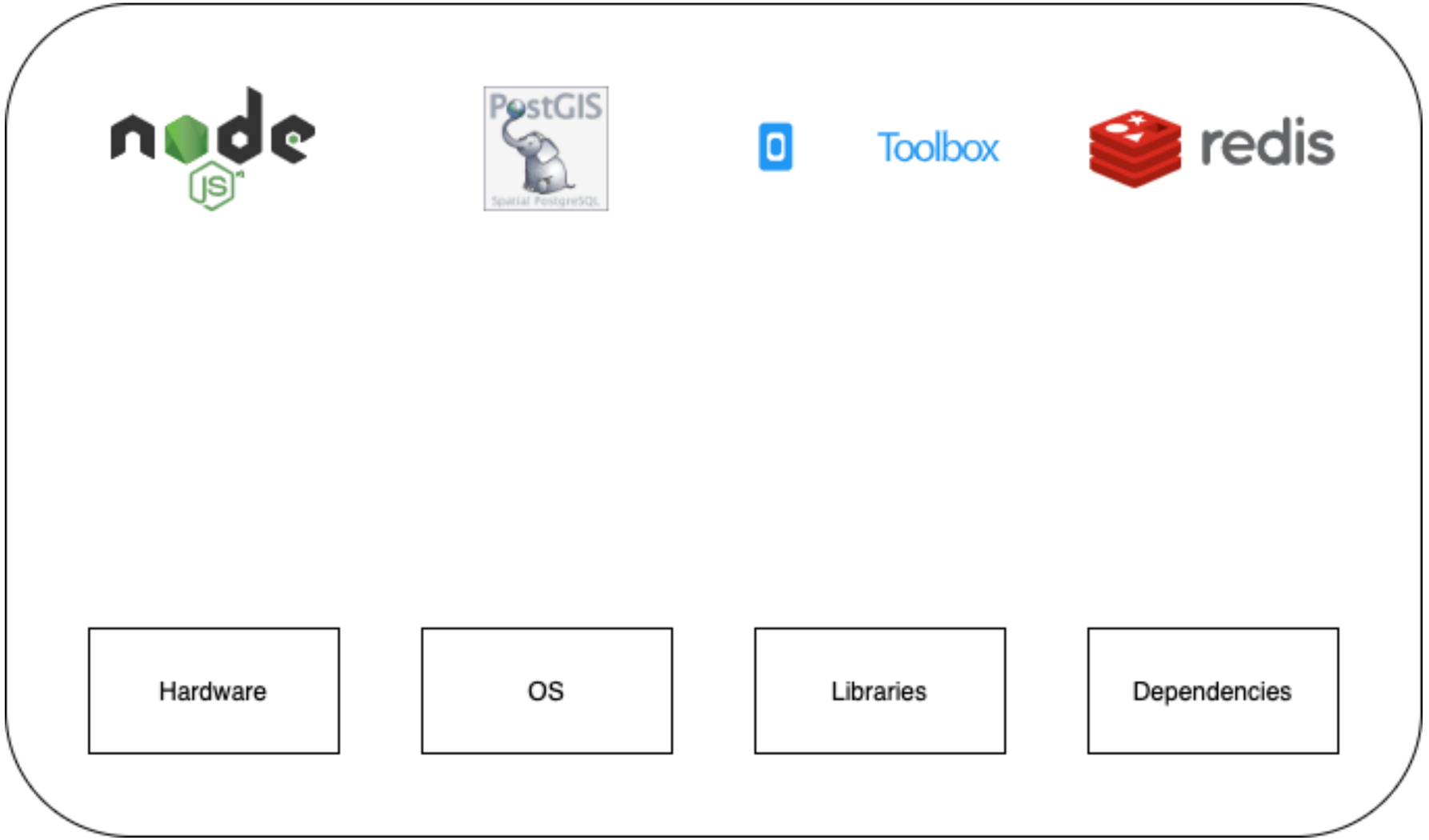
BJÖRN UHLIG
S0554192@HTW-BERLIN.DE
BIG DATA MANAGEMENT UND ANALYTICS IN DATENZENTRIERTEN WISSENSCHAFTEN
WINTERSEMESTER 2019/2020

