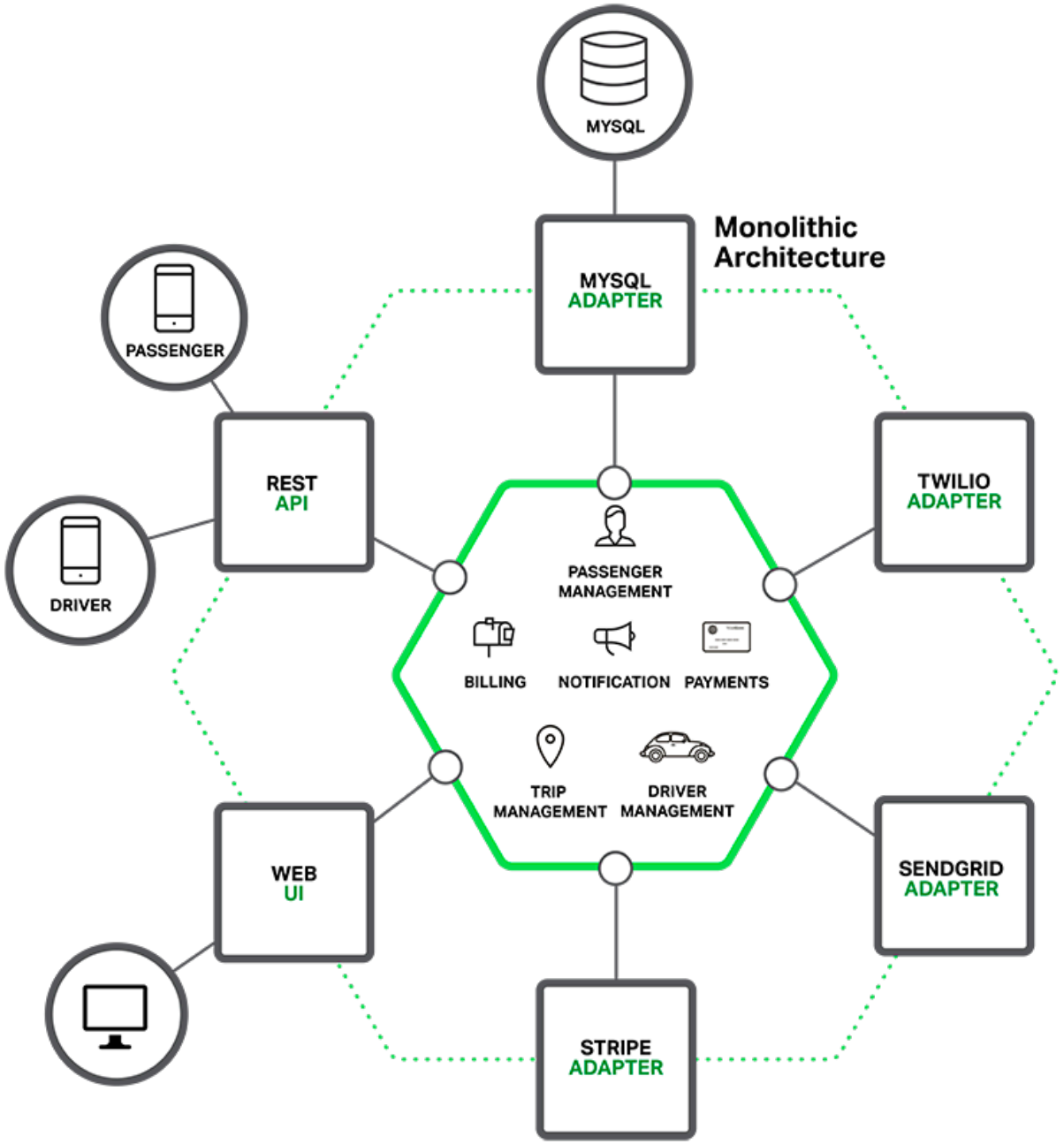
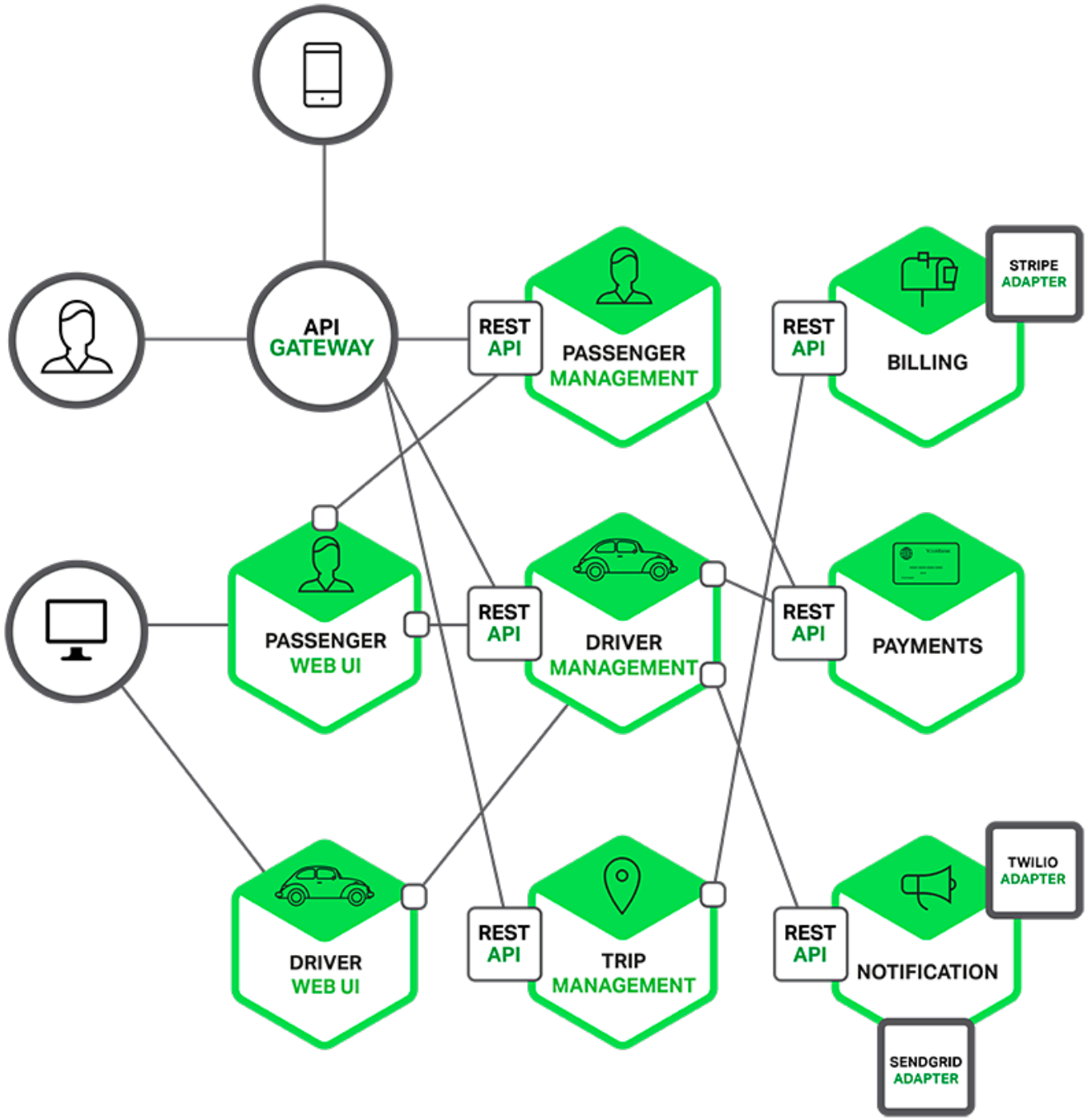
Container Internals & Security

Hyunsang Choi

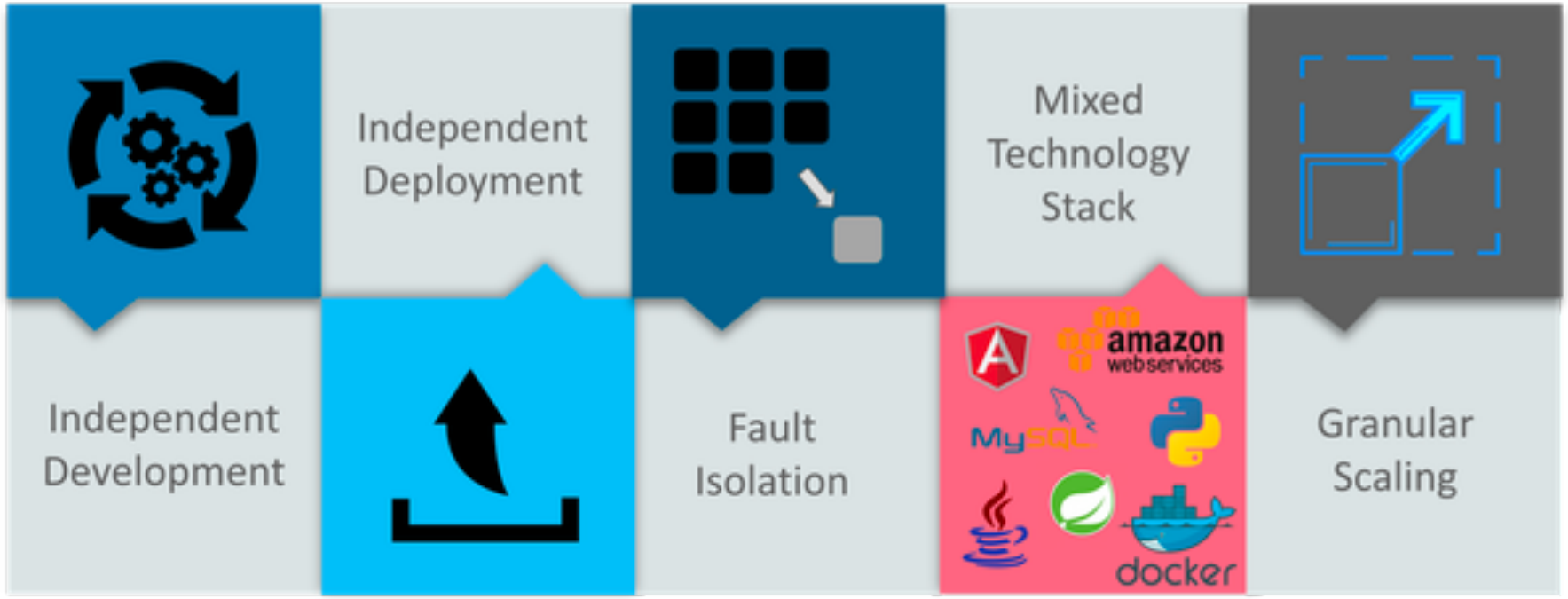
Monolithic


MYSQL
Monolithic Architecture
MYSQL ADAPTER
PASSENGER
REST API
DRIVER
TWILIO ADAPTER
PASSENGER MANAGEMENT
BILLING
NOTIFICATION
PAYMENTS
TRIP MANAGEMENT
DRIVER MANAGEMENT
WEB UI
SENDGRID ADAPTER
STRIPE ADAPTER

Microservice



Microservice Benefits



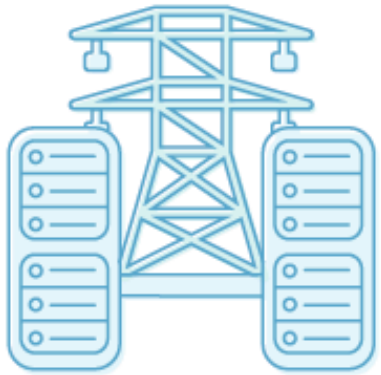
Container History

- **1979: Unix V7 (chroot)**
- **2000: FreeBSD Jails**
- **2004: Solaris Containers**
- **2005: Open VZ (Open Virtuzzo)**
- **2006: Process Containers (Google)**
- **2008: LXC** cgroups and namespaces
- **2011: OpenShift (Redhat)**
- **2013: LMCTFY (**Let Me Contain That For You) Google LMCTFY core -> libcontainer
- **2013: Docker** LXC in its initial stages and replaced to libcontainer
- **2014: Kubernetes (Google), GKE (GCP)**
- **2014: rkt (CoreOS)**
- **2015: CNCF (Linux Foundation)**
- **2015: Kubernetes to CNCF**
- **2015: OCI (runC) to CNCF**
- **2014: containerd (Docker)**
- **2016: Skopeo (Redhat)**
- **2016: Dirty COW**
- **2016: Helm**
- **2016: Docker Swarm**
- **2016: CRI (Kubernetes)**
- **2017: moby project**
- **2017: containerd to CNCF**
- **2017: Buildah**
- **2017: CRI-O**
- **2017: Podman**
- **2017: Kata Containers (OpenStack)**
- **2017: AKS (Azure)**
- **2018: gVisor (Google)**
- **2018: Nabla**
- **2018: FireCracker (AWS)**
- **2018: EKS (AWS)**

Container Internals

Docker Internals

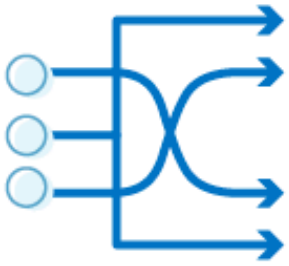
Virtualization: Why?



