June 16, 2023, CCE Colloquium, Kyoto University

2023年6月16日 京都大学 大学院 情報学研究科 通信情報システムコース 談話会

The internals and the latest trends of container runtimes

コンテナランタイムの仕組み及び最新の動向

Akihiro Suda (NTT)

akihiro.suda.cz@hco.ntt.co.jp

Self-Introduction

NTT

2008-2012: Department of Information Science, Kyoto University (Yuasa Lab.)
2012-2014: Graduate School of Informatics, Kyoto University (Takagi Lab.)
2014-Present: Software Innovation Center, NTT Corporation

A maintainer of several container-related open source projects:

Moby (≈ Docker)
The open source upstream of Docker

Docker : Moby ≈ Chrome : Chromium

containerd
The underlying runtime of Docker and Kubernetes

runc
The low-level runtime below containerd

And also BuildKit, OCI Runtime Spec, nerdctl (Founder), Lima (Founder), etc.

Topics

- Introduction to containers
- Internals of container runtimes
- Latest trends in container runtimes

Introduction to containers

What are containers?

- Lightweight methods to isolate filesystems, CPU resources, memory resources, system permissions, etc.
- Not really well-defined, actually (discussed later)
- Pros and cons compared to virtual machines:

Pros 😀

- Low overhead
  - No hardware emulation
  - The kernel is shared with the host operating system
- Direct access to host filesystems, networks, GPUs, etc. (when permitted to do so)

Cons 😢

- Can't run Windows on Linux hosts
- Can't change kernel config
- Weak isolation

Docker

- The most popular container engine
- Supports Linux and Windows
  (But Windows is out of the scope of my talk)

A pure nginx image,
without systemd, sshd, ...

```
$ docker run -p 8080:80 -v .:/usr/share/nginx/html nginx:1.25
```

Forwards the TCP port:
8080 (host) → 80 (container)

Mounts the current directory on the host onto
`/usr/share/nginx/html` in the container

- Using Docker is assumed for the most part of this talk (with its default config)
- Non-Docker containers will be discussed later too;
  most of them are very similar to Docker under the hood

Docker

- An image can be built using a language called `Dockerfile`

```
FROM debian:12
RUN  apt-get update && apt-get install -y openjdk-17-jre
COPY myapp.jar /myapp.jar
CMD  ["java", "-jar", "/myapp.jar"]
```

```
$ docker build -t myimage -f Dockerfile .
$ docker run myimage ...
```

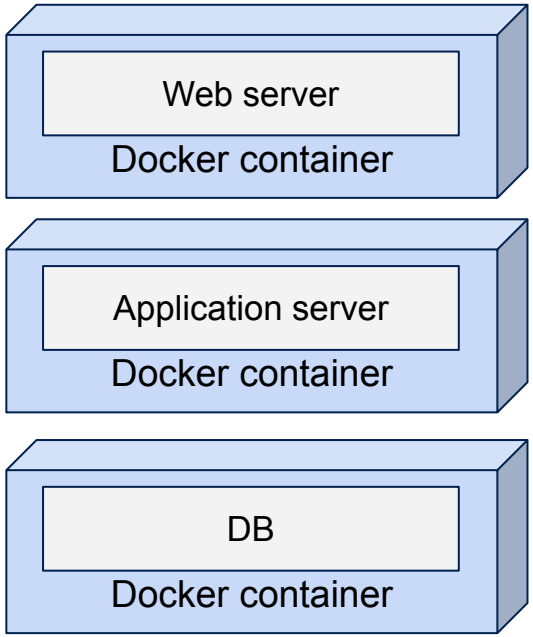
- The built image can be pushed to registries like Docker Hub

```
$ docker login example.com
$ docker push example.com/myimage
```

Docker Compose

- Composes containers using a declarative YAML

```
services:
  web:
    image: nginx:1.25
    ports: 8080:80
  app:
    image: example.com/myimage
  db:
    image: postgresql:15.3
 ...
```

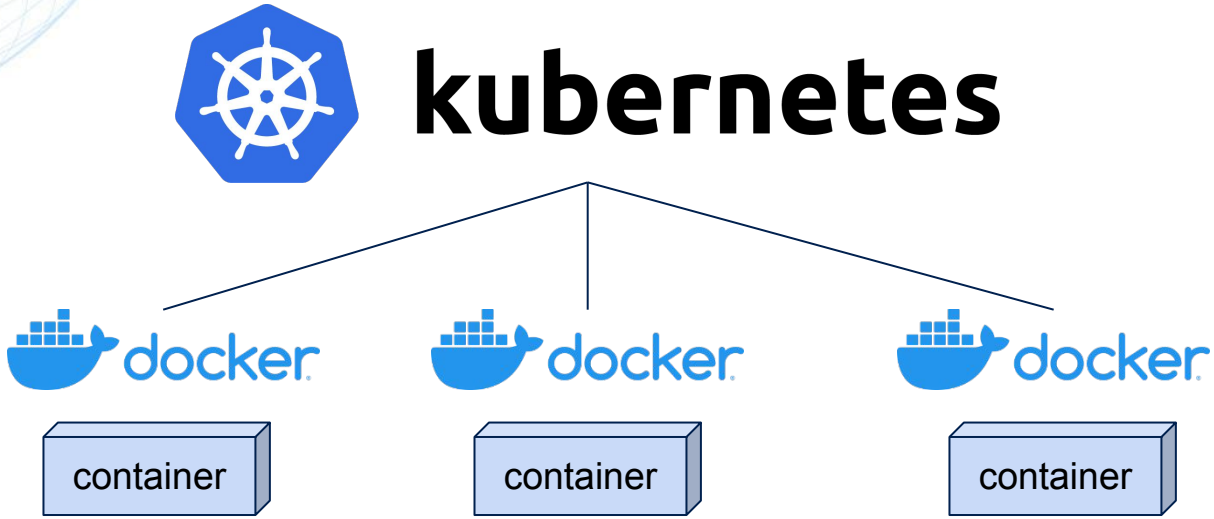



```
$ docker compose up
```

Kubernetes

- Kubernetes clusterizes container hosts
  (such as, but not limited to, Docker hosts)

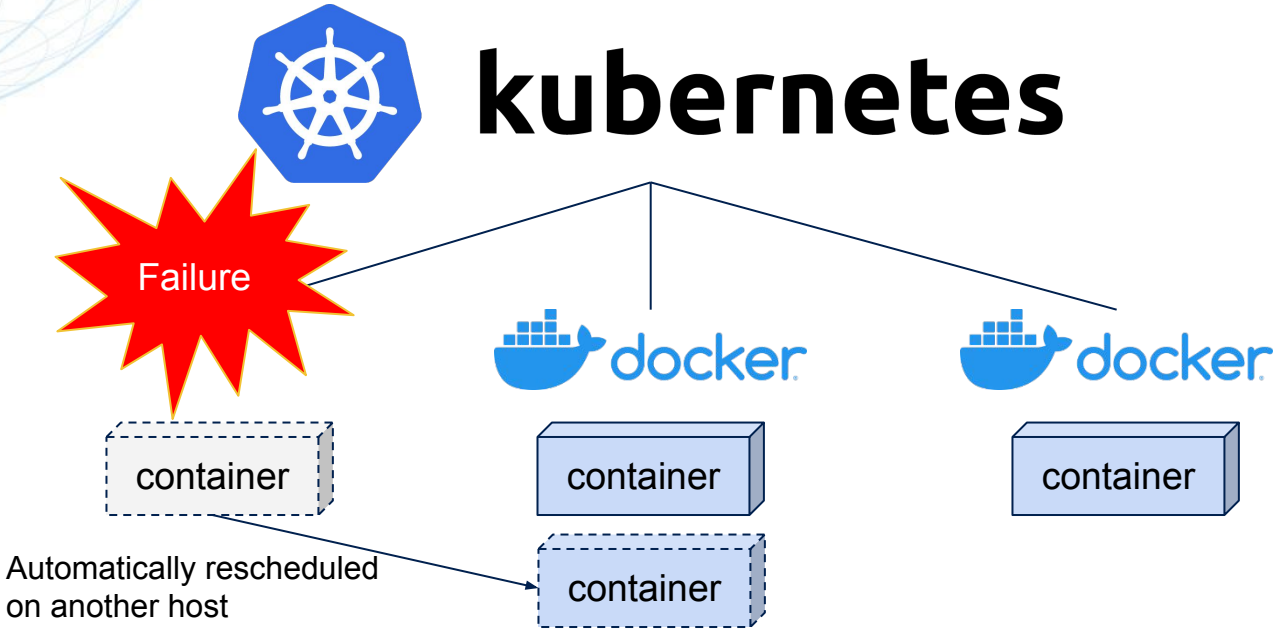
- Provides load balancing & fault-tolerance across container hosts



Kubernetes

- Kubernetes clusterizes container hosts
  (such as, but not limited to, Docker hosts)

- Provides load balancing & fault-tolerance across container hosts



Docker vs pre-Docker containers

- Docker (2013) wasn't the first container platform

- 1999: FreeBSD Jail
- 2000: Virtual Environment system for Linux (precursor to Virtuozzo and OpenVZ)
- 2001: Linux Vserver
- 2002: Virtuozzo
- 2004: BSD Jail for Linux
- 2004: Solaris Containers
- 2005: OpenVZ
- 2008: LXC
- 2013: Docker

Apparently, the term "container" was coined this time

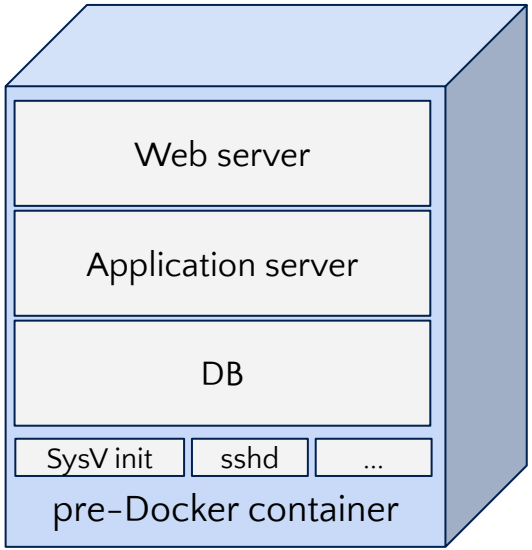
The basis of the modern container ecosystem was almost established by 2008

Docker was just a wrapper for LXC until 2014

Other precursors to containers: chroot in UNIX 7th Ed. (1979), chroot with a security enhancement in System III (1982), Plan9 namespaces (1992)