CONTAINER TECHNOLOGIES

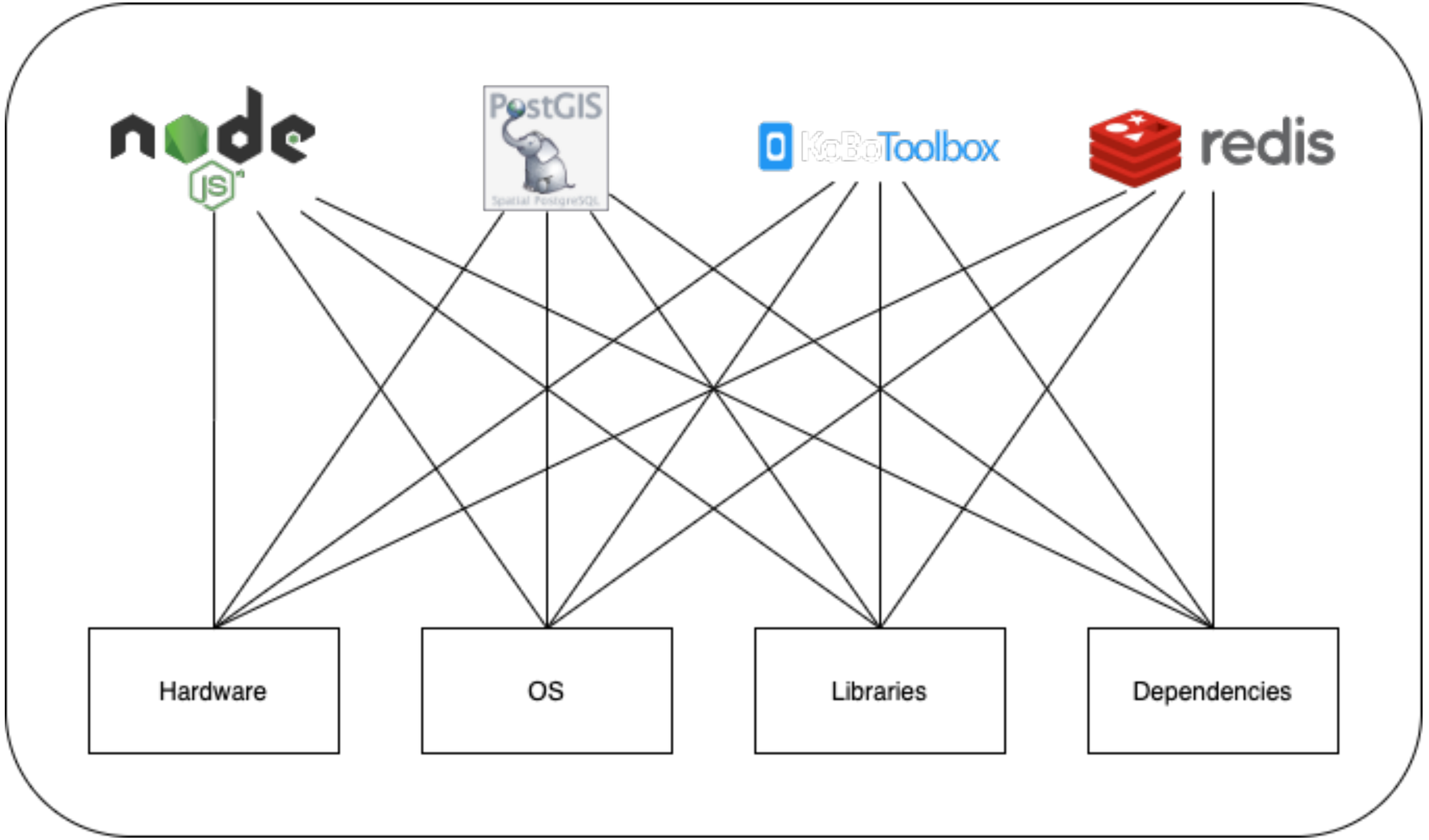
INHALT

1. Problem: Matrix From Hell
2. Solution: Container Technologies
3. OS-Level-Virtualisierung in a Nutshell
4. Container und Security: Docker und Singularity