Reduce physical resource costs

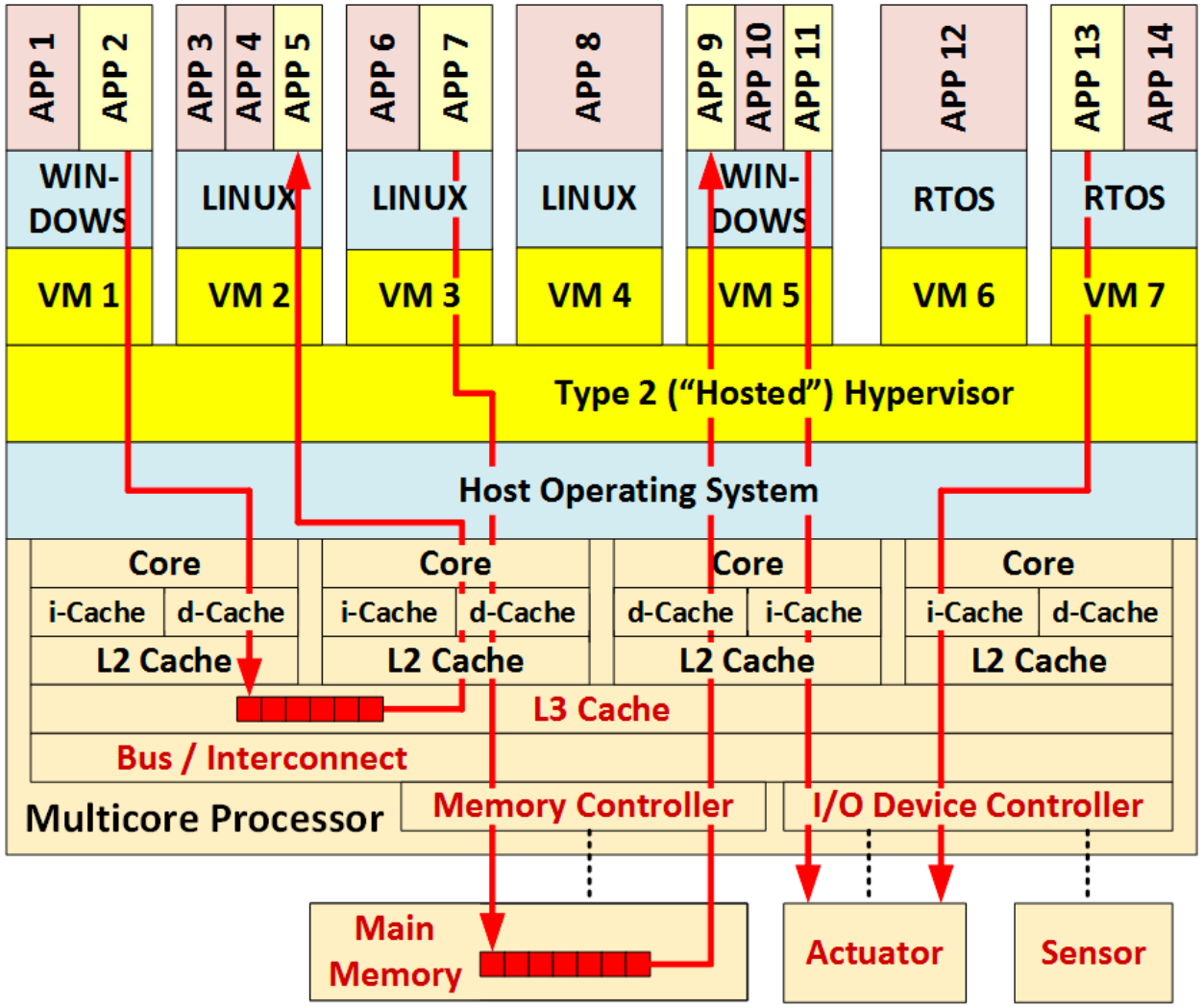
Speed setup

Create backups

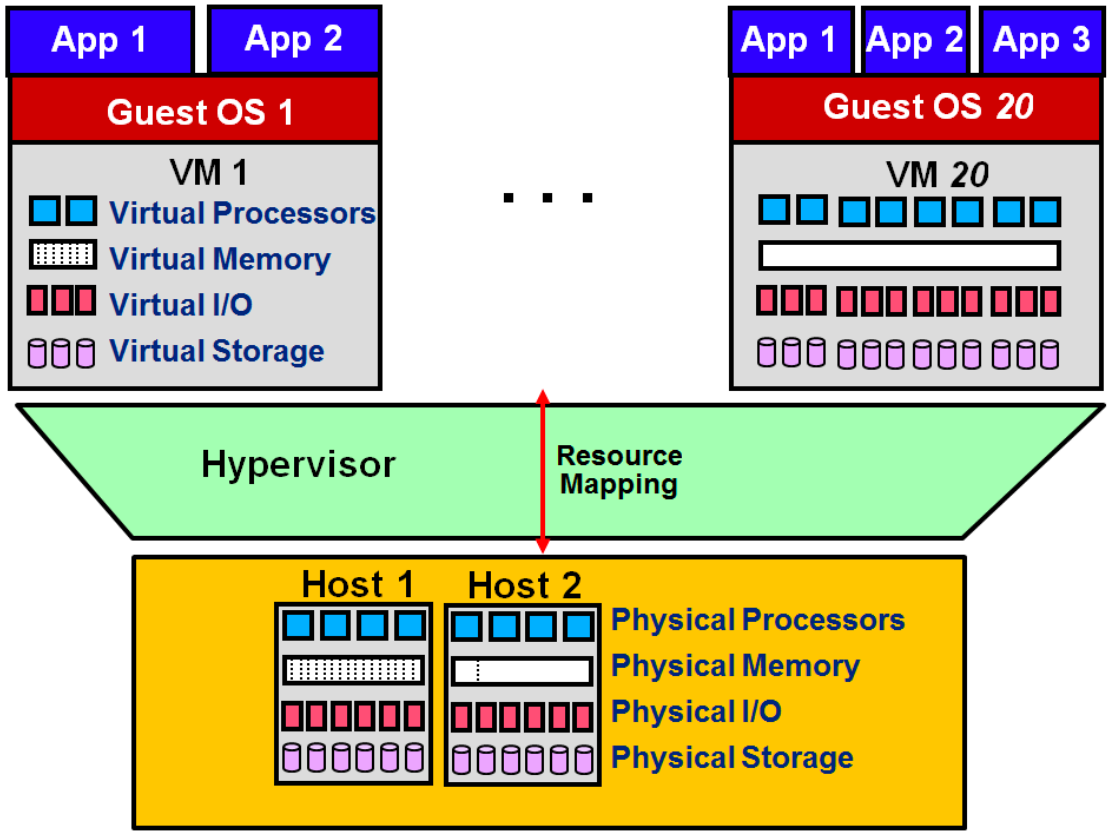


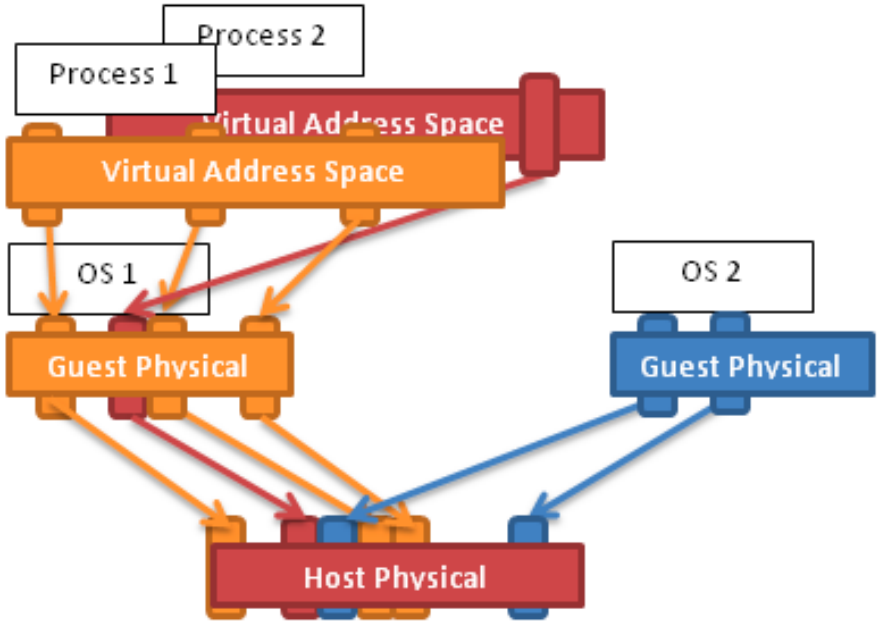
Gain flexibility

Virtualization: Resource Sharing

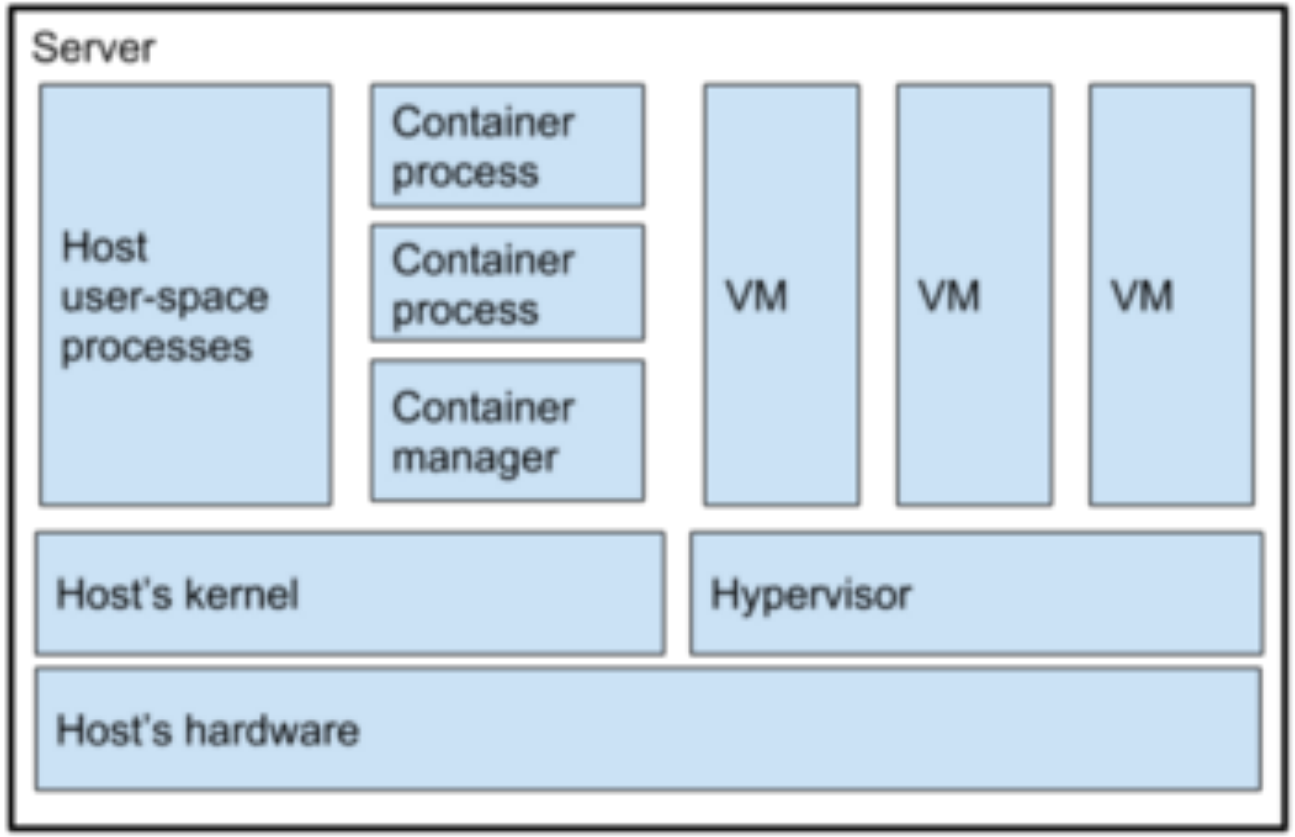


VM: Resource Mapping

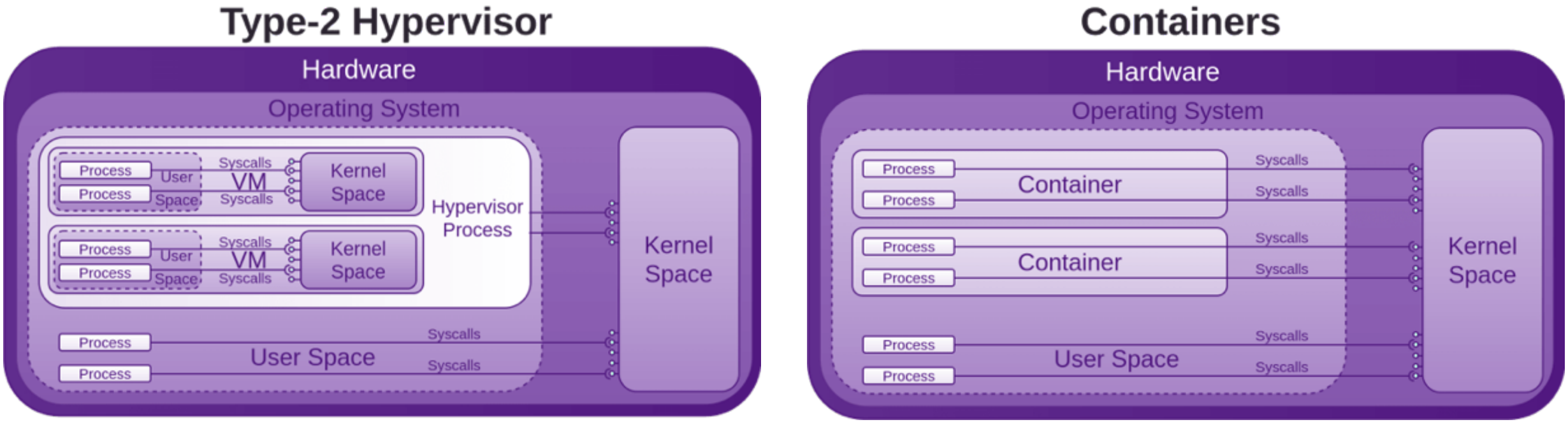




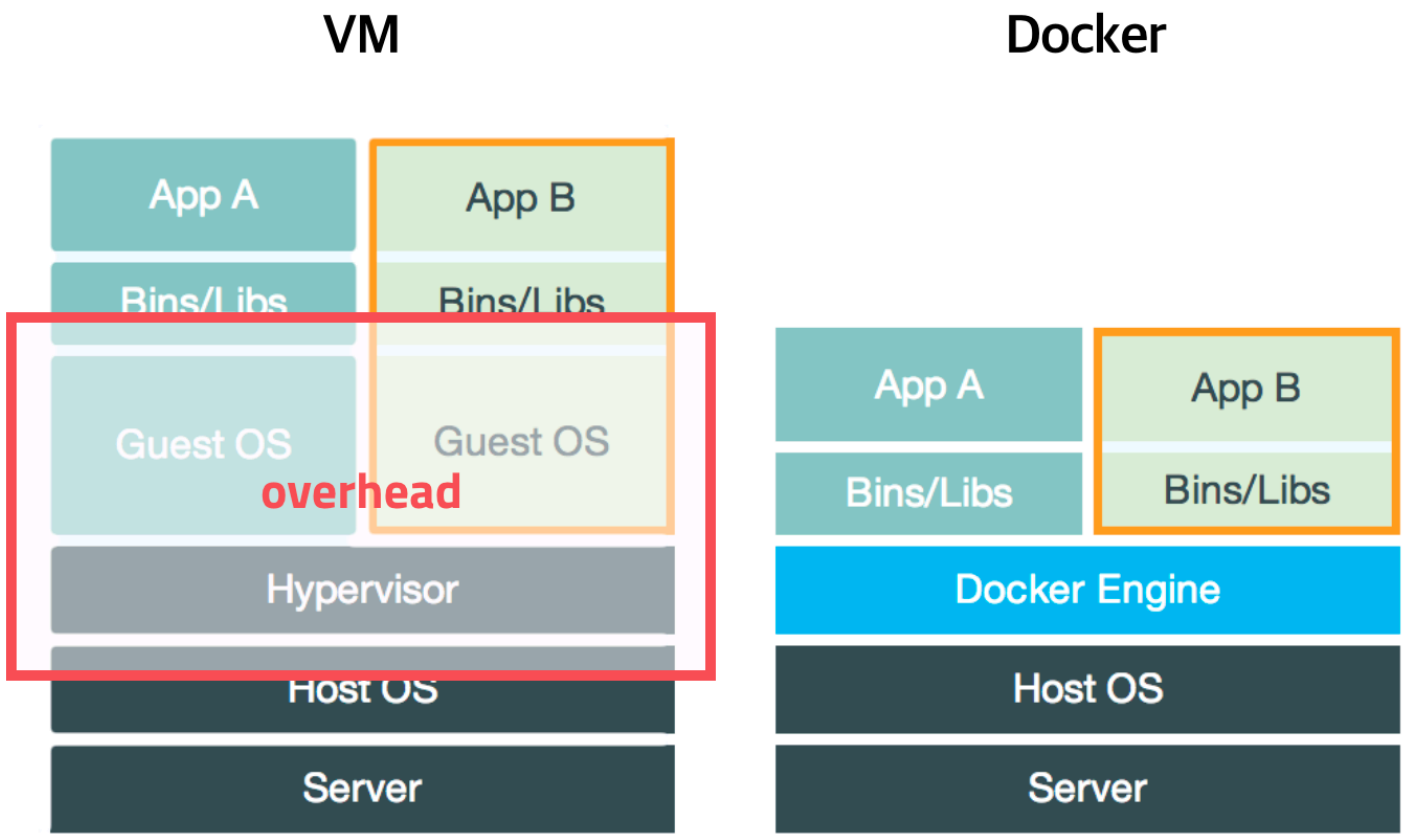
Container: No Resource Mapping



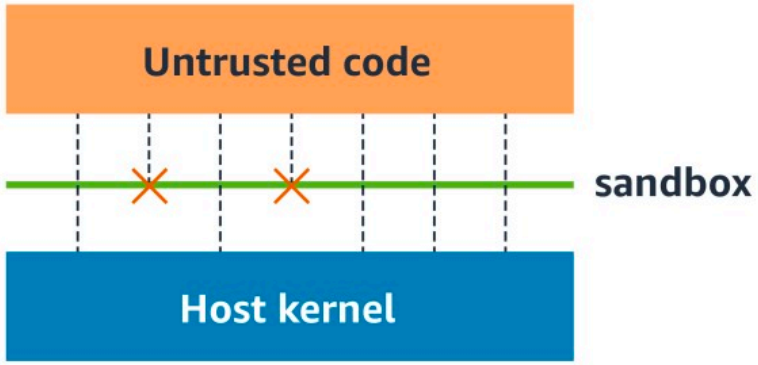
Hypervisor vs Container



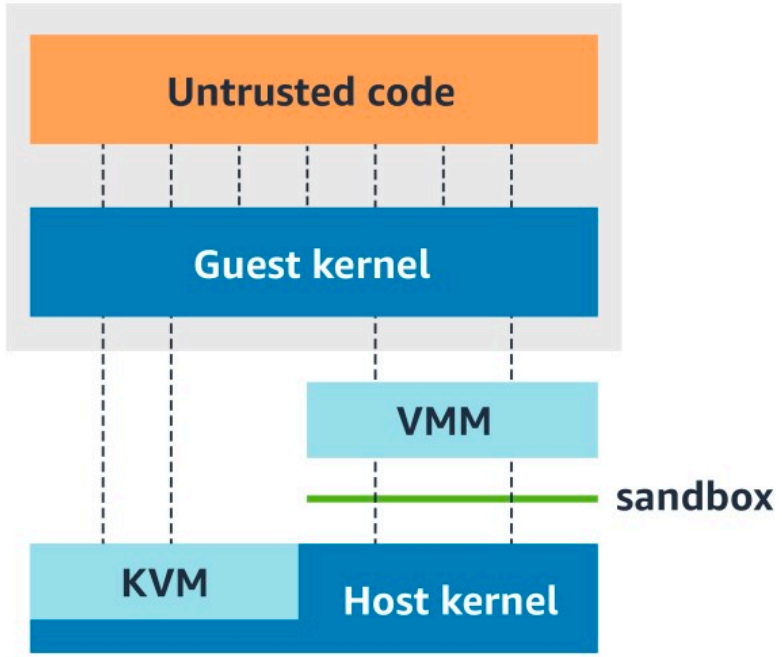
Container vs VM: Performance Perspective



Container vs VM: Security Perspective



(a) Linux container model

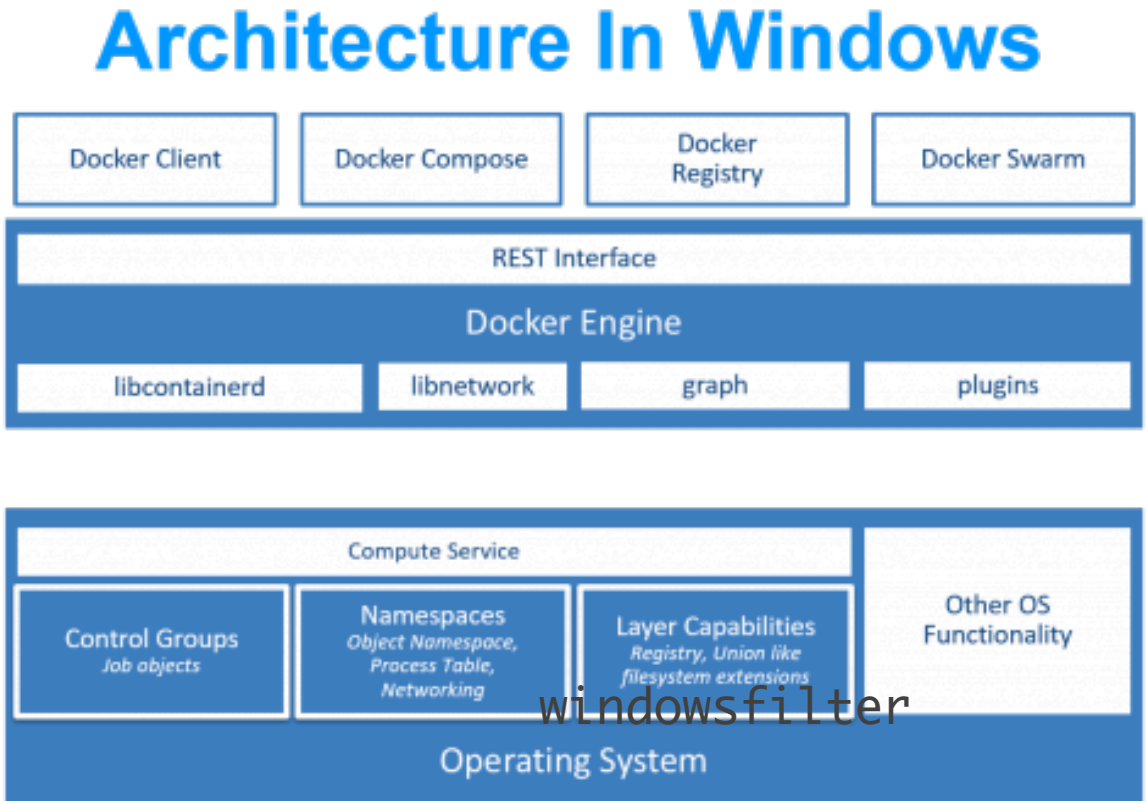


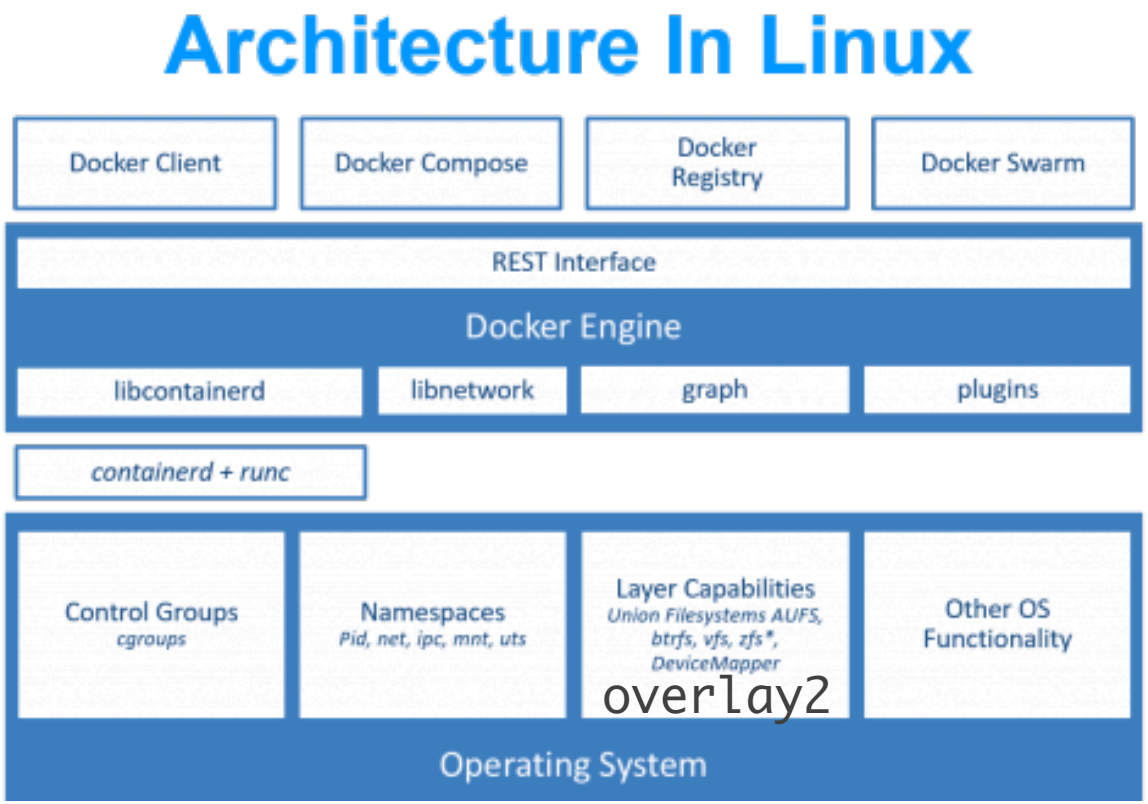
(b) Linux kernel-based virtual machine (KVM) virtualization model

VM vs Containers

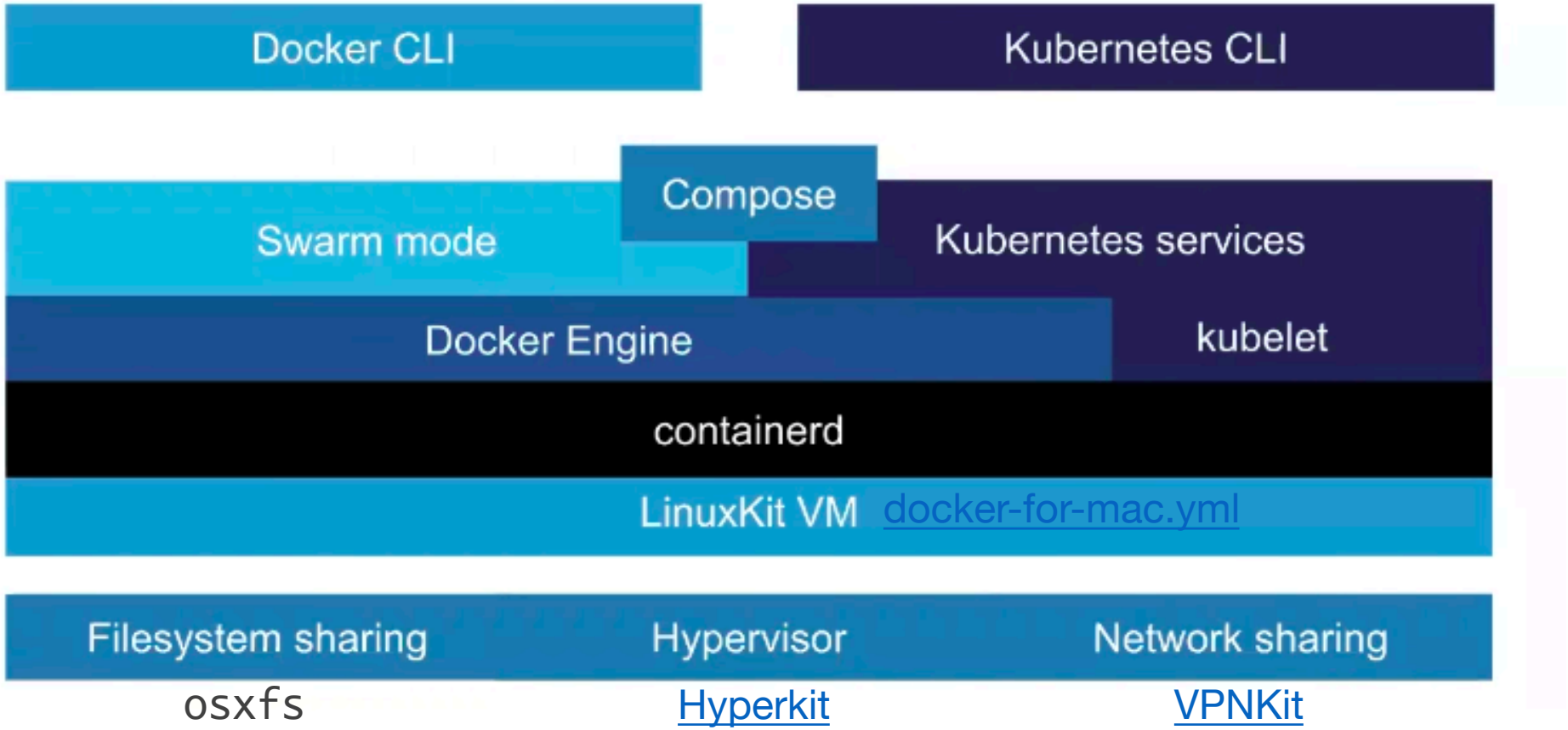
Criteria	VMs	Containers
Portability—number of operating systems	One or more	One
Portability—number of OS versions	One or more	One
Portability—number of OS types	One or more	Primarily Linux
Size of Applications	Medium or large	Small or medium
Security	More isolation	Less isolation
Number of applications per server	Lower	Higher
Number of copies of single application	One	Many
Performance (throughput, not response time)	Lower	Higher
Overhead—administration	Higher	Lower
Overhead—resource usage	Much higher	Much lower
Readily share resources (devices, services)	No	Yes
Robustness via failover and restart	Not supported	Supported
Scalability and load balancing via dynamic deployment	Slower and harder	Faster and easier
Application runs on bare metal	Not supported	May be supported

Docker Architecture

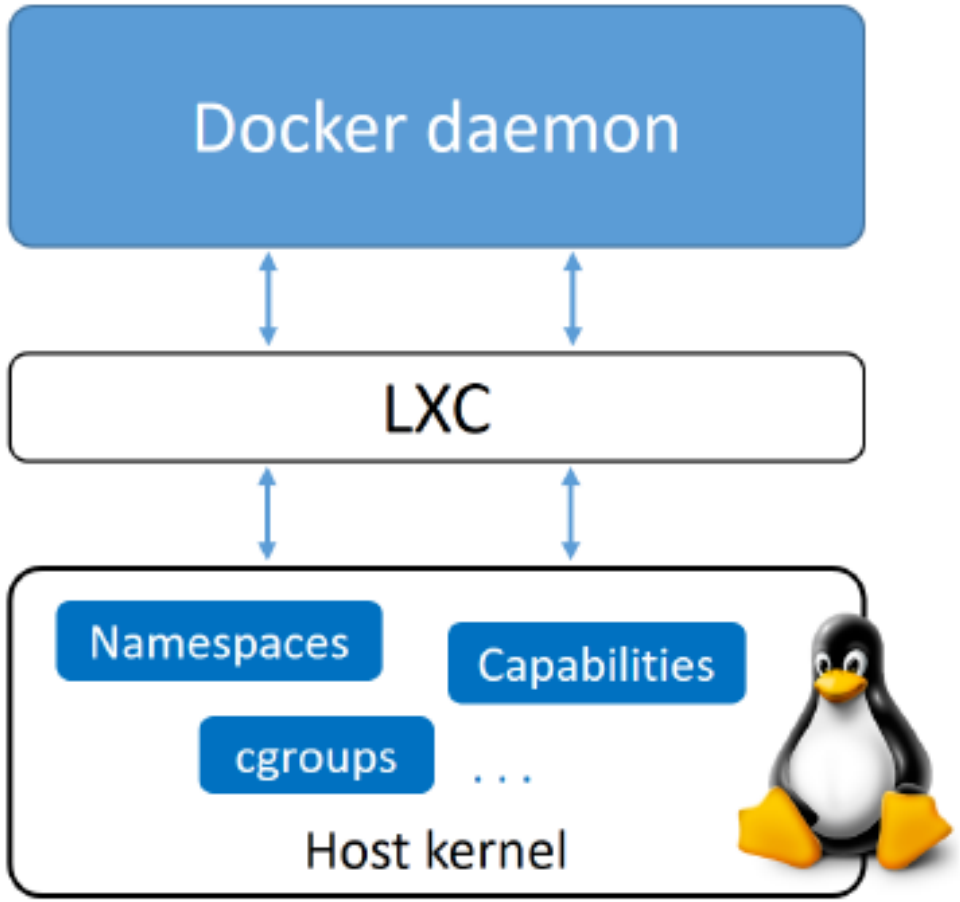


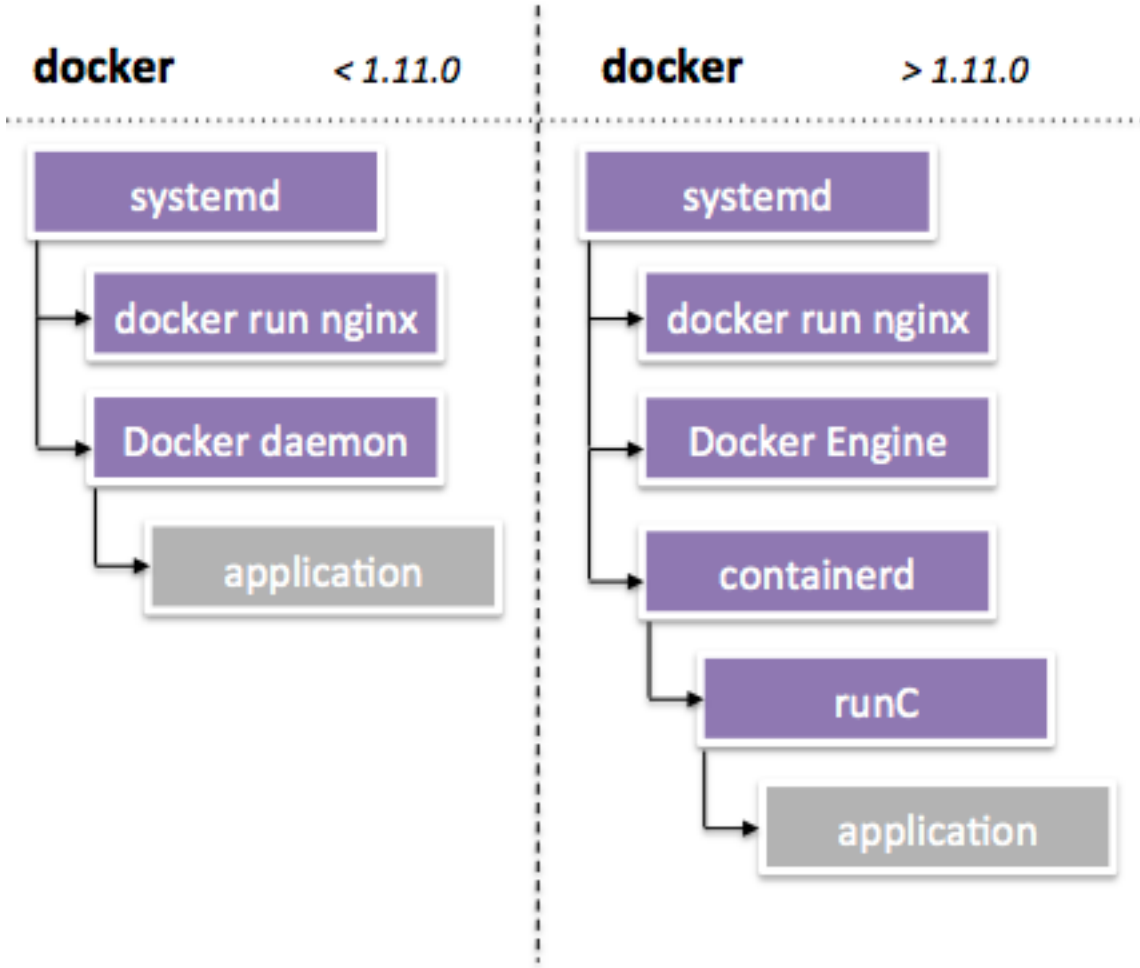


Docker for Mac

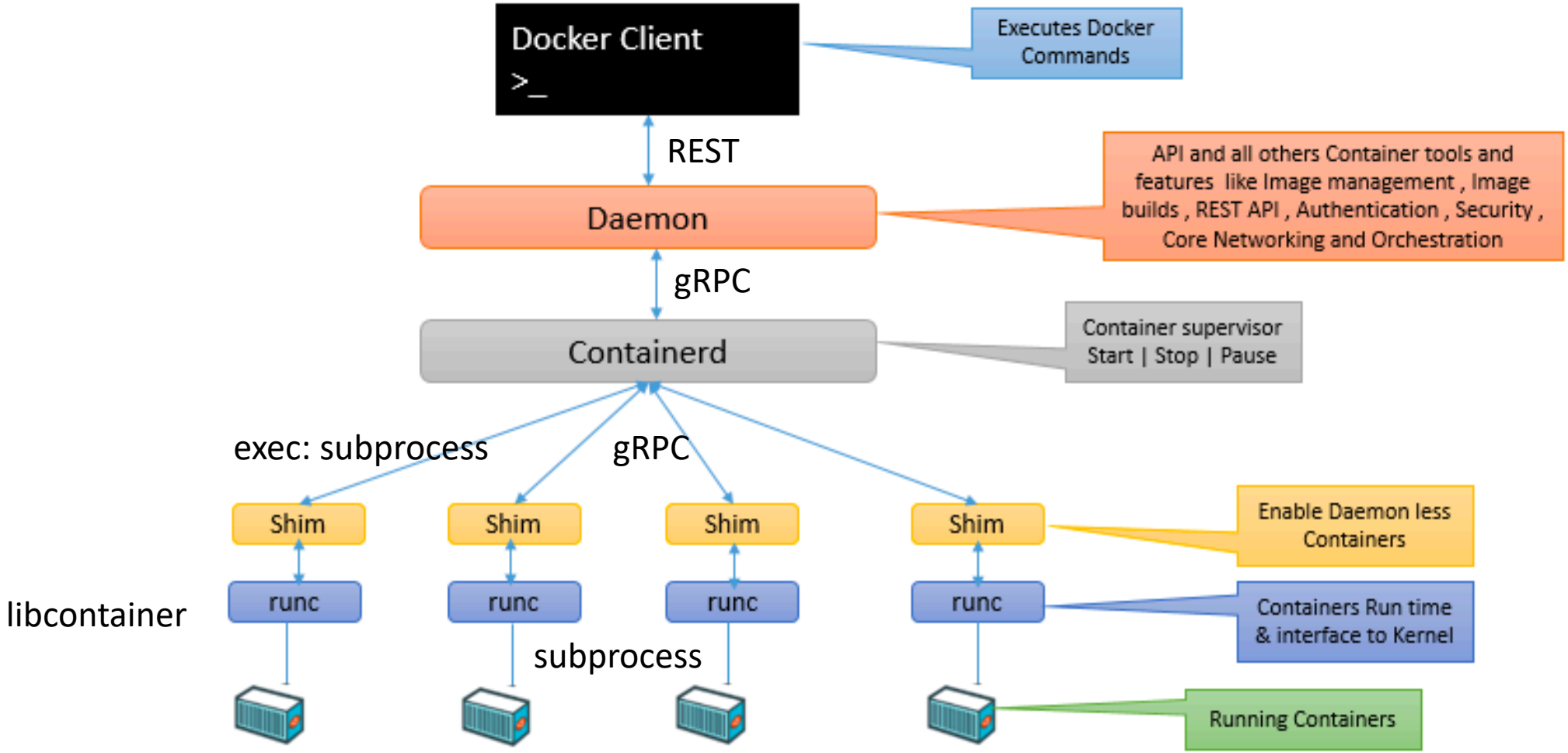


Docker Initial Architecture

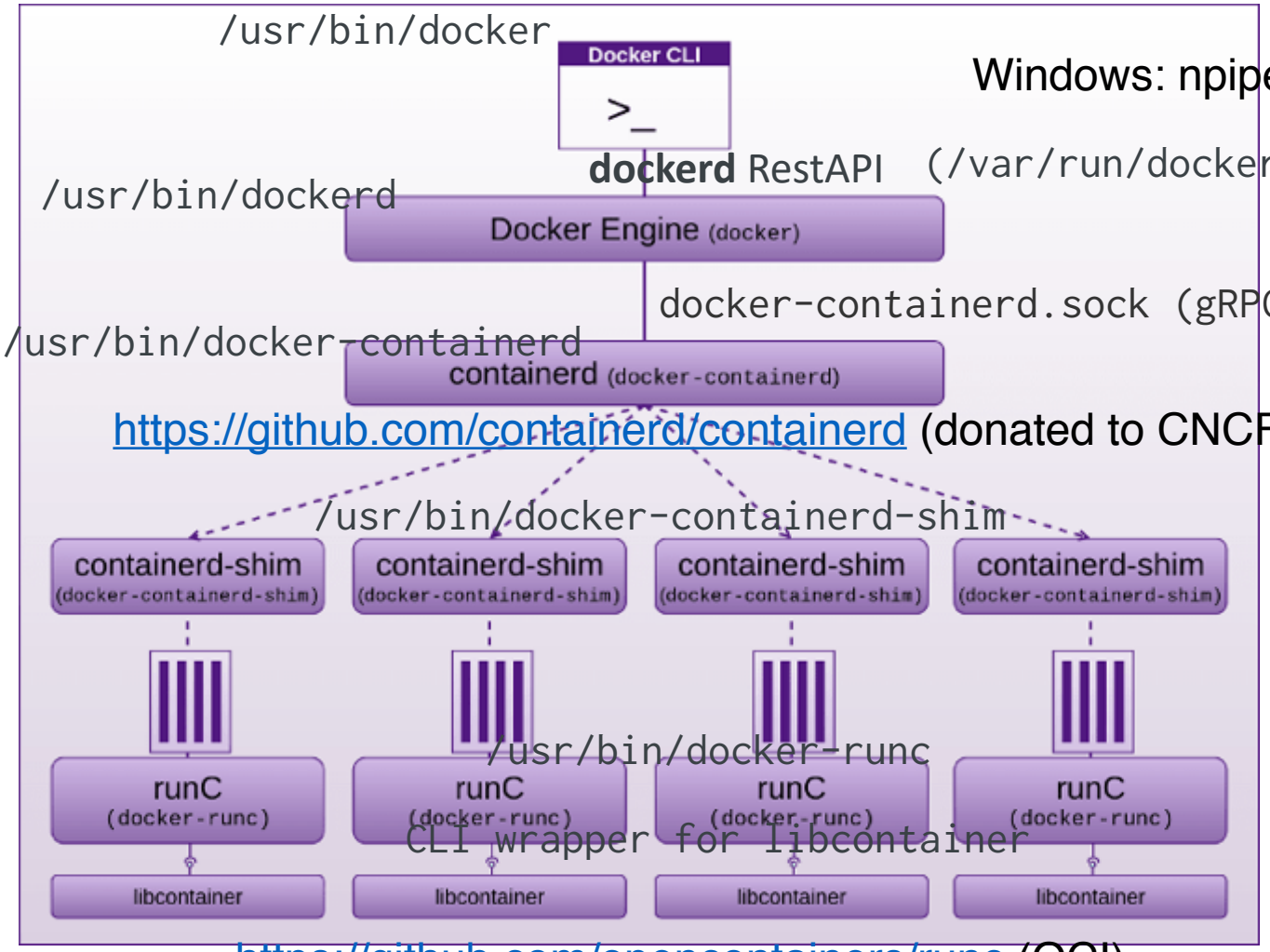




Docker Architecture: Now



Docker Processes (Linux)