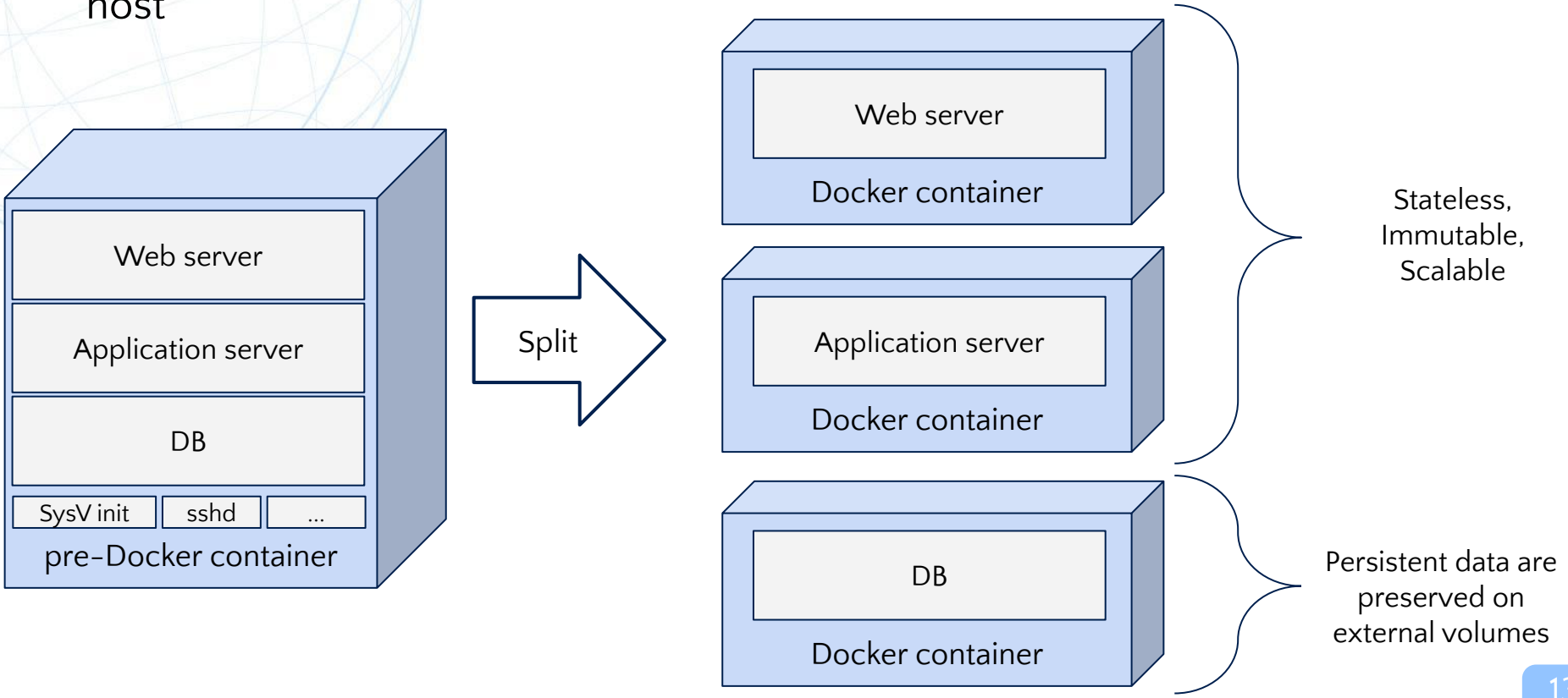
Docker vs pre-Docker containers

- Pre-Docker containers had focused on mimicking an entire VM, with SysV init, sshd, syslogd, etc., inside it
- For a single-host environment, it was often common to put Web server + Application server + DB in a single container



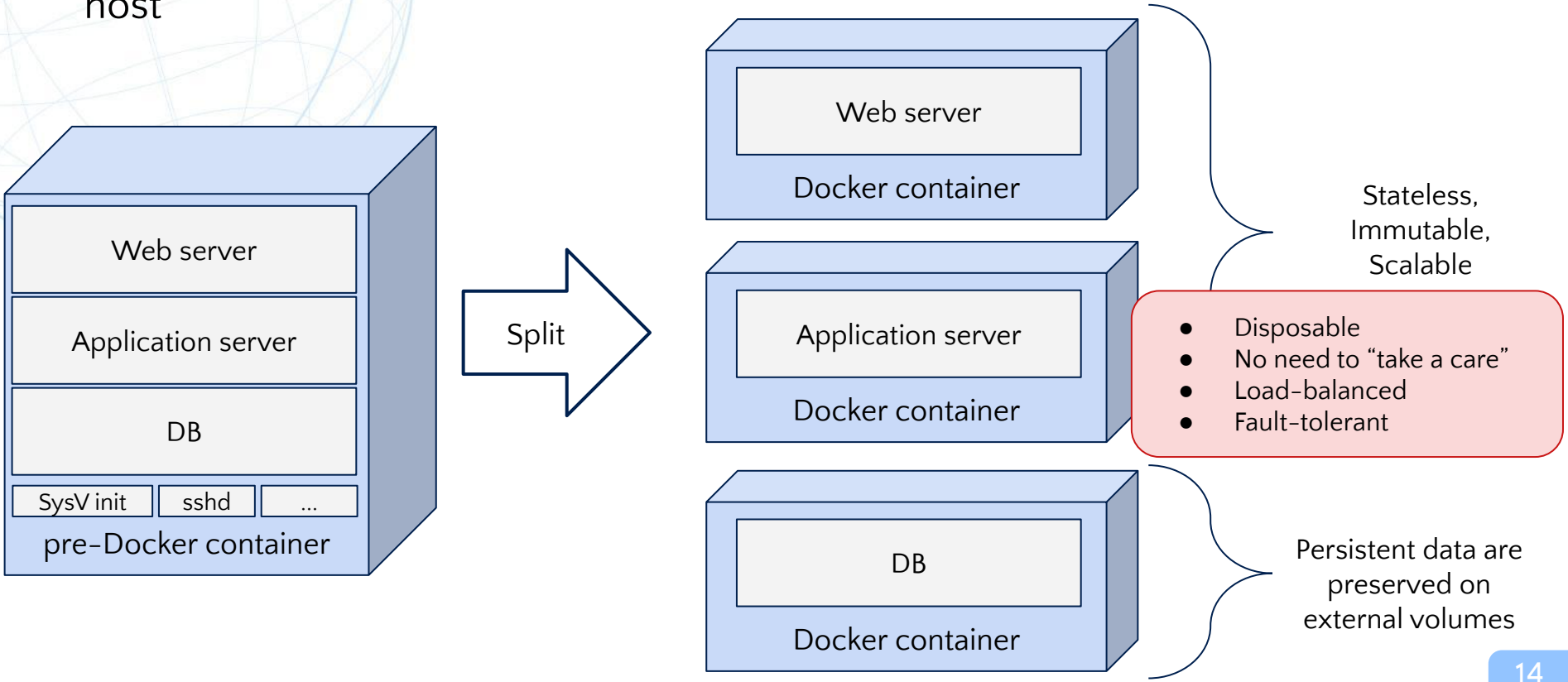
Docker vs pre-Docker containers

- In the case of Docker, services are split to separate containers, even on a single host



Docker vs pre-Docker containers

- In the case of Docker, services are split to separate containers, even on a single host



Docker vs pre-Docker containers

NTT

- Filesystem images can be shared with other users on Docker Hub
- `docker push` to Docker Hub, just like `git push` to GitHub

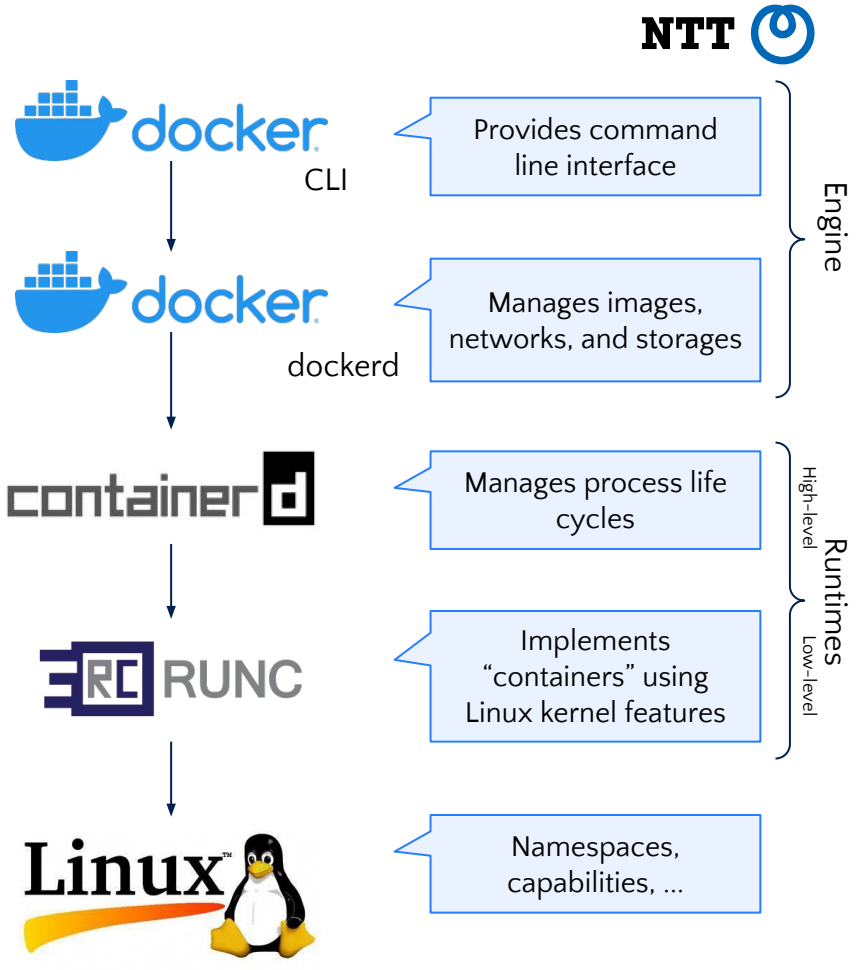
→ Docker became the de facto standard

Internals of container runtimes

Using Docker (v24) is assumed, but most parts are applicable to non-Docker containers too

Docker under the hood

- Consists of client (`docker` CLI) and daemon (`dockerd`)
- `dockerd` talks to `containerd` to manage process life cycles
  (and also images, since Docker v24, depending on config)
- `containerd` executes `runc` to create "containers", which are implemented by several kernel features such as namespaces and capabilities
- No "container" object exists in Linux kernel



