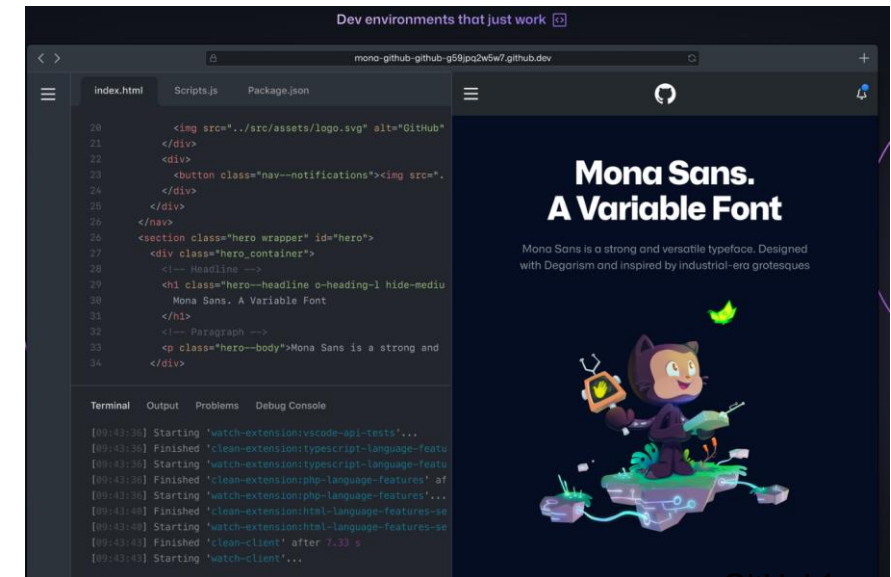
# GitHub Codespaces - Architektur



<sup>1</sup> <https://docs.github.com/en/codespaces/overview>

# Übung 8: GitHub Codespaces

- Wir schauen uns Umsetzung von GitHub Codespaces mal im Details an.
- Arbeiten Sie sich durch das GitHub Codespaces Tutorial unter Verwendung ihres FH GitHub Accounts:
  - <https://docs.github.com/en/codespaces/getting-started/quickstart>
- Anschließend beantworten Sie bitte die Fragen zu GitHub Codespaces auf dem Miro-Board unter
  - [https://miro.com/app/board/uXjVP\\_m8EXo=/?share\\_link\\_id=297204738278](https://miro.com/app/board/uXjVP_m8EXo=/?share_link_id=297204738278)
- Sie haben 20 Minuten Zeit das Tutorial zu bearbeiten und die Fragen zu beantworten. Danach diskutieren wir gemeinsam ihre Ergebnisse.



<sup>1</sup> <https://github.com/features/codespaces>



# Containerisierung - Definition

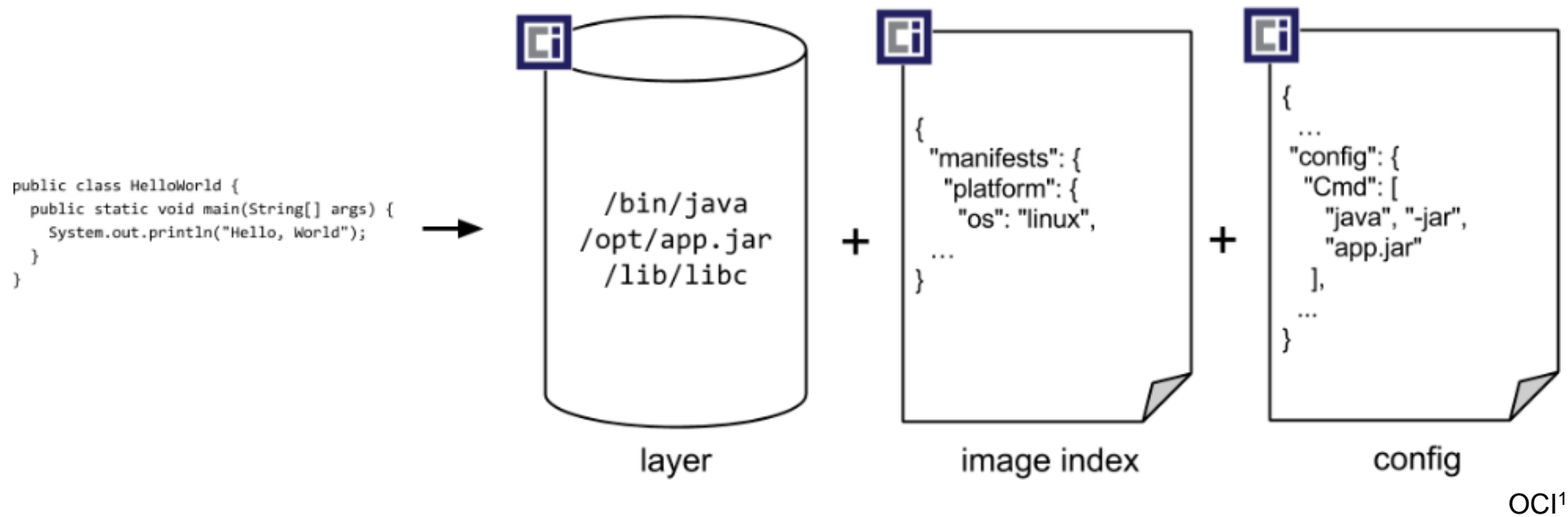
---

Container-Anwendungen sind weit verbreitet, eine beliebte Deployment-Variante und elementar im Aufbau von Microservice bzw. Serverless IT-Architekturen

- Ähnlich wie VMs sind Container eine Art „Behälter“ für Anwendungen, in dem diese laufen können.
  - VMs bilden jedoch eine ganze Computer-Umgebung ab (OS, Libs, etc)
  - Container enthalten lediglich die wichtigen Daten und Bibliotheken, die für die Ausführung der Applikation benötigt werden (OS-Komponenten, Libraries, Binaries etc)
  - Container ermöglichen eine leichtgewichtigere Form der Virtualisierung und weniger Ressourcen
  - Virtualisierung auf höherer Ebene im Vergleich zur VM und ohne Hypervisor
  - Bekannteste Container-Technologie ist Docker bzw. Docker Container
- Im Folgenden legen wir einen Fokus auf Anwendungs-Container und Docker als Container-Umgebung
  - Unveränderliche Container mit so wenig Code wie nötig zu betreiben, um die Anwendung auszuführen