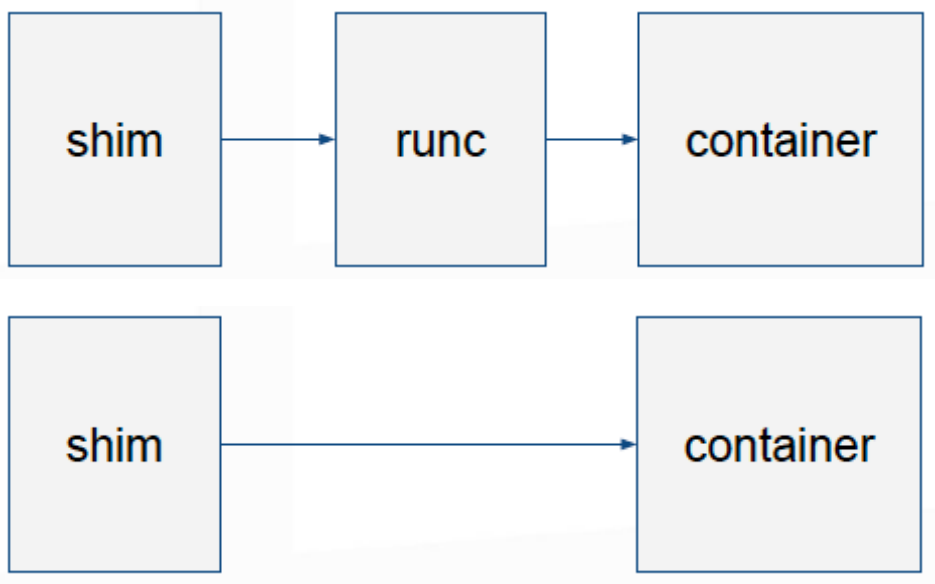
Windows: npipe:////./pipe/docker_engine

(/var/run/docker.sock by default or system fd://, tcp)

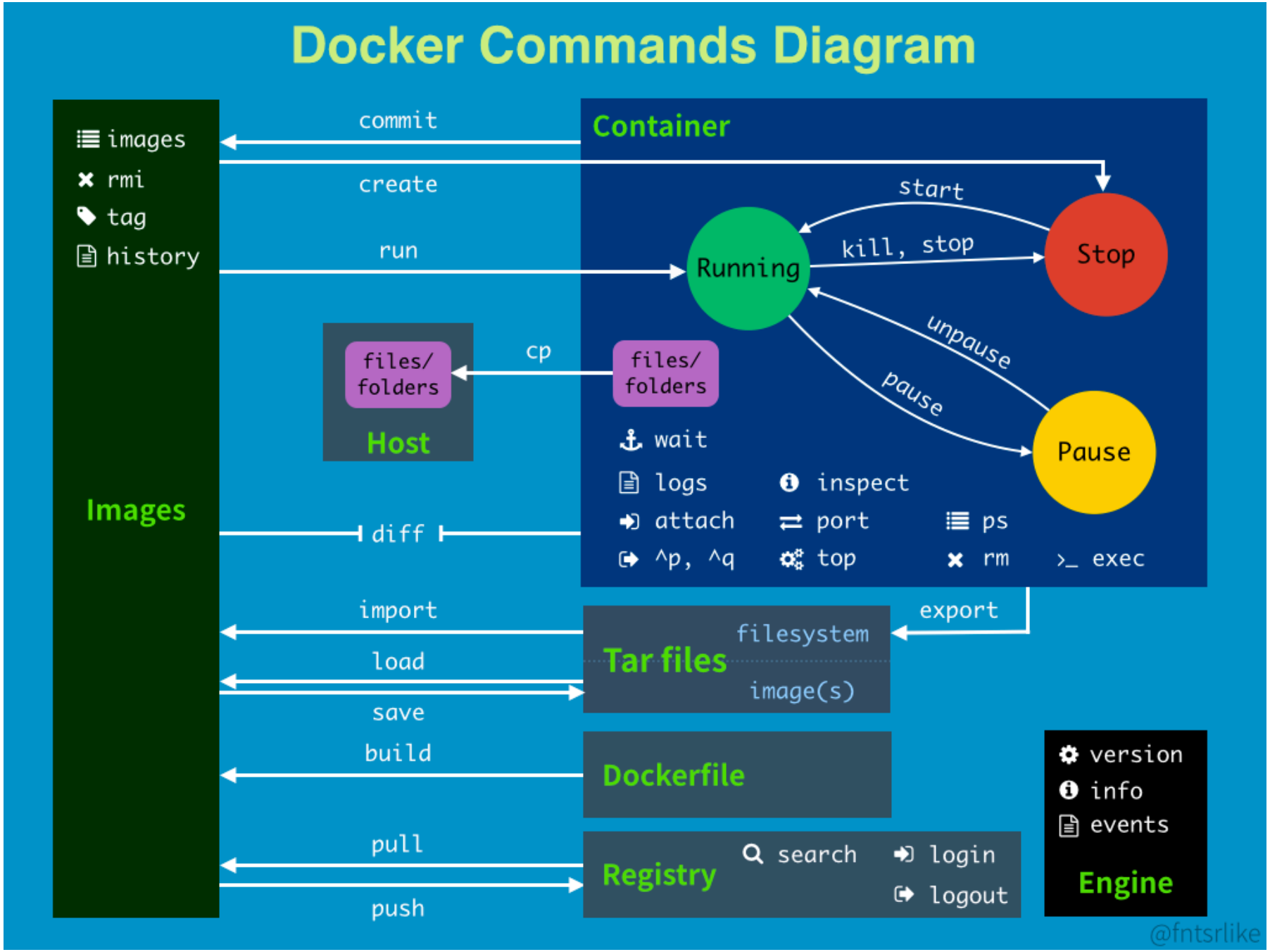
```
docker run -d alpine sleep 60

/usr/bin/dockerd -H fd://
\_ docker-containerd -l
         unix:///var/run/docker/libcontainerd/
                docker-containerd.sock ...
    \_ docker-containerd-shim 3da7...
         /var/run/docker/libcontainerd/ 3da7..
                docker-runc
        \_ sleep 60
```

Re-Parenting

1. shim launches runc
2. runc launches container
3. runc exits
4. shim becomes parent of container



- We can run hundreds of containers without having to run hundreds of runc instances
- Shim keeps any STDIN and STDOUT streams open so that when the daemon is restarted, the container doesn't terminate due to pipes being closed etc (daemon upgrade in production…)
- Shim reports the container's exit status back to the daemon


Docker Commands Diagram
images
rmi
tag
history
Images
commit
create
run
cp
files/
folders
Host
files/
folders
Container
start
Running
kill, stop
Stop
unpause
pause
Pause
wait
logs
inspect
attach
port
ps
^p, ^q
top
rm
>_ exec
diff
import
load
save
export
Tar files
filesystem
image(s)
build
Dockerfile
pull
push
Registry
search
login
logout
version
info
events
Engine
@fntsrlike

Container Internals

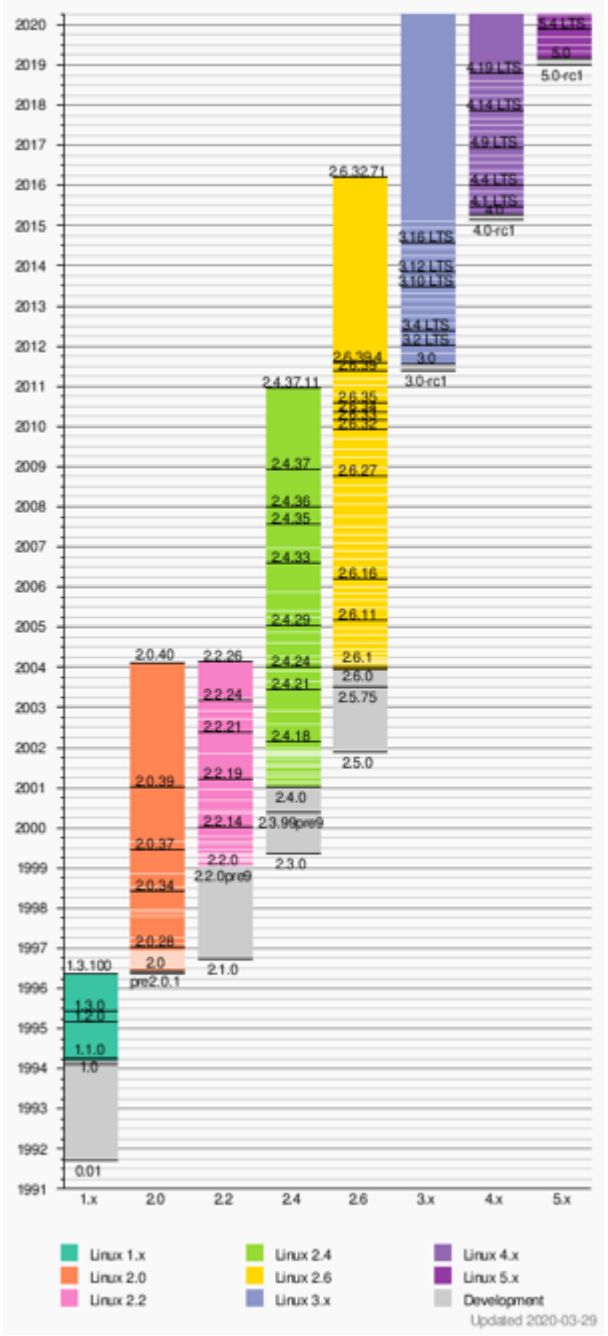
Namespace & Cgroup

Namespaces in Kernel

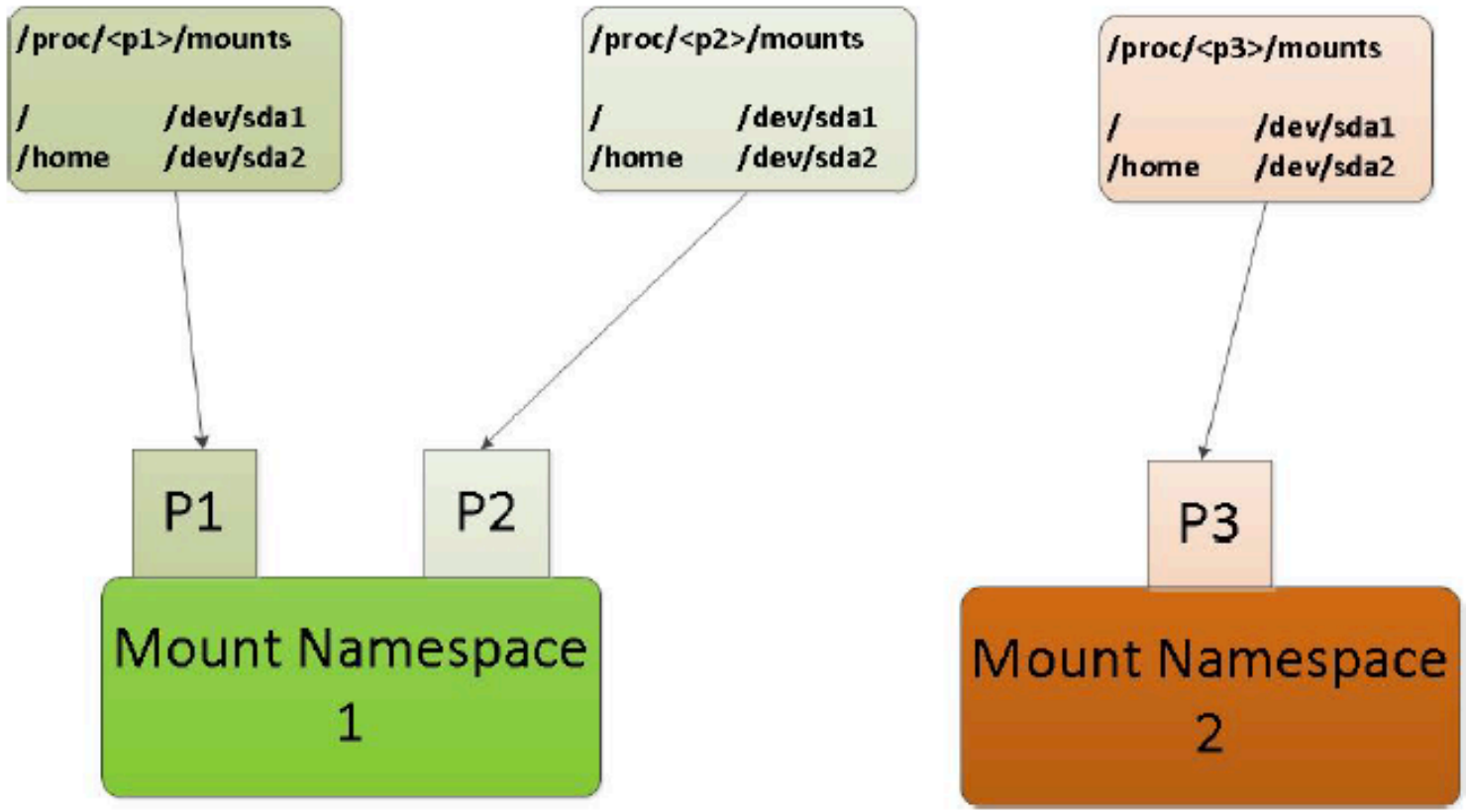
Restricting visibility

Namespaces:

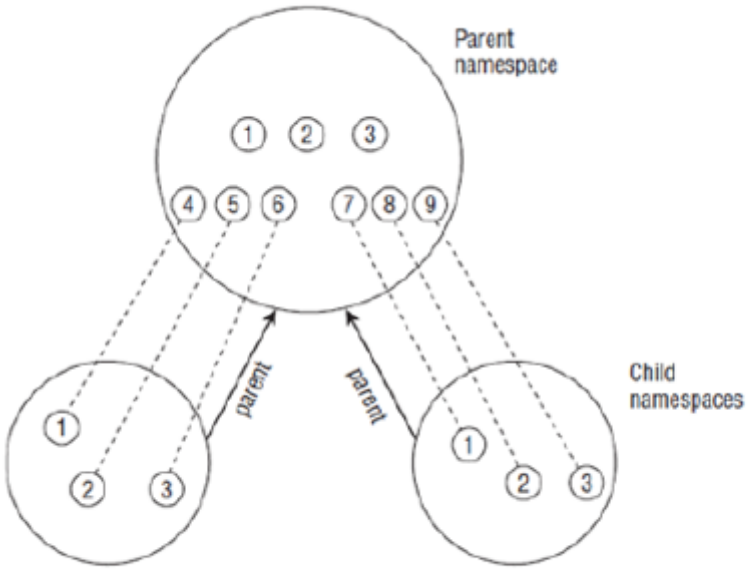
- cgroup 4.6, 2016
- ipc 2.6.19, 2006
- mnt 2.4.19, 2002
- net 2.6.24, 2006
- pid 2.6.24, 2006
- user 3.8, 2013
- uts 2.6.19, 2006

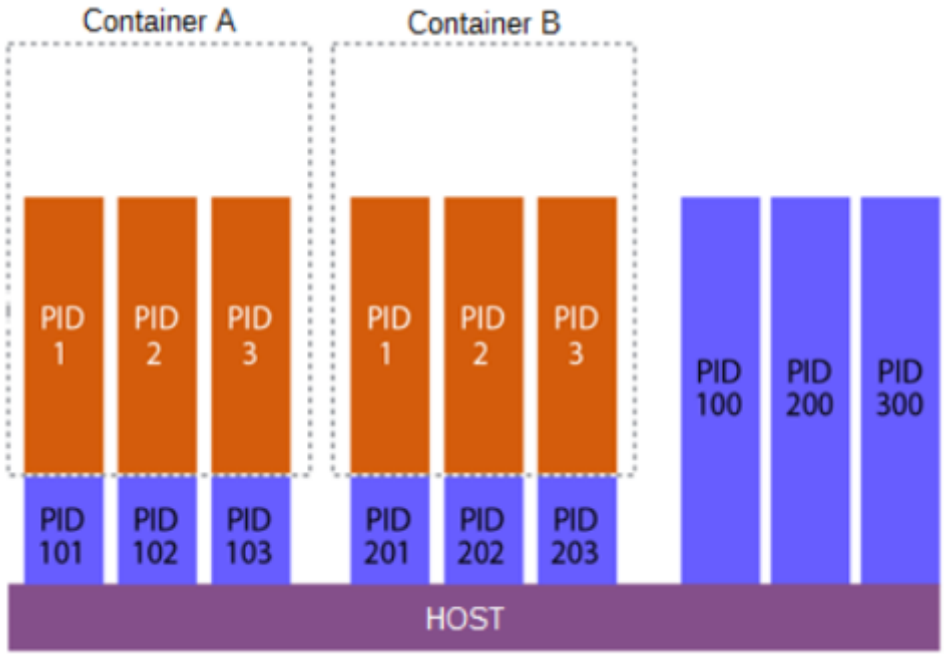


Mount Namespace

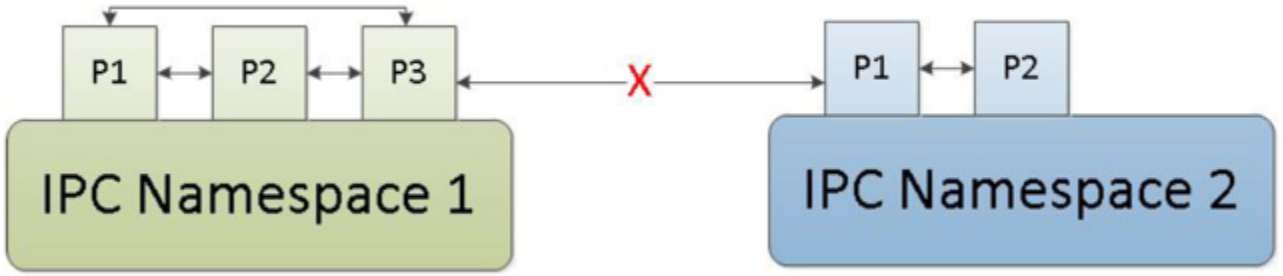


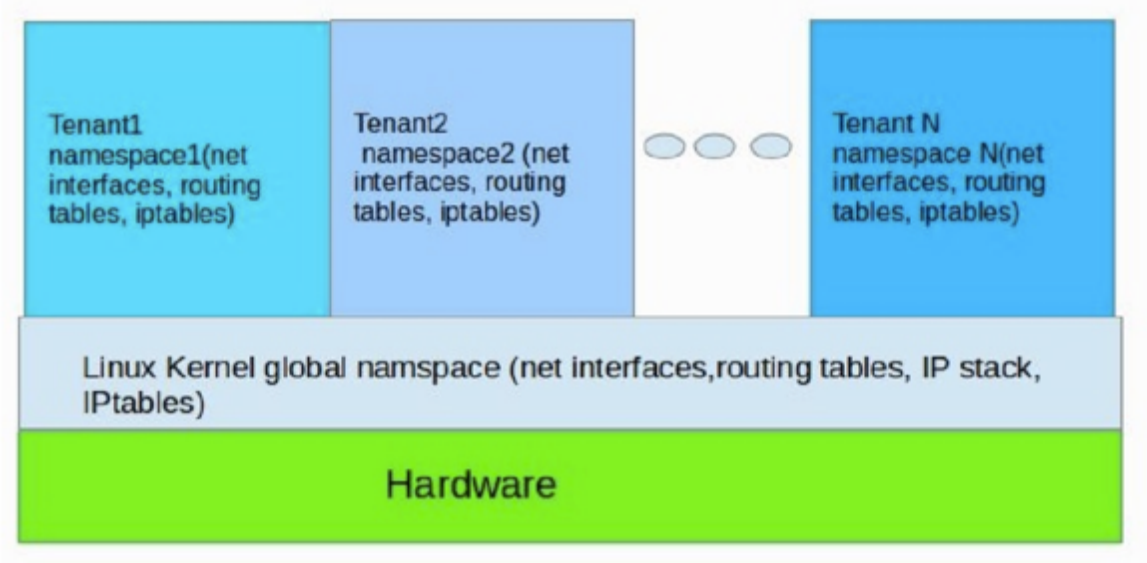
PID Namespace





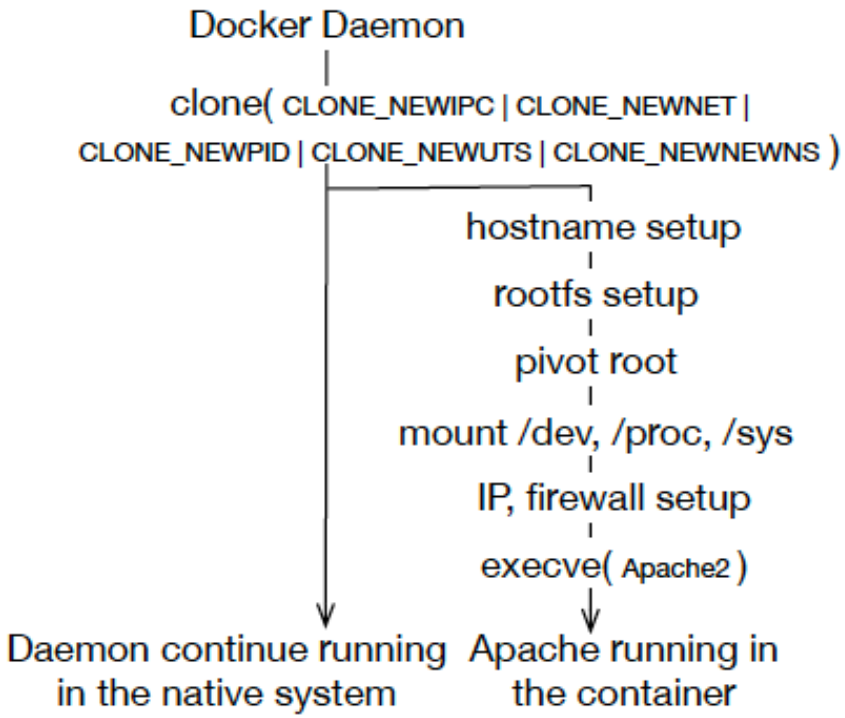
IPC & Network Namespace





Docker Namespaces

Syscalls: clone(2), setns(2), unshare(2)

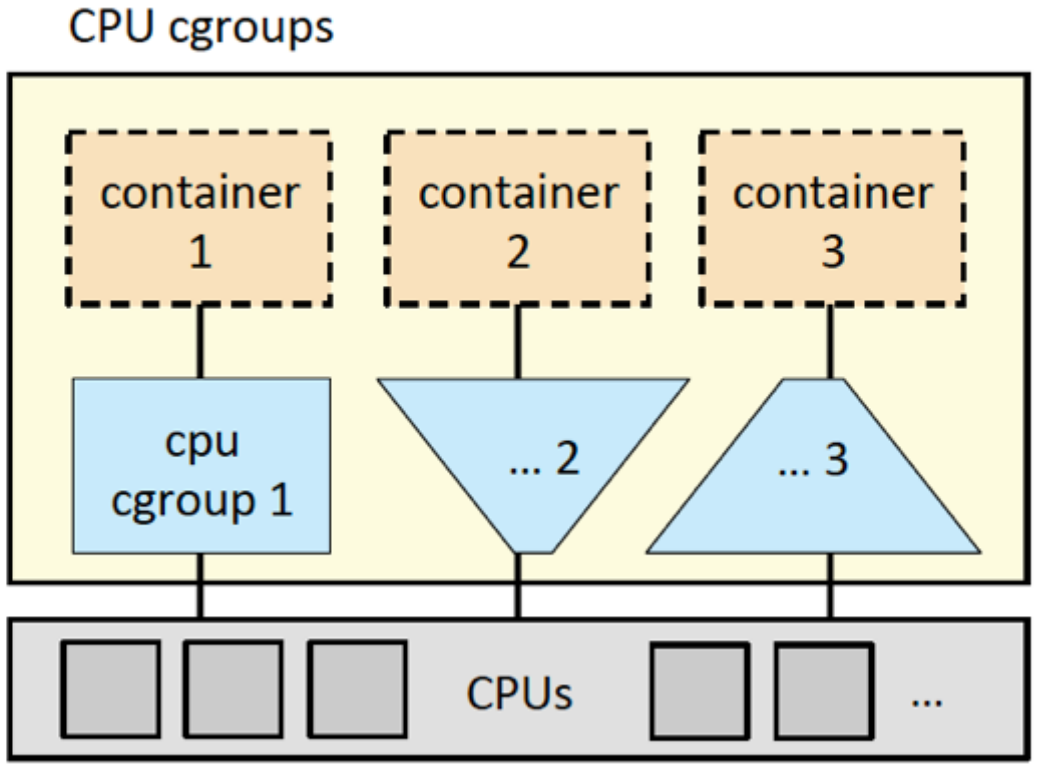


cgroups

Restricting usage

cgroups:

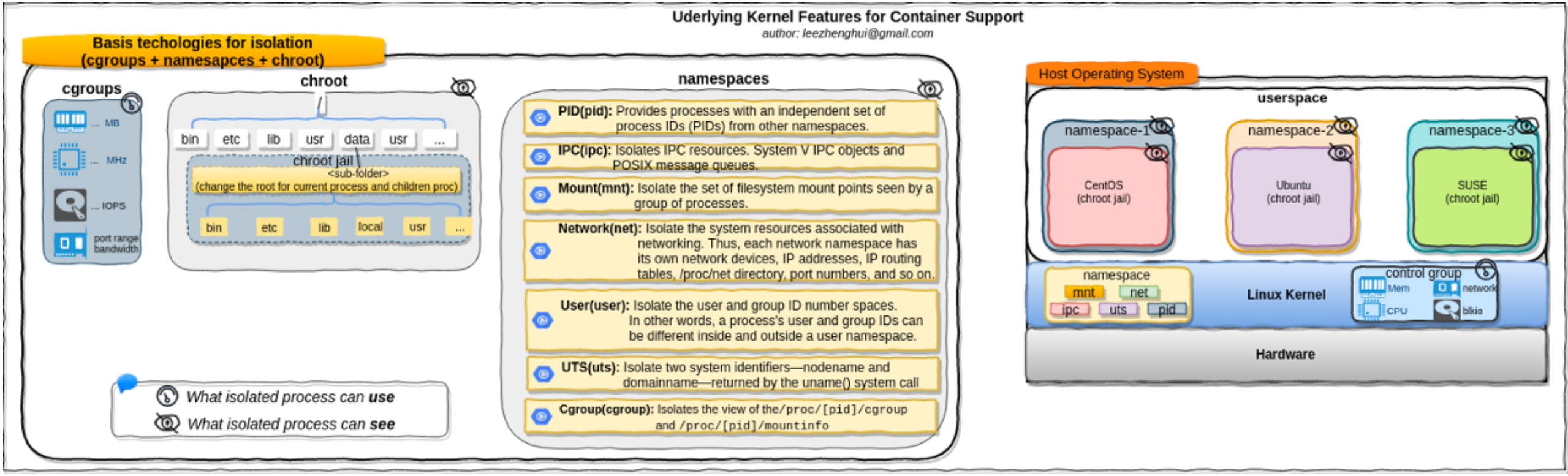
- blkio
- **cpu,cpuacct**
- cpuset
- devices
- hugetlb
- **memory**
- net_cls,net_prio
- pids
- …