CLI: Command Line Interface

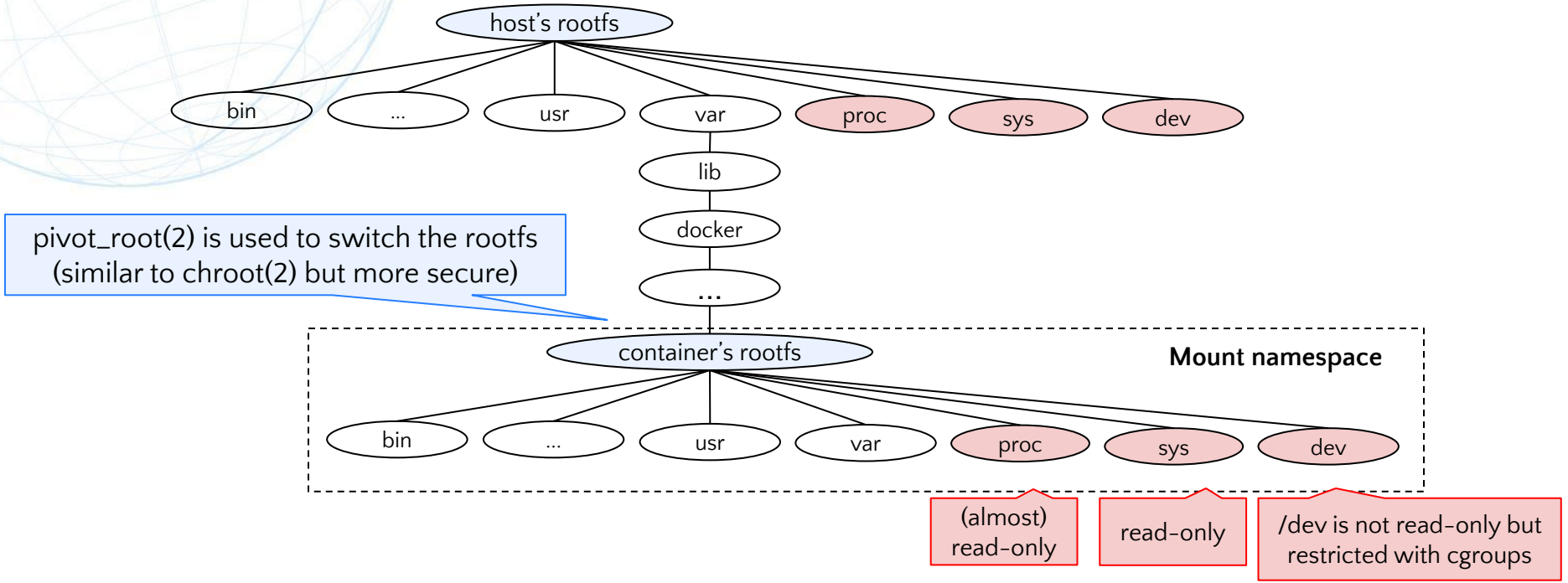
Engines can be also classified as runtimes in a broader sense

“Container” technologies offered by the kernel

- Mount namespaces
  - Isolates the rootfs from the host (with `pivot_root(2)`)
- Network namespaces
  - Allows assigning dedicated IP addresses to containers
- PID namespaces
  - Hides the host processes from containers
- Cgroups
  - Limits container resources such as memory and CPU
- Capabilities & Seccomp
  - Limits syscalls
- AppArmor XOR SELinux
  - Strictly limits file accesses

Mount namespaces

- Isolates the filesystem view from the host (and other containers)



Mount namespaces were merged in Linux v2.4.19 (2002), enhanced in Linux v2.6.16 (2006); Inspired by Plan9's namespaces

Mount namespaces + File protections

- Mount namespaces don't protect host system files by themselves

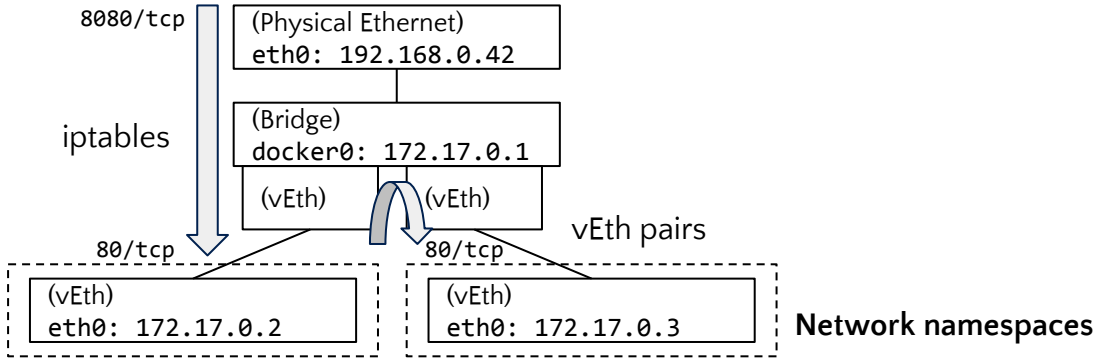
- Read-only bind mounts:
  - Remount `/proc/sys` as a read-only to prohibit sysctl

- Masks:
  - Mount `/dev/null` over `/proc/kcore` to hide the RAM
  - Mount an empty `tmpfs` over `/sys/firmware` to hide the firmware data

- Accesses to `/dev` are restricted by Cgroups Device Controller (discussed later)

Network namespaces

- Allows assigning dedicated IP addresses to containers
- Containers can talk to each other by IP on the same host, via a bridge
- Container ports can be exposed to the Internet via iptables
- Multi-host networking can be implemented by combining network namespaces with VXLAN, etc.



Network namespaces were merged in Linux v2.6.24 (2008)

Network namespaces aren't just for networks

- Network namespaces isolate abstract UNIX sockets too

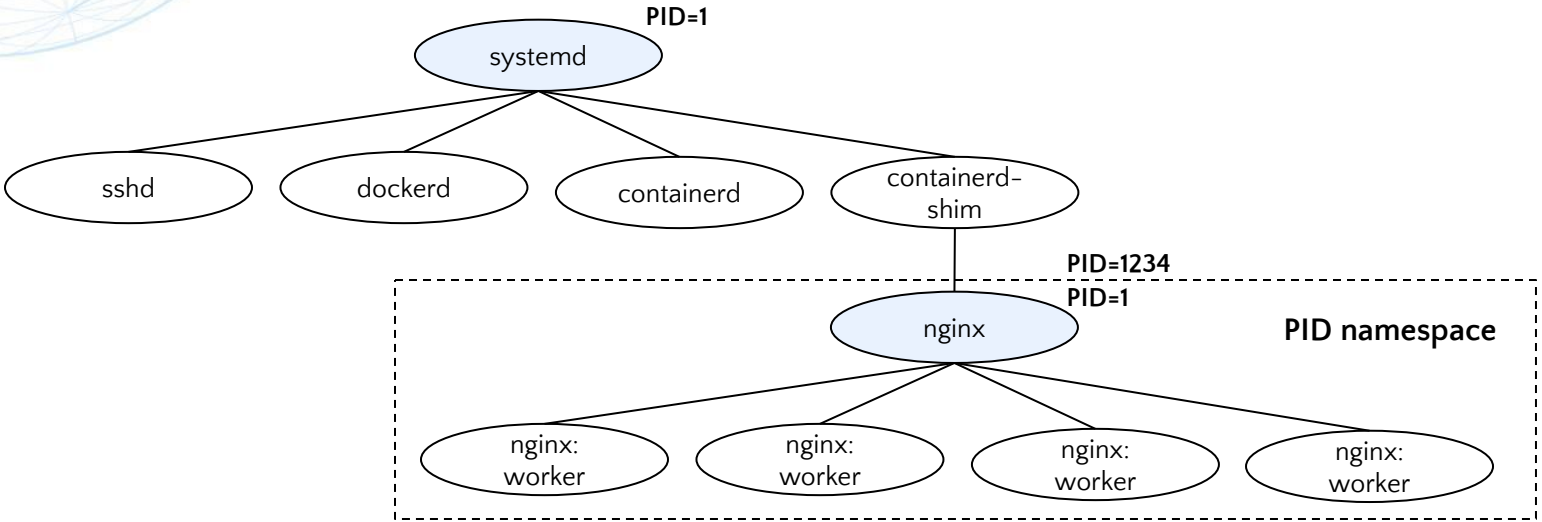
- Abstract UNIX sockets:
  - UNIX sockets but their paths begin with `\0` (`NUL`)
  - Not visible as named files
  - Used by dbus, ibus, irqbalance, iscsid, LXD, multipathd, X Window System, etc. (depending on configurations)
  - Historically also used by systemd, upstart, containerd, etc.

- Disabling network namespaces (`docker run --net=host`) may result in allowing a container to connect to a system daemon on the host (especially when the container is running as the root)

https://medium.com/nttlabs/dont-use-host-network-namespace-f548aeeef575 (My own blog)

PID namespaces

- Hides processes of the host and other containers
- PID=1 is an application, not systemd (usually)
- No sshd, journald, etc. in containers (usually)



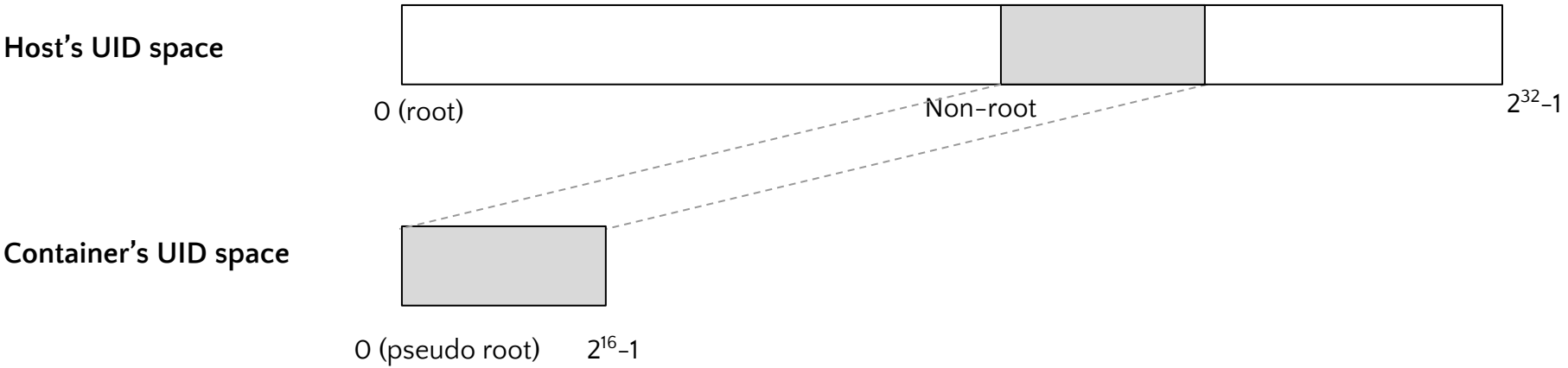
PID namespaces were merged in Linux v2.6.24 (2008)

(Optional) User namespaces

Don't confuse this with "User space" (the antonym of "kernel space")

NTT

- Maps a non-root user to the pseudo "root" in a container
- Pretends to be the root in the container (`apt-get`, `dnf`, …)
- Just a non-root user outside the container
- Mitigates potential container breakout attacks



User namespaces were merged in Linux v2.6.23 (2007), enhanced in Linux v3.8 (2013)

Other namespaces

- IPC namespaces
  - Isolates System V inter-process communication objects, etc.
- UTS namespaces
  - Isolates the hostname and the domainname
  - “UTS” (Unix Time Sharing system) sounds like a misnomer
- (Optional) Cgroup namespaces
  - Isolates `/sys/fs/cgroup` hierarchy
- (Optional) Time namespaces
  - Isolates clocks
  - Not supported by most container implementations yet

Cgroups

- Imposes several quotas:
  - CPU
  - Memory
  - Block I/O
  - Number of processes

- Filesystem quota is not a part of cgroups

- Also controls access to device nodes
  - Allowed by default: `/dev/null`, `/dev/zero`, `/dev/urandom`, ...
  - Disallowed by default: `/dev/sda` (disk devices), `/dev/mem`, ...