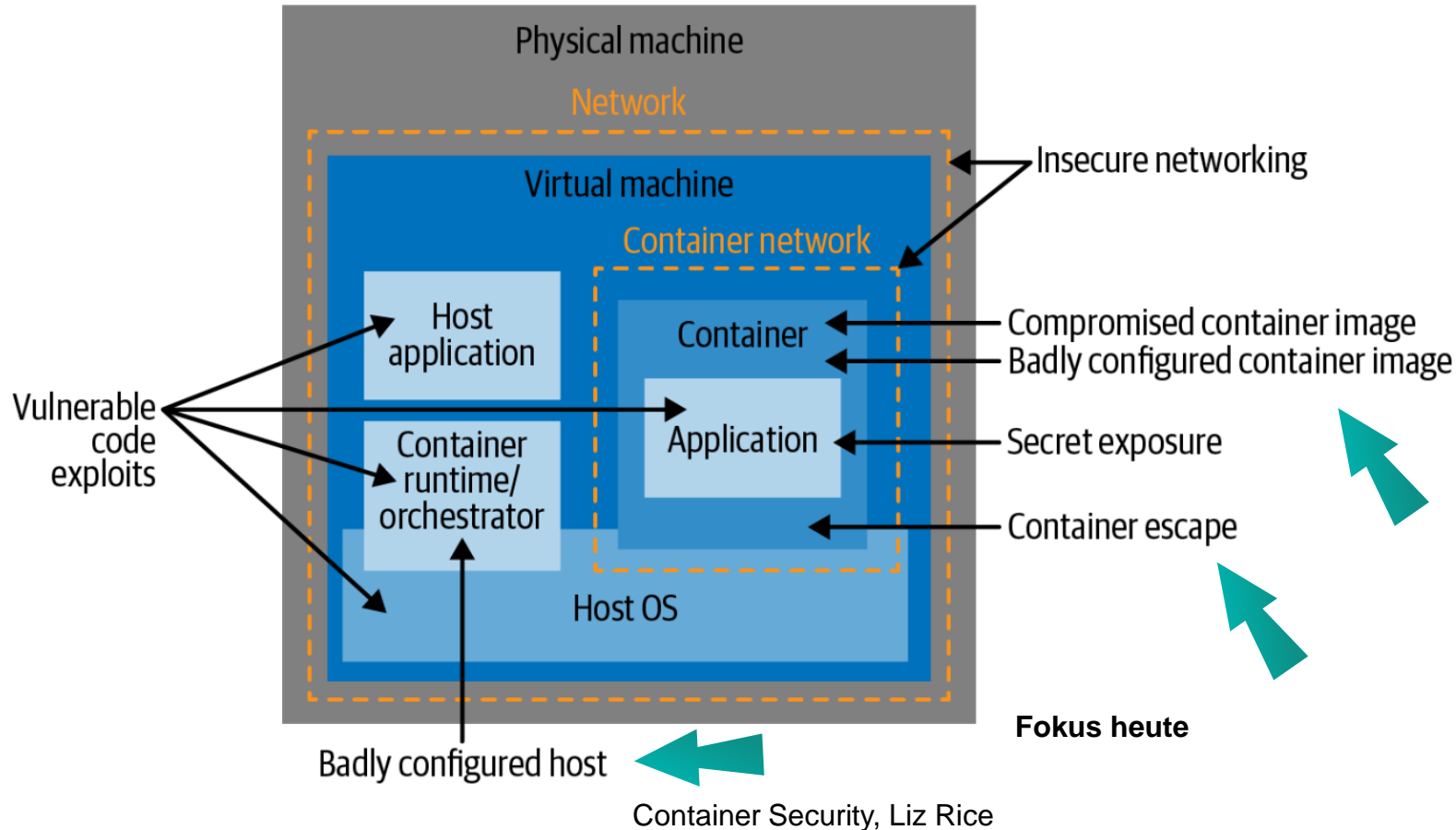
# OCI Standards

- Die Open Container Initiative (OCI) wurde gegründet, um Standards für Container-Images und -Laufzeiten zu definieren.
  - In Anlehnung an Docker Mechanismen und Techniken – viele Gemeinsamkeiten dementsprechend
  - Ziel der OCI: Standards für Container zu definieren und gleichzeitig kompatibel mit Docker bleiben
  - OCI-Spezifikationen: Image-Format



<sup>1</sup> <https://github.com/opencontainers/image-spec>

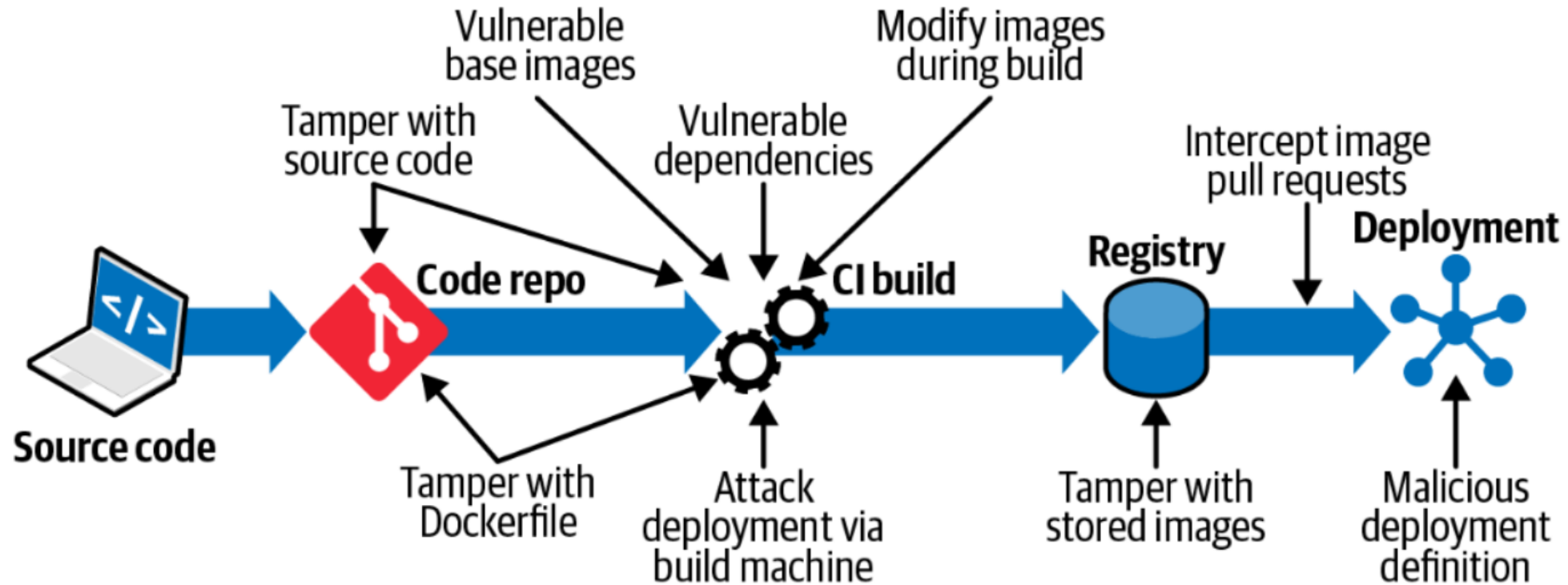
# Container-Anwendungen – Bedrohungen und Angriffe



- Externe Angreifer
- Allgemeine Interne Angreifer
  - Mit Absicht und Ausversehen
- DevSecOps Interne Angreifer
  - Devs, Admins, IT Sec etc
- Kompromittierte Anwendungen

# Container Images – Angriffsszenarien

**Hauptziel:** Image-Integrität im gesamten Image-Lebenszyklus sichern!



Container Security, Liz Rice

# Container Images

- Basis für Container bilden sogenannte Images, vereinfacht ausgedrückt:
  - Datei durch die die Installation und das Updaten einer Software wegfällt
  - Image-Inhalt: Root Filesystem und Konfigurationen
  - Beinhaltet alle Komponenten, um eine Anwendung plattformunabhängig auszuführen
  - Image kann einfach auf ein anderes System übertragen werden (Kopieren, aus Registry laden)
  - Container lässt sich aus dem Image in entsprechender Container-Umgebung (z.B. Docker) starten
- Verfügbar gemacht werden Images über eine Container Registry
  - Dort werden sie gespeichert, verwaltet und bereitstellt
  - Die bekannteste öffentliche Registry ist Docker Hub<sup>1</sup> – viele frei verfügbare Images
  - Wir werden mit GitHub Packages als Container Registry im Praktikum arbeiten – unter <https://ghcr.io><sup>2</sup>