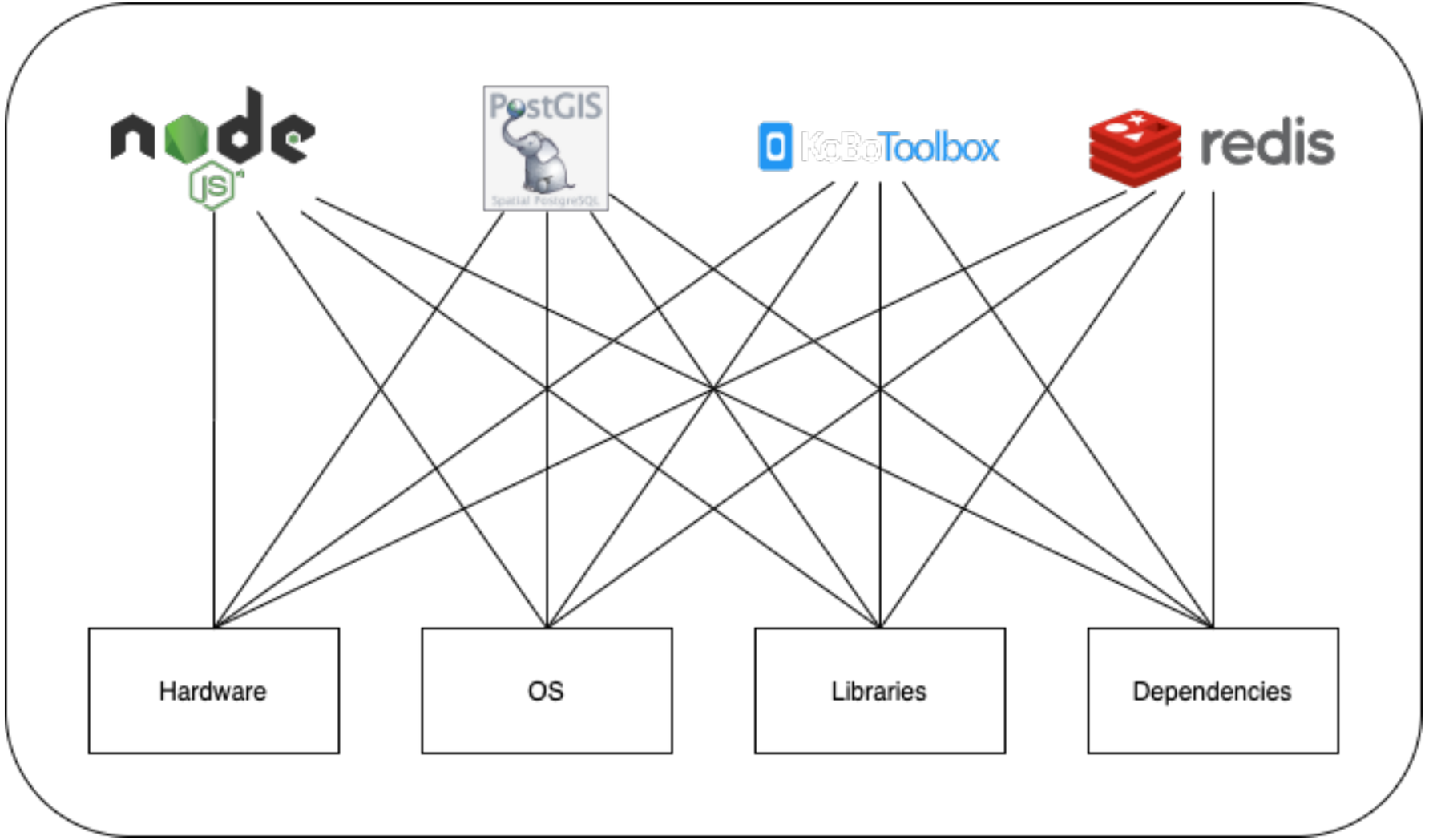
PROBLEM


node
PostGIS
Spatial PostgreSQL
Toolbox
redis
Hardware
OS
Libraries
Dependencies

DEPENDENCIES OVERKILL



MATRIX FROM HELL



SOLUTION: CONTAINER

- Betriebssystem-Virtualisierung (keine volle Virtualisierung)
- Modifizierte Laufzeitumgebung für eine Software, die verhindert, dass die Software auf geschützte Ressourcen zugreift und nur Ressourcen nutzt, die ihr zugewiesen wurden
- Ein Prozess wird abgeschirmt und sieht nicht mehr alle Host Ressourcen
- Anders: Technologie zu Isolation von Softwarekomponenten
- Technologien: chroot, *namespaces*, *croups*, capabilities, …