Cgroups were merged in Linux v2.6.24 (2008)

Capabilities

- The root privilege can be decomposed to 64-bit capability flag set

- Retained by default: (Docker v24)
  - Bit 0: `CAP_CHOWN` : for chown
  - Bit 10: `CAP_NET_BIND_SERVICE` : for binding TCP ports below 1024, etc.
  - Bit 13: `CAP_NET_RAW`: for old `ping` implementations that craft raw Ethernet packets
    - Dangerous, as it allows impersonating to be another host
    - Expected to be dropped by default in future

- Dropped by default: (Docker v24)
  - Bit 12: `CAP_NET_ADMIN`: for disallowing reconfiguration of iptables, etc.
  - Bit 21: `CAP_SYS_ADMIN`: for disallowing reconfiguration of mounts, etc.

Capabilities were merged in Linux v2.1.92 (1998); Based on an abandoned draft of IEEE 1003.1e (aka POSIX.1e)

(Optional) Seccomp

- Allows specifying an explicit allowlist (or denylist) of syscalls
  - About 350 syscalls are allowed by default in Docker v24
- Defense of depth; used in conjunction with capabilities
- Kubernetes does not use seccomp by default for compatibility sake
  (Unless `seccompDefault: true` is specified in `KubeletConfiguration`)

Seccomp was merged in Linux v2.6.12 (2005), enhanced in Linux v3.5 (2012)

(Optional) AppArmor XOR SELinux

- These LSMs provide further fine-grained configuration knobs
- Mutually exclusive; one is chosen by host OS distributors (not by container image distributors)
- **AppArmor** is chosen by Debian, Ubuntu, SUSE, etc.
- **SELinux** is chosen by Fedora, Red Hat Enterprise Linux, and similar host OS distributions

LSM: Linux Security Module

(Optional) AppArmor

- The default profile almost just overlaps with capabilities, mount masks, etc. (Defense of depth)
  - `deny @{PROC}/kcore rwklx`
  - `deny mount`
  - `ptrace (trace,read,tracedby,readby) peer=docker-default`
  - …

- Custom settings can be added to the profile for further security
  - `deny /** w` : Completely prohibits writing files

AppArmor was originally known as SubDomain (c. 2000); Merged in Linux v2.6.36 (2010)

(Optional) SELinux

- Similar to AppArmor, but takes a different approach
  - **AppArmor**: checks file path strings
  - **SELinux**: checks xattrs (extended attributes recorded in the filesystem)

- **Type Enforcement (TE)**: Protects the host from containers

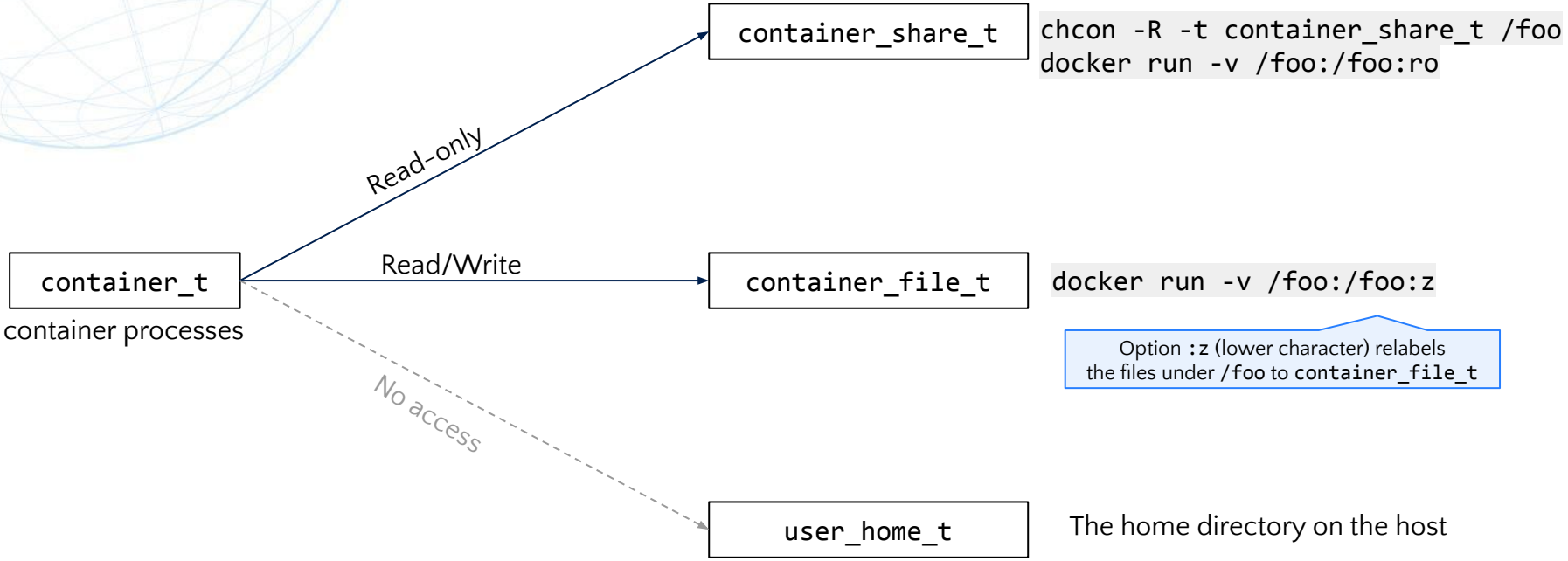
- **Multi-category Security (MCS)**: Protects a container (as well as the host) from another container

- SELinux also supports **Multi-level Security (MLS)** and **Role-Based Access Control (RBAC)**, but these are rarely utilized for containers

SELinux was merged in Linux v2.6.0-test3 (2003)

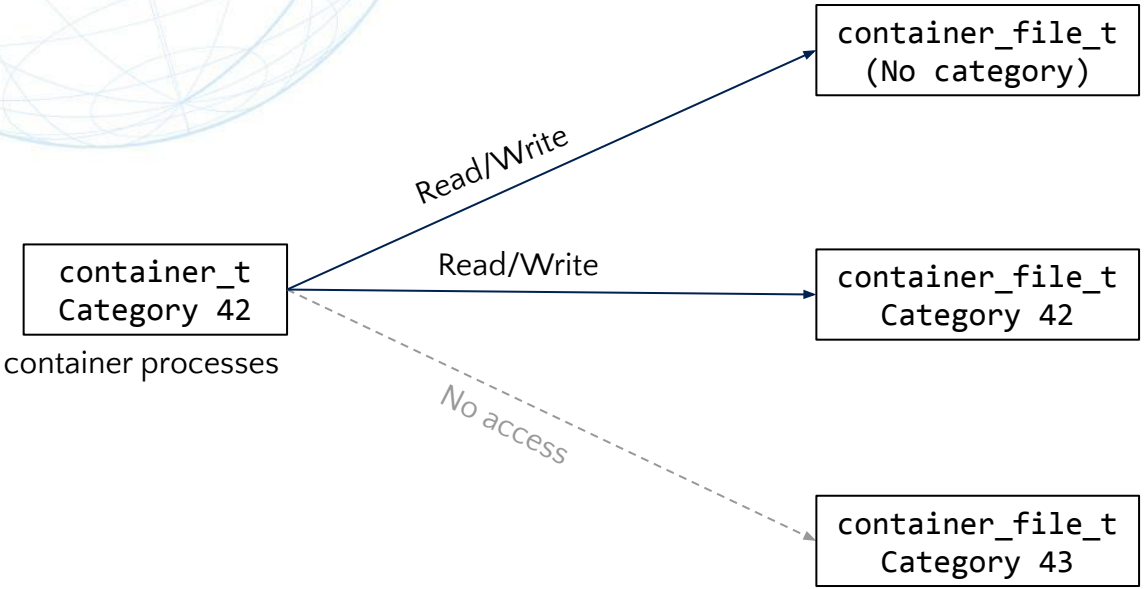
(Optional) SELinux: Type Enforcement (TE)

- Container processes run with the process type (aka “domain”) `container_t` to limit file accesses



(Optional) SELinux: Multi-category Security (MCS)

- Container processes also have category numbers for isolation across containers



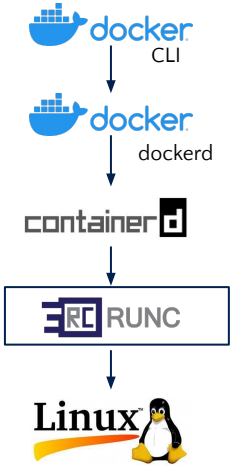

```
docker run -v /foo:/foo:Z
```

Option `:Z` (upper character) relabels the files under `/foo` to have the category that corresponds to the container

Files for another container

Specifications

- Configuration knobs for these runtime components are standardized as the OCI Runtime Spec
  - Defines the structure of OCI runtime bundles: `config.json` + `rootfs/`
  - High-level runtimes (containerd) produce OCI runtime bundles
  - Low-level runtimes (runc) consume OCI runtime bundles



- Aside from the OCI Runtime Spec, OCI also provides:
  - **OCI Image Spec**: defines JSON and tar.gz files for archiving container images
  - **OCI Distribution Spec**: defines HTTP REST API for distributing OCI Image Spec blobs

OCI: Open Container Initiative (Don't confuse with "Oracle Cloud Infrastructure")

What about Docker for Mac/Win ?