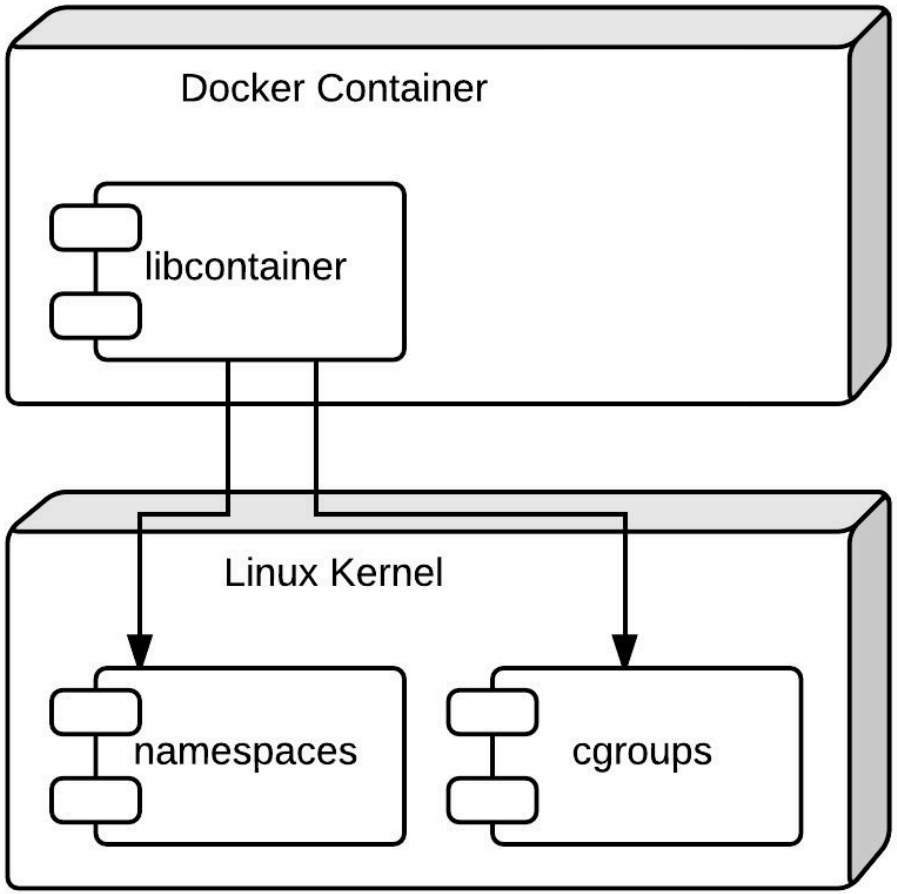
Namespace and cgroups Test

- Namespace creation test
  - unshare --fork --pid --mount-proc bash
- cgroups test
  - sudo cgcreate -a ssut -g memory:testgrp (test group create)
  - echo 2000000 > /sys/fs/cgroup/memory/testgrp/memory.kmem.limit_in_bytes (memory limit)
- docker exec = nsenter + cgroups (resource limit)
- sudo ls /proc/`docker inspect -f '{{ .State.Pid }}' container0`/ns –liah
  - 1722671 lrwxrwxrwx 1 root root 0 Mar 14 17:33 cgroup -> cgroup:[4026531835]
  - 1722667 lrwxrwxrwx 1 root root 0 Mar 14 17:33 ipc -> ipc:[4026532634]
  - 1722670 lrwxrwxrwx 1 root root 0 Mar 14 17:33 mnt -> mnt:[4026532632]
  - 1589019 lrwxrwxrwx 1 root root 0 Mar 14 16:35 net -> net:[4026532637]
  - 1722668 lrwxrwxrwx 1 root root 0 Mar 14 17:33 pid -> pid:[4026532635]

Namespace and Cgroups



libcontainer

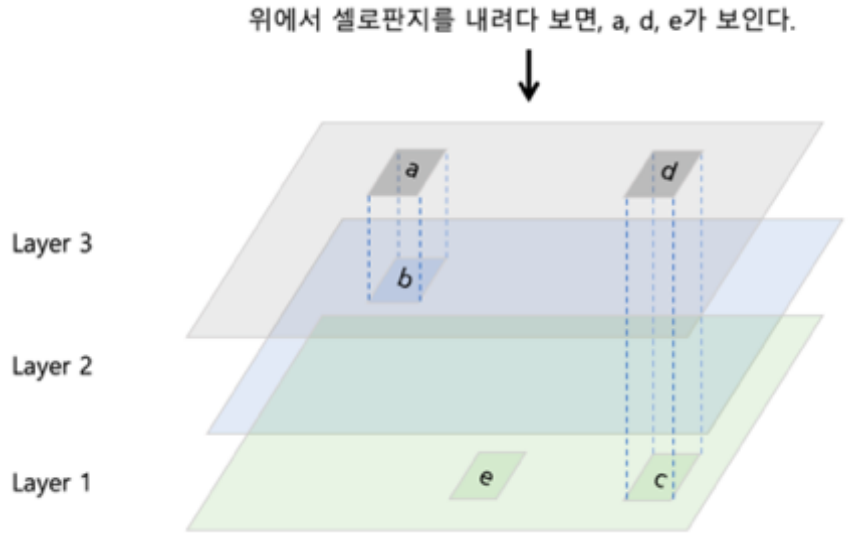


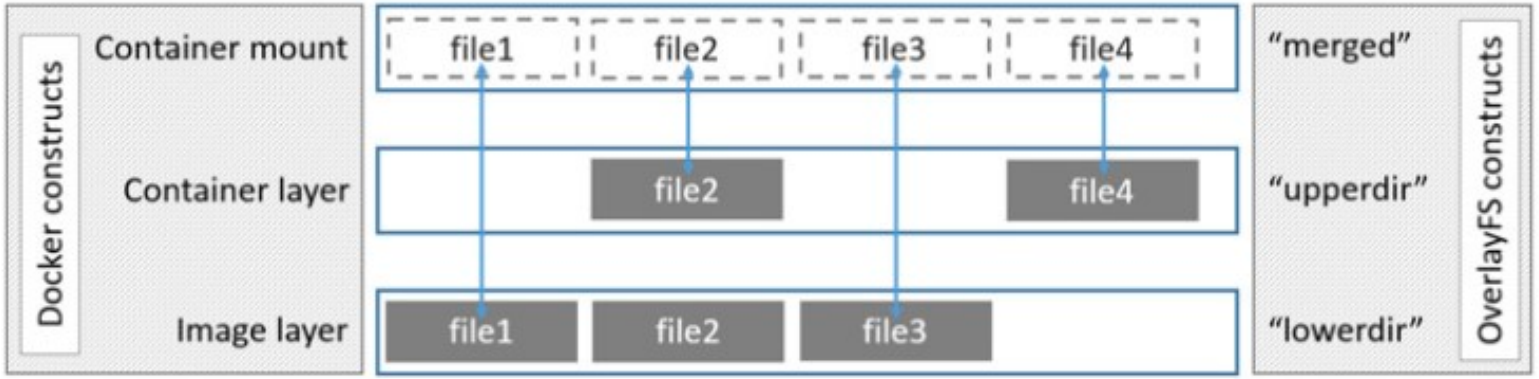
Container Internals

Storage, Image, Network

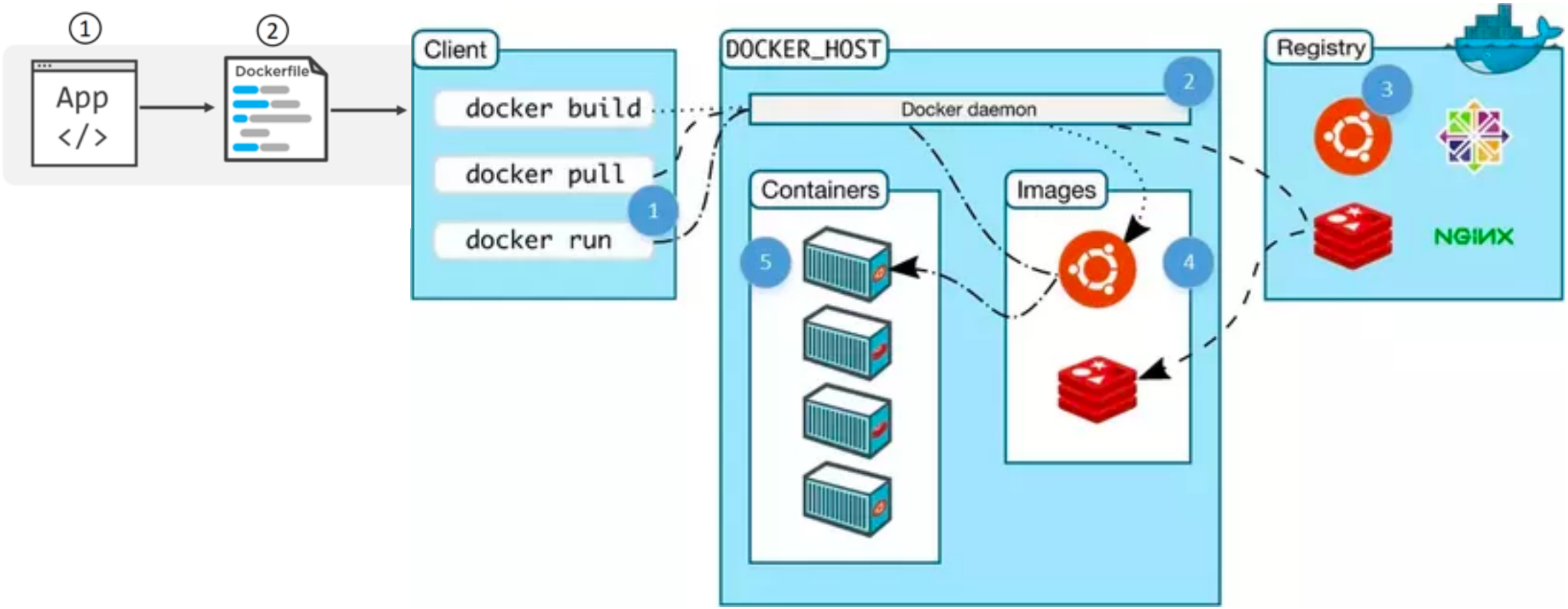
Docker Storage Driver

- /etc/docker/daemon.json (overlay2 by default since ver. 4.0)
- Union FS
  - File system which implements a union mount for other file system (AUFS: old, overlay2: kernel ~3.18, device mapper: block level CoW, btrfs: for big data, zfs and windowsfilter: for windows)



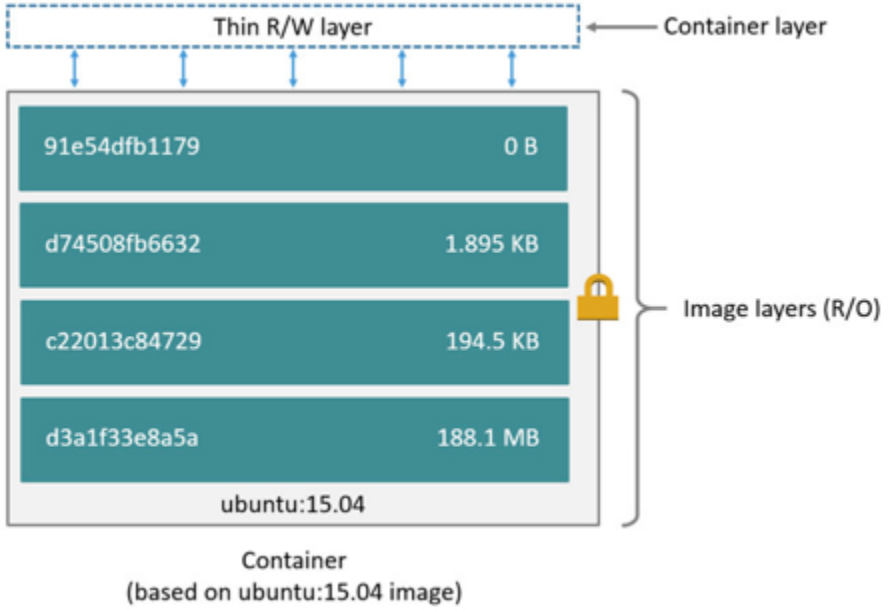


Docker Image



Inside Docker Image

- Location
  - Linux: /var/lib/docker/<storage-driver>
  - Windows: C:\ProgramData\docker\windowsfilter
- Digest
  - Cryptographic content hash (immutable)



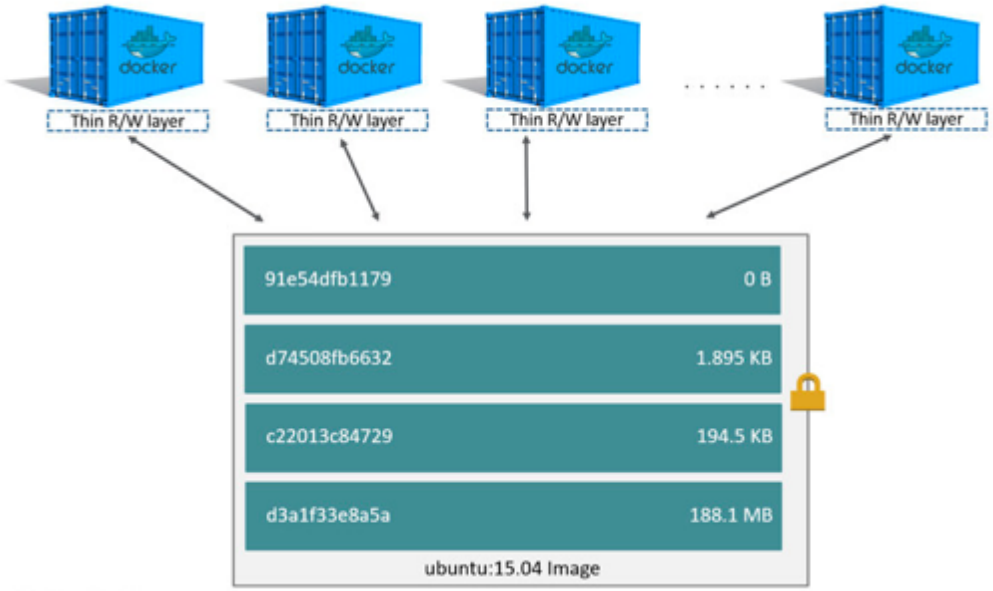
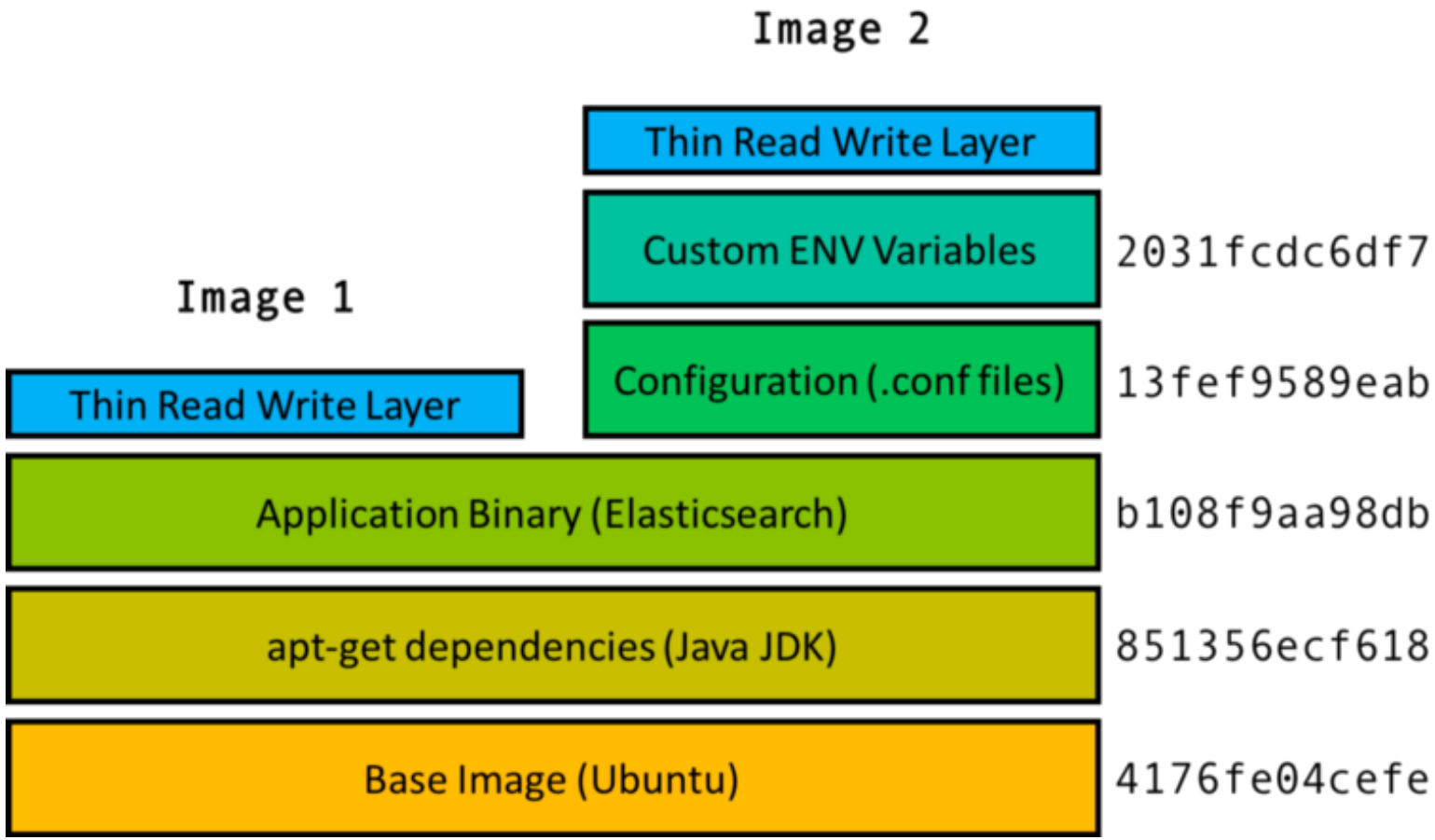
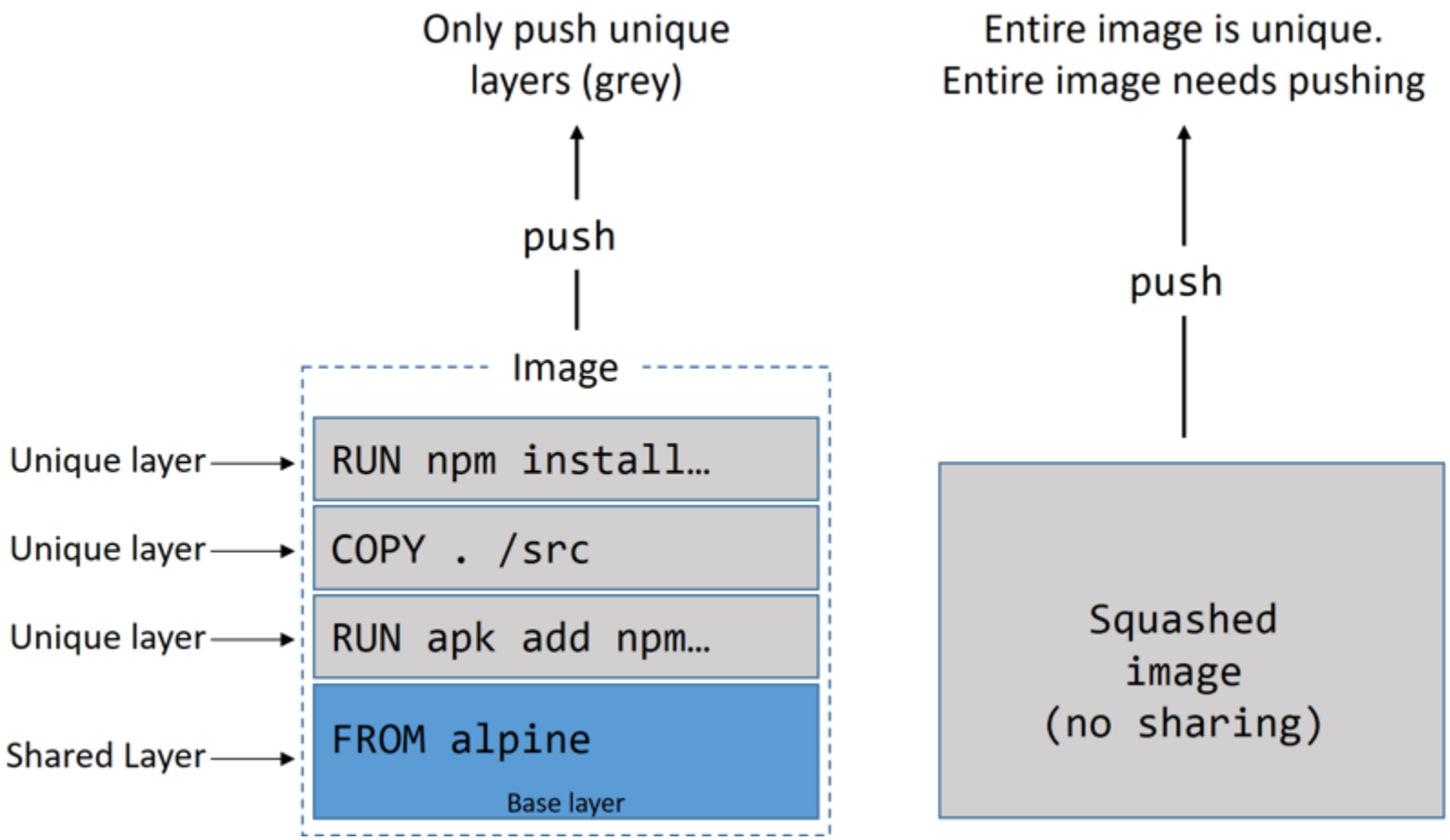
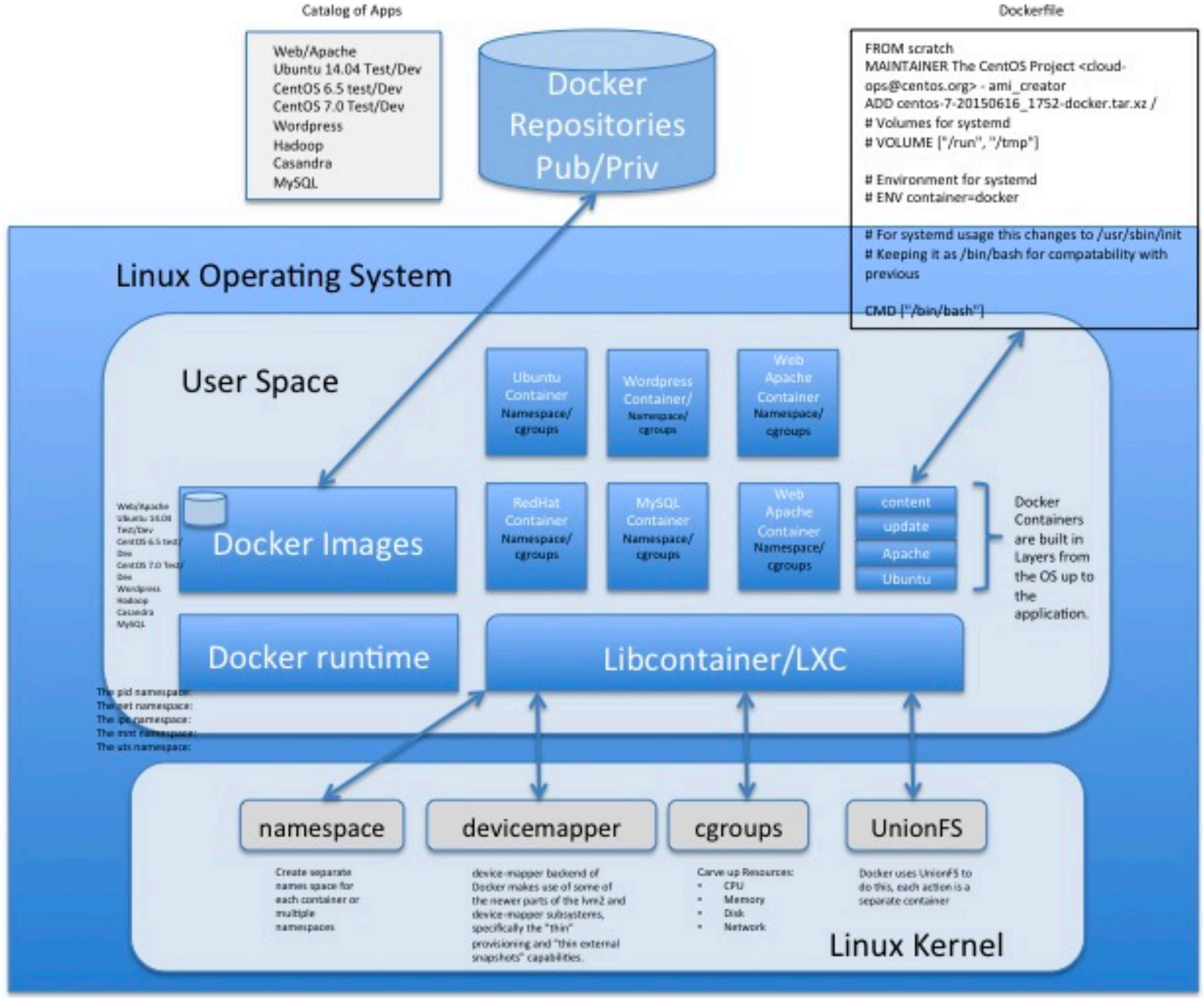