<sup>1</sup> <https://hub.docker.com/>

<sup>2</sup> <https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry>

# Container Images – Aufbau in Layern

`docker run <imagename>`

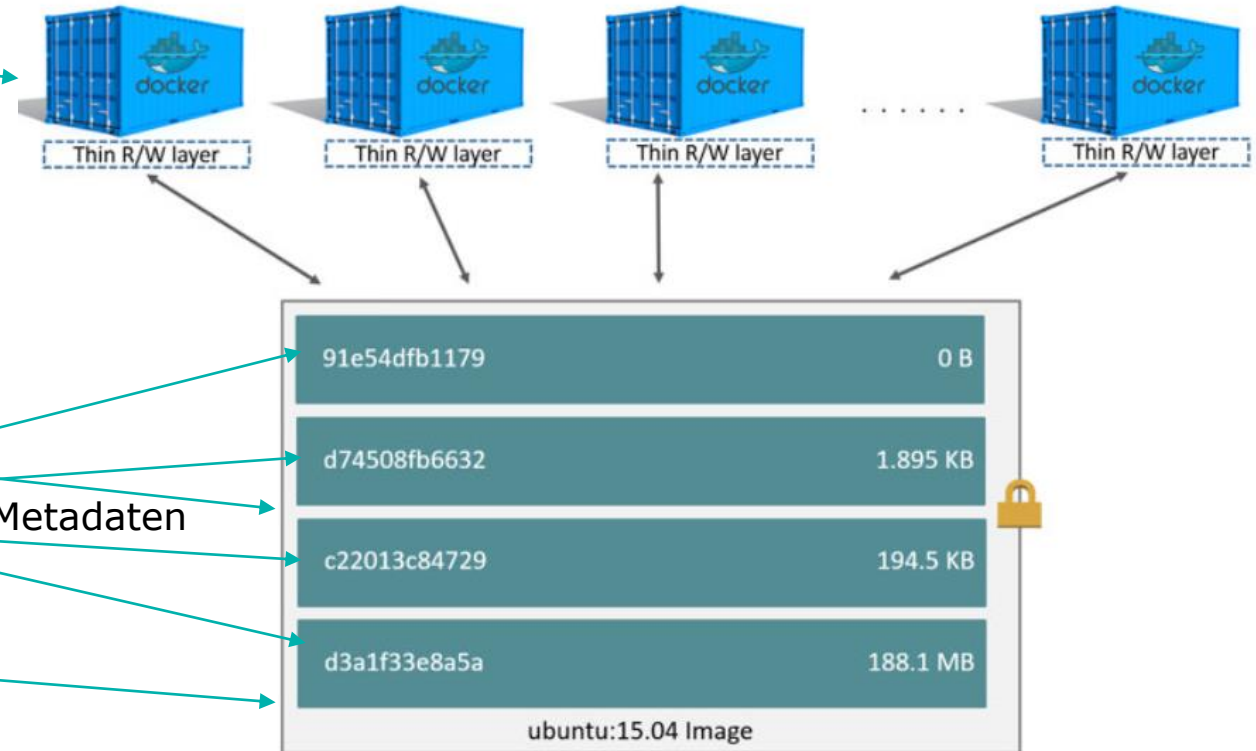
Container <-> Image

Dockerfile

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
LABEL org.opencontainers.image.authors="org@example.com"
COPY . /app
RUN make /app
RUN rm -r $HOME/.cache
CMD python /app/app.py
```

Metadaten

Metadaten



Docker Docs<sup>1</sup>

<sup>1</sup> <https://docs.docker.com/storage/storagedriver/#images-and-layers>

# Docker Image – Konfigurationsanalyse

- Analyse von Images und deren Konfiguration via *docker inspect*
  - Z.B. `docker inspect alpine:latest`

```
[
{
  "Id": "sha256:49176f190c7e9cdb51ac85ab6c6d5e4512352218190cd69b08e6fd803ffbf3da",
  "RepoTags": [
    "alpine:latest"
  ],
  "RepoDigests": [
    "alpine@sha256:8914eb54f968791faf6a8638949e480fef81e697984fba772b3976835194c6d4"
  ],
  "Parent": "",
  "Comment": "",
  "Created": "2022-11-22T22:19:29.008562326Z",
  "Container": "4700accf8884be7b6e6eb7c3fc8ea8af0d01e91787f8c446c56ee841f779a323",
  "ContainerConfig": {
    "Hostname": "4700accf8884",
    "Domainname": "",
    "User": "",
    "AttachStdin": false,
    "AttachStdout": false,
    "AttachStderr": false,
    "Tty": false,
    "OpenStdin": false,
    "StdinOnce": false,
    "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
    ],
    "Cmd": [
      "/bin/sh",
      "-c",
      "#(nop) ",
      "CMD [\"/bin/sh\"]"
    ],
    "Image": "sha256:60643c78796d4d33b3533adf6df1994ab846fb22ca117abe6f6cbc53d93e5205",
    "Volumes": null,
    "WorkingDir": "",
    "Entrypoint": null,
    "OnBuild": null,
    "Labels": {}
  },
  "DockerVersion": "20.10.12",
  "Author": ""
}
```

```
"Config": {
  "Hostname": "",
  "Domainname": "",
  "User": "",
  "AttachStdin": false,
  "AttachStdout": false,
  "AttachStderr": false,
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
    "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  ],
  "Cmd": [
    "/bin/sh"
  ],
  "Image": "sha256:60643c78796d4d33b3533adf6df1994ab846fb22ca117abe6f6cbc53d93e5205",
  "Volumes": null,
  "WorkingDir": "",
  "Entrypoint": null,
  "OnBuild": null,
  "Labels": null
},
"Architecture": "amd64",
"Os": "linux",
"Size": 7044846,
"VirtualSize": 7044846,
"GraphDriver": {
  "Data": {
    "MergedDir": "/var/lib/docker/overlay2/a3e6cd8a9f05c47a7e895dc16de87681da56e3b101ddb4ebf92bf08a0a85757f/merged",
    "UpperDir": "/var/lib/docker/overlay2/a3e6cd8a9f05c47a7e895dc16de87681da56e3b101ddb4ebf92bf08a0a85757f/diff",
    "WorkDir": "/var/lib/docker/overlay2/a3e6cd8a9f05c47a7e895dc16de87681da56e3b101ddb4ebf92bf08a0a85757f/work"
  },
  "Name": "overlay2"
},
"RootFS": {
  "Type": "layers",
  "Layers": [
    "sha256:ded7a220bb058e28ee3254fbb04ca90b679070424424761a53a043b93b612bf"
  ]
},
"Metadata": {
  "LastTagTime": "0001-01-01T00:00:00Z"
}
```



# Docker Image – Konfigurationsanalyse – 2. Beispiel

- Docker Docs – Getting Started Beispiel
  - docker inspect getting-started

```

1  [
2  {
3    "Id": "sha256:e5fb3733d5e0a78cdbf85492a4a35dc5608ee1e9650a48066fee1c1d493c7066",
4    "RepoTags": [
5      "getting-started:latest"
6    ],
7    "RepoDigests": [],
8    "Parent": "",
9    "Comment": "buildkit.dockerfile.v0",
10   "Created": "2022-12-01T14:30:26.167945529Z",
11   "Container": "",
12   "ContainerConfig": {
13     "Hostname": "",
14     "Domainname": "",
15     "User": "",
16     "AttachStdin": false,
17     "AttachStdout": false,
18     "AttachStderr": false,
19     "Tty": false,
20     "OpenStdin": false,
21     "StdinOnce": false,
22     "Env": null,
23     "Cmd": null,
24     "Image": "",
25     "Volumes": null,
26     "WorkingDir": "",
27     "Entrypoint": null,
28     "OnBuild": null,
29     "Labels": null
30   },
31   "DockerVersion": "",
32   "Author": "",
33   "Config": {
34     "Hostname": "",
35     "Domainname": "",
36     "User": "",
37     "AttachStdin": false,
38     "AttachStdout": false,
39     "AttachStderr": false,
40     "ExposedPorts": {
41       "3000/tcp": {}
42     },
43     "Tty": false,
44     "OpenStdin": false,
45     "StdinOnce": false,
46     "Env": [
47       "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",

```