NTT

- **Docker for Mac**
  - **Linux containers**: Supported, but under the hood it just runs Linux VM
  - **macOS containers**: Never supportable, unless Apple implements containers for macOS

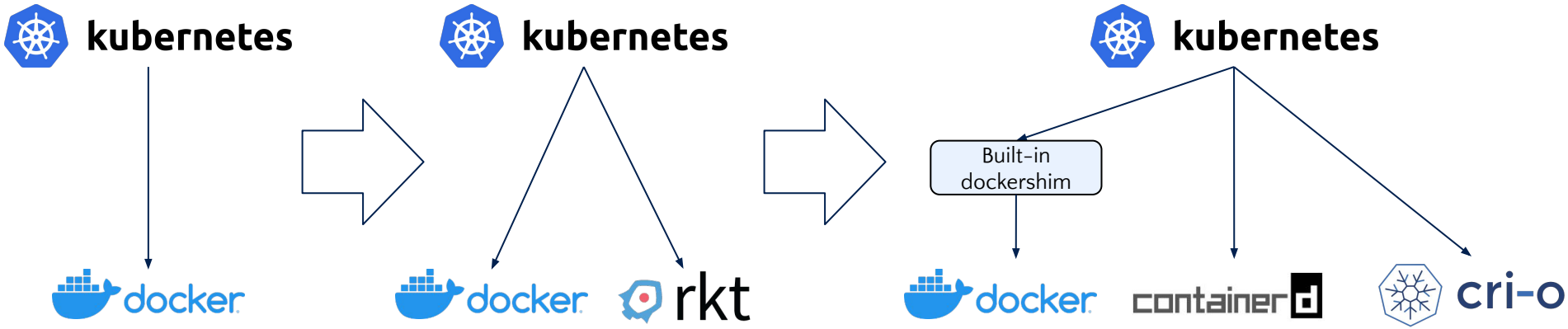
- **Docker for Windows**
  - **Linux containers**: Supported, but under the hood it just runs Linux VM
  - **Windows containers**: Supported, natively

Latest trends in container runtimes

- Alternatives to Docker
- Running containers on Mac
- Docker being refactored
- Lazy-pulling
- User Namespaces
- Rootless Containers
- Kata Containers
- gVisor
- WebAssembly

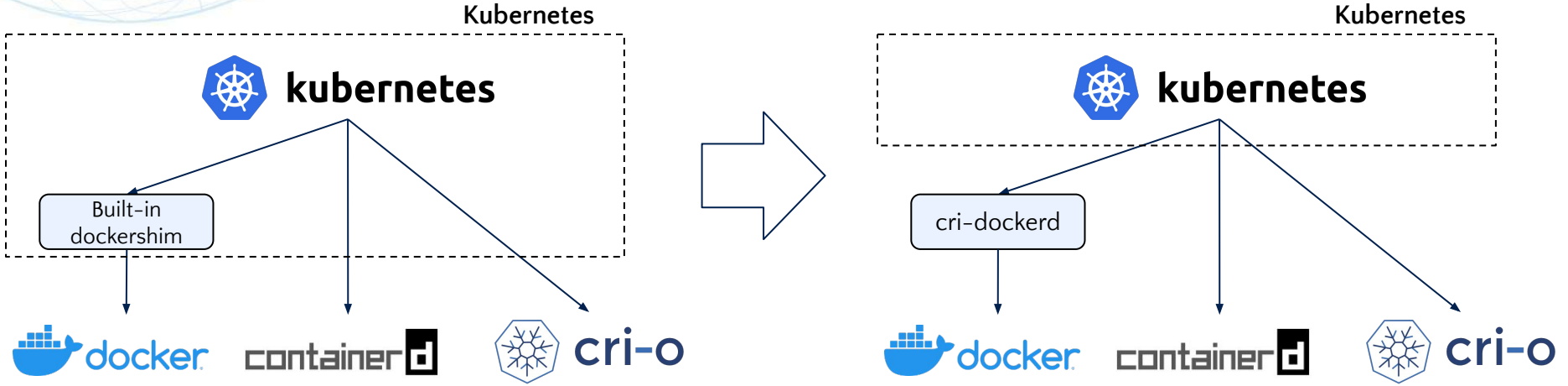
Alternatives to Docker (as Kubernetes runtimes)

- **Kubernetes v0.2 (2014)**: Docker was the only supported runtime
- **Kubernetes v1.3 (2016)**: Introduced support for rkt as an alternative container runtime (rkt was retired in 2019)
- **Kubernetes v1.5 (2016)**: Introduced the Container Runtime Interface



Alternatives to Docker (as Kubernetes runtimes)

- **Kubernetes v1.24 (2022)**: Dropped the *built-in* support for Docker
  - Docker still continues to work for Kubernetes, via `cri-dockerd`
  - But Docker is now seeing less adoptions for Kubernetes



More info: https://medium.com/nttlabs/docker-to-containerd-4f3a56e6f2b6 (My own blog)

Alternatives to Docker (as Kubernetes runtimes)

NTT

- **containerd**
  - **Adopters**: Amazon Elastic Kubernetes Service, Azure Kubernetes Service, Google Kubernetes Engine, k3s, etc.
  - Originally made for Docker in 2015
  - Supports Kubernetes too since 2017
  - Focuses on extensibility

- **CRI-O**
  - **Adopters**: Red Hat OpenShift, Oracle Container Engine for Kubernetes, etc.
  - Solely made for Kubernetes in 2016
  - Focuses on simplicity

CRI-O was originally known as OCID

Alternatives to Docker (as CLI)

- Kubernetes has become the standard for multi-node production clusters
- Users still want Docker-like CLI for building and testing containers locally on their laptops
- Runtime developers also want Docker-like CLI for implementing and experimenting new features
  - It is often hard to propose new features to Docker and Kubernetes
  - Developers want their own “lab” platform to incubate new features

CLI: Command Line Interface

Alternatives to Docker (as CLI)

- Podman (2018–): Docker-compatible standard container engine
  - Daemonless
  - Often confused with CRI-O (CRI API daemon)
  - Shares data with CRI-O (`podman ps --external`)
  - Manages pods as well as containers, but most users seem to just use Podman for non-pod containers
    - **Pod**: a set of containers that share the same network namespace and data volumes, etc. on the same host for efficient communication

Podman was originally known as “kpod” (2016); it was written as a debugging tool for ocid (later CRI-O)

Alternatives to Docker (as CLI)

- nerdctl (2020-): contaiNERD ConTroL
  - Docker-compatible CLI for containerd
  - An official subproject of containerd (non-core)
  - Made for experimenting new features, ahead of Docker
    - Lazy-pulling (explained later)
    - Faster rootless containers (explained later)
    - ...
  - Also useful for debugging Kubernetes nodes that are running containerd:

```
nerdctl --namespace=k8s.io ps
```

More info: https://medium.com/nttlabs/nerdctl-v1-0-fb6bf8e1b0b (My own blog)

Solutions for running containers on Mac

- Docker for Mac/Win is no longer free[-as-in-beer] since 2021
  - It was free (no charge) until then, but was never free software (open source software)

- Windows users can just run the free (opensource) version of Docker (Apache License 2.0) in WSL2
  - No GUI though

- No equivalent for macOS users so far

Solutions for running containers on Mac

- Lima (2021-): Linux virtual machines for running containerd
  - Similar to WSL2 but for macOS hosts, using QEMU
  - Automatic port forwarding with macOS host
  - Automatic file system sharing with macOS host
  - Originally made for promoting experiments on containerd + nerdctl
  - For that sake, containerd + nerdctl is the default runtime
  - Supports Docker and Podman too, optionally
  - No GUI; not a full alternative to Docker for Mac

More info: https://medium.com/nttlabs/lima-is-now-a-cncf-project-a7affde4f03c (My own blog)

Solutions for running containers on Mac

NTT

- colima (2021–): “Containers in Lima”
  - Provides an alternative CLI for running containers in Lima
  - Slightly misnomer, as Lima itself was already written for running containers
  - colima uses Docker by default, while Lima uses containerd + nerdctl by default

- Finch (2022–): nerdctl + Lima in a single “finch” command
  - More extensions are likely to come

- Rancher Desktop (2020–): Kubernetes on Desktop
  - Supports Docker and nerdctl too
  - macOS version uses Lima (since 2021), Windows version uses WSL2
  - Provides GUI

Solutions for running containers on Mac

NTT

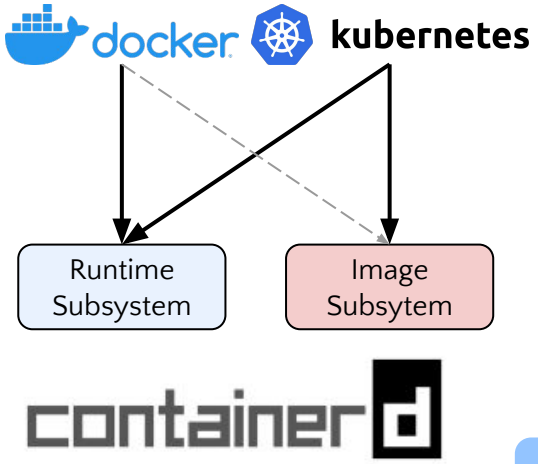
- Podman Machine (2021–)
  - Podman's built-in feature for creating Linux virtual machines with Podman installed in it

- Podman Desktop (2022–)
  - Provides GUI for Podman, Docker, and Kubernetes
  - Supports Lima as well as Podman Machine

Docker is being refactored to make more use of containerd

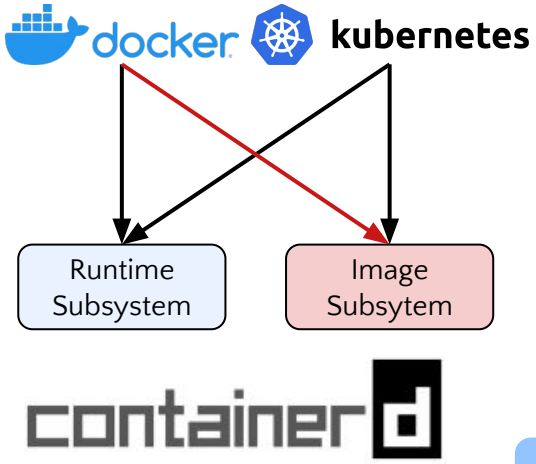
NTT

- containerd provides runtime subsystem and image subsystem
- The image subsystem is not used by Docker
- Docker's legacy image subsystem is far behind containerd's modern image subsystem
  - No support for lazy-pulling (on-demand image pulling)
  - Limited support for multi-platform images (e.g., AMD64/ARM64 dual-platform images)
  - Limited compliance of OCI Image Spec