Image Sharing

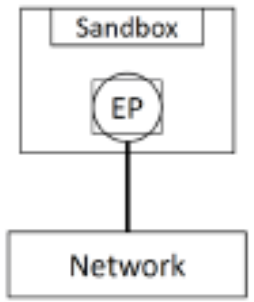


Squash the Image



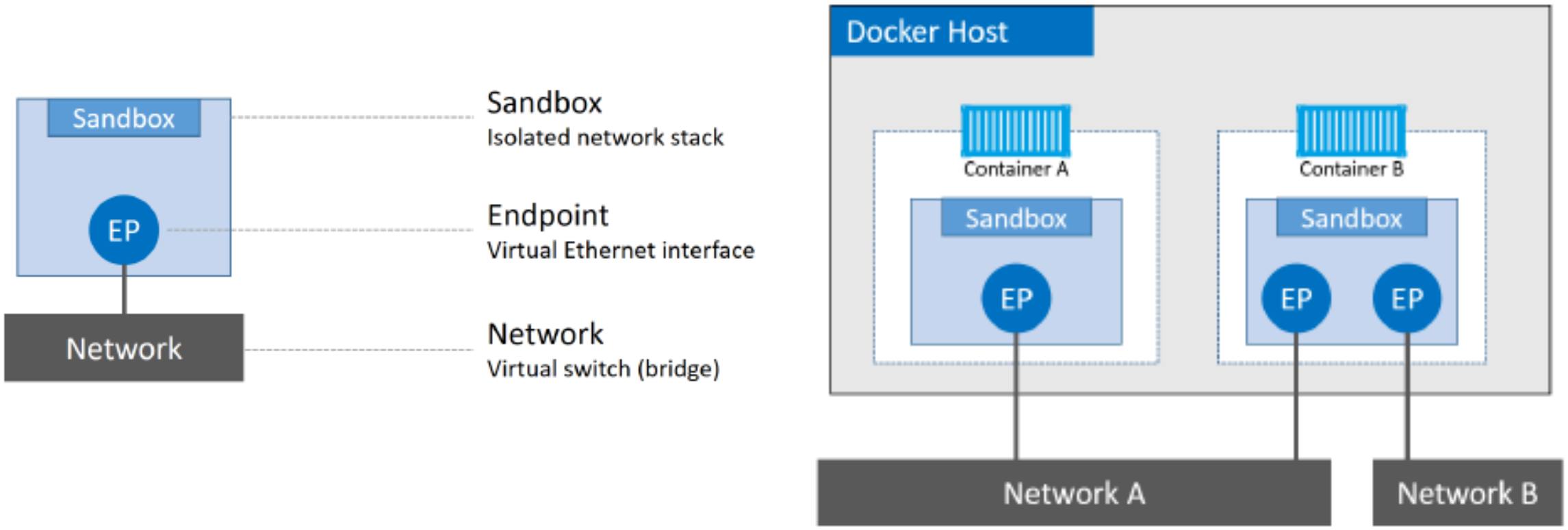

Catalog of Apps
Web/Apache
Ubuntu 14.04 Test/Dev
CentOS 6.5 test/Dev
CentOS 7.0 Test/Dev
Wordpress
Hadoop
Casandra
MySQL
Docker Repositories Pub/Priv
Dockerfile
FROM scratch
MAINTAINER The CentOS Project <cloud-ops@centos.org> - ami_creator
ADD centos-7-20150616_1752-docker.tar.xz /
Volumes for systemd
VOLUME ["/run", "/tmp"]
Environment for systemd
ENV container=docker
For systemd usage this changes to /usr/sbin/init
Keeping it as /bin/bash for compatability with previous
CMD ["/bin/bash"]
Linux Operating System
User Space
Ubuntu Container Namespace/ cgroups
Wordpress Container/ Namespace/ cgroups
Web Apache Container Namespace/ cgroups
Docker Images
RedHat Container Namespace/ cgroups
MySQL Container Namespace/ cgroups
Web Apache Container Namespace/ cgroups
content
update
Apache
Ubuntu
Docker Containers are built in Layers from the OS up to the application.
Docker runtime
Libcontainer/LXC
namespace
devicemapper
cgroups
UnionFS
Create separate names space for each container or multiple namespaces
device-mapper backend of Docker makes use of some of the newer parts of the lvm2 and device-mapper subsystems, specifically the "thin" provisioning and "thin external snapshots" capabilities.
Carve up Resources:
CPU
Memory
Disk
Network
Docker uses UnionFS to do this, each action is a separate container
Linux Kernel

Docker Networking

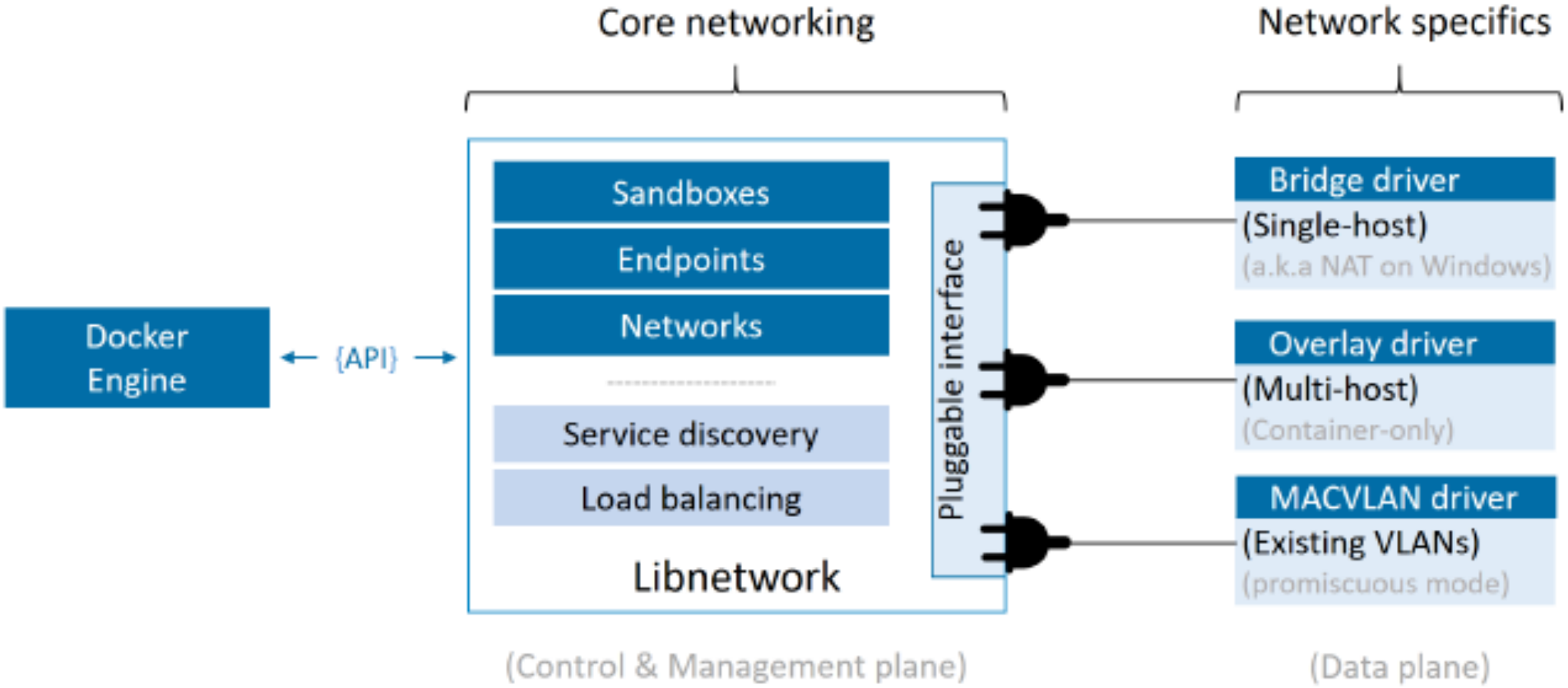


CNM	libnetwork	Drivers
Design (DNA)	Primitives Control & Management plane	Networks Data plane

Container Network Model (CNM)



Drivers

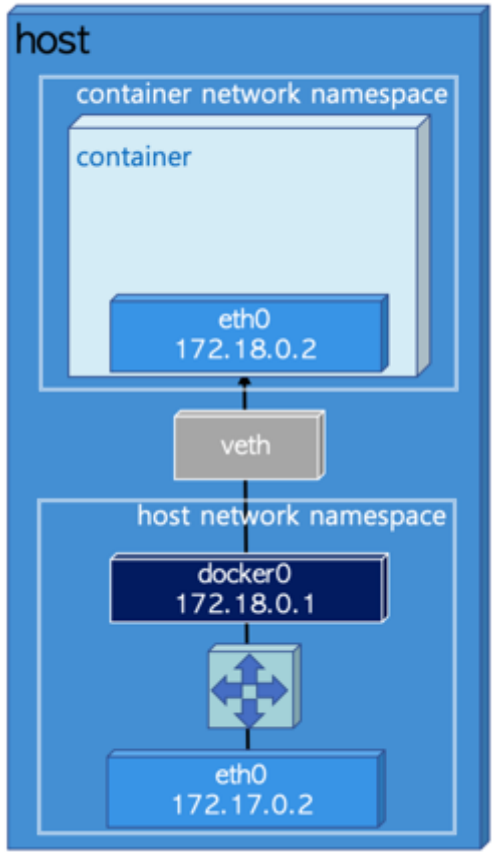


Windows: NAT, overlay, transparent

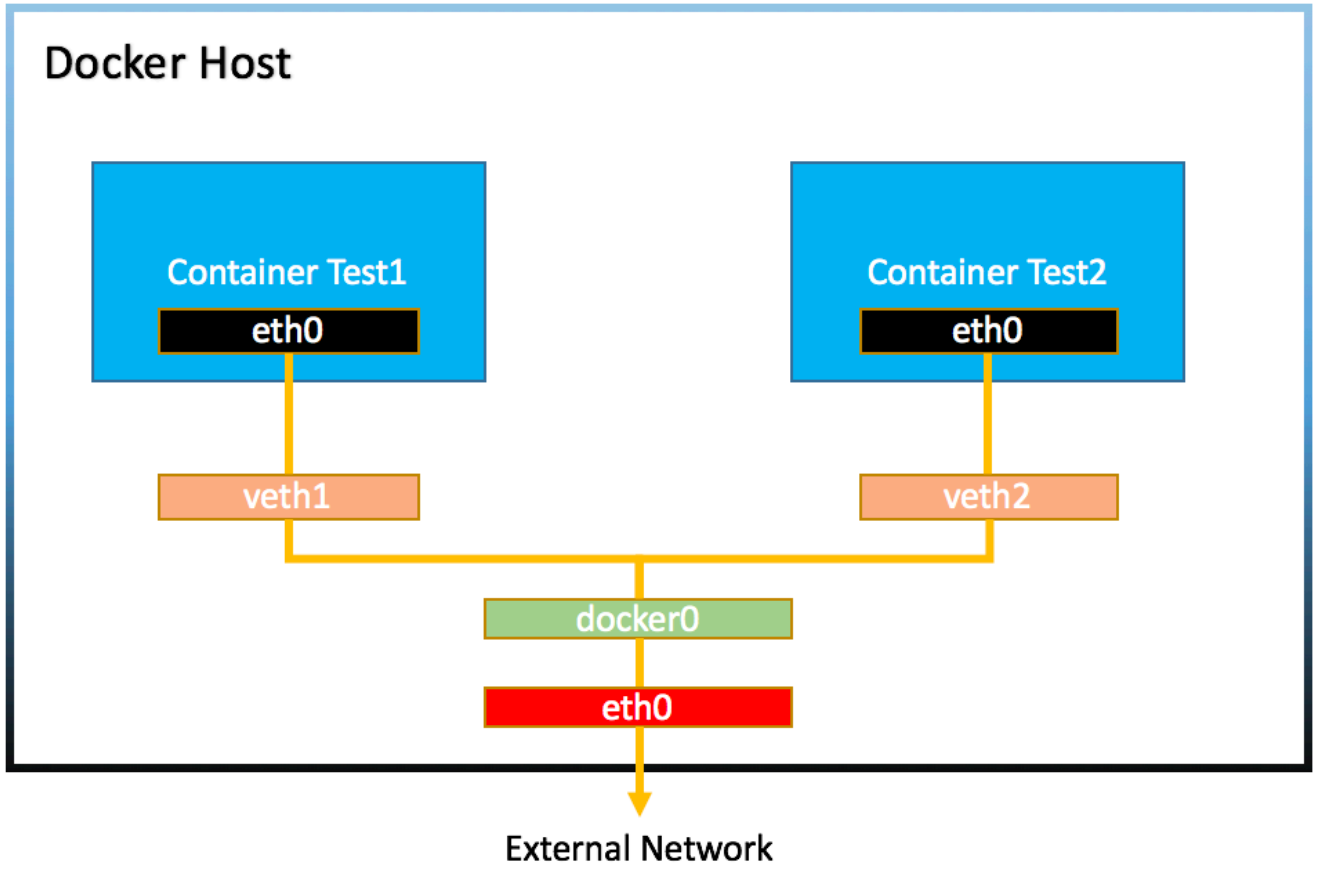
3rd parties: calico, contiv, kuryr, weave

Bridge Network

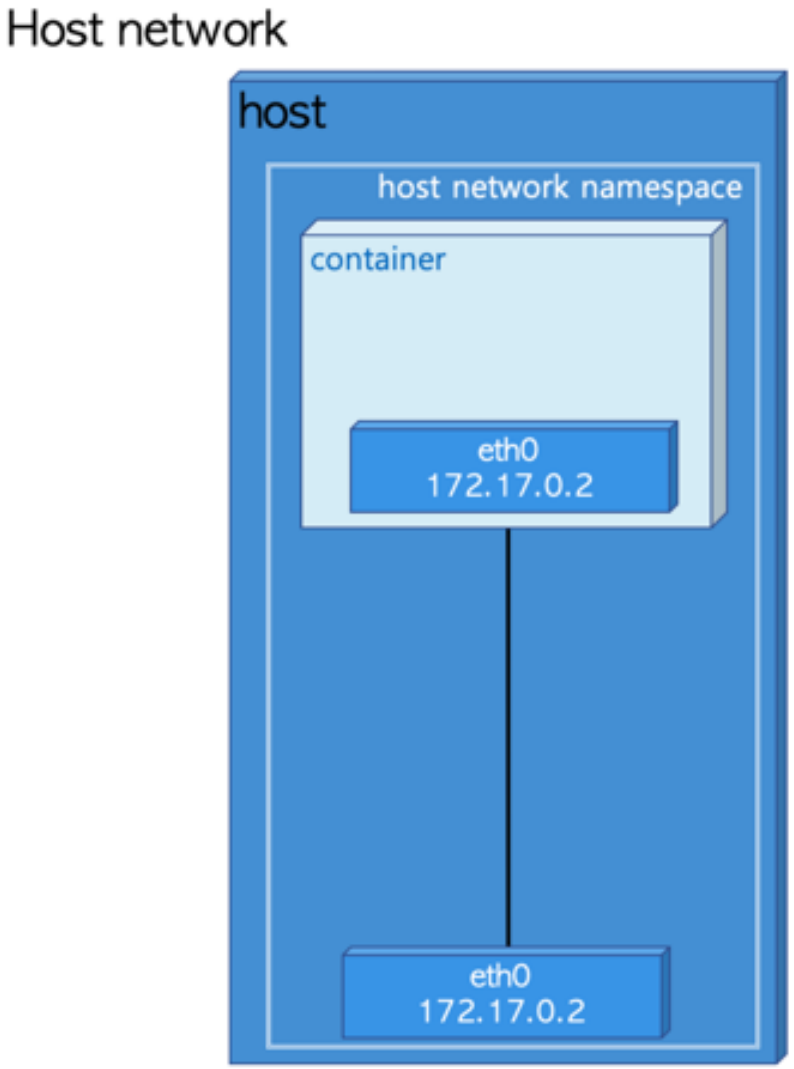
Bridge network



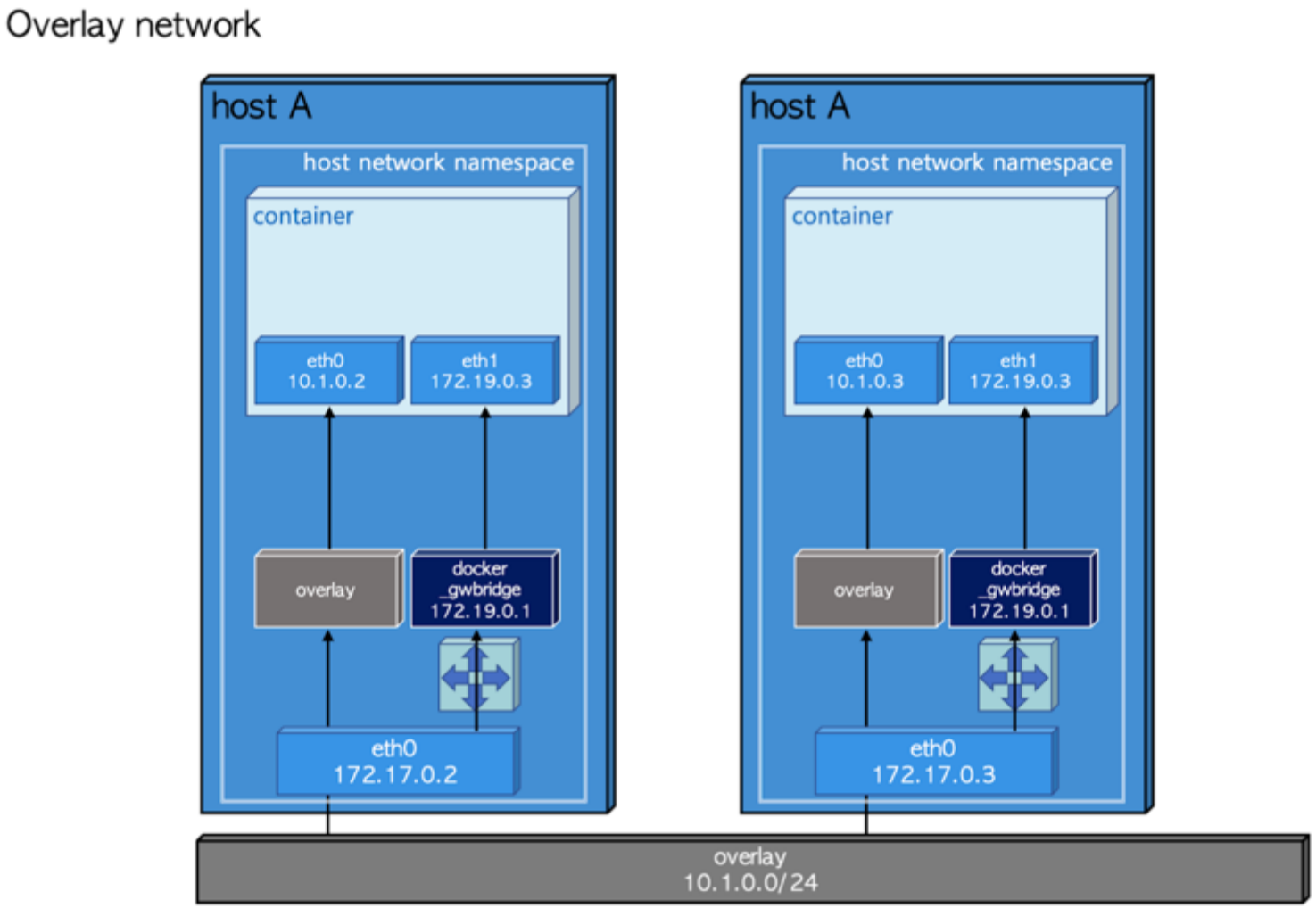
802.1d bridge (layer 2 switch)



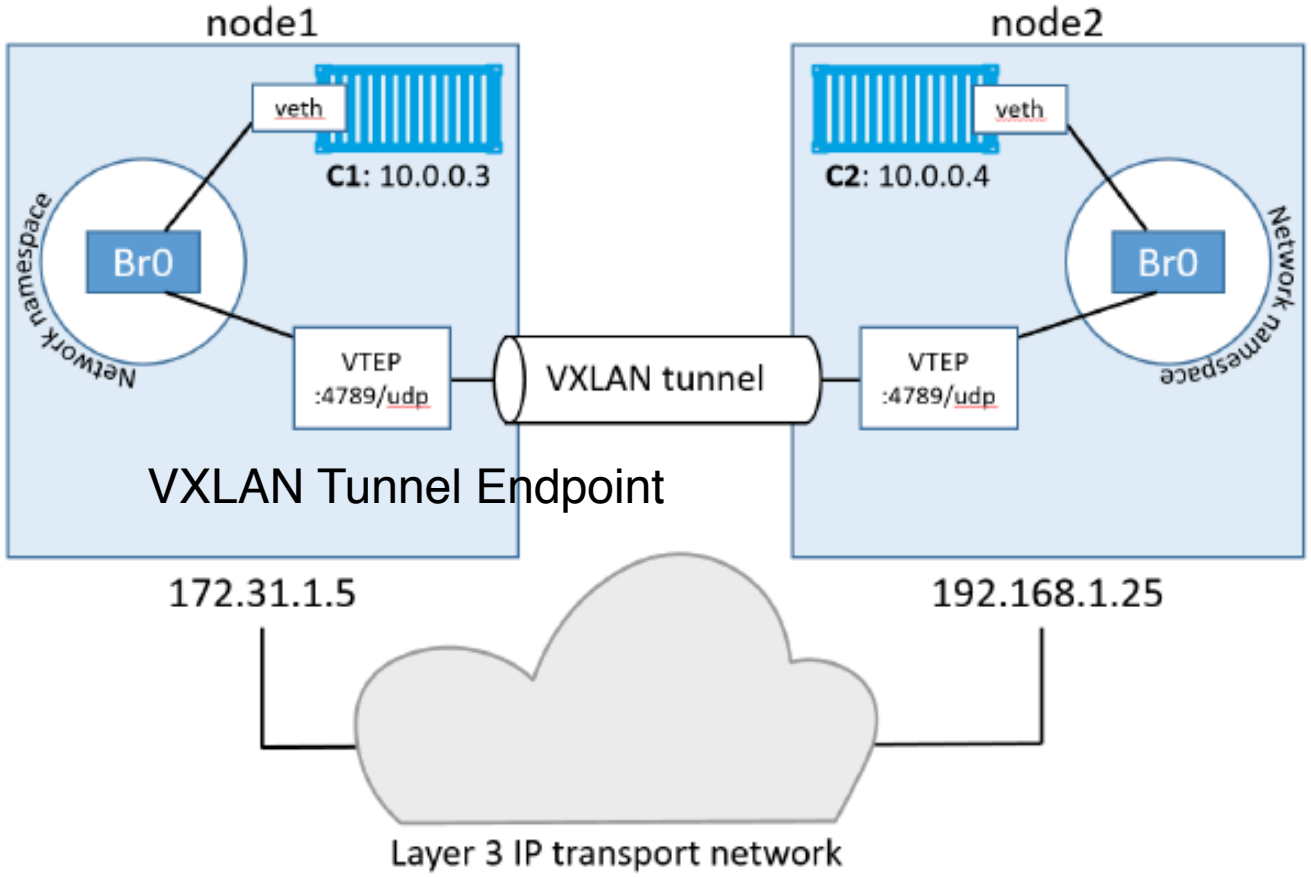
Host Network



Overlay Network

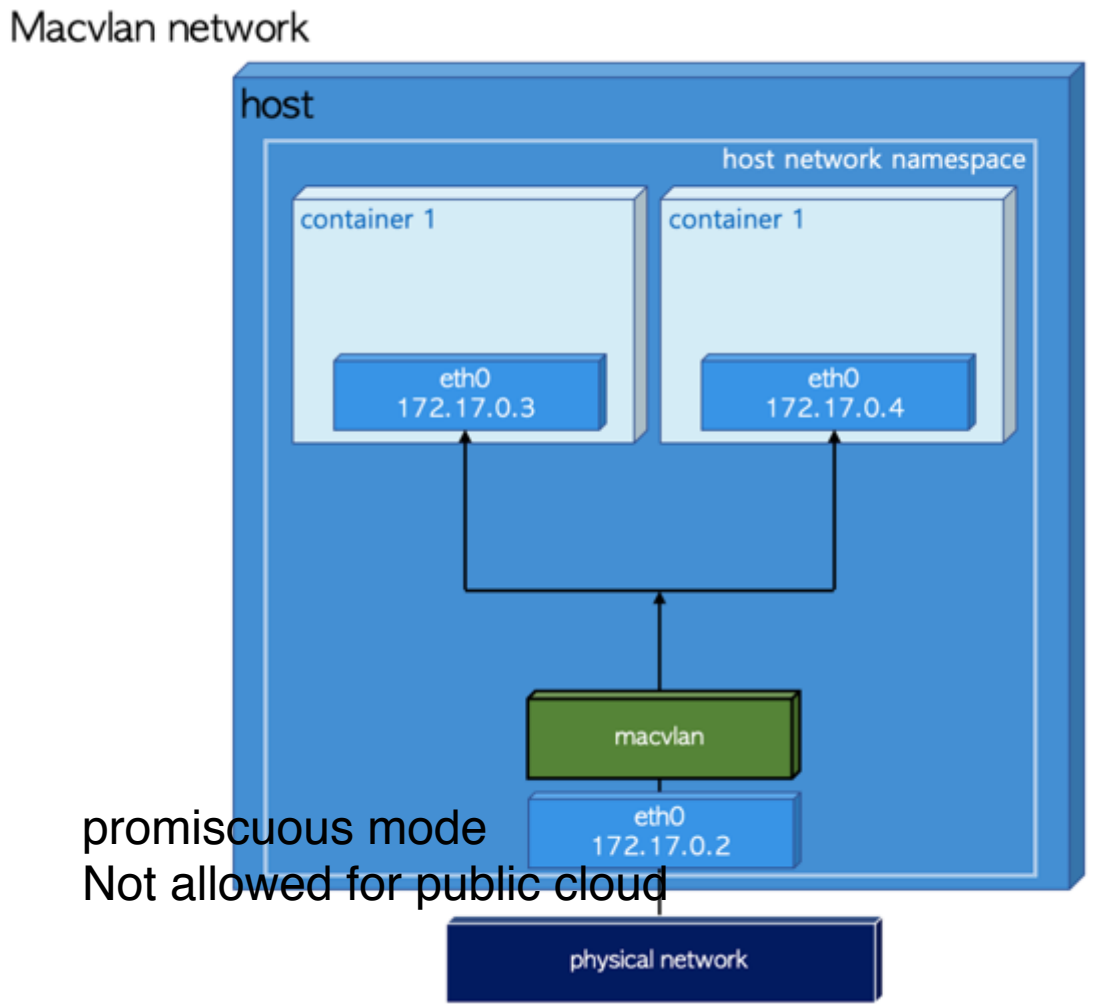


Overlay Network Details



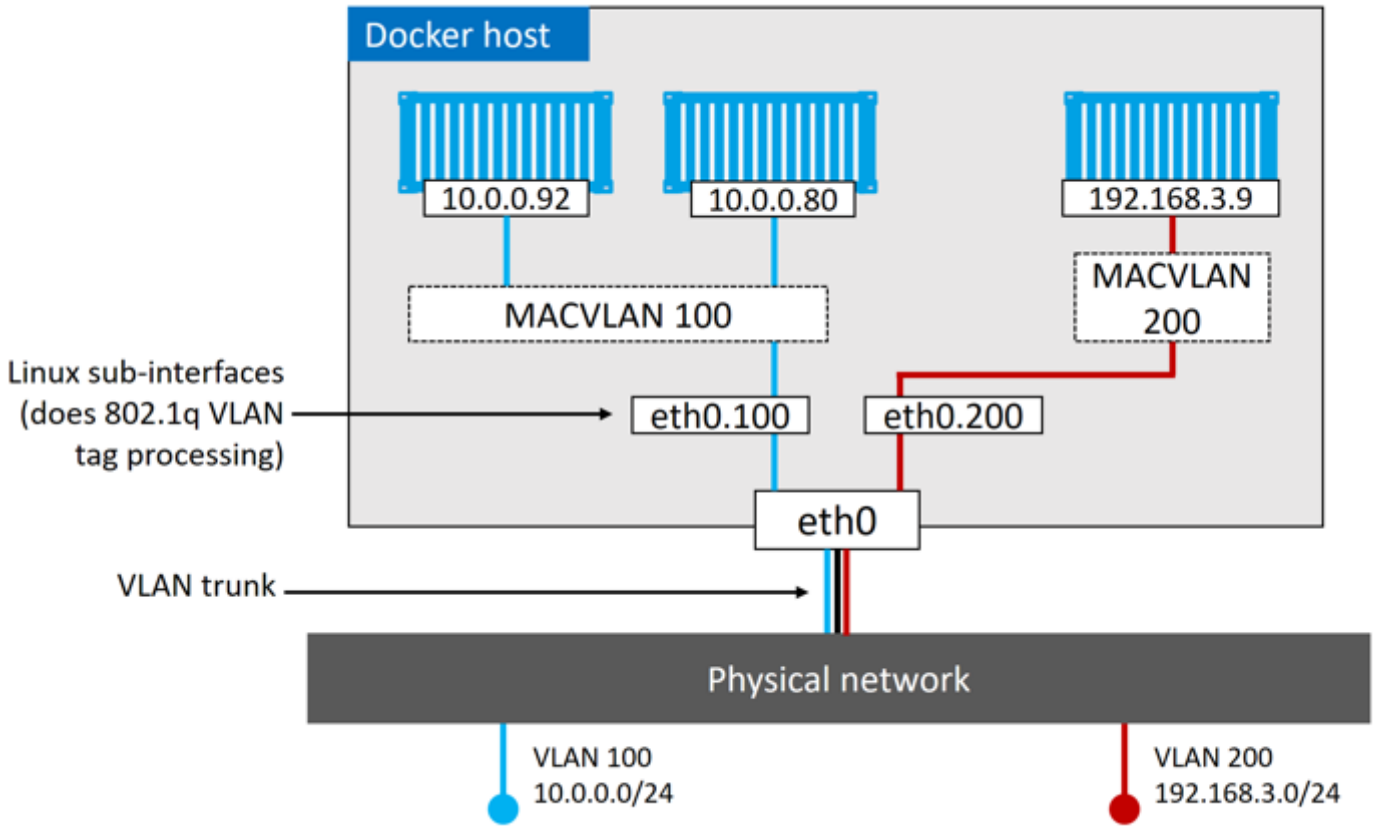

```
$ docker swarm init \
--advertise-addr=172.31.1.5 \
--listen-addr=172.31.1.5:2377

$ docker swarm join \
--token SWMTKN-1-0hz2ec...2vye \
172.31.1.5:2377

$ docker network create -d overlay uber-net
c740ydi1lm89khn5kd52skrd9
```