```

48     "NODE_VERSION=18.12.1",
49     "YARN_VERSION=1.22.19"
50   ],
51   "Cmd": [
52     "node",
53     "src/index.js"
54   ],
55   "ArgsEscaped": true,
56   "Image": "",
57   "Volumes": null,
58   "WorkingDir": "/app",
59   "Entrypoint": [
60     "docker-entrypoint.sh"
61   ],
62   "OnBuild": null,
63   "Labels": null
64 },
65 "Architecture": "amd64",
66 "Os": "linux",
67 "Size": 255001278,
68 "VirtualSize": 255001278,
69 "GraphDriver": {
70   "Data": {
71     "LowerDir": "/var/lib/docker/overlay2/z4zp51hhs229xjreswtaaajowg/diff:/var/l
72     "MergedDir": "/var/lib/docker/overlay2/sj5ggc3kbbkcdcrdxl21lgxibom/merged",
73     "UpperDir": "/var/lib/docker/overlay2/sj5ggc3kbbkcdcrdxl21lgxibom/diff",
74     "WorkDir": "/var/lib/docker/overlay2/sj5ggc3kbbkcdcrdxl21lgxibom/work"
75   },
76   "Name": "overlay2"
77 },
78 "RootFS": {
79   "Type": "layers",
80   "Layers": [
81     "sha256:e5e13b0c77cbb769548077189c3da2f0a764ceca06af49d8d558e759f5c232bd",
82     "sha256:0d519f3bcae31782bf2de9c562962860e645d7c58241e16bb090af802f97a836",
83     "sha256:3d3b9564a8d2ab55aca4864cebac1691854faf4072cd13a50238026c6b1f3a95",
84     "sha256:b2d2930f52072004c7d096195bacbc820b4fc68d4eeale276f88a4e32a4181fe",
85     "sha256:5137e8576456d70f7d25551ce72d6d633adfa1248fcfbc219e1906182696a9e5",
86     "sha256:8ecdaeb24188eac50485d1444b6bf44b875439e623e9b3f4428e6acc8a904469",
87     "sha256:a7ba526c104e80fe68596d4fa8843eca4cfceec27a2b87e7970c5e86e1cffadf"
88   ]
89 },
90 "Metadata": {
91   "LastTagTime": "2022-12-01T14:30:28.800378368Z"
92 }
93 }
94 ]

```



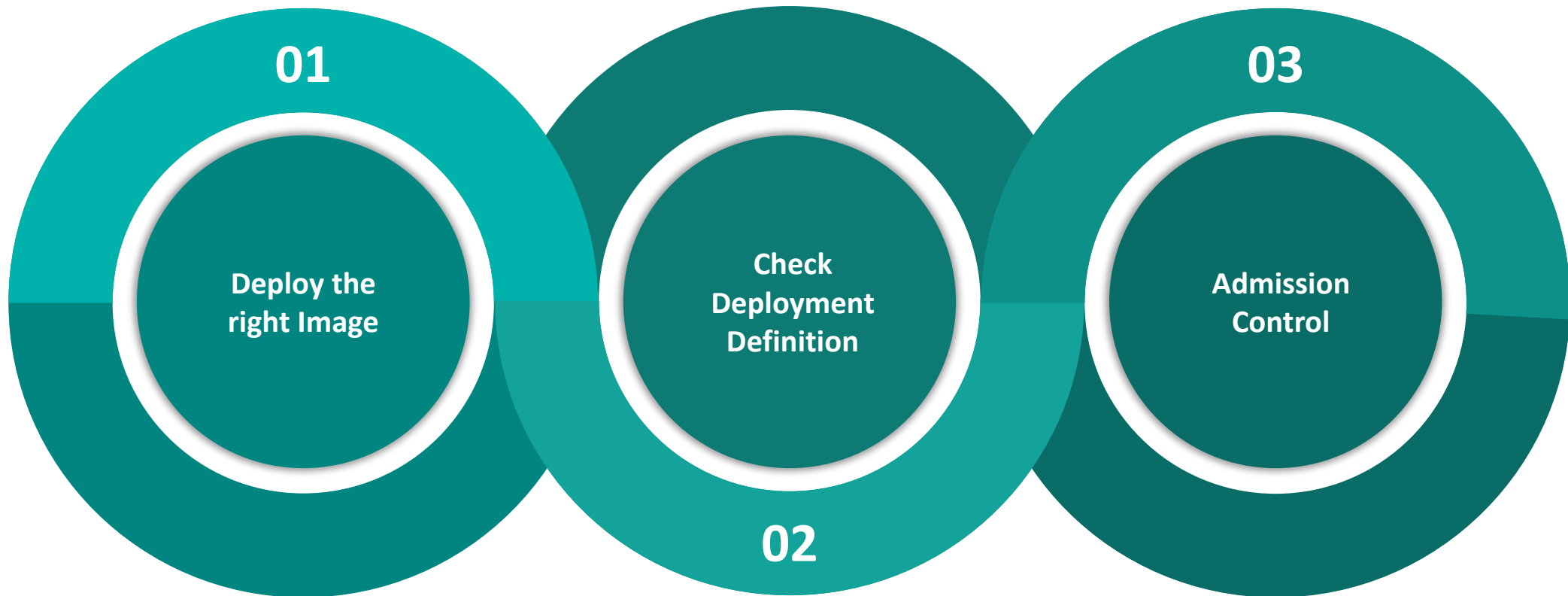
# Container Lebenszyklus – High-Level Overview

---

1. Erstellung eines Docker-Image oder Download eines existierenden Image aus einer Container Registry
2. Ausführen des Image oder mehrerer Images (Base Image + Anwendung) durch *docker run* in Docker-Umgebung
  - Z.b. Ubuntu OS + Apache Webserver
3. Anpassung der Konfiguration der Images und ggf. Anpassung der Images bzw. der Anwendung
  - Erzeugung eines neuen Image basierend auf einem existierenden Image (meist Base Image)
  - Hier z.B. neues Image mit PHP-Anwendung basierend auf dem Image mit Ubuntu OS + Apache Webserver
4. Erstellen des neuen Image (PHP, Apache, Ubuntu) und speichern in Container Registry
5. Abruf des finalen Image aus gesicherter Container Registry und Ausführung des Image (Schritt 2)

# IT-Sicherheit beim Deployment von Docker Images

- Das Hauptaugenmerk bei der Bereitstellung liegt darauf, sicherzustellen, dass das richtige Image gezogen (pull) und ausgeführt (run) wird!



# Admission Control - Anwendungscontainer

1. Sind Bibliotheken und Abhängigkeiten auf dem neuesten Stand?
  - Image neu erstellen und Container von diesem neuen Image aus starten
    - ❖ Nicht im laufenden Container patchen (andere Vorgehensweise als bei VMs)
  - Zusätzlich wird durch die Neuerstellung sichergestellt, dass alle üblichen Qualitätsprüfungen durchlaufen werden – Vermeidung eines „Quick and Dirty“ Fix
2. Wurde das Image auf Schwachstellen/Malware/Richtlinien gescannt?
3. Stammt das Abbild aus einer vertrauenswürdigen Registrierung?
4. Ist das Image signiert?
5. Läuft das Image als Benutzer (nicht Root) bzw. ist die Ausführung eingeschränkt?



# Sichere Containerausführung

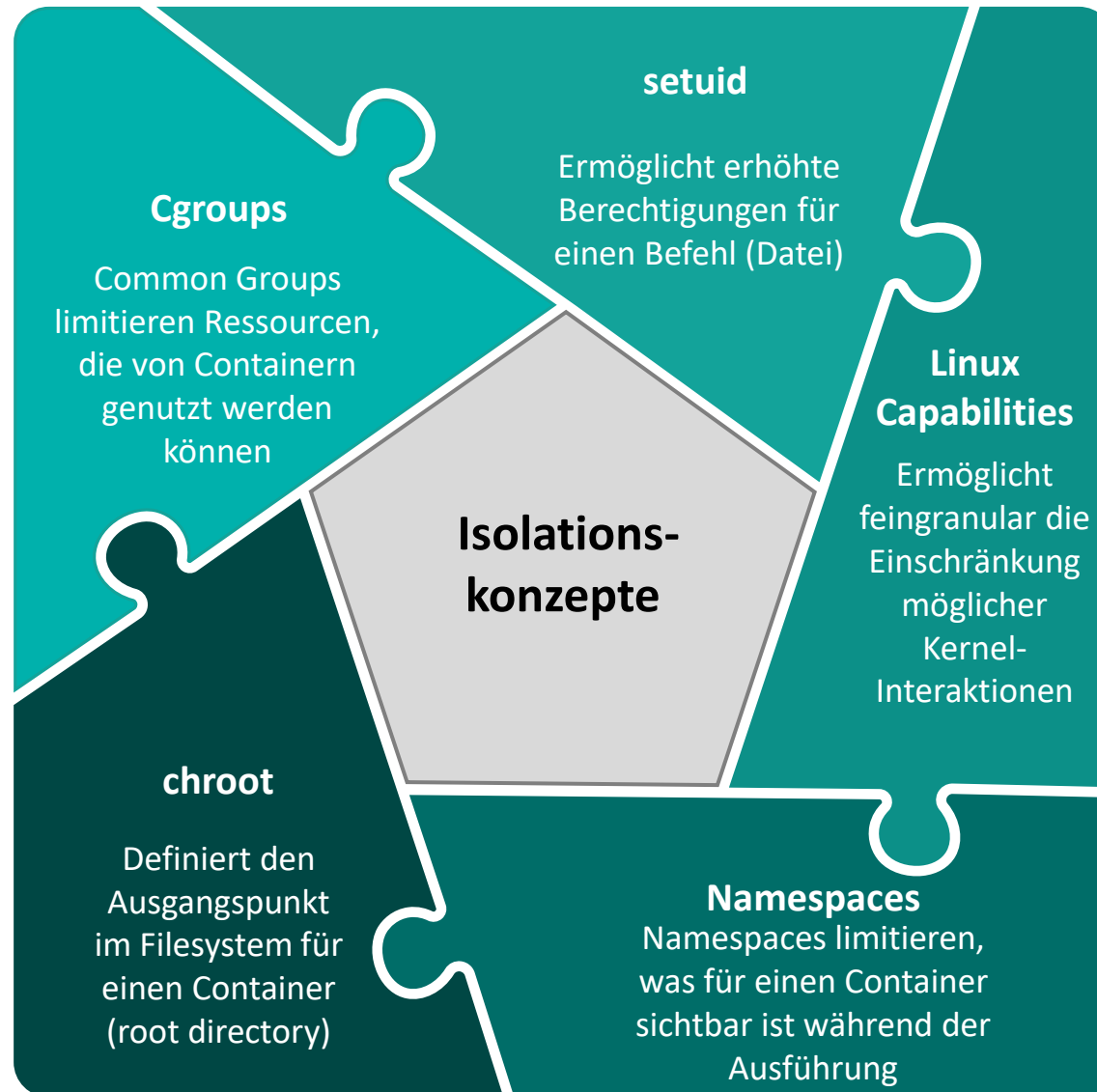
**Erinnerung: Hauptziel:** Image-Integrität im gesamten Image-Lebenszyklus sichern!

Insbesondere wichtig, dass die Container-Anwendung sicher (CIA) ausgeführt wird (zur Laufzeit).

Das schauen wir uns jetzt im Detail an!



# Container-Architekturen



# setuid

- Kurze Wiederholung: Dateiberechtigungen