Docker is being refactored to make more use of containerd

- Docker v24 (2023) experimentally supports using containerd's image subsystem
- Future version will use containerd's image subsystem by default



Lazy-pulling of images

- Most files in the images are never used
  - Dynamic libraries (`/usr/lib`)
  - Command binaries (`/usr/bin`)
  - Document files (`/usr/share/doc`)
  - …

“pulling packages accounts for 76% of container start time, but only 6.4% of that data is read”
“Slacker: Fast Distribution with Lazy Docker Containers” (Harter, et al., FAST 2016)

- But containers cannot be started until downloading the entire images
- Because OCI-standard tar.gz images are not seek()-able
- “Lazy-pulling” eliminates this issue

OCI: Open Container Initiative (Don’t confuse with “Oracle Cloud Infrastructure”)

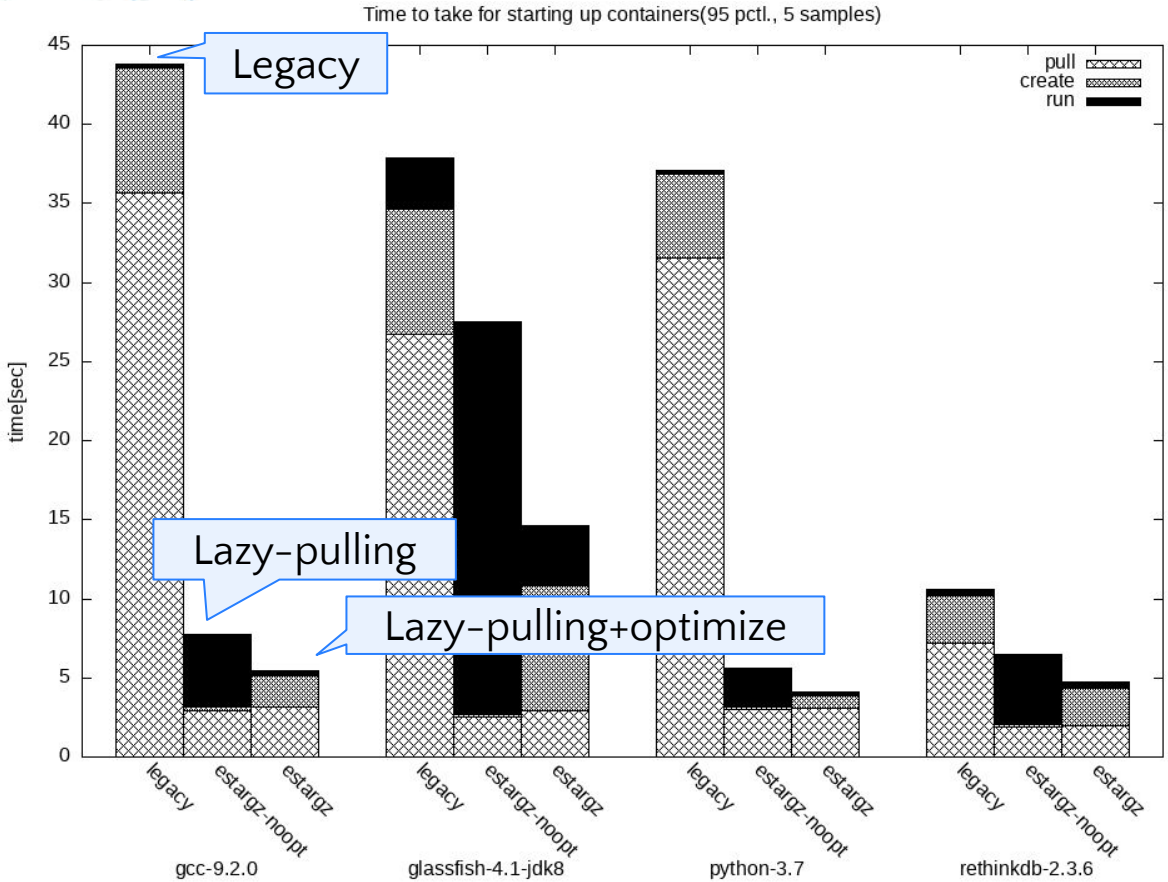
Lazy-pulling of images

- Lazy-pulling: pulling image contents on demand
- No need to pull an entire image
- Several formats are being proposed (mostly for containerd)

Format	Implementation for containerd	Description
eStargz (2019-)	github.com/containerd/stargz-snapshotter	Optimizes gzip granularity for seek()-ability; Forward compatible with OCI v1 tar.gz
SOCI (2022-)	github.com/awslabs/soci-snapshotter	Captures a checkpoint of tar.gz decoder state; Forward compatible with OCI v1 tar.gz
Nydus (2022-)	github.com/containerd/nydus-snapshotter	An alternate image format; Not compatible with OCI v1 tar.gz
OverlayBD (2021-)	github.com/containerd/overlaybd	Block devices as container images; Not compatible with OCI v1 tar.gz

OCI: Open Container Initiative (Don't confuse with "Oracle Cloud Infrastructure")

Lazy-pulling of images

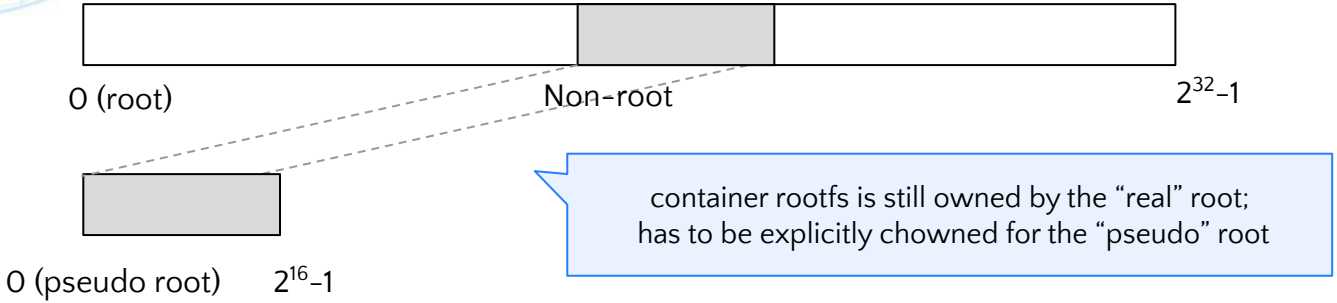


https://github.com/containerd/stargz-snapshotter/blob/v0.14.3/docs/images/benchmarking-result-ecdb227.png

Expanding adoption of User namespaces

- User namespaces are still rarely used in the Docker and Kubernetes ecosystem, although Docker has been supporting it since v1.9 (2015)

- One of the reason is that the complexity and the overhead of “chowning” are not negligible



- Linux kernel v5.12 (2021) added “idmapped mounts” to eliminate the necessity for chowning
  - runc v1.2 will be released soon (2023 Q2? Q3?) to support this

Expanding adoption of User namespaces

NTT

- Kubernetes v1.25 (2022) added preliminary support for User Namespaces (KEP-127)
- For compatibility sake, it is unlikely that Kubernetes will ever enable User Namespaces by default
- Users will still have to explicitly enable User Namespaces for enhanced security
- Docker may still potentially enable User Namespaces by default in future, but nothing is decided yet (Discussed in PR #38795)

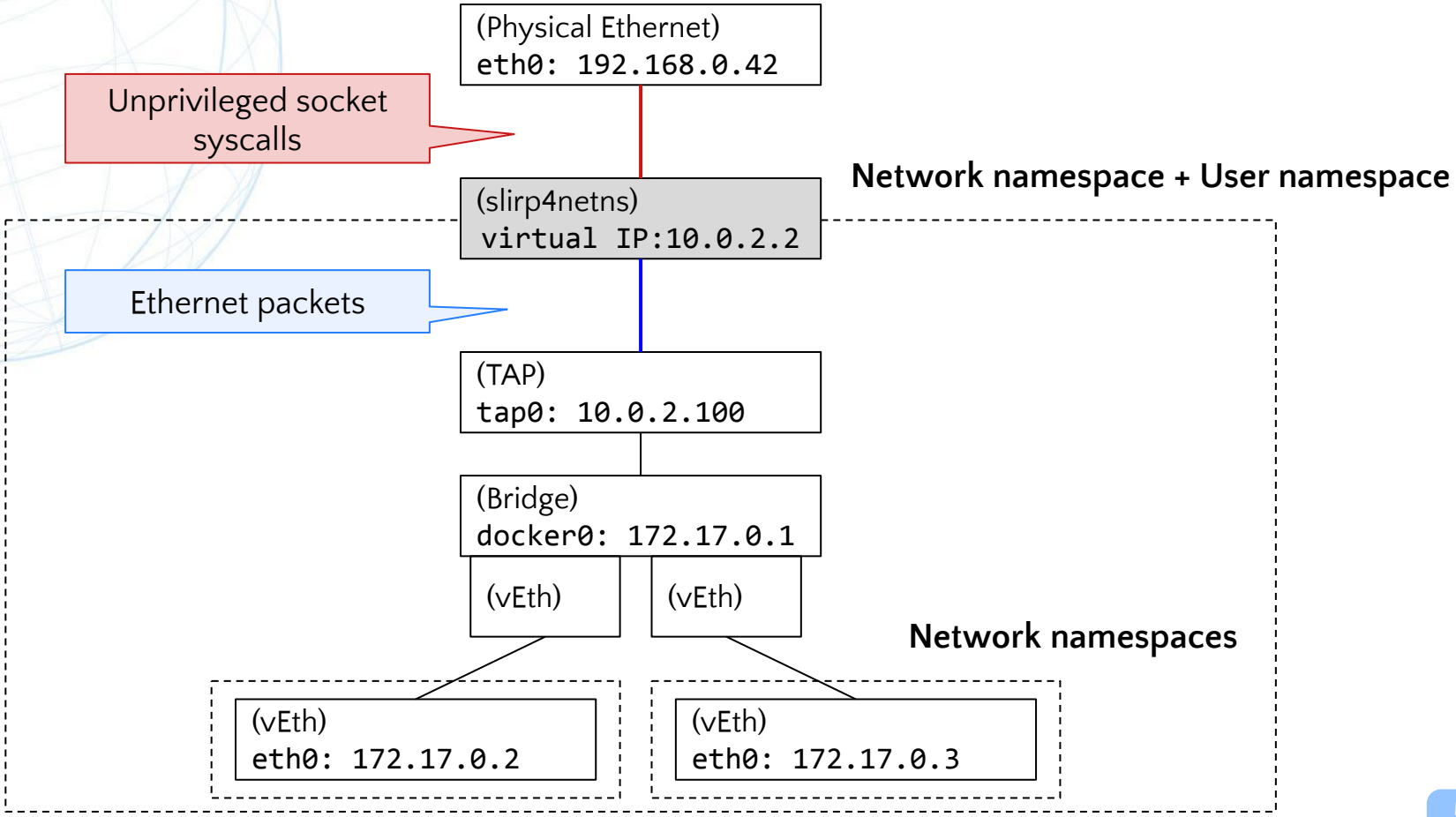
Rootless containers

- Puts container runtimes (as well as containers) in a user namespace that is created by a non-root user
  - No overhead of chowning, as everything is in the same user namespace
- Can mitigate potential vulnerabilities of the runtimes
  - No access to read/write other users' files
  - No access to modify the kernel
  - No access to modify the firmware
  - No ARP spoofing
  - No DNS spoofing
- Also useful for shared hosts (High-performance Computing, etc.)