Macvlan Network



promiscuous mode
Not allowed for public cloud

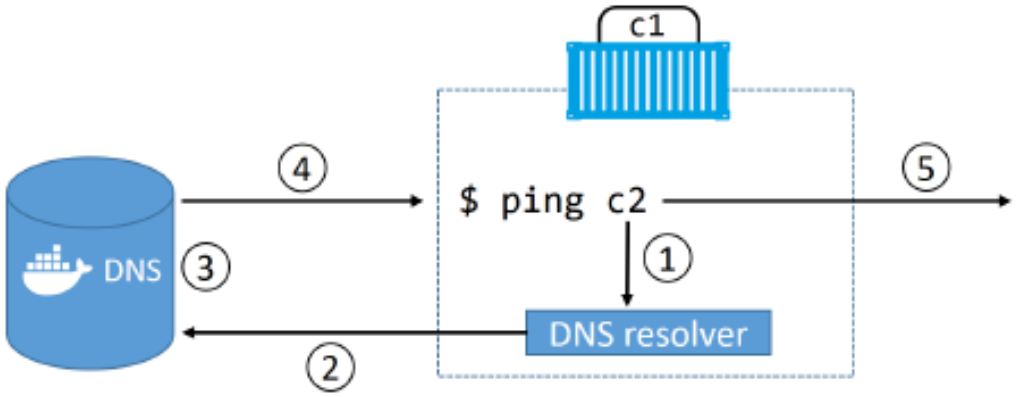
Macvlan Network Details

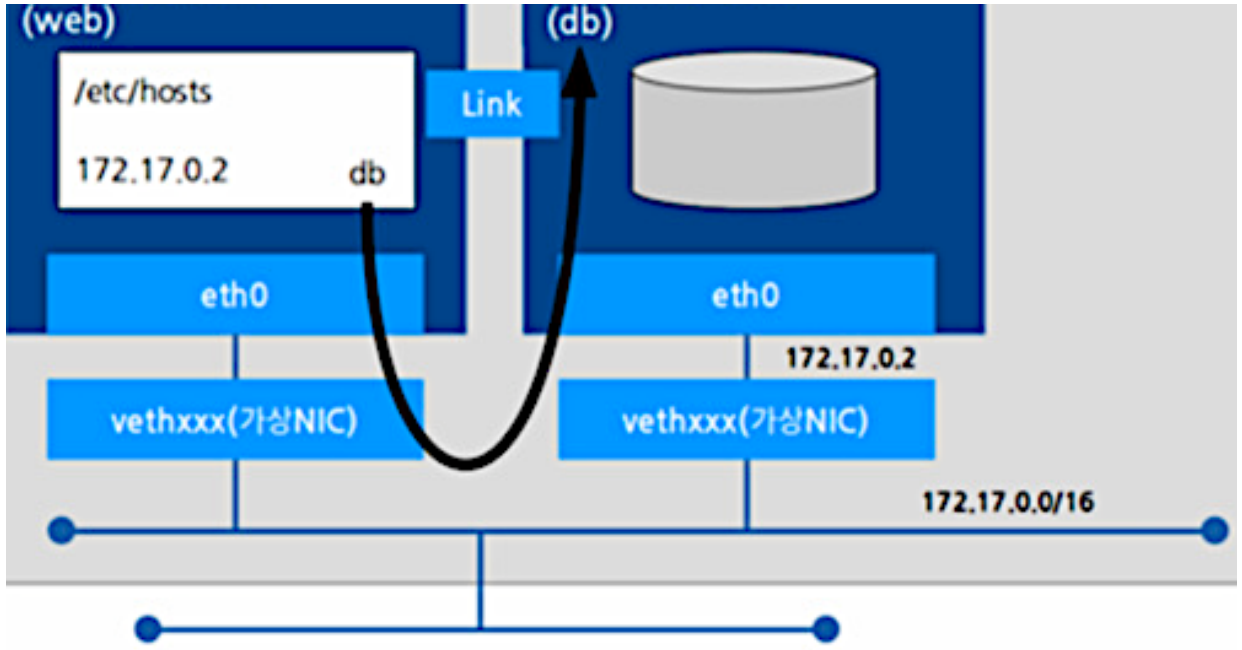



```
$ docker network create -d macvlan \
--subnet=10.0.0.0/24 \
--ip-range=10.0.00/25 \
--gateway=10.0.0.1 \
-o parent=eth0.100 \
macvlan100
```

```
$ docker container run -d --name mactainer1 \
--network macvlan100 \
alpine sleep 1d
```

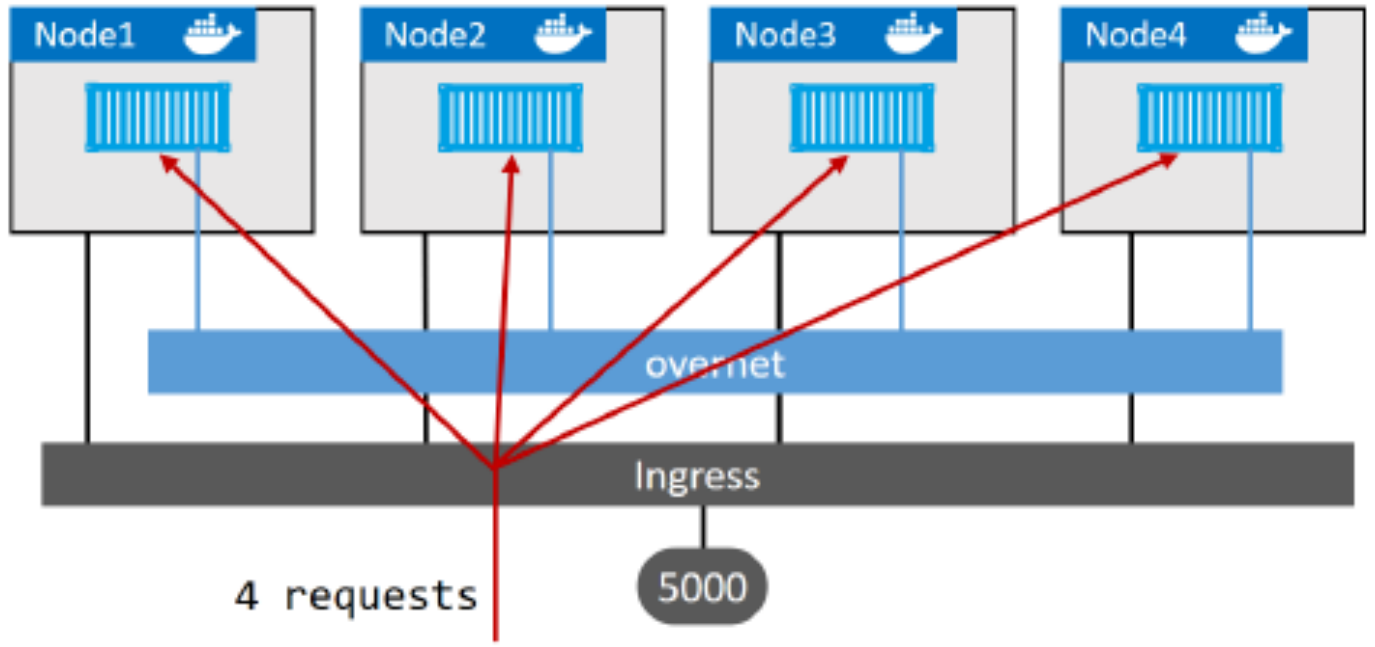
Service Discovery





Ingress Load Balancing

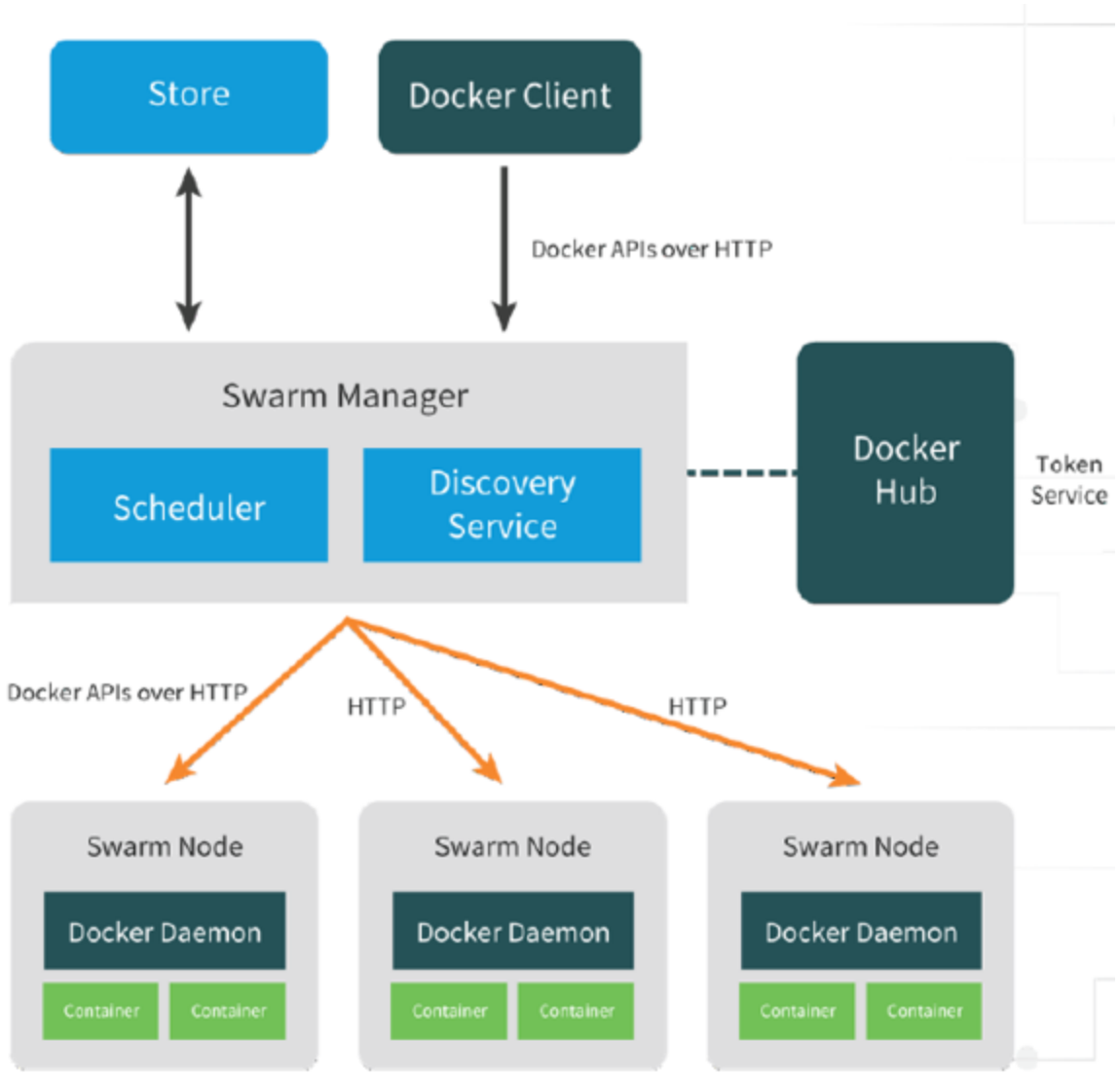
```
$ docker service create -d --name svc1 --network overnet \
  --replicas 4 \
  --publish published=5000,target=80 nginx
```



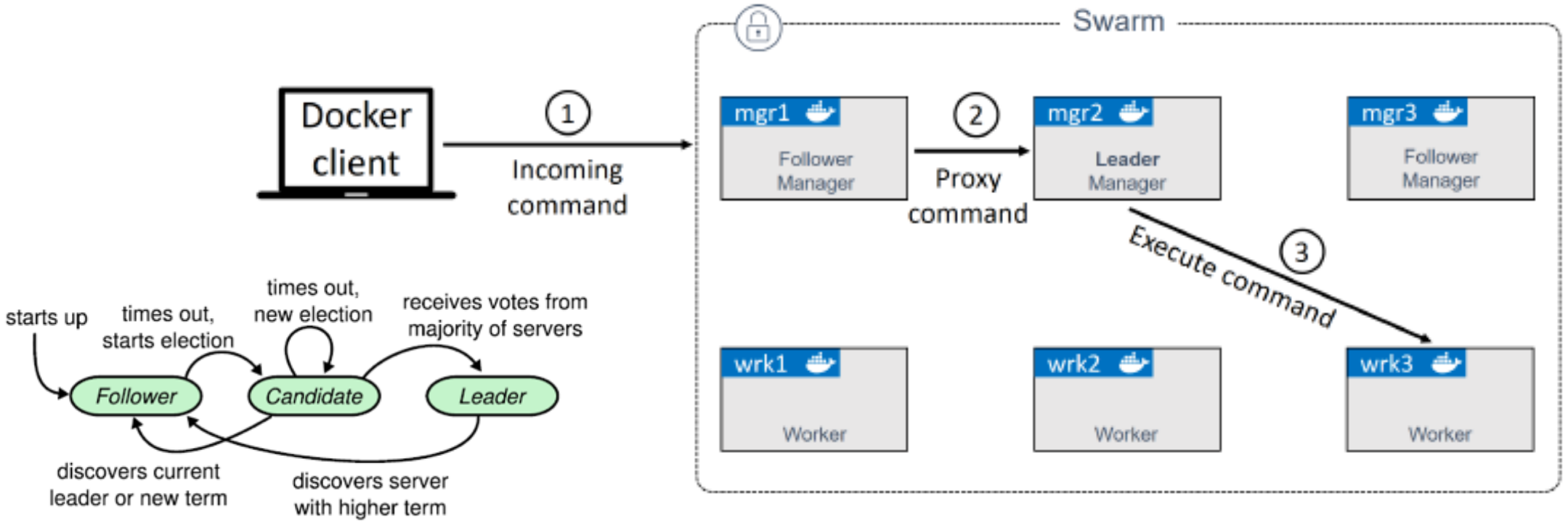
Container Internals

Docker Swarm

Docker Swarm Architecture



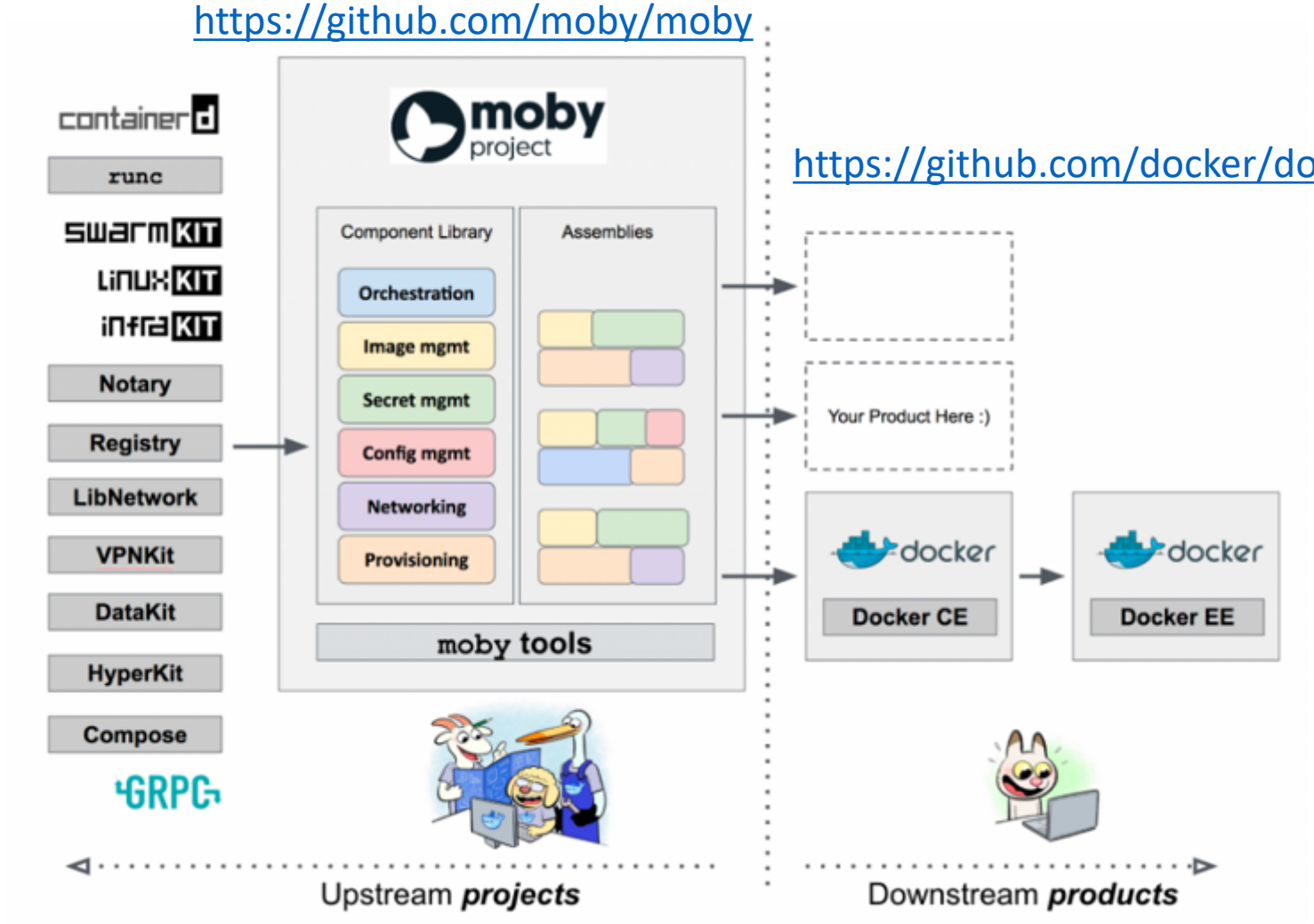
Docker Swarm (Managers)



Raft Consensus Algorithm

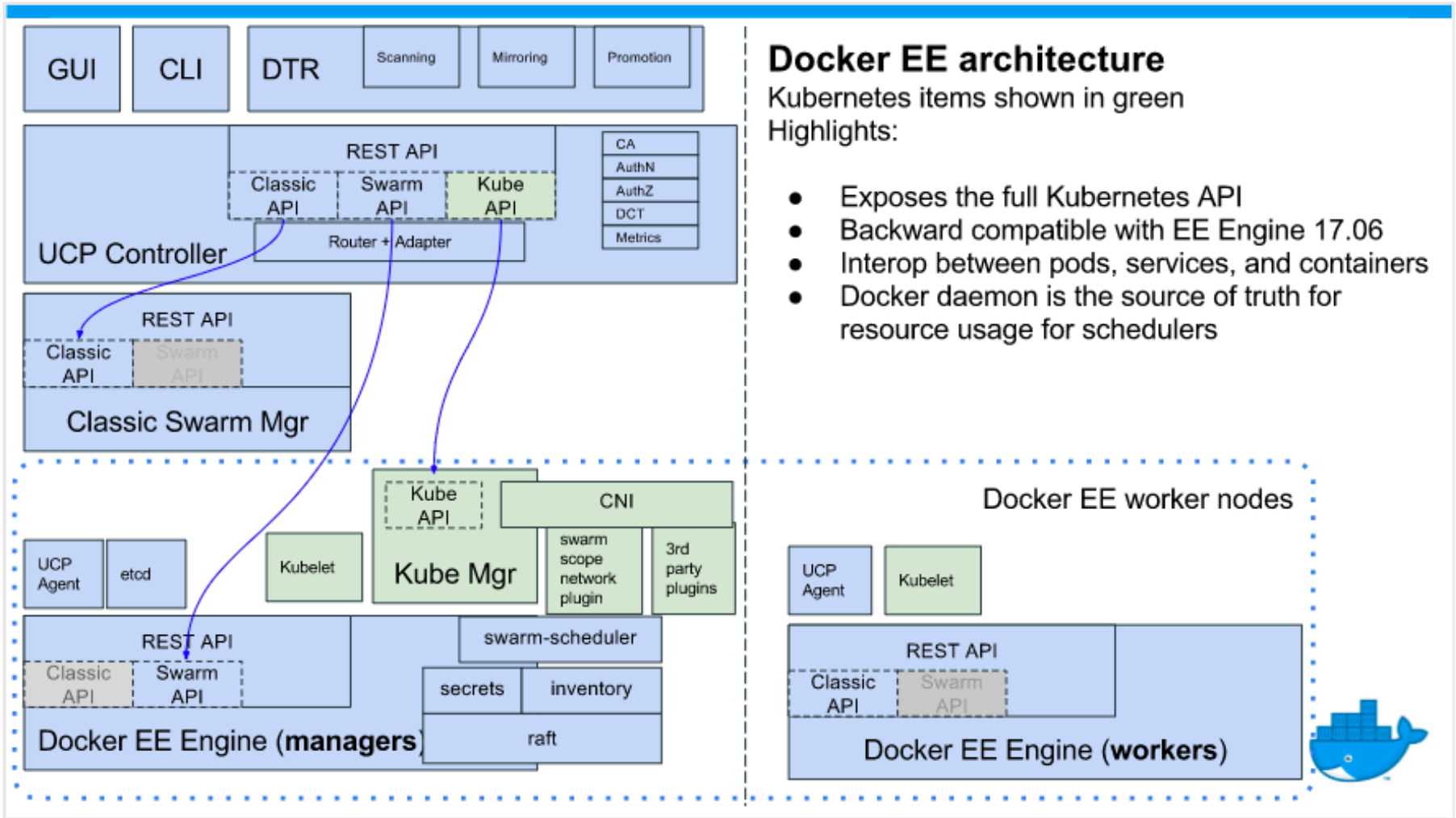
Moby project

https://github.com/moby/moby



https://github.com/docker/docker-ce

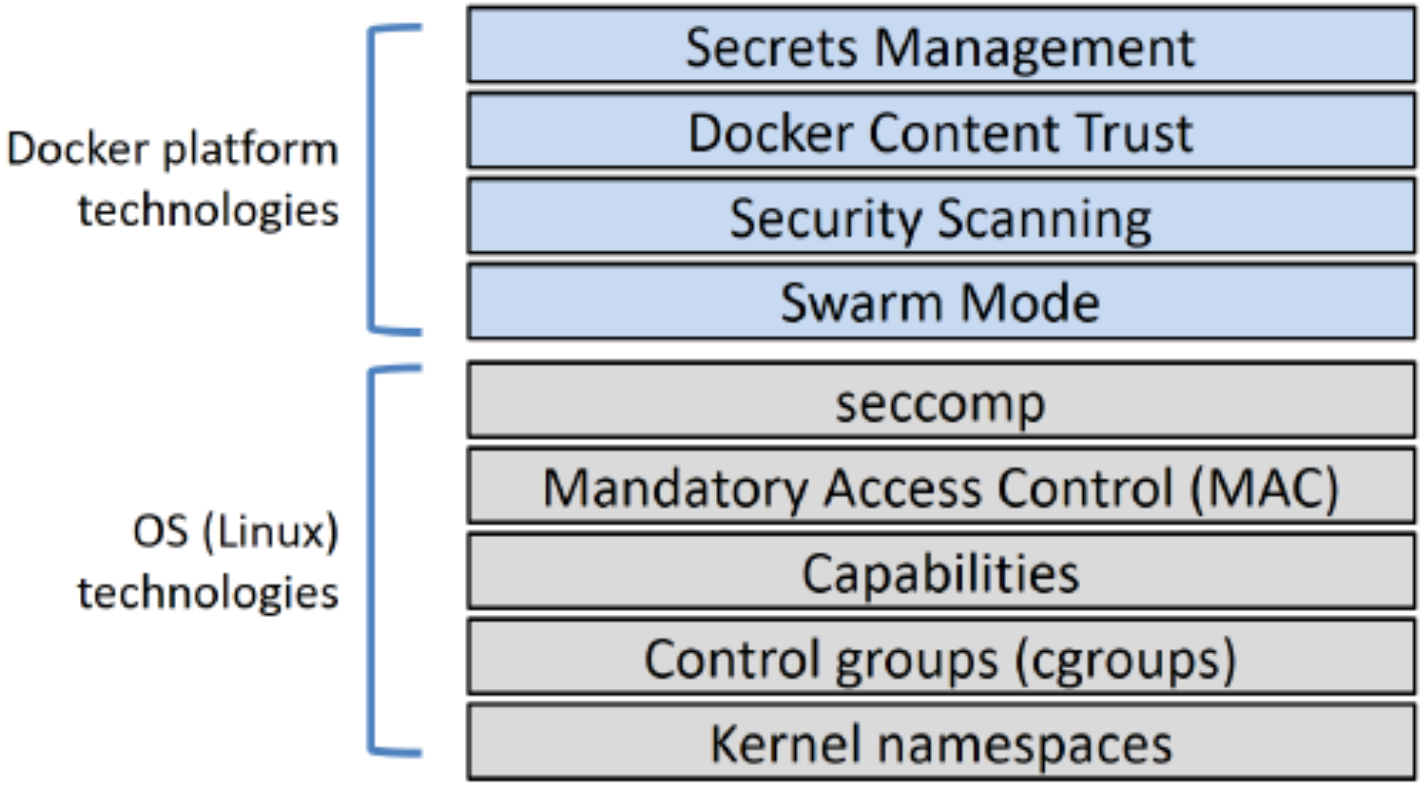
Docker EE Architecture



Container Security

Capabilities, Seccomp, SELinux, Apparmor

Docker Security Model



Capabilities

```
$ cat /proc/$$/task/$$/status
....
CapEff: 0000000000000000
CapPrm: 0000000000000000
CapInh: 0000000000000000
CapAmb: 0000000000000000
CapBnd: 0000001fffffffff
securebits
```

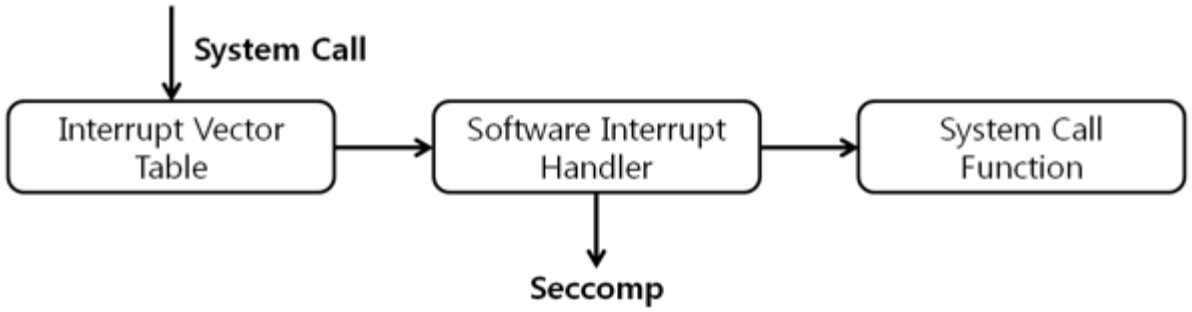
- Traditional UNIX privilege scheme
  - Process running with EUID 0 (superuser): all privilege actions allowed
  - Process running with EUID ≠ 0 : no privilege actions allowed
- Capabilities
  - Since kernel 2.2 (late 90's), 38 capabilities total
  - Collections of distinct privileges that can be enabled per process (thread)
  - CAP_SYS_BOOT, CAP_SYS_ADMIN, CAP_CHOWN, CAP_SYS_TIME, CAP_KILL …
  - capabilities(7), libcap(3)
  - 5 capability sets (effective, permitted, inheritable, ambient, bounding)
  - capset(2), capget(2), prctl
  - systemd: Default, User=…, AmbientCapabilities=X, CapabilityBoundingSet=X