```
-rwxr-xr-x 1 neugebauer fb5 76672 Nov 30 13:58 myping
```

Berechtigungen      Besitzer      Gruppe

- Weitere Anpassungen der Berechtigungen durch: *setuid*, *setgid*, und das *sticky bit*
- Kleines Beispiel mit Sleep-Befehl – ohne suid:*

```
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ cp /bin/sleep ./mysleep
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ ls -l mysleep
-rwxr-xr-x 1 neugebauer neugebauer 35328 Dez  1 11:03 mysleep
```

```
7508 7526 7526 7526 pts/0 8033 Ss 1000 0:00 bash
7526 8033 8033 7526 pts/0 8033 S+ 0 0:00 \_ sudo ./mysleep 100
8033 8034 8034 8034 pts/1 8035 Ss 0 0:00 \_ sudo ./mysleep 100
8034 8035 8035 8034 pts/1 8035 S+ 0 0:00 \_ ./mysleep 100
```

- Mit suid:

```
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ chmod +s mysleep
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ ls -l mysleep
-rwsr-sr-x 1 neugebauer neugebauer 35328 Dez  1 11:03 mysleep
```

```
7508 8020 8020 8020 pts/2 8070 Ss 1000 0:00 bash
8020 8070 8070 8020 pts/2 8070 S+ 0 0:00 \_ sudo ./mysleep 100
8070 8071 8071 8071 pts/1 8072 Ss 0 0:00 \_ sudo ./mysleep 100
8071 8072 8072 8071 pts/1 8072 S+ 1000 0:00 \_ ./mysleep 100
```

- Was hat das Ganze jetzt mit IT-Sicherheit zu tun?



```
-rwsr-sr-x 1 root neugebauer 35K Dez  1 11:03 mysleep
```

# Setuid – IT-Sicherheitsauswirkungen

- Eine falsch gesetzte setuid kann gravierende Konsequenzen haben (z.B. bei bash) – jeder der einen Befehl über die bash ausführt, hätte *root*-Berechtigungen
  - Typische Privilege Escalation Angriffe
  - Eine Vielzahl von Angriffen sind erfolgreich aufgrund von fehlerhaft konfigurierten Berechtigungen im Dateisystem!
    - Beispiel: `sudo chmod -R 777 <Verzeichnis>`
- Ähnliche Problematik bei setgid - Programm läuft unter Gruppenkennung, die der Datei zugewiesen ist
- Anmerkung: Ganz so trivial ist der Umgang mit setuid, setgid nicht, da es weitere OS-Sicherheitsmaßnahmen gibt, die fehlerhafte Konfiguration verhindern (Stichwort: reset user ID)
- Gegenmaßnahme in Docker-Umgebungen:
  - `docker run --security-opt=no-new-privileges <imagename>`





# Linux Capabilities

- Es gibt insgesamt über 30 Linux Capabilities (Privilegien) in heutigen Kernels.
  - Beispiel-Capabilities sind z.B.
    - `CAP_NET_BIND_SERVICE`: Benötigt, um Ports kleiner 1024 zu verwenden
    - `CAP_SYS_BOOT`: Benötigt, um Systemneustarts zu kontrollieren
    - Hilfe-Seite: *man capabilities*<sup>1</sup>
  - Anzeigen der Capabilities einer Datei oder eines Prozesses über `getcap` bzw. *getpcaps*
  - Setzen von Capabilities über *setcap* – z.B. `setcap 'cap_net_raw+ep' <Datei/Prozess>`

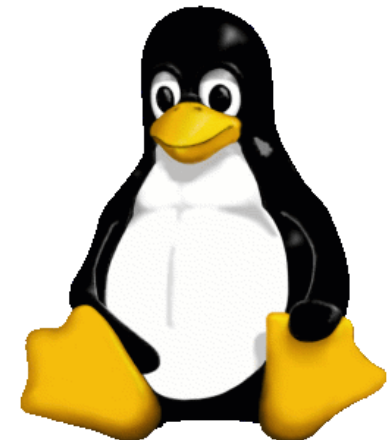
```
vagrant@vagrant:~$ sudo bash
root@vagrant:~# ps
  PID TTY          TIME CMD
 25061 pts/0        00:00:00 sudo
 25062 pts/0        00:00:00 bash
 25070 pts/0        00:00:00 ps
root@vagrant:~# getpcaps 25062
Capabilities for '25062': = cap_chown,cap_dac_override,cap_dac_read_search,
cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap
cap_linux_immutable,cap_net_bind_service,cap_net_broadcast,cap_net_admin,
cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,
cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,
cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,
cap_lease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override
cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read+ep
```

<sup>1</sup> <https://man7.org/linux/man-pages/man7/capabilities.7.html>



# Linux Capabilities und Docker

- Das Linux-Capabilities-Feature unterteilt die Privilegien, die Prozessen zur Verfügung stehen, die als Root-Benutzer laufen, in kleinere Gruppen von Privilegien.
- Docker unterstützt Linux Capabilities über den *docker run* Befehl
  - Optionen sind `--cap-add` und `--cap-drop` zum Hinzufügen und Entfernen von Capabilities
  - Standardmäßig startet ein Container mit einer Reihe von Permissions (z.B `cap_sys_module`)
  - Gemäß Principle of Least Privilege alle Capabilities entfernen + Whitelist Kernel-Calls
    - `docker run --cap-drop ALL --cap-add SYS_TIME <imagename> /bin/sh`
  - Minimale Capabilities hängen stark vom Einsatzszenario ab, ggf. viel Try-and-Error bis die richtige Kombination von Capabilities für die Docker-Anwendung gefunden wurde und die Anwendung fehlerfrei läuft...



# Linux Caps Extended - seccomp

- Secure Computing Mode (seccomp) ist eine Kernel-Funktion, um Systemaufrufe an den Kernel feingranular zu filtern
- Eingeschränkte und erlaubte Systemaufrufe werden in Profilen zusammengefasst
  - Verschiedene Profile können unterschiedlichen Containern zugewiesen werden
- Seccomp bietet eine feingranularere Kontrolle als Capabilities
  - Ein Angreifer hat eine begrenzte Anzahl von Syscalls, die aus dem Container aufgerufen werden können (und die Menge ist vorab genau definiert!)
- Standard seccomp-Profil für Docker ist eine frei verfügbare JSON-Datei<sup>1</sup>
  - 44 von über 300 Systemcalls standardmäßig blockiert<sup>2</sup>
  - Weitere Einschränkung der Systemaufrufe - Trade-Off zur Anwendungskompatibilität
  - Option für *docker run* ist `--security-opt` – Beispielbefehl:
    - `docker run --security-opt seccomp=<PfadzumProfil>/myprofile.json <imagename>`
  - ❖ Achtung: Standard-Profil muss geladen werden, Kernel ist entsprechend konfiguriert und Docker-Umgebung mit seccomp gebaut
    - `docker run --rm -it --security-opt seccomp=seccomp_dockerstandard_profil.json hello-world`

<sup>1</sup> <https://github.com/moby/moby/blob/master/profiles/seccomp/default.json>

<sup>2</sup> <https://docs.docker.com/engine/security/seccomp/>

# Cgroups

- Cgroups begrenzen die Ressourcen, wie Speicher, CPU und Netzwerkeingabe/-ausgabe, die Container nutzen können (Version 1 und 2 verfügbar)
  - Aus Sicherheitsperspektive ermöglichen gut konfigurierte cgroups, dass ein Docker-Container das Verhalten des Systems und andere Prozesse/Container nicht beeinflusst
  - Z.B. gesamte CPU beansprucht oder Filesystem lahmlegt durch zu viele IO-Operationen
  - Es gibt auch eine cgroup *pid*: Hiermit kann man beispielsweise eine Fork-Bombe verhindern