Rootless containers

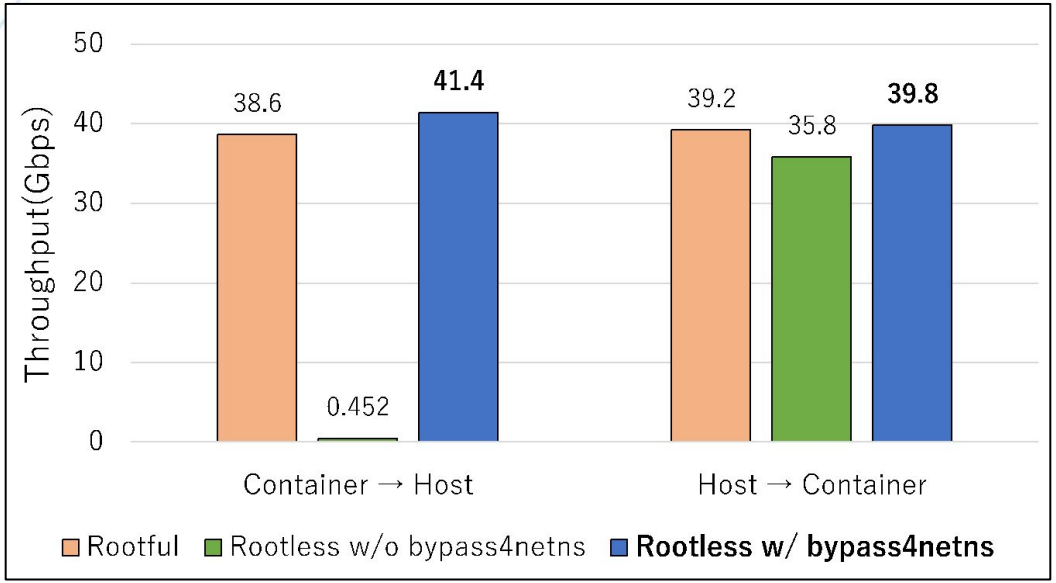
- **2014**: LXC v1.0 introduced support for Rootless containers (called “unprivileged containers” at that time)
  - Networking depends on a SETUID binary, which is hard to configure and also insecure
- **2017**: runc v1.0-rc4 gained initial support for Rootless
- **2018**: Several works has begun to support Rootless in containerd, BuildKit, Docker, Podman, etc.
  - slirp4netns (usermode TCP/IP) eliminated the need to use a SETUID binary for bringing up the network
- **2019**: Docker v19.03 was released with an experimental Rootless support
- **2020**: Docker v20.10 was released with general availability of Rootless

Rootless containers



Faster Rootless containers

- Bypasses slirp4netns (usermode TCP/IP) by using `SECCOMP_IOCTL_NOTIF_ADDFD`
  - Captures socket syscalls inside the NetNS, reconstructs the FDs outside the NetNS, and replaces the FDs inside the NetNS
- Even faster than rootful



Accelerating TCP/IP Communications in Rootless Containers by Socket Switching (Naoki Matsumoto and Akihiro Suda, SWoPP 2022)
https://speakerdeck.com/mt2naoki/ip-communications-in-rootless-containers-by-socket-switching?slide=4

Criticisms against Rootless containers

- It is controversial whether non-root users should be allowed to create user namespaces
  - **Yes**, for container users, because rootless containers are much safer than running everything as the root
  - **No**, for others, because it can be rather an attack surface
    CVE-2023-32233: Privilege escalation in Linux Kernel due to a Netfilter nf_tables vulnerability

- Ubuntu and Debian provide a sysctl knob to allow/disallow unprivileged user namespaces: `kernel.unprivileged_userns_clone=<bool>`
  - But not upstreamed

- Linux v6.1 (2022) introduced a new LSM hook: `userns_create`
  - Hookable from KRSI (eBPF LSM)
  - Userspace tools have to be improved to provide a human-friendly UX for this

LSM: Linux Security Module, KRSI: Kernel Runtime Security Instrumentation

More LSMs

- Landlock LSM was merged into Linux v5.13 (2021)
  - Restricts file accesses by paths
    - `LANDLOCK_ACCESS_FS_EXECUTE`
    - `LANDLOCK_ACCESS_FS_READ_FILE`
    - `...`
  - No privilege is needed to set up the profile
  - Slightly similar to OpenBSD's `pledge(2)`

- Landlock is not supported by the OCI Runtime Spec yet , hope that it can be supported very soon (PR #1111)

“Non-container” containers

- “Containers” are not well defined

- Almost anything can be called a “container runtime” when it accepts OCI formats 😅

OCI: Open Container Initiative (Don’t confuse with “Oracle Cloud Infrastructure”)

“Non-container” containers: Kata Containers

- Virtual machines, with container-ish user experiences
- As secure as virtual machines (because they are virtual machines)
- Same images as regular containers
- Same runtime configuration as regular containers
- Implemented as a containerd plugin

Kata Containers was released in 2017 , as a merger of Intel Clear Containers and Hyper runV

“Non-container” containers: gVisor

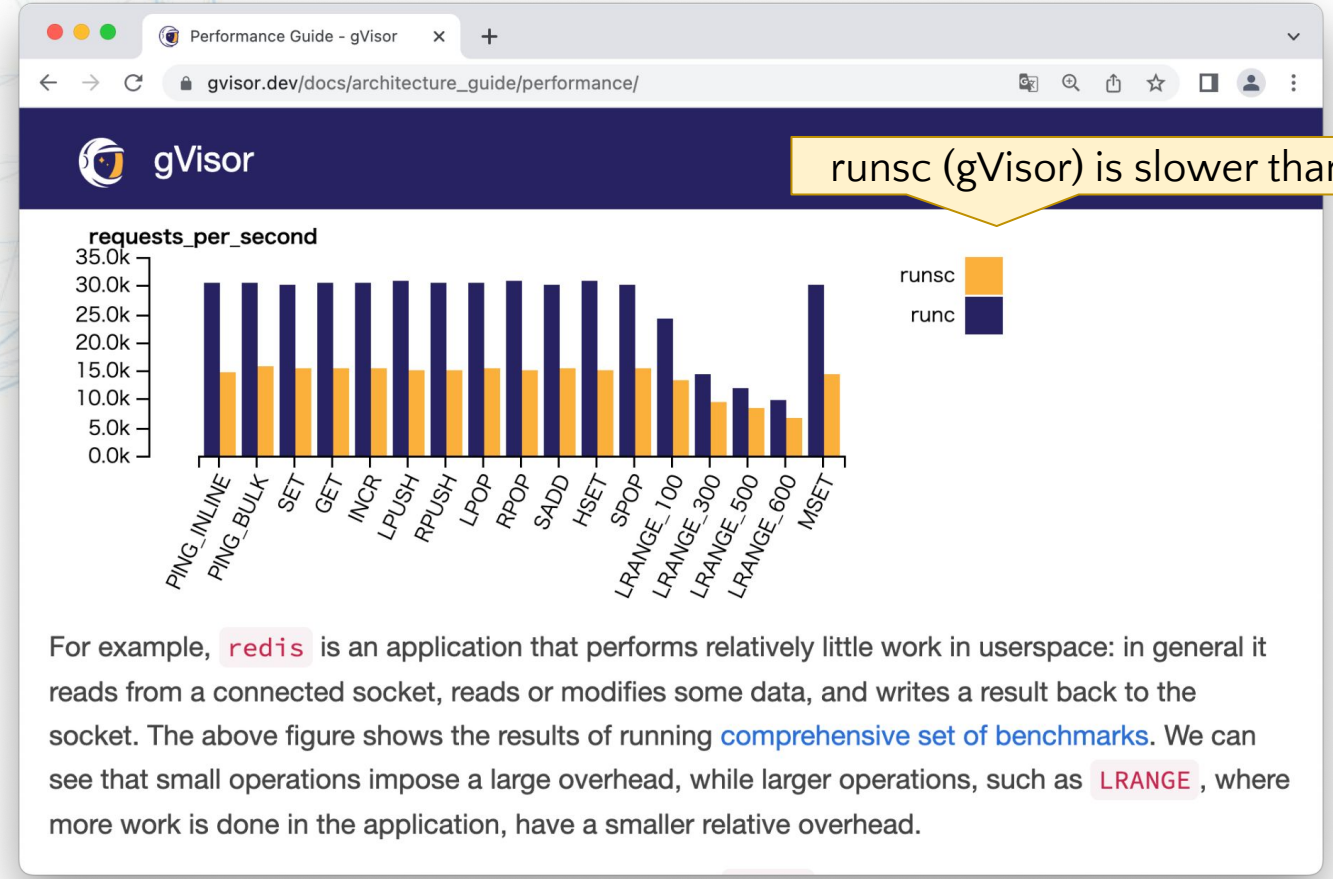
NTT

- Traps syscalls and execute them in yet another kernel (“sandbox”) to mitigate attacks
  - **KVM mode**: rarely used, but the best option for bare-metal hosts
  - **ptrace mode**: usermode kernel implementation; the most common option but slow
  - **SIGSYS trap mode** (since 2023): expected to replace ptrace mode eventually

- Seccomp is applied to limit calling host syscalls
- gVisor’s kernel is highly compatible with Linux kernel, but not 100% compatible
- Implemented as a containerd plugin;
  Also available as a runc-compatible binary (`runsc`)

gVisor was released in 2018

“Non-container” containers: gVisor



Screenshot on June 2023

“Non-container” containers: gVisor

- Google Cloud Run was using gVisor, but they switched away to microVM in 2023
  - *“This means that software that previously didn’t run in Cloud Run due to unimplemented system call issues can now run in Cloud Run’s second-generation execution environment.”*

https://cloud.google.com/blog/products/serverless/cloud-run-jobs-and-second-generation-execution-environment-ga/?hl=en