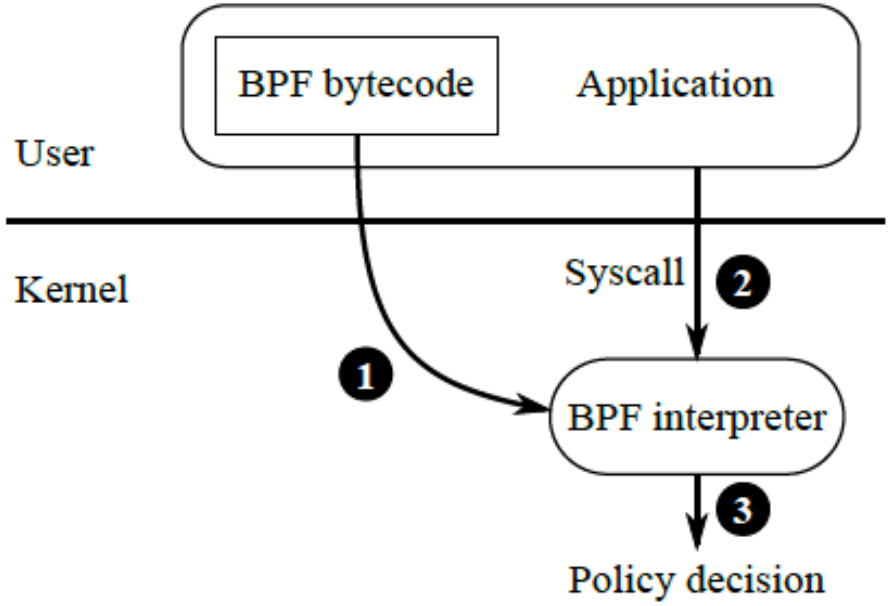
Dockerd Capabilities

- By default, Docker starts containers with a restricted set of capabilities
  - SETPCAP, MKNOD, AUDIT_WRITE, CHOWN, NET_RAW, DAC_OVERRIDE, FOWNER, FSETID, KILL, SETGID, SETUID, NET_BIND_SERVICE, SYS_CHROOT, SETFCAP
  - Removed: NET_ADMIN, SYS_ADMIN, …
- Activated command
  - Runs as native process, by default under EUID 0
  - Runs with selected set of capabilities
- Capability-related arguments
  - Add capabilities: --cap-add *list*
  - Drop capabilities: --cap-drop *list*
  - User: --user name|uid (no capabilities)

Seccomp

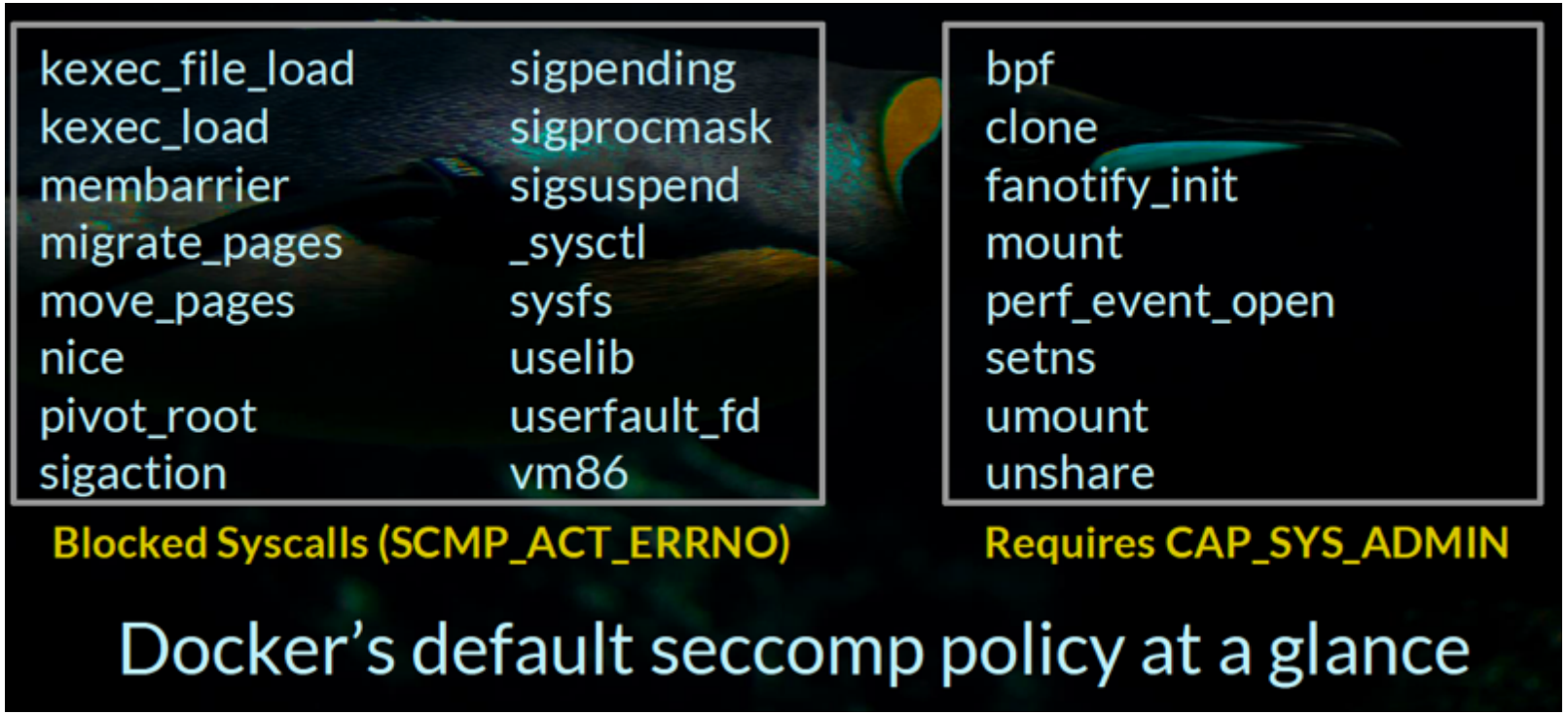
- Since kernel 2.6.12 (2005)
- Seccomp-bpf
  - Allows filtering of system calls using a configurable policy implemented using Berkeley Packet Filter rules
  - For Google Chrome



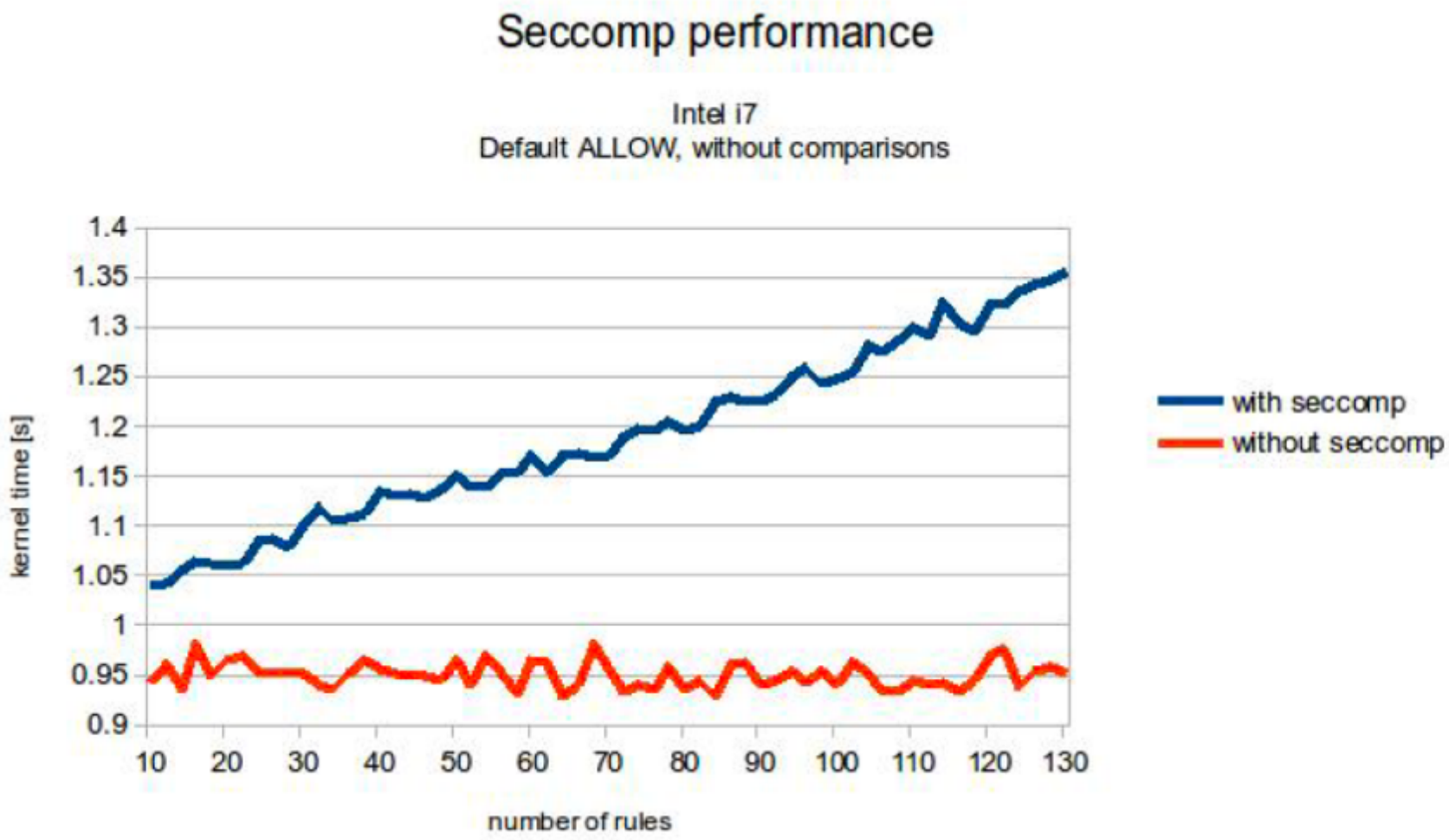


Seccomp for Docker

- Docker command
  - docker run -it --security-opt seccomp=/path/to/seccomp/profile.json …
  - Docker's default seccomp profile: predefined whitelist (44 out of 300 sys calls)



Seccomp Performance

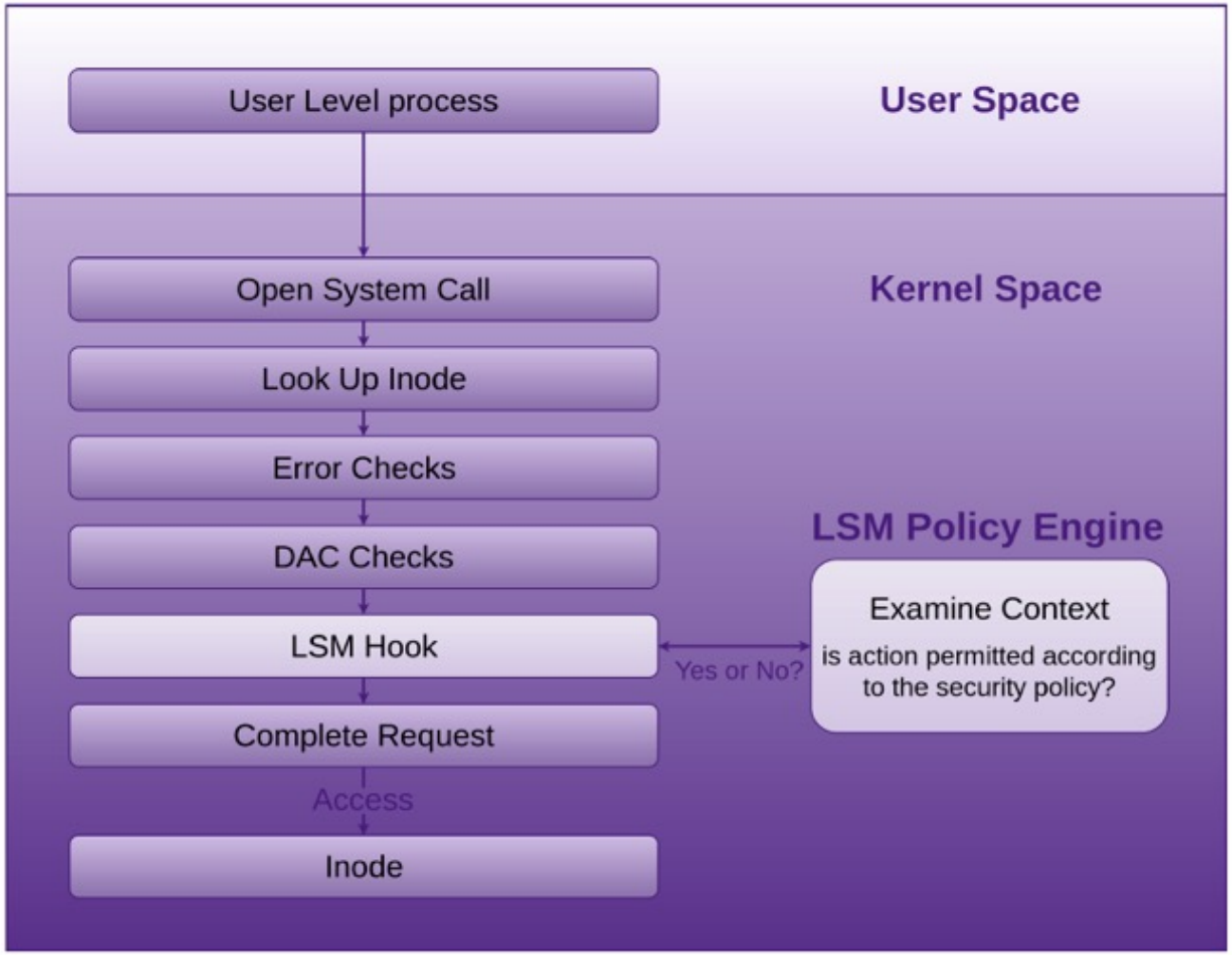


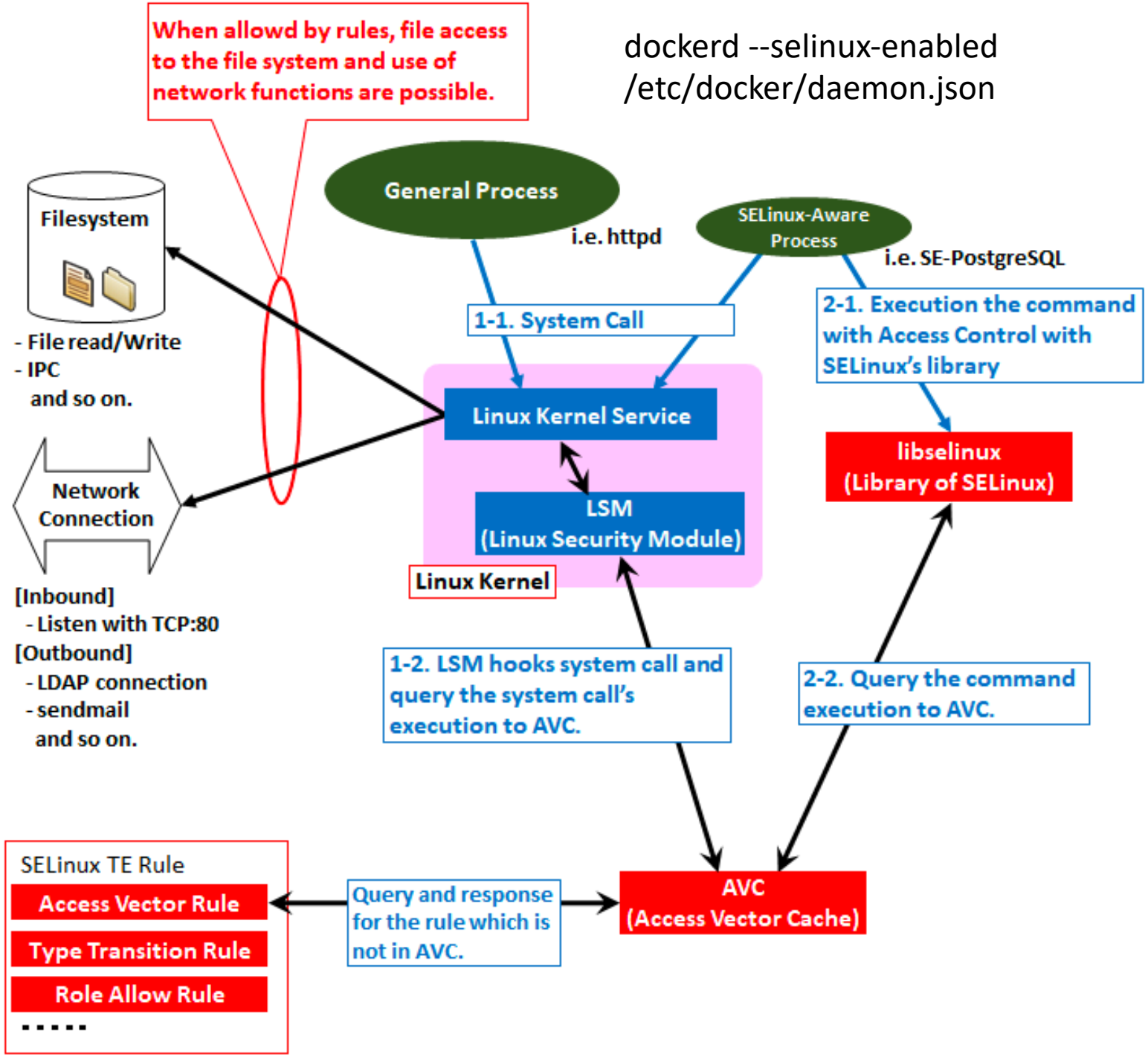
Mandatory Access Control Systems: SELinux

- SELinux (Security Enhanced Linux)
  - sudo setenforce 1 → sudo sestatus
  - Set security labels, 3 modes (enforce, permissive, disable)
  - Even if Apache HTTP Server is compromised, an attacker cannot use that process to read files in user home directories by default, unless a specific SELinux policy rule was added or configured to allow such access
- Protecting host file system from container breakout

Linux Security Modules (LSM)

- Linux Security Modules (LSM)
  - Kernel API for access control
  - Hooks: Like Netfilter for whole kernel
  - Pluggable: Smack, SELinux, AppArmor, YAMA, TOMOYO, LoadPin
  - Shared by all containers: security namespace [USENIX SEC'18]



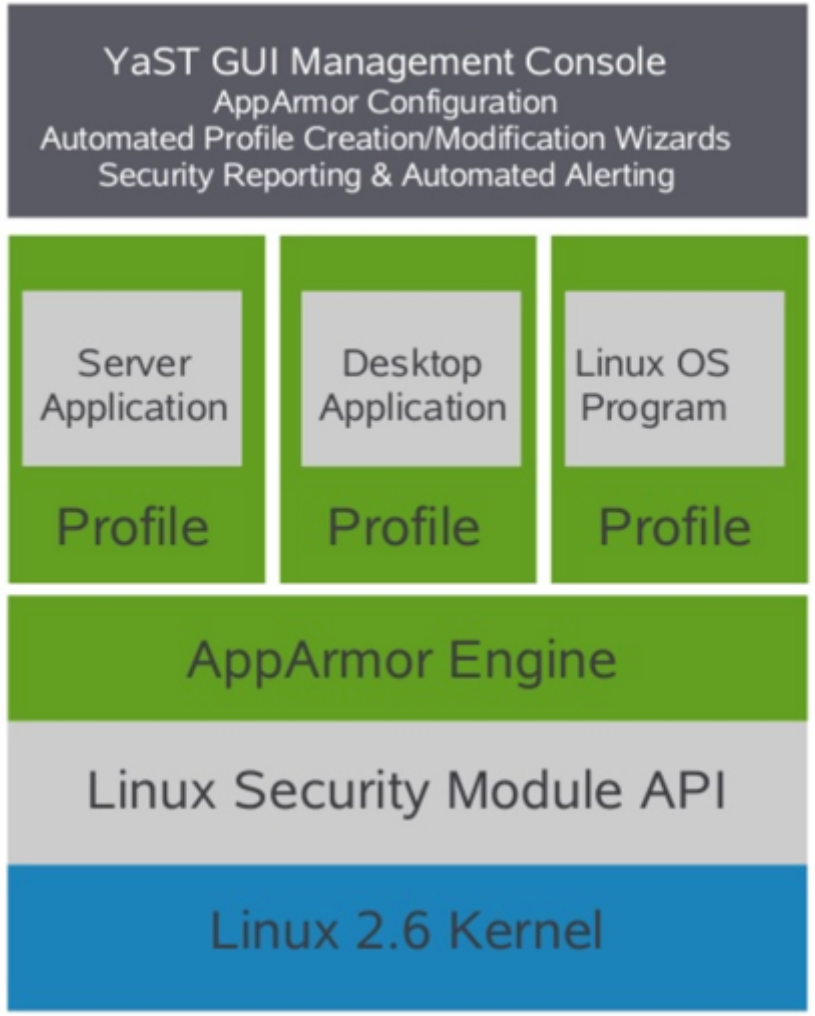

When allowd by rules, file access to the file system and use of network functions are possible.
dockerd --selinux-enabled
/etc/docker/daemon.json
General Process
i.e. httpd
SELinux-Aware Process
i.e. SE-PostgreSQL
Filesystem
- File read/Write
- IPC
and so on.
1-1. System Call
2-1. Execution the command with Access Control with SELinux's library
Linux Kernel Service
libselinux (Library of SELinux)
Network Connection
LSM (Linux Security Module)
Linux Kernel
[Inbound]
- Listen with TCP:80
[Outbound]
- LDAP connection
- sendmail
and so on.
1-2. LSM hooks system call and query the system call's execution to AVC.
2-2. Query the command execution to AVC.
SELinux TE Rule
Access Vector Rule
Type Transition Rule
Role Allow Rule
.....
Query and response for the rule which is not in AVC.
AVC (Access Vector Cache)

SELinux Namespace

- Prototype
  - Developed by Stephen Smalley
  - https://github.com/stephensmalley/selinux-kernel/tree/selinuxns
  - Defines: struct selinux_ns, init_selinux_ns
    - Encapsulates global SELinux state
    - Passed to internal APIs (“security server”)
    - Initial / global namespace
- Current
  - https://git.kernel.org/pub/scm/linux/kernel/git/pcmoore/selinux.git
  - https://github.com/SELinuxProject/selinux-kernel

Mandatory Access Control Systems: AppArmor

- AppArmor
  - Restrict programs' capabilities with per-program profiles
  - Profiles: /etc/apparmor.d
  - sudo apparmor_parser /etc/apparmor.d/… → sudo aa-status
  - 2 modes (enforcement, complain)



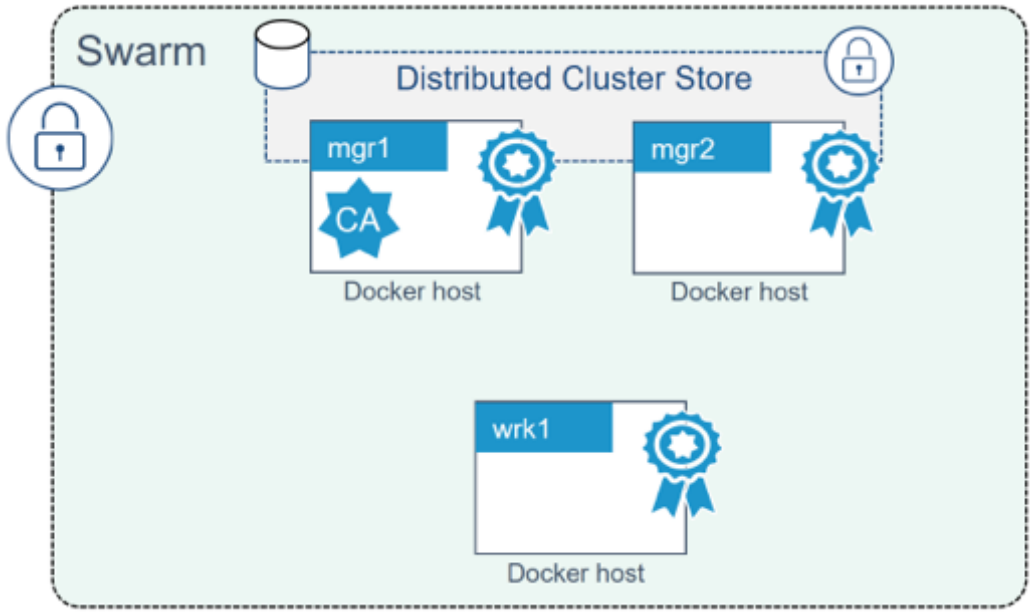
AppArmor for Docker

- AppArmor
  - Docker automatically generates and loads a default profile for containers named docker-default (tmpfs) and load it into the kernel
  - https://github.com/moby/moby/tree/master/contrib/apparmor
- Command
  - $ docker run --rm -it --security-opt apparmor=docker-default hello-world
- Create profile
  - Profile language: https://gitlab.com/apparmor/apparmor/-/wikis/QuickProfileLanguage
  - $ apparmor_parser -r -W /path/to/your_profile

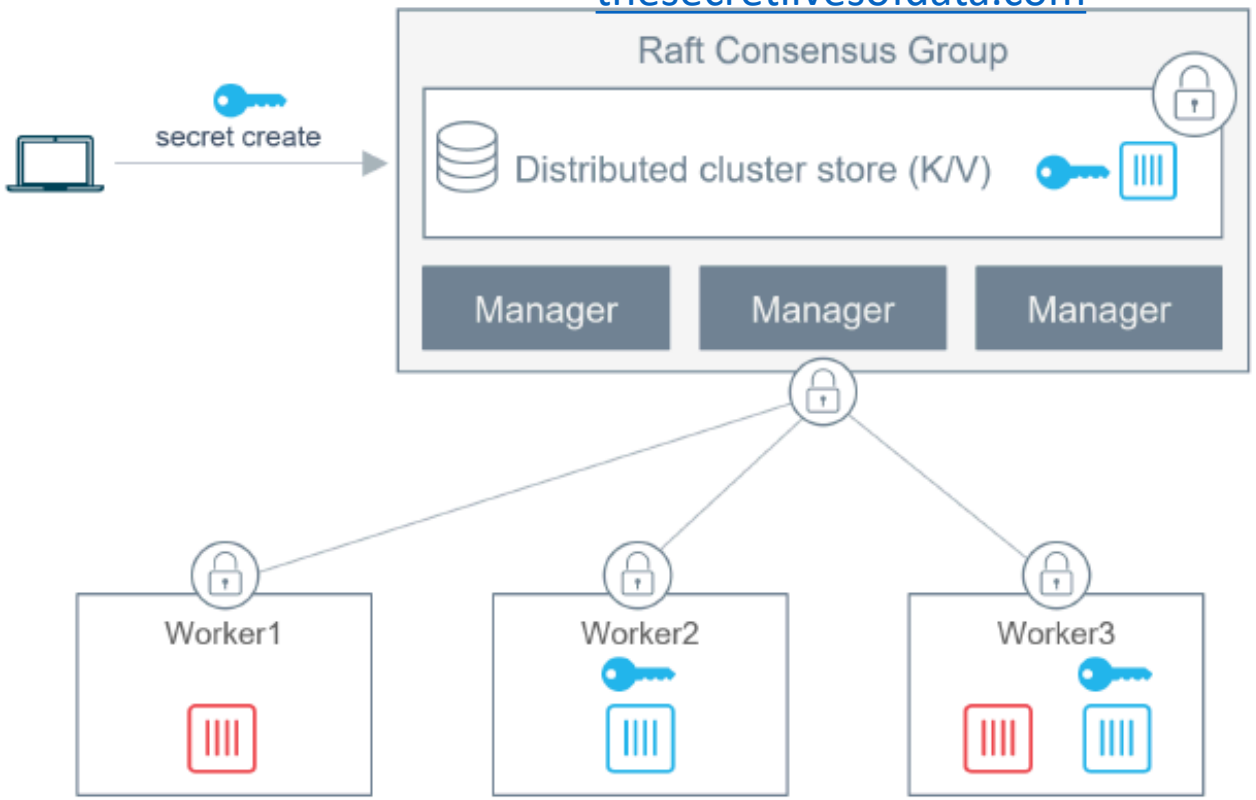
Docker Platform Security

- Docker Swarm
  - Default: cryptographic node IDs, mutual authentication, automatic CA configuration, automatic certificate rotation, encrypted cluster store, encrypted networks, and more
- Docker Content Trust (DCT)
  - Sign images and verify the integrity and publisher
- Docker Security Scanning
  - Analyses images, detects known vulnerabilities, and provides detailed reports
- Docker Secrets
  - Stored in the encrypted cluster store, encrypted in-flight when delivered to containers, stored in in-memory filesystems when in use (least privilege model)