CHROOT

- 1979: Unix cmd chroot (change root)
- Simuliert einem Prozess, dass ein Pfad / ist (e.g. FTP Server/ Clients: Isolation im Filesystem)
- Kein *security feature*: VirtEnv, Testing, Honeypots (Jail)
- Problem: Network und OS-Prozesse sind nach wie vor durch SysCalls kontrollierbar (root-user Rechte im chroot Verzeichnis)
- Keine Einschränkung von Ressourcen (I/O, CPU, Memory)

JAILS

- Chroot nur sporadisch in kleinen Kreisen verwendet
- BDS fast als einziges System, welches Technologie weiterentwickelt als BDS-Jails (erste Container)
- (Andere Beispiele: Solaris Zones und OpenVZ)
- Anwendungsfälle führten zur Übernahme der Technologie in den Linux-Kernel (cgroups, namespaces, OpenVZ Funktionen (neu geschrieben))

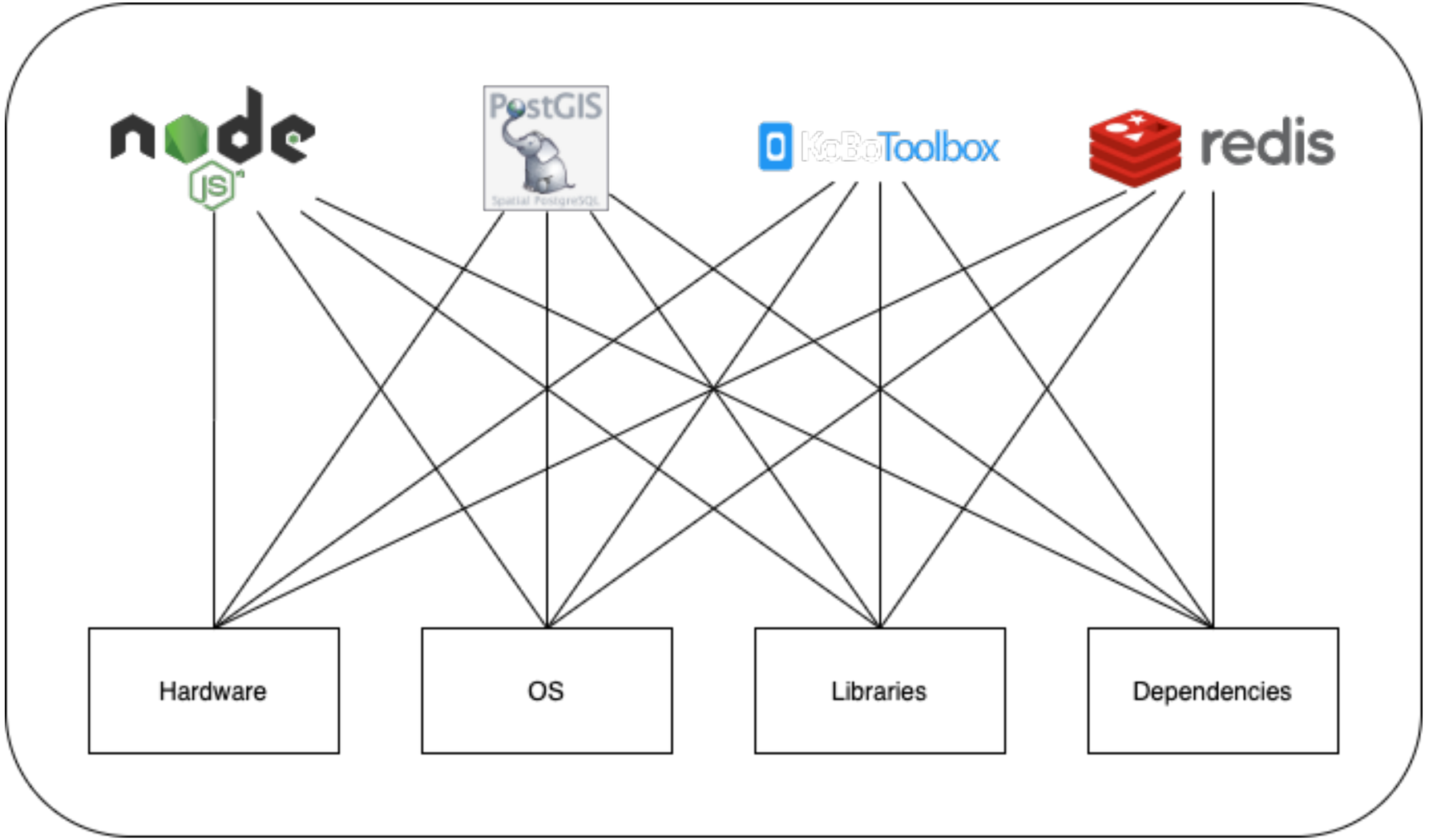
CGROUPS

- Host-OS begrenzt die Ressourcen (CPU, Memory, I/O, …), die ein Prozess (Container) nutzen kann
- Genutzt von nice bis zu OS-Level-Virtualisierung
- Ressourcen-limiting (e.g. Memory limit): Verhindert DOS-Angriffe auf den Host (nur der Container ist betroffen)
- Priorisierung (wie *nice*, CPU-Zeit oder I/O-Bandbreite)
- Accounting (Billing der Ressourcennutzung)
- Kontrolle (freeze, restart, …): Garantiert konsistente Uptime im Falle eines Container-Failure