```
@gneugeb-fhaachen → /sys/fs/cgroup $ ls
blkio      cpu,cpuacct  freezer     net_cls     perf_event  systemd
cpu        cpuset       hugetlb     net_cls,net_prio  pids
cpuacct    devices      memory      net_prio    rdma
```

```
@gneugeb-fhaachen → /sys/fs/cgroup/cpu $ ls
cgroup.clone_children  cpuacct.usage_percpu_sys  cpu.shares
cgroup.procs           cpuacct.usage_percpu_user  cpu.stat
cpuacct.stat           cpuacct.usage_sys         cpu.uclamp.max
cpuacct.usage          cpuacct.usage_user        cpu.uclamp.min
cpuacct.usage_all      cpu.cfs_period_us         notify_on_release
cpuacct.usage_percpu   cpu.cfs_quota_us          tasks
```

# Cgroups – Konfiguration Docker-Container

- Wenn ein Docker-Container gestartet wird, wird unter `/sys/fs/cgroup` im Unterordner *docker* automatisch ein neues Set an cgroups angelegt
  - Cgroup name: Container ID

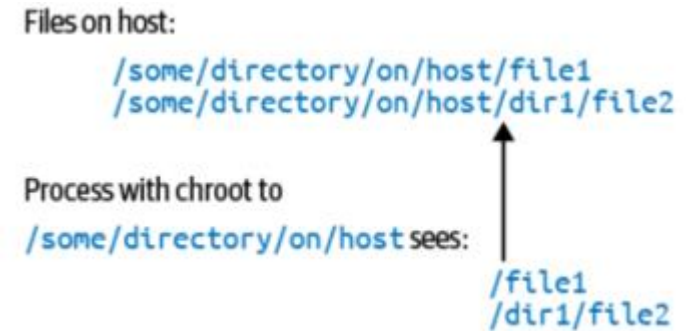
```
@gneugeb-fhaachen → .../cgroup/cpu,cpuacct/docker/6ba78190b66d07c2209407ea72678357e6037
97c4024a9d11199a486de17266b $ ls
cgroup.clone_children  cpuacct.usage_percpu      cpu.cfs_period_us      cpu.uclamp.min
cgroup.procs           cpuacct.usage_percpu_sys  cpu.cfs_quota_us       notify_on_release
cpuacct.stat           cpuacct.usage_percpu_user cpu.shares              tasks
cpuacct.usage          cpuacct.usage_sys         cpu.stat
cpuacct.usage_all      cpuacct.usage_user        cpu.uclamp.max
```

- Cgroup Limits können sehr einfach über den *docker run* Befehl als Parameter übergeben werden. Beispiele:
  - `docker run --memory 256m <containername>`
  - `docker run --cpus 0.5 <containername>`
  - Auch möglich über Konfiguration in *docker-compose.yml*

# chroot

➤ Innerhalb eines Containers sehen Sie nicht das gesamte Dateisystem des Hosts, sondern nur eine Teilmenge, da das „Root Directory“ bei der Erstellung des Containers geändert wird – *chroot* Befehl

- Jeder Container hat sein eigenes „Root Directory“ d.h.
  - Alle nötigen Dateien und Libraries müssen enthalten sein, z.B. auch eine Shell (/bin/bash)
  - Eingeschränkte Linux-Umgebung im Container (alle Use Cases betrachten)



Container Security, Liz Rice

```
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ mkdir new_root
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ sudo chroot new_root/
[sudo] password for neugebauer:
chroot: failed to run command '/bin/bash': No such file or directory
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$
```

# Namespaces - Einführung

---

- Ein Container ähnelt „von innen“ einer VM.
  - Über Docker `exec <imagename> bash` erhalten Sie Shell-Zugriff
  - Wenn Sie `ps` ausführen: Nur Prozesse sichtbar, die im Container laufen
  - Der Container verfügt über einen eigenen Netzwerkstack
  - Der Container hat ein eigenes Dateisystem mit einem Root-Verzeichnis
  - Es scheint keine Beziehung zum Dateisystem der VM zu geben
- Wie geht das? Mit der Technik Linux Namespaces!
- Namespaces kontrollieren was sichtbar im Container ist.
  - Prozesse (Laufende Docker-Container) im selben Namespaces sehen die selben Ressourcen

# Arten von Namespaces

- Es existieren eine Reihe von unterschiedlichen Typen von Namespaces

- Unix Timesharing System (UTS)
- Process Ids
- Mount points
- Network
- User and group Ids
- Inter-process communications (IPC)
- Control groups (cgroups)

NS	TYPE	NPROCS	PID	USER	COMMAND
4026531834	time	82	2704	neugebauer	/lib/systemd/systemd --user
4026531835	cgroup	82	2704	neugebauer	/lib/systemd/systemd --user
4026531836	pid	82	2704	neugebauer	/lib/systemd/systemd --user
4026531837	user	82	2704	neugebauer	/lib/systemd/systemd --user
4026531838	uts	82	2704	neugebauer	/lib/systemd/systemd --user
4026531839	ipc	82	2704	neugebauer	/lib/systemd/systemd --user
4026531840	net	82	2704	neugebauer	/lib/systemd/systemd --user
4026531841	mnt	80	2704	neugebauer	/lib/systemd/systemd --user
4026532591	mnt	1	3220	neugebauer	/snap/snap-store/599/usr/bin/snap-stor
4026532592	mnt	1	3334	neugebauer	/snap/snapd-desktop-integration/43/usr

➤ Ein Prozess (Container) ist immer **exakt** in einem Namespace jedes Typs!

- Über *lsns* können Sie sich alle existierenden Namespaces anzeigen lassen – Standardmäßig 1 NS



# UTS – Eigener Hostname im Docker-Container

```
neugebauer@neugebauer-VM:~$ hostname  
neugebauer-VM
```

**VS**

```
@gneugeb-fhaachen → /workspaces/codespaces-blank/getting-started/app (master) $ docker run --rm -it ge  
tting-started /bin/sh  
/app # hostname  
308898e25b0f
```

# Unshare Befehl für Namespace-Experimente

- Über den Unshare Befehl können Sie neue Namespaces erstellen (unabhängig vom Host-System) und dann einen bestimmten Befehl mit dem neuen Namespace starten (z.B. Shell).