Docker Swarm Security



thesecretlivesofdata.com



Container Security Mechanisms by Default

Available Container Security Features, Requirements and Defaults			
Security Feature	**LXC 2.0**	**Docker 1.11**	**CoreOS Rkt 1.3**
User Namespaces	Default	Optional	Experimental
Root Capability Dropping	Weak Defaults	Strong Defaults	Weak Defaults
Procfs and Sysfs Limits	Default	Default	Weak Defaults
Cgroup Defaults	Default	Default	Weak Defaults
Seccomp Filtering	Weak Defaults	Strong Defaults	Optional
Custom Seccomp Filters	Optional	Optional	Optional
Bridge Networking	Default	Default	Default
Hypervisor Isolation	Coming Soon	Coming Soon	Optional
MAC: AppArmor	Strong Defaults	Strong Defaults	Not Possible
MAC: SELinux	Optional	Optional	Optional
No New Privileges	Not Possible	Optional	Not Possible
Container Image Signing	Default	Strong Defaults	Default
Root Interation Optional	True	False	Mostly False

Container Security

Vulnerabilities

Namespace Vulnerabilities

- Process
  - CVE-2009-1338: Kernel did not consider PID namespaces when executing KILL system call, allowing attacker to send arbitrary signals to every process on the system
- Filesystem
  - CVE-2016-2853: aufs incorrectly handled mount namespaces leading to a possible privilege escalation
  - CVE-2015-1328, CVE-2016-1576: similar to CVE-2016-2583 in overlayfs
  - CVE-2013-1957, CVE-2013-1959, CVE-2014-5206, CVE-2014-5207, CVE-2014-7970, CVE-2014-9717, CVE-2015-4176, CVE-2015-4177, and CVE-2015-4178
- Etc
  - CVE-2013-1956, CVE-2013-1858, CVE-2014-4014, CVE-2014-8989, CVE-2015-2925, CVE-2016-4997, CVE-2016-4998, …

Capability Vulnerabilities

- CAP_NET_ADMIN
  - CVE-2010-4655: Sensitive heap memory disclosure
  - CVE-2011-1019: Granted the CAP_SYS_MODULE capability to load arbitrary modules and was exploited trivially using ifconfig
  - CVE-2013-4514: Denial of Service, possibly arbitrary code execution
- Capabilities related
  - CVE-2014-7975: CAP_SYS_ADMIN
  - CVE-2013-4588: CAP_NET_ADMIN
  - CVE-2013-6383: CAP_SYS_RAWIO
  - CVE-2011-2517: CAP_NET_ADMIN
  - CVE-2011-1019: CAP_NET_ADMIN, CAP_SYS_MODULE
  - …

Docker CVEs

CVE ID	Description	Date	Patch
CVE-2016-8867	Incorrect application of ambient capabilities	Oct 27, 2016	Engine 1.12.3
CVE-2014-8178	Attacker controlled layer IDs lead to local graph content poisoning	Oct 12, 2015	Engine 1.8.3, 1.6.2-CS7
CVE-2014-8179	Manifest validation and parsing logic errors allow pull-by-digest validation bypass	Oct 12, 2015	Engine 1.8.3, 1.6.2-CS7
CVE-2015-3629	Symlink traversal on container respawn allows local privilege escalation	May 7, 2015	Engine 1.6.1
CVE-2015-3627	Insecure opening of file-descriptor 1 leading to privilege escalation	May 7, 2015	Engine 1.6.1
CVE-2015-3630	Read/write proc paths allow host modification & information disclosure	May 7, 2015	Engine 1.6.1
CVE-2015-3631	Volume mounts allow LSM profile escalation	May 7, 2015	Engine 1.6.1

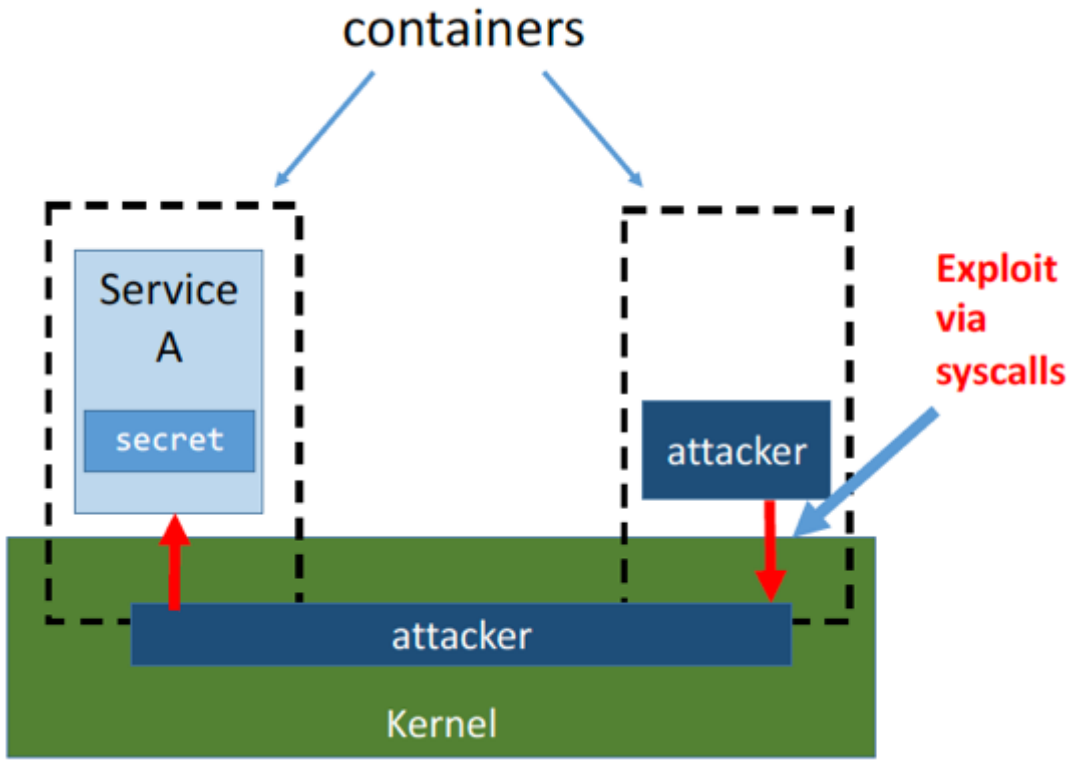
https://www.docker.com/legal/docker-cve-database

Conatainers are Isolated?

Containers == namespaced processes → Kernel exploits mostly work

- Sep 2018: CVE-2018-14634
- DirtyCOW (CVE-2016-5195)
- Many more (CVE database), 2018: Codexec (3), Mem. Corrupt (8)

Horizontal attack possible via shared privileged component (kernel)



Dirty CoW

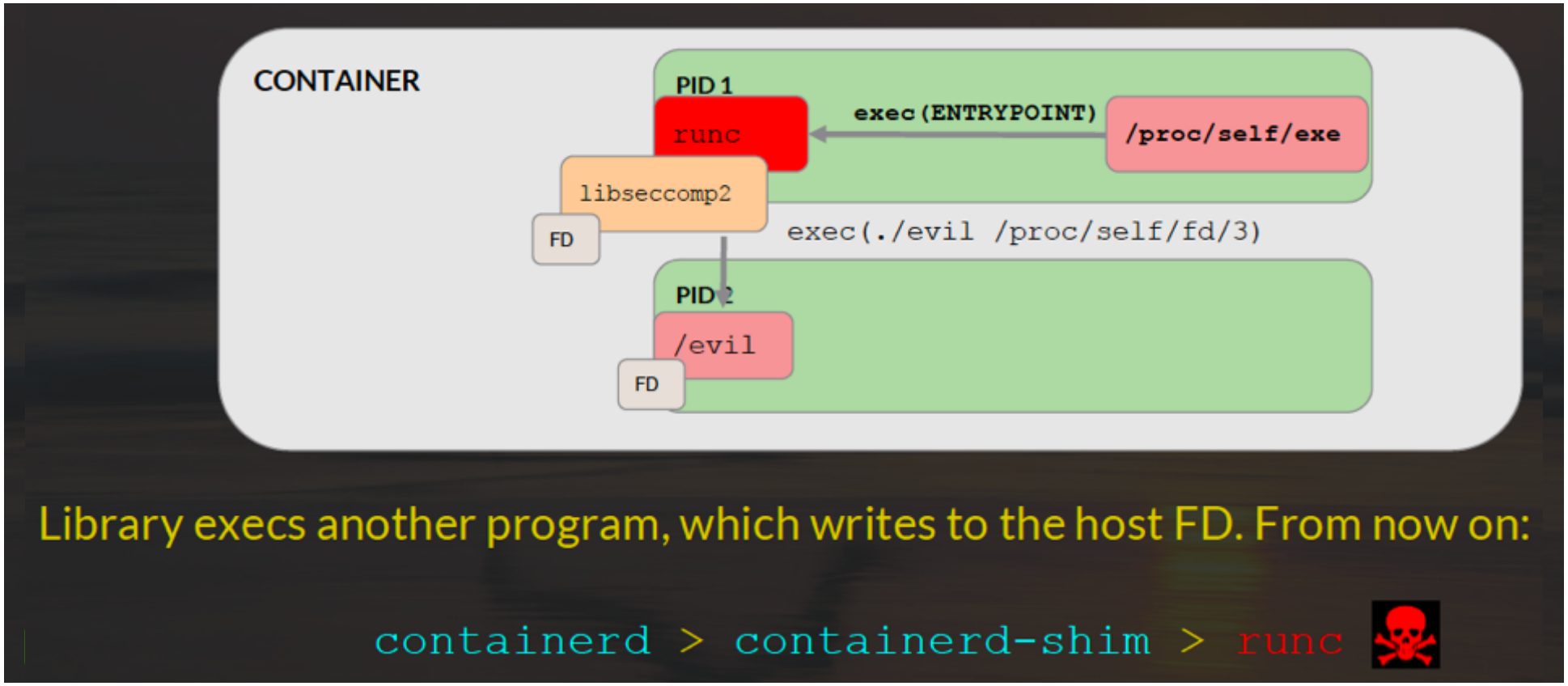
DityCow Exploit Sketch:

- **mmap** a page
- Create a thread that invokes **madvise**
- Create a thread that invokes **Read/Write procfs**

Triggers race condition in Kernel Mem. management code

```
// FROM: https://dirtycow.ninja/


map=mmap(NULL,st.st_size,PROT_READ,MAP_PRIVATE
,f,0); printf("mmap %zx\n\n",(uintptr_t) map);


/* You have to do it on two threads. */
pthread_create(&pth1,NULL,madviseThread,argv[1
]);  //madvise
pthread_create(&pth2,NULL,procselfmemThread,ar
gv[2]);
// R/W procfs


/* You have to wait for the threads to finish.
*/ pthread_join(pth1,NULL);
pthread_join(pth2,NULL); return 0;
```

The RunC Escape (CVE-2019-5736)



Recent Container Trend

Rootless Docker

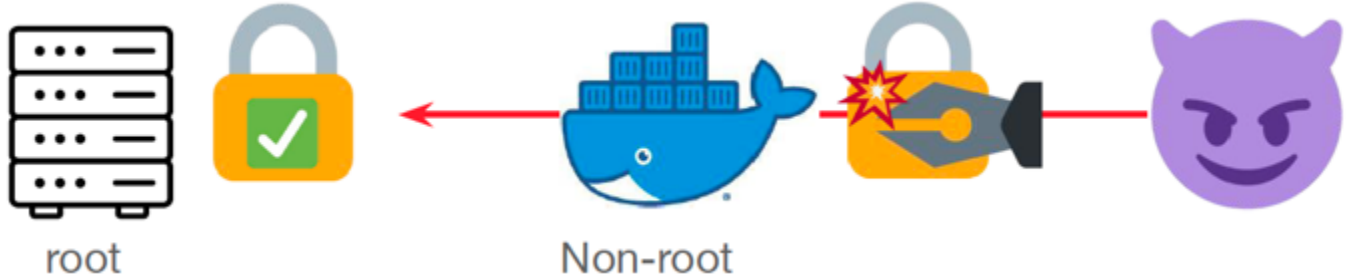
Rootless Docker?

```
$ sudo docker
$ usermod -aG docker penguin
$ docker run --user 42
$ dockerd --userns-remap
```

All of them run the daemon as the root!

Rootless Docker!

- Running the Docker daemon as a non-root user
  - $ curl -fsSL https://get.docker.com/rootless | sh
- Limitations
  - No OverlayFS (except on Ubuntu)
  - Limited network performance by default
  - TCP/UDP port numbers below 1024 can’t be listened on
  - No cgroup
    - docker run --memory and --cpu-* flags are ignored
    - docker top: does not work



For Rootless Docker

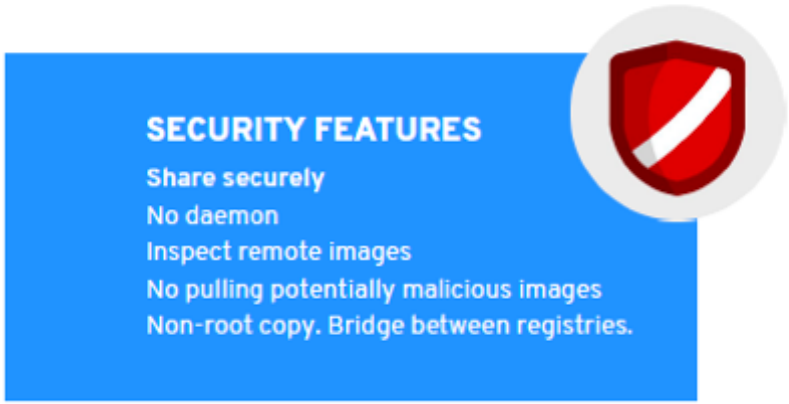
- If OverlayFS is not available, use XFS to deduplicate files
- The default network stack (VPNKit) is slow
  - Install slirp4netns (v0.3.0+) to get better throughput: 514Mbps → 9.21 Gbps
  - still slow compared to native vEth 52.1 Gbps
  - install lxc-user-nic to get native performance
- Exposing port numbers below 1024 requires CAP_NET_BIND_SERVICE

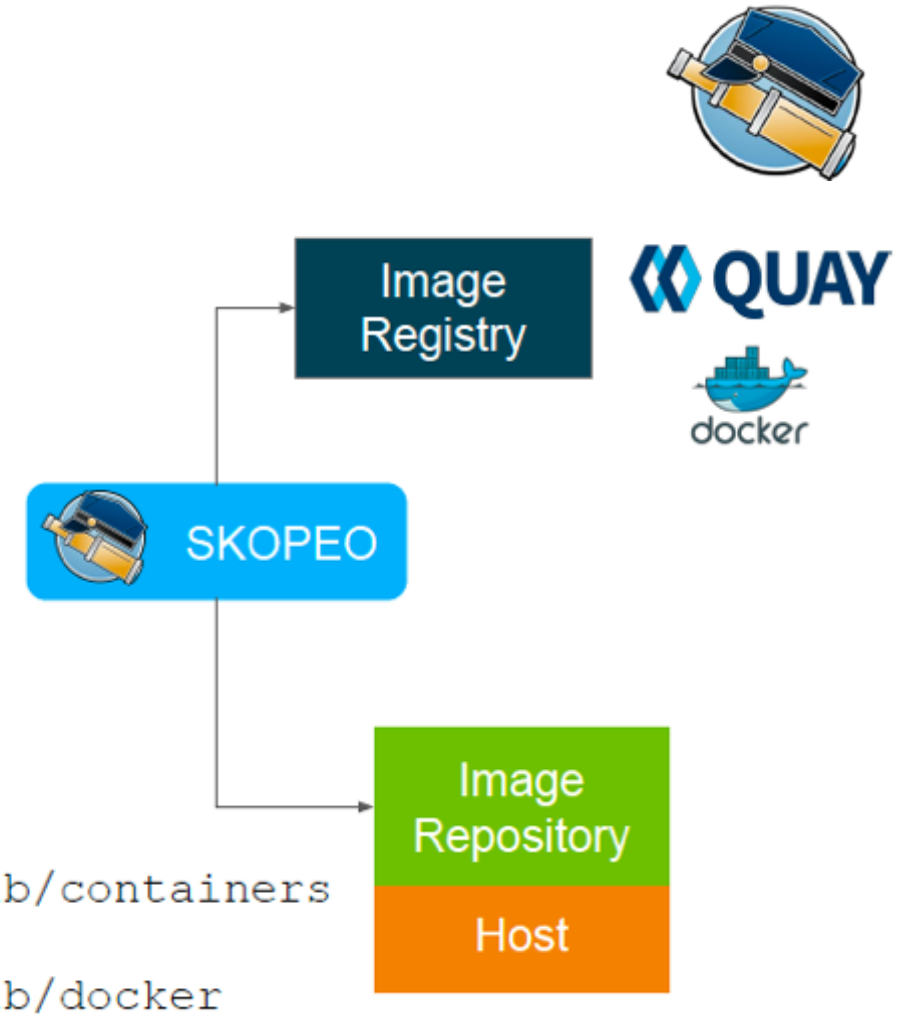
Recent Container Trend

Daemonless, Dockerless

Skopeo

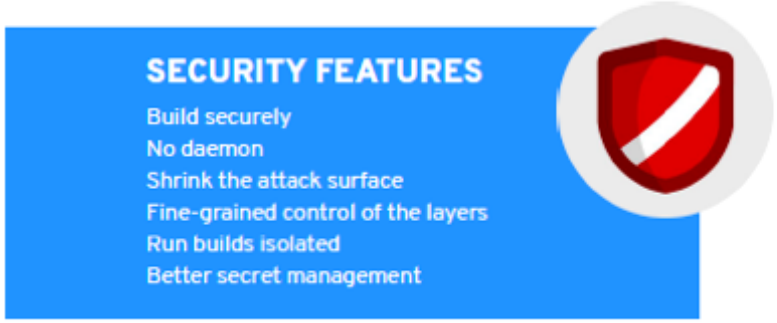
Built for interfacing with Docker registry
CLI for images and image registries
Rejected by upstream Docker ¯_(ツ)_/¯
Allows remote inspection of image metadata - no downloading
Can copy from one storage to another

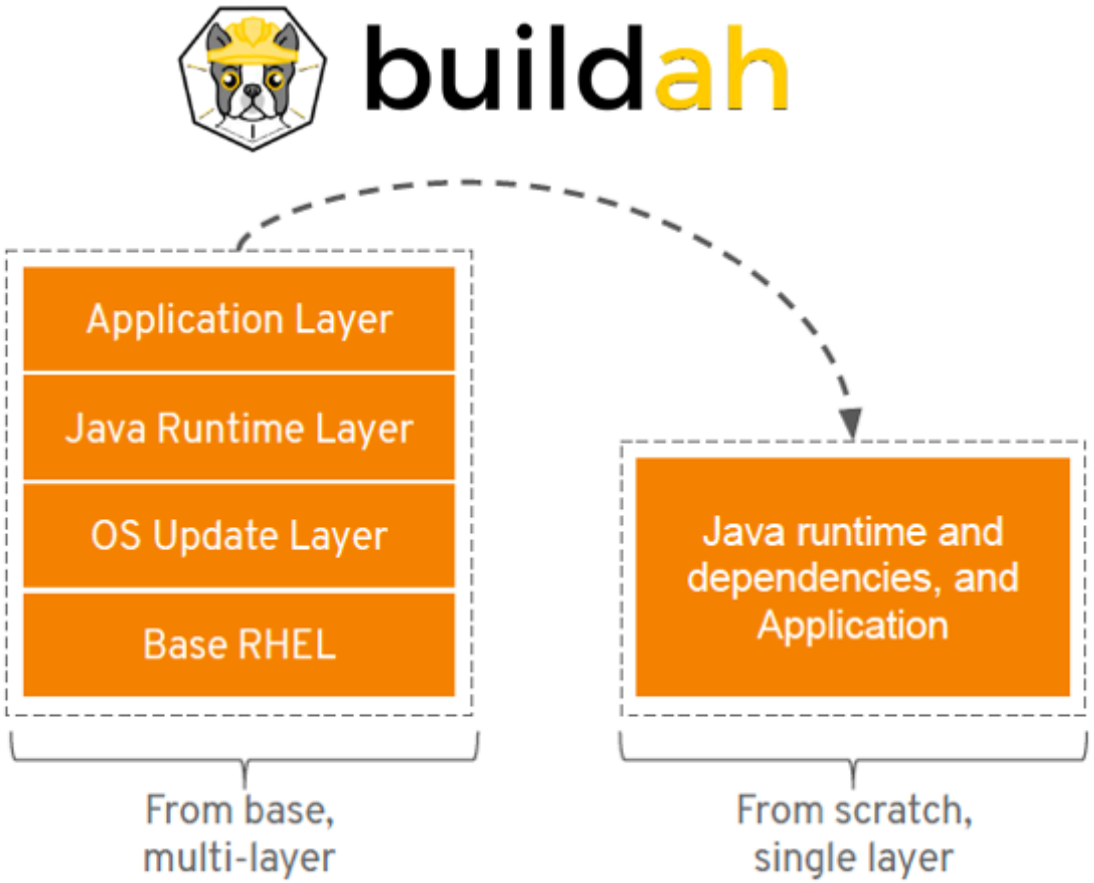




Buildah

Now buildah.io
Builds OCI compliant images
No daemon - no “docker socket”
Does not require a running container
Can use the host’s user’s secrets.
Single layer, from scratch images are made easy and it ensures limited manifest.
If needed you can still maintain Dockerfile based workflow





What Do You Need to Run a Container?

Standard Definition of what makes up a container image.

- OCI Image Bundle Definition

Mechanism to pull images from a container registry to the host

- github.com/containers/image

Ability to explode images onto COW file systems on disk

- github.com/containers/storage

Standard mechanism for running a container

- OCI Runtime Spec (1.0)
- runc default implementation of OCI Runtime Spec (Same tool Docker uses to run containers)

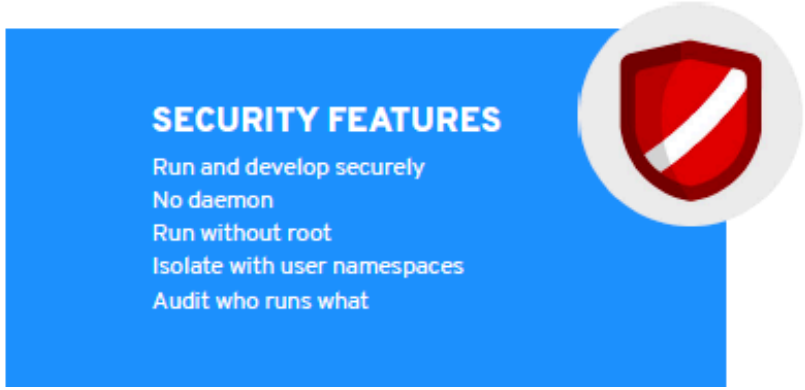
Standard Way to setup networking for containers

- Container Networking Interface

Podman

@ podman.io
Client only tool, based on the Docker CLI. (same+)
No daemon!
Storage for
- Images - `containers/ima`
- Containers - `containers/s`

Runtime - `runc`
Shares state with CRI-O and with

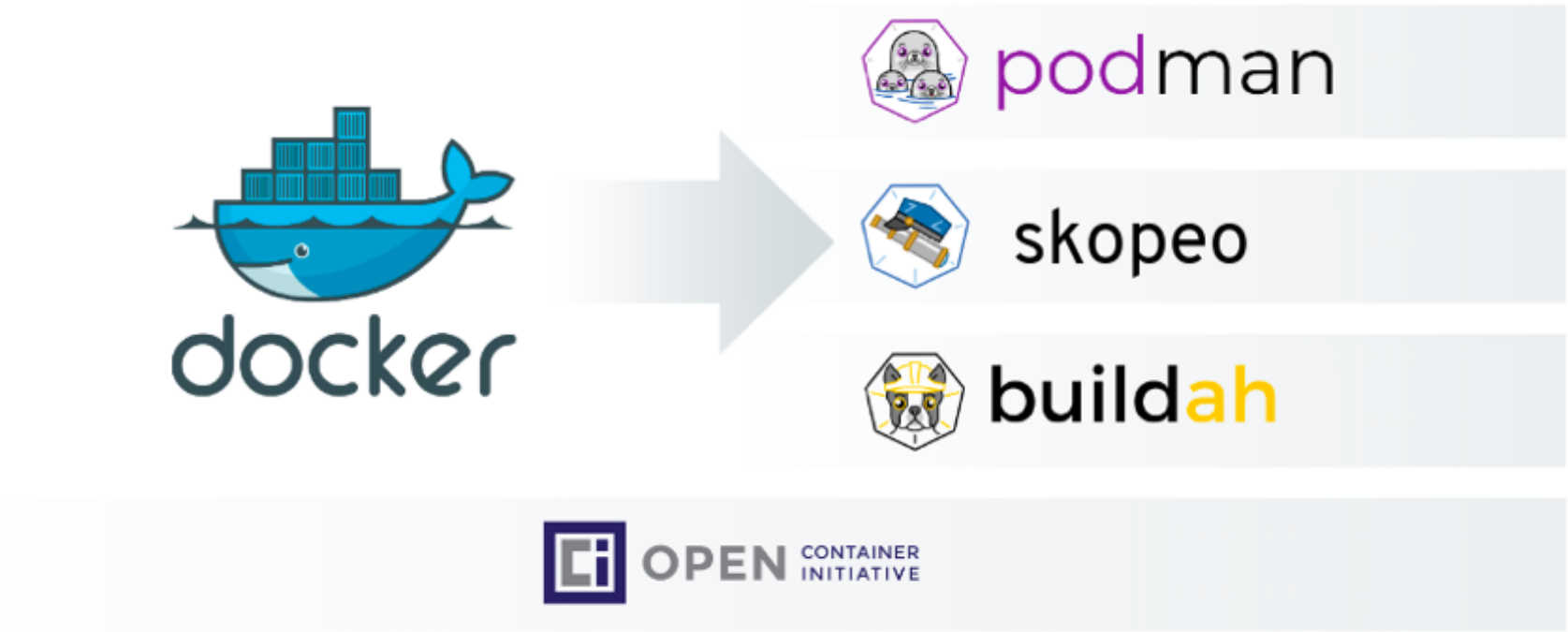


	User Outside - Root Inside

Docker	$ whoami
	fatherlinux
	
	$ docker run --privileged -itv /:/host ubi8 touch /host/etc/hacker
	
	$ ls -alh /etc/hacker
	-rw-r--r--. 1 root root 0 Oct 23 17:02 /etc/hacker

Podman	$ whoami
	fatherlinux
	
	$ podman run --privileged -itv /:/host ubi8 touch /host/etc/hacker
	touch: cannot touch '/host/etc/hacker': Permission denied

There No Docker in OpenShift 4 and RHEL 8



Recent Container Trend

Runtimes

CLOUD NATIVE COMPUTING FOUNDATION

KubeCon CloudNativeCon

Sandbox

+20

Incubating

- OpenTracing — Distributed Tracing API
- gRPC — Remote Procedure Call
- CNI — Networking API
- TUF — Software Update Spec
- Notary — Security
- NATS — Messaging
- LINKERD — Service Mesh
- HELM — Package Management
- ROOK — Storage
- HARBOR — Registry
- etcd — Key/Value Store
- Open Policy Agent — Policy
- cri-o — Container Runtime
- KV — Key/Value Store
- cloudevents — Serverless

Graduated

- kubernetes — Orchestration
- Prometheus — Monitoring
- JAEGER — Distributed Tracing
- CoreDNS — Service Discovery
- envoy — Service Proxy
- Vitess — Storage
- containerd — Container Runtime
- fluentd — Logging


Alena Prokharchyk
Apple


Brendan Burns
Microsoft


Justin Cormack
Docker


Saad Ali
Google


Matt Klein
Lyft


Katie Gamanji
American Express


Liz Rice
Aqua Security (TOC Chair)