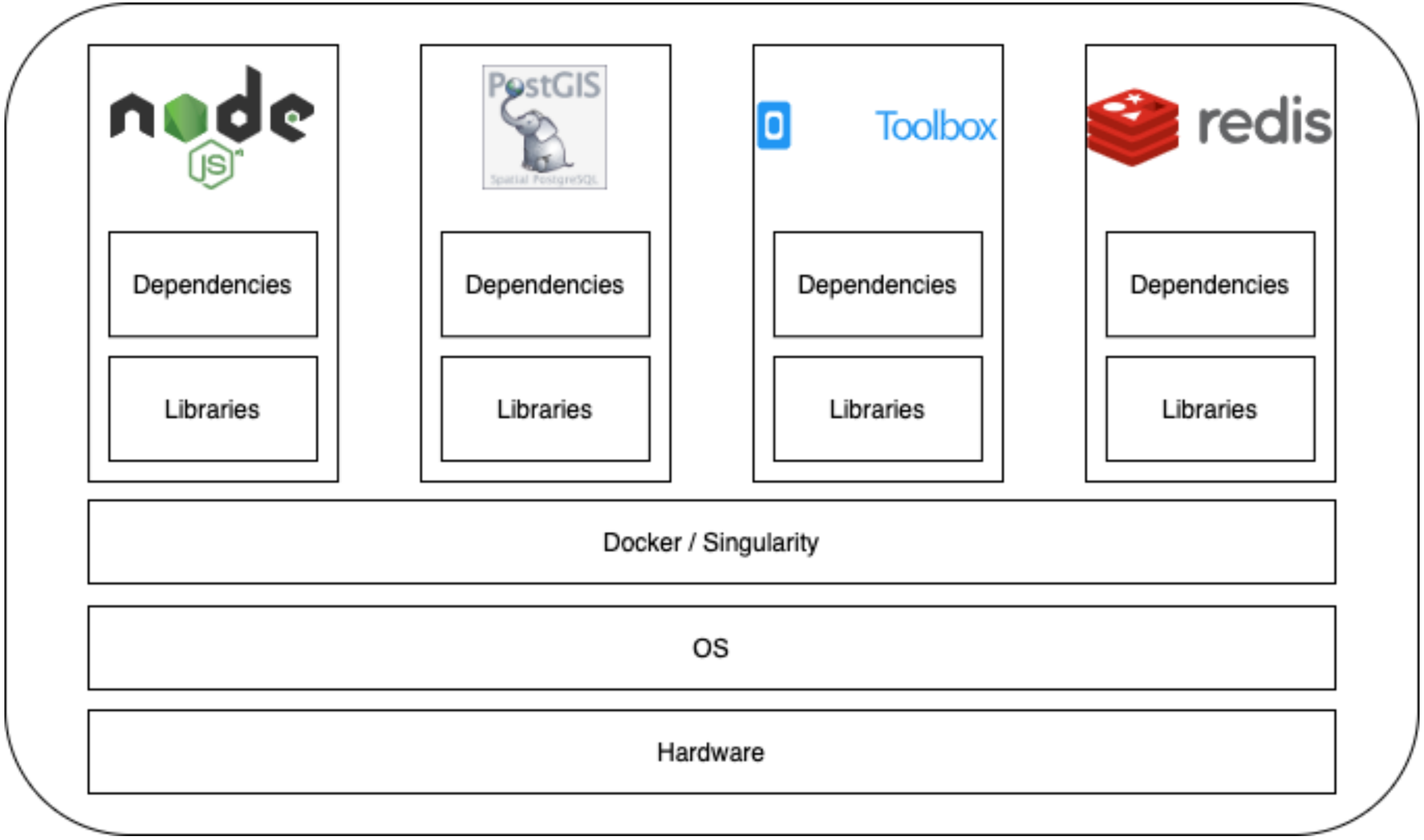
NAMESPACES

- Linux namespaces: mnt, uid, pid, net, ipc, uts, cgroup
- Einfache Anwendung: User namespaces
- Beispiel PID: Ein Prozess sieht nicht die Prozess-IDs von anderen Systemprozessen außerhalb seines Containers
- Beispiel NET: ein network-namespace isoliert Zugriff