```
neugebauer@neugebauer-VM:~$ sudo unshare --uts sh
# hostname
neugebauer-VM
# hostname freewilly
# hostname
freewilly
# exit
neugebauer@neugebauer-VM:~$ hostname
neugebauer-VM
```

# Process ID Namespace – Prozessisolation im Docker-Container

```
neugebauer@neugebauer-VM:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 101200 11996 ?        Ss   08:44   0:00 /sbin/init sp
root         2  0.0  0.0      0     0 ?        S    08:44   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   08:44   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   08:44   0:00 [rcu_par_gp]
root         5  0.0  0.0      0     0 ?        I<   08:44   0:00 [netns]
root         7  0.0  0.0      0     0 ?        I<   08:44   0:00 [kworker/0:0H]
root         9  0.0  0.0      0     0 ?        I<   08:44   0:00 [kworker/0:1H]
root        10  0.0  0.0      0     0 ?        I<   08:44   0:00 [mm_percpu_wq]
root        11  0.0  0.0      0     0 ?        S    08:44   0:00 [rcu_tasks_ru]
root        12  0.0  0.0      0     0 ?        S    08:44   0:00 [rcu_tasks_tr]
root        13  0.0  0.0      0     0 ?        S    08:44   0:00 [ksoftirqd/0]
root        14  0.0  0.0      0     0 ?        I    08:44   0:00 [rcu_sched]
root        15  0.0  0.0      0     0 ?        S    08:44   0:00 [migration/0]
root        16  0.0  0.0      0     0 ?        S    08:44   0:00 [idle_inject/
```

VS

```
@gneugeb-fhaachen → /workspaces/codespaces-blank/getting-started/app (master) $ docker exec -it dc7 /bin/sh
/app # ps aux
PID    USER      TIME  COMMAND
   1   root         0:00 node src/index.js
  25   root         0:00 /bin/sh
  31   root         0:00 ps aux
/app # █
```

# Namespaces kombiniert mit Chroot

```
neugebauer@neugebauer-VM:~/DevSecOps/linuxplayground$ sudo unshare --pid --fork chroot alpine sh
/ # ls -l
total 3248
-rwxrwx--- 1 1000 1000 3252303 Dec 2 08:17 alpine-minirootfs-3.17.0-x86_64.tar.gz
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 bin
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 dev
drwxr-xr-x 17 1000 1000 4096 Nov 22 13:06 etc
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 home
drwxr-xr-x 7 1000 1000 4096 Nov 22 13:06 lib
drwxr-xr-x 5 1000 1000 4096 Nov 22 13:06 media
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 mnt
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 opt
dr-xr-xr-x 2 1000 1000 4096 Nov 22 13:06 proc
drwx----- 2 1000 1000 4096 Dec 2 08:18 root
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06 run
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06/sbin
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06/srv
drwxr-xr-x 2 1000 1000 4096 Nov 22 13:06/sys
drwxrwxr-x 2 1000 1000 4096 Nov 22 13:06/tmp
drwxr-xr-x 7 1000 1000 4096 Nov 22 13:06/usr
drwxr-xr-x 12 1000 1000 4096 Nov 22 13:06/var
/ #
```

```
/ # ps
PID    USER      TIME  COMMAND
/ # ps aux
PID    USER      TIME  COMMAND
/ # mount -t proc proc proc
/ # ps
PID    USER      TIME  COMMAND
      1 root        0:00  sh
      6 root        0:00  ps
/ #
```

## Weitere Namespaces

- Ähnliches Prinzip für die weiteren Namespaces
  1. Mount: Anderes Dateisystem verfügbar im Container als auf dem Hostsystem
  2. Network: Eigene Netzwerkschnittstellen und Routingtabellen
    - Wichtiges Feature hinsichtlich IT-Sicherheit – aber auch hohe Fehlerrate für Kommunikationsprobleme (Routing)
  3. User Namespace: Eigene User und GruppenIDs im Container
  4. Inter-Process Communications: Kommunikation zwischen Prozessen
  5. Cgroup: Ähnlich einem chroot auf dem cgroup Dateisystem (/sys/fs/cgroup)

```
/app # ip route
default via 172.17.0.1 dev eth0
172.17.0.0/16 dev eth0 scope link src 172.17.0.2
/app # █
```

**VS**

```
@gneugeb-fhaachen → /workspaces/codespaces-blank/getting-started/app (master) $ ip route
default via 172.16.5.1 dev eth0 proto dhcp src 172.16.5.4 metric 100
168.63.129.16 via 172.16.5.1 dev eth0 proto dhcp src 172.16.5.4 metric 100
169.254.169.254 via 172.16.5.1 dev eth0 proto dhcp src 172.16.5.4 metric 100
172.16.5.0/24 dev eth0 proto kernel scope link src 172.16.5.4
172.17.0.0/16 dev docker0 proto kernel scope link src 172.17.0.1
```

# Sichere Image-Konfiguration

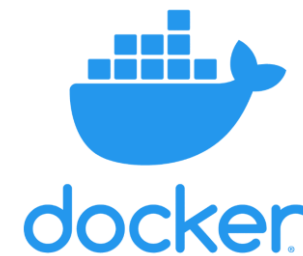
## config.json