WebAssembly

NTT

- Platform-independent byte codes, originally designed for Web browsers in 2015
- Similar to Java applets (1995), but puts more focus on portability and security
- Can be compiled from C, Go, Java, Rust, .NET, etc.
- Harvard architecture: code address space != data address space
  - No instruction for "`JMP <immediate>`", "`JMP *<reg>`"
  - Only jumpable to labels that are resolved on compilation time
  - Less possibility of arbitrary code execution bugs

WebAssembly

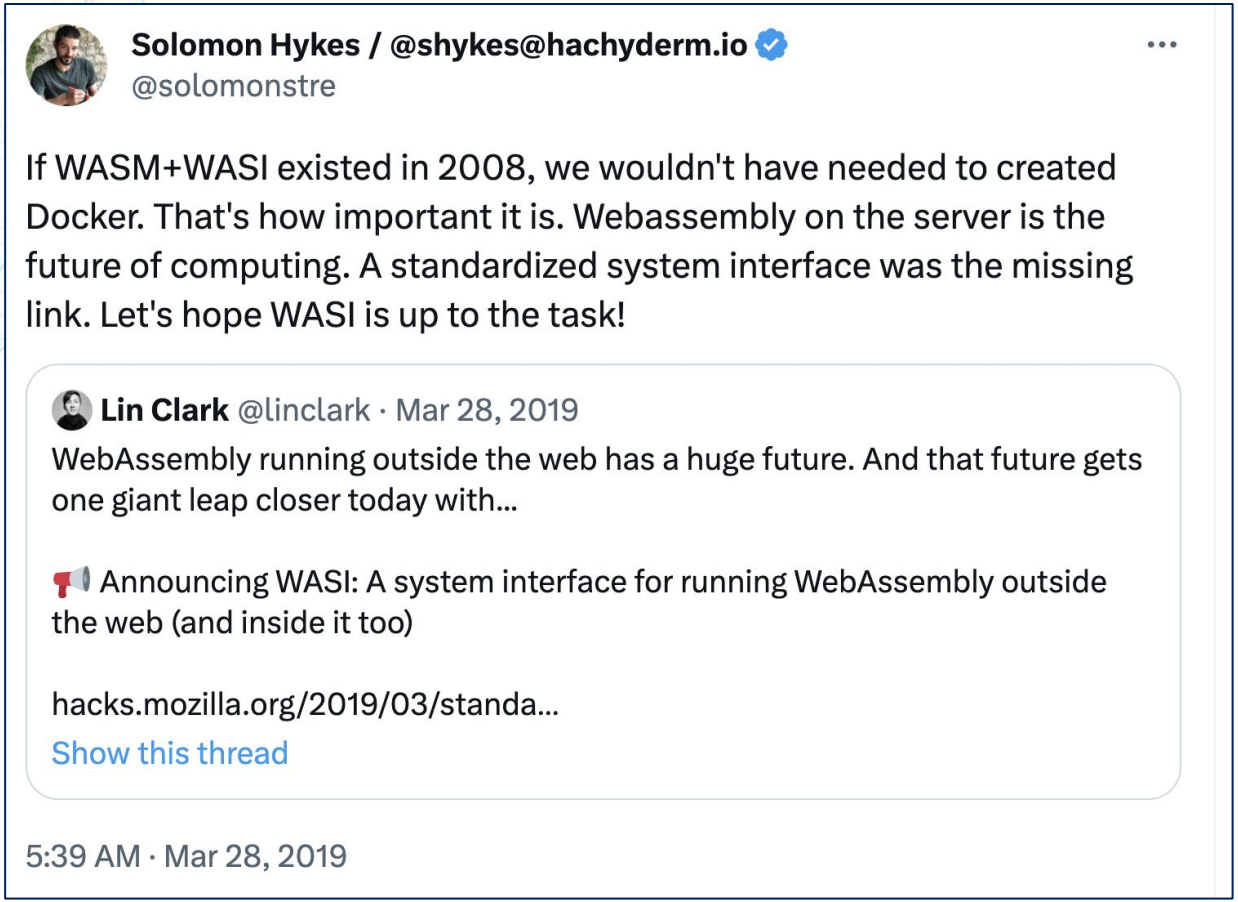
NTT

- WebAssembly isn't just for Web browsers today

- WASI (2019-): WebAssembly System Interface
  - Provides low-level API for implementing POSIX-like layers on it
  - Operates on file descriptors passed from a runtime: `fd_read()` , `fd_write()`, `sock_receive()` , `sock_send()`, …

- WASIX (2023-): Extends WASI to provide more convenient (and somewhat controversial) functions
  - **Threads**: `thread_spawn()` , `thread_join()`, …
  - **Processes**: `proc_fork()` , `proc_exec()`, …
  - **Sockets**: `sock_listen()` , `sock_connect()`, …

WebAssembly



Solomon Hykes / @shykes@hachyderm.io
@solomonstre

If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!

> **Lin Clark** @linclark · Mar 28, 2019
> WebAssembly running outside the web has a huge future. And that future gets one giant leap closer today with...
>
> 📢 Announcing WASI: A system interface for running WebAssembly outside the web (and inside it too)
>
> hacks.mozilla.org/2019/03/standa...
> Show this thread

5:39 AM · Mar 28, 2019

https://twitter.com/solomonstre/status/1111004913222324225

WebAssembly

NTT

- containerd has “runWASI” plugin since 2022

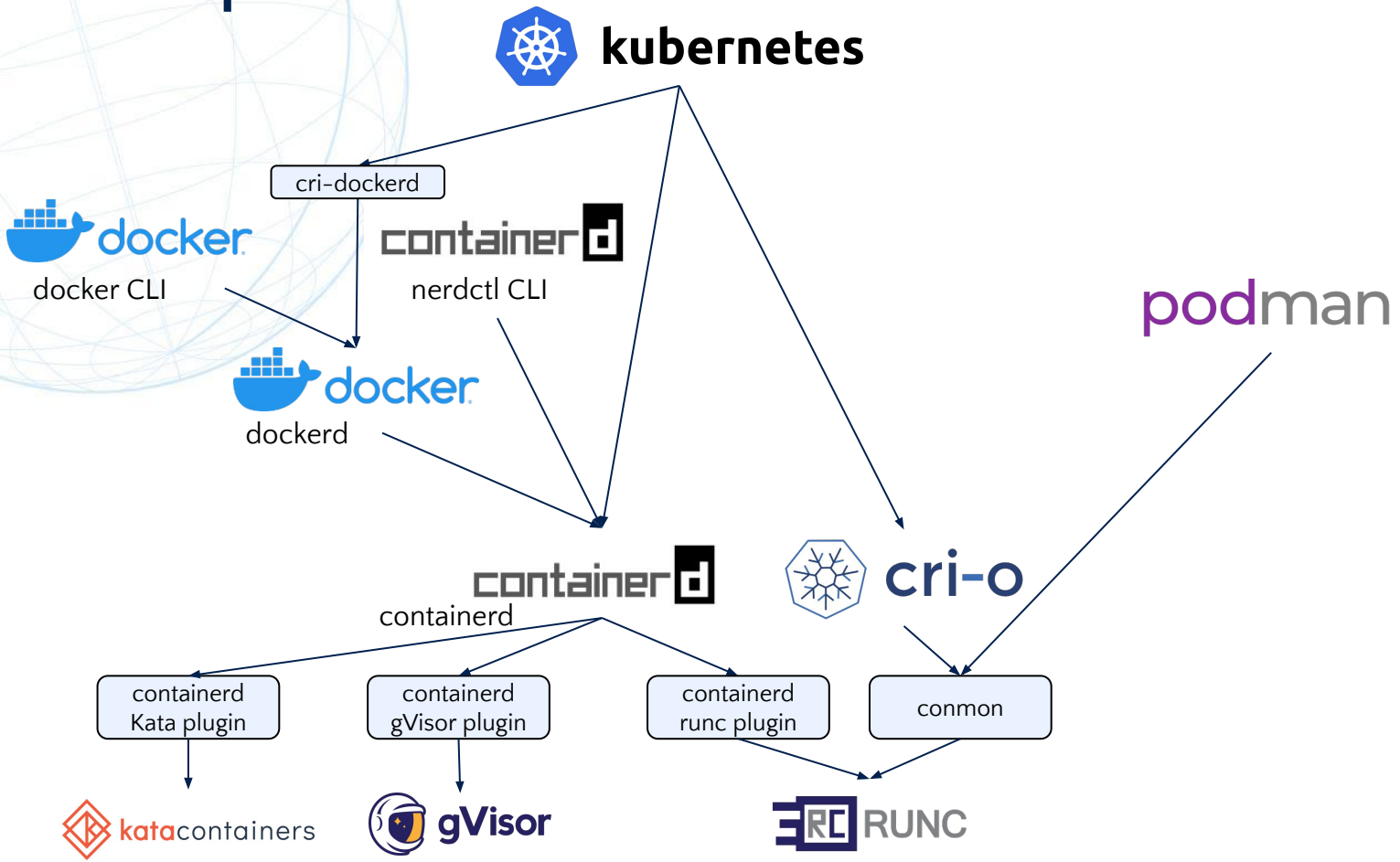
- Supports WasmEdge and WasmTime as underlying WASI runtimes

Recap

Recap

- Containers are more efficient, but often less secure, than virtual machines
  - Lots of security technologies are being introduced to harden containers: UserNS, Rootless, LSMs, ...
- Alternatives to Docker are arising, but Docker isn't fading out
  - **Kubernetes runtimes**: containerd, CRI-O
  - **CLI**: Podman, nerdctl, Finch
- "Non-container" containers are trends too
  - **Kata**: VM-based, **gVisor**: user mode kernel, **runWASI**: WebAssembly, ...

Landscape


kubernetes
cri-dockerd
docker
docker CLI
containerd
nerdctl CLI
podman
docker
dockerd
containerd
containerd
cri-o
containerd Kata plugin
containerd gVisor plugin
containerd runc plugin
conmon
kata containers
gVisor
RUNC
Orchestrator
Engines
High-level Runtimes
Shims
Low-level Runtimes

Other topics (Not covered in this talk, feel free to chat with me)

NTT

- Copy-on-Write filesystems
  - overlayfs, btrfs, zfs, devicemapper, ...
- Image security
  - SBOM, SLSA, Scanning, Signing, Reproducible builds, ...
- Auditing
  - auditd, falco, ...
- Trend of reimplementing runtimes in Rust
  - youki, containerd rust-extensions, conmon-rs, ...
- Checkpointing
  - CRIU
- Multi-node networking
  - VXLAN, BGP, ...
- Service mesh (almost out of the scope of container runtimes)
  - Sidecars, eBPF, Ambient mesh, ...