auf Netzwerk-Interfaces und Konfigurationsdateien. Dies erlaubt die Trennung von Interfaces, Routes und Firewall-Regeln. Container auf einem Host kommunizieren miteinander wie zwei Remote Hosts im Netzwerk (Requests, etc.)

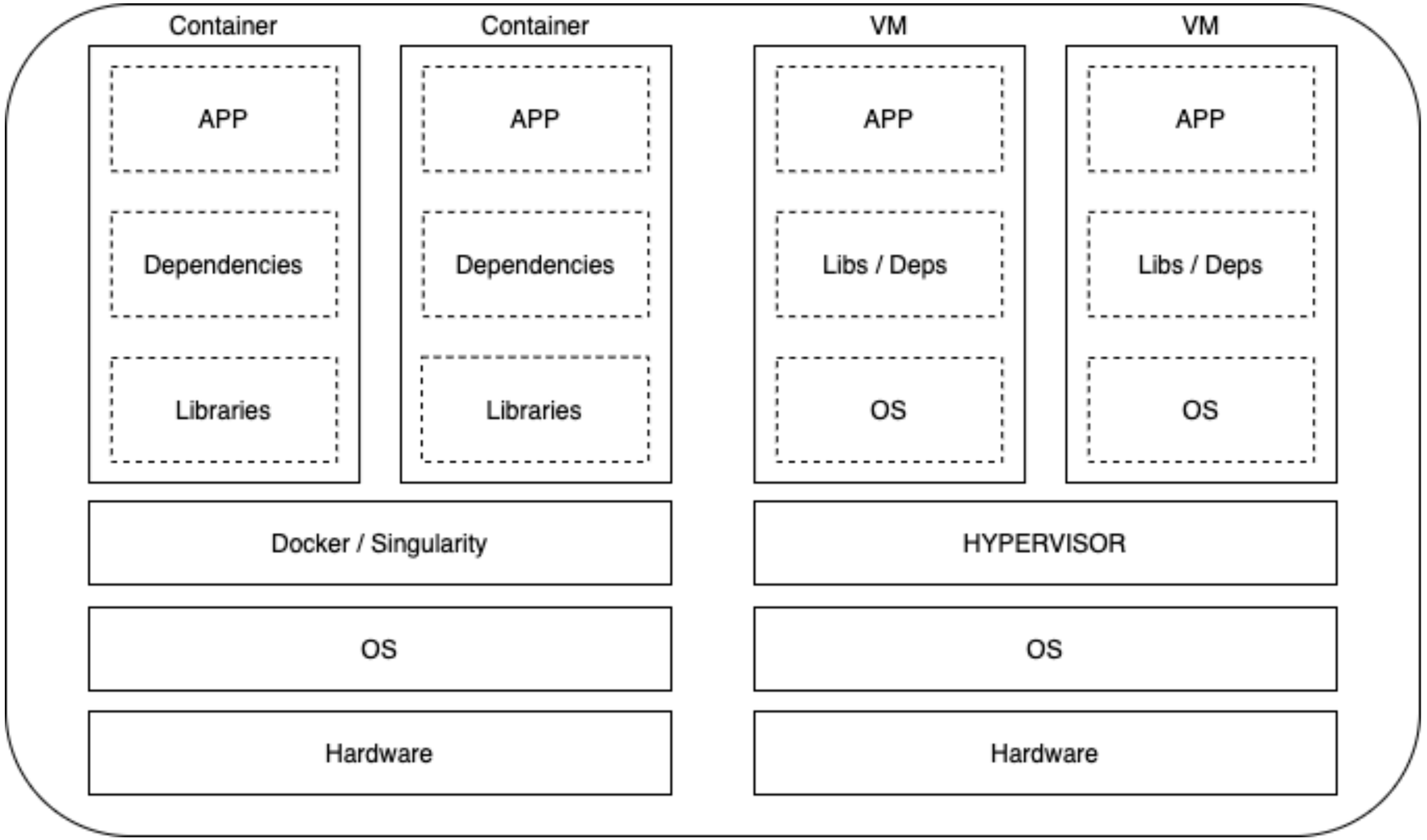
MATRIX FROM HELL: WIE SIEHT DIE LOSUNG AUS?