```
"linux": {
  "resources": {
    "memory": {
      "limit": 1000000
    },
    "devices": [
      {
        "allow": false,
        "access": "rwm"
      }
    ]
  },
  "namespaces": [
    {
      "type": "pid"
    },
    {
      "type": "network"
    },
    {
      "type": "ipc"
    },
    {
      "type": "uts"
    },
    {
      "type": "mount"
    }
  ]
}
```

Container Security, Liz Rice

# Übung 9: Ihre erste eingeschränkte Docker-Anwendung

- Wir schauen uns die Isolationskonzepte mal im Detail an.
- Arbeiten Sie sich mit Hilfe ihrer GitHub Codespaces Umgebung durch das Docker-Tutorial unter Verwendung ihres FH GitHub Accounts:
  - [https://docs.docker.com/get-started/02\\_our\\_app/](https://docs.docker.com/get-started/02_our_app/)
- Anschließend bearbeiten Sie bitte die folgenden Aufgaben zum Thema „Absicherung von laufenden Docker-Containern“ auf dem Miro-Board unter
  - [https://miro.com/app/board/uXjVP9rMr8M=/?share\\_link\\_id=687821523811](https://miro.com/app/board/uXjVP9rMr8M=/?share_link_id=687821523811)
- Sie haben 15 Minuten Zeit das Tutorial zu bearbeiten und die Aufgaben/Fragen zu bearbeiten/beantworten. Danach diskutieren wir gemeinsam ihre Ergebnisse.





# Rootless Container

- **Im Allgemeinen läuft die Docker-Umgebung und Docker-Container als Root**

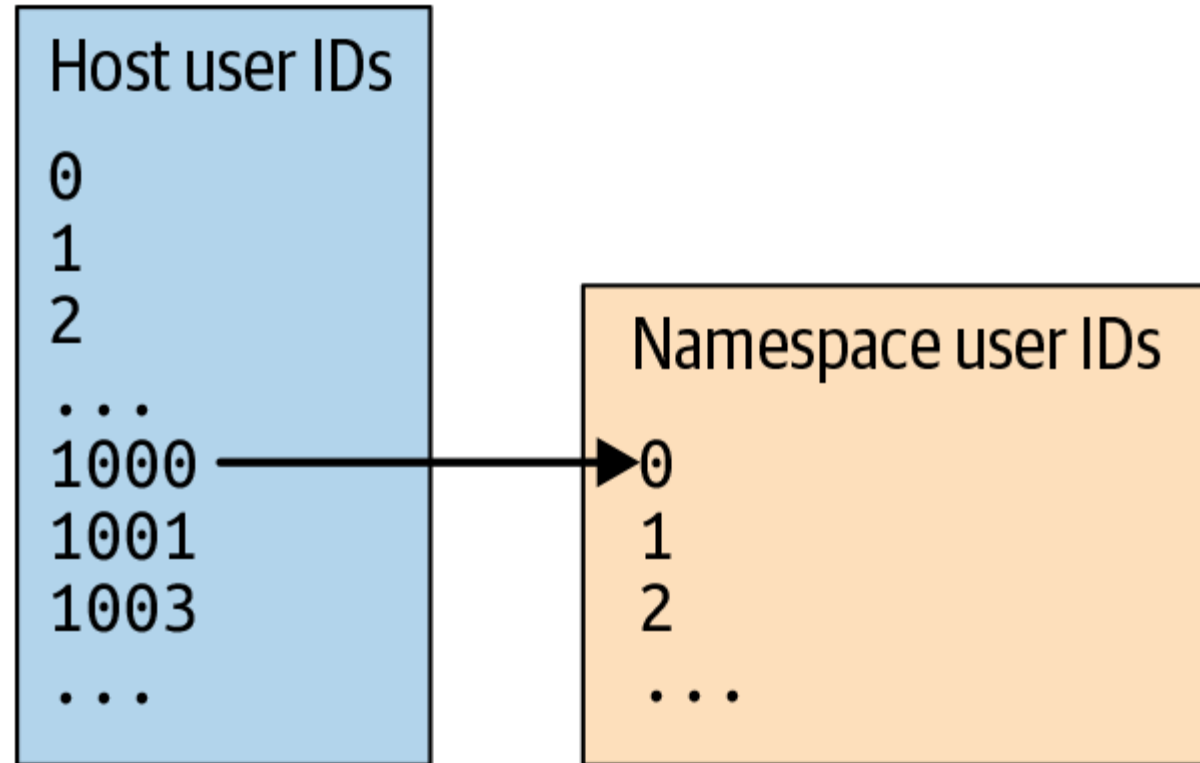
- Z.B. um benötigt um neue Container zu erstellen
- Genauer muss man Mitglied in der docker Gruppe sein, um über den Docker socket mit dem Docker daemon zu kommunizieren
- Zugriff zum Hostsystem mit Docker daemon bedeutet Root-Zugriff zum System!
- Angreifer könnte versuchen den Container mit gemountetem Dateisystem als Volume zu starten
  - `docker run -v`

- Rootless Container nutzen die Funktion des User Namespace

- Eine normale Nicht-Root-Benutzer-ID auf dem Host wird innerhalb des Containers auf Root abgebildet
- Bei Privilege Escalation nur Rechte des Benutzers auf Hostsystem und nicht Root
- Problematisch ist das Mapping der Berechtigungen (file and group ownership) im Dateisystem in den User Namespace – auch Netzwerk muss indirekt zugegriffen werden
- Docker-Umgebung unterstützt Rootless mode mit gewissen Einschränkungen<sup>1</sup>
- Alternative: Podman Container-Umgebung von Red Hat (Daemonless Architektur)

<sup>1</sup> <https://docs.docker.com/engine/security/rootless/>

# Rootless Container – User Namespace Mapping



Container Security, Liz Rice



Bei erfolgreichem Container-Escape-Angriff ist man nur „Non-Root“ User auf dem Hostsystem!

# Best Practices - Containerdesign

- Idealerweise Service pro Container – Stichwort Microarchitecture
- Keine Nutzdaten (persistente Daten) im Container speichern
  - Container sind standardmäßig „Immutable Components“
  - Beim Beenden oder neuem Deployment sind alle zur Laufzeit erzeugten Daten weg
  - Für Nutzdaten wird ein externes, persistentes Volume verwendet (*docker -v*)
- Containerverwaltung und Konfiguration sollte über Automatisierungstools erfolgen
  - Jenkins
  - Terraform
  - Ansible



# Best Practices - Dockerfile

- Base Image
  - Erste Zeile des Dockerfiles definiert das Base Image (FROM-Anweisung)
  - Base Image sollte aus einer vertrauenswürdigen Registry stammen
  - Je kleiner das Basis-Image, desto geringer Wahrscheinlichkeit für unnötigen oder schadhaften Code
- Image Tags
  - Vorsicht mit Tags – besser den SHA256-digest zur Prüfung verwenden
- Multi-Stage Builds<sup>1</sup> – Ziel: unnötige Inhalte im endgültigen Image eliminieren
  - Technik: Verschiedene Abschnitte im Dockerfile, die jeweils auf ein anderes Base Image verweisen
  - Erste Stage: alle Pakete, Libs und die Toolchain – nicht zur Laufzeit benötigt
  - Zweite Stage: Umgebung vorbereiten, Dateien kopieren, etc
  - Dritte Stage: Golden Image erstellen mit Lean Base Image

Gutes Schreiben von Docker-Image Definitionen ist komplex!<sup>2</sup>

```
# syntax=docker/dockerfile:1

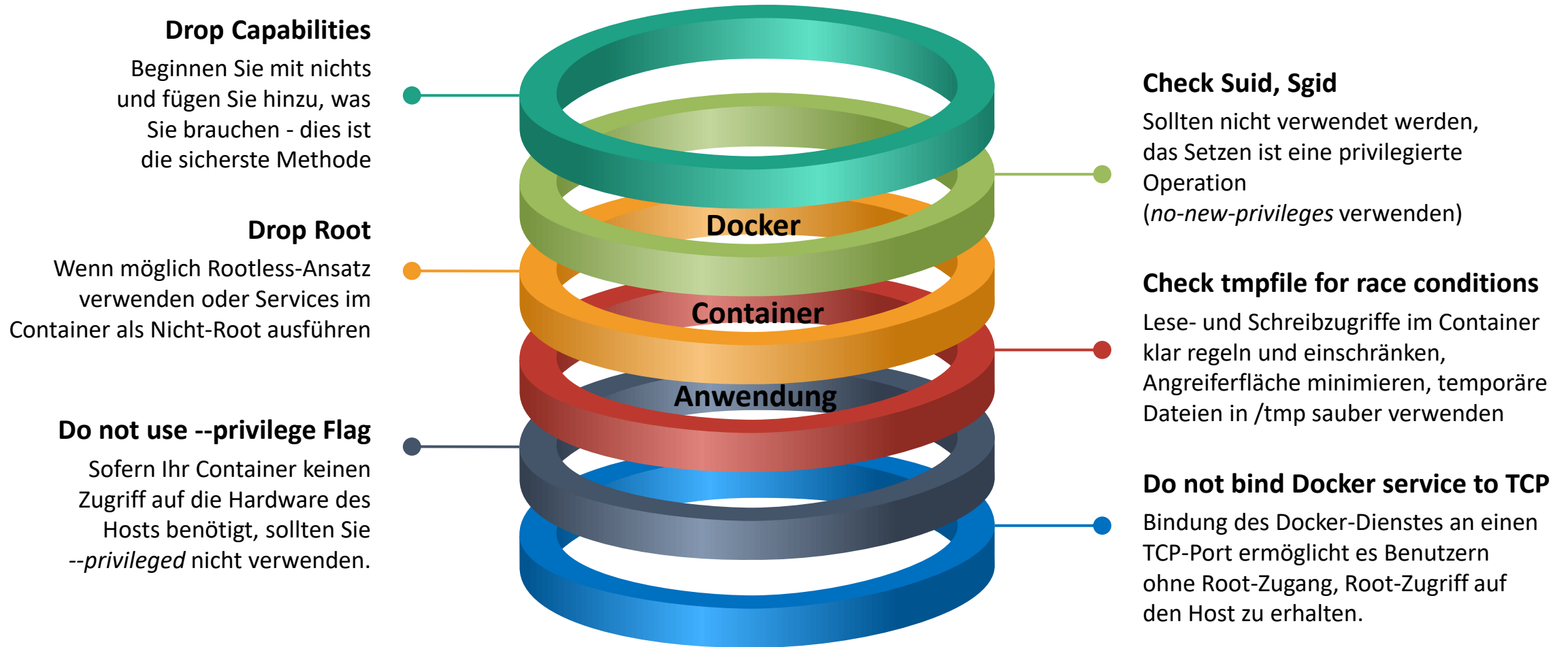
FROM golang:1.16
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go ./
RUN CGO_ENABLED=0 go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app ./
CMD ["/app"]
```

<sup>1</sup> <https://docs.docker.com/build/building/multi-stage/>

<sup>2</sup> [https://docs.docker.com/develop/develop-images/dockerfile\\_best-practices/](https://docs.docker.com/develop/develop-images/dockerfile_best-practices/)

# Best Practices – Betrieb von Container-Anwendungen



# Zusammenfassung

---

- CI/CD ist das Kernstück von DevSecOps und ist letztlich das Automatisierungsframework für alle IT-Sicherheitsaktivitäten
- Es gibt eine Vielzahl von Deployment-Optionen und die Tendenz geht Richtung Cloud-Deployment, insbesondere Container-basierte Deployments
- Container-Sicherheit ist ein komplexes Thema und muss Stück für Stück umgesetzt werden
- Hier haben wir die wichtigsten Techniken zur Container-Absicherung beim Erstellen von Images als auch beim Betrieb von Containern kennengelernt



**Vielen Dank für ihre Aufmerksamkeit!  
Fragen?**