OS-LEVEL-VIRTUALISIERUNG AUF EINEM HOST



OS-LEVEL-VIRTUALISIERUNG VS FULL VIRTUALIZATION



CONTAINER UND VMS

- Container:

Virtualisierung auf OS-Level

Weniger Isolierung (geteilte Ressourcen)

Containerize Apps

Prozess in Containern nutzen SysCalls auf das Host-OS

Wenig Overhead: Neuer Container startet wie ein Linux Prozess (ms)

- VM:

Volle Virtualisierung

Volle Isolierung (keine geteilten Ressourcen)

Containerize OS-Kernels

SysCalls auf das OS müssen vollständig emuliert werden

Viel Overhead: Neue VM (OS, etc.) startet in Sekundenzeit (s)

CONTAINER: OS-LEVEL VIRTUALISIERUNG

- Container: Von „Shipping Container“, standardisiertes Format für Software-Packaging, die an eine Container-Plattform (Docker, Singularity, rkt, …) passt
- „Ship a container“ von der Entwicklung zum Deployment und die Software verhält sichgleich, überall
- Prozesse in einer Sandbox-Umgebung (wie eine Instanz einer VM)

BEISPIEL: DOCKER CONTAINER

- Was passiert wenn ein Docker Container gestartet wird?

Namespaces und cgroups werden angelegt

Der Prozess sieht nur die ihm zugewiesenen Ressourcen (I/O, CPU, Memory,...)

Der Container erhält seinen eigenen Network-Stack (er kann nicht mit anderen sockets und interfaces anderer Container interagieren

Konfigurationseinstellungen des Host-Systems (Root-Rechte, IPC, ...)

CONTAINER UND SECURITY

- Container verhindern DOS-Angriffe auf das Host-System (nur der Container ist betroffen
- Honeypot: eindringende Malware bemerkt nicht, dass sie in einer Virtuellen Umgebung isoliert ist (Jail)
- Root-Right-Escalation: Jailbreak/Container-Escape
- ! Die Isolierungstechniken wurden nicht im Hinblick auf Security entworfen und weisenSchwachstellen auf. Zugriff auf das Host-Filesystem und Host-OS-Prozesse sind u.U. möglich

CONTAINER UND SECURITY

- Beispiel Docker:

Docker Daemon läuft im Hintergrund und benötigt Root-Rechte

Docker erlaubt geteilte File-Systeme zwischen Host und Guest

Guest kann das File-System und die Prozesse des Host ändern!

Angriffe auf den Docker Daemon sind möglich durch image-loads (Container sind mutable)

etc. ...

! Sicherheitsarchitektur muss erweitert werden (https/certificates/vpn/ checksums von container images...)

CONTAINER UND SECURITY

- Docker Capabilities (Whitelisted privileges):

Deny mount

Deny socket access

Deny filesystem operations

...

= eingeschränkte Root-Privilegien um Root-Escalation zu verhindern

Erfordert viel Aufwand und zusätzliche Konfiguration

(siehe auch: https://docs.docker.com/engine/security/security/)

CONTAINER UND SECURITY

- Singularity

OS-Virtualization Plattform für HPC Environments

Entwickelt mit dem Ziel der Verifizierbarkeit und Sicherheit von Containern

Kryptographische Signaturen, immutable Comtainerimage Format, In-Memory-Decryption

SIF Single File Container Format

CONTAINER UND SECURITY

- Sicherheit in Singularity

Verhinderung von Root-Escalation by Default (unprivilegierter User behält seine Privilegien im Container, d.h. er erhält keine Root-Rechte im Container). Wie? UID/GID wird in den Container geschrieben

Container werden ohne "Substitute User“ erstellt (keine Userwechsel durch sudo, etc.)

Integration over Isolation: Es wird nicht versucht Container komplett zu isolieren (durch Zusatzmaßnahmen; Docker), sondern stärker in den Host integriert. Dadurch wird die Interaktion mit dem Host-System in den Vordergrund gestellt. Isolation kann dann durch das Hostsystem erhöht werden, indem dem Container Optionen übergeben werden (namespaces, prevent mounting, ...).

CONTAINER UND SECURITY

- Sicherheit in Singularity: SIF

Singularity Image Format: Immutable Container Image / Single File

Container Image ist kryptographisch signiert (OpenPGP)

Docker Container in der Docker Registry können verändert werden. Die Identität von Singularity Containern ist kryptographisch gesichert

Root Filesystem eines Singularity Containers kann verschlüsselt werden

*.sif image file wird direkt aus dem Image geladen und verifiziert (Single Step Execution: Weniger Angriffsoberfläche).

Siehe: https://sylabs.io/guides/3.5/user-guide.pdf#page=19&zoom=100,96,286

CONCLUSION

- Container erlauben light-weight Virtualisierung auf OS-Ebene
- Haben wenig overhead und sind daher extrem gut skalierbar (freeze/restart im ms-Bereich)
- Vielseitige Anwendungsgebiete (HPC, Dev, Ops, ...)
- Beschleunigen Build und Deployment Prozesse von Software von Wochen/Monaten zu Tagen/Stunden
- Haben bekannt Security Issues (jedes neue Docker Feature erweitert die Angriffsfläche)

LITERATUR

- Baier, Jonathan [2015]: Getting Started With Kubernetes, Packt Publishing Ltd., Birmingham.
- Holla, Shrikrishana [2015]: Orchestrating Docker, Packt Publishing Ltd., Birmingham.
- Hightower, Kelsey; Beda, Joe; Burns, Brendan [2017]: Kubernetes: Up & Running, O'Reilly, Sebastopol.
- Nickoloff, Jeff []: Docker in Action, Manning, helter Island.
- Rosen, Rami: Namespaces and croups, the basis of Linux containers; Präsentation bei NetDev 1.1: The Technical Conference on Linux Networking (10. - 12. Februar 2016 in Sevilla, Spanien; https://netdevconf.info/1.1/proceedings/slides/rosen-namespaces-cgroups-lxc.pdf, Zugriff am 22. Februar 2020.
- https://robertnorthard.com/jailbreak-escaping-the-container/, Zugriff am 22. February 2020
- https://de.wikipedia.org/wiki/LXC
- https://de.wikipedia.org/wiki/Containervirtualisierung
- Singularity User Guide, https://sylabs.io/guides/3.5/user-guide.pdf, Zugriff am 21. Februar 2020
- https://docs.docker.com/engine/security/security/, Zugriff am 21. Februar 2020.
- https://web.archive.org/web/20081210042915/http://wiki.netbsd.se/How_to_break_out_of_a_chroot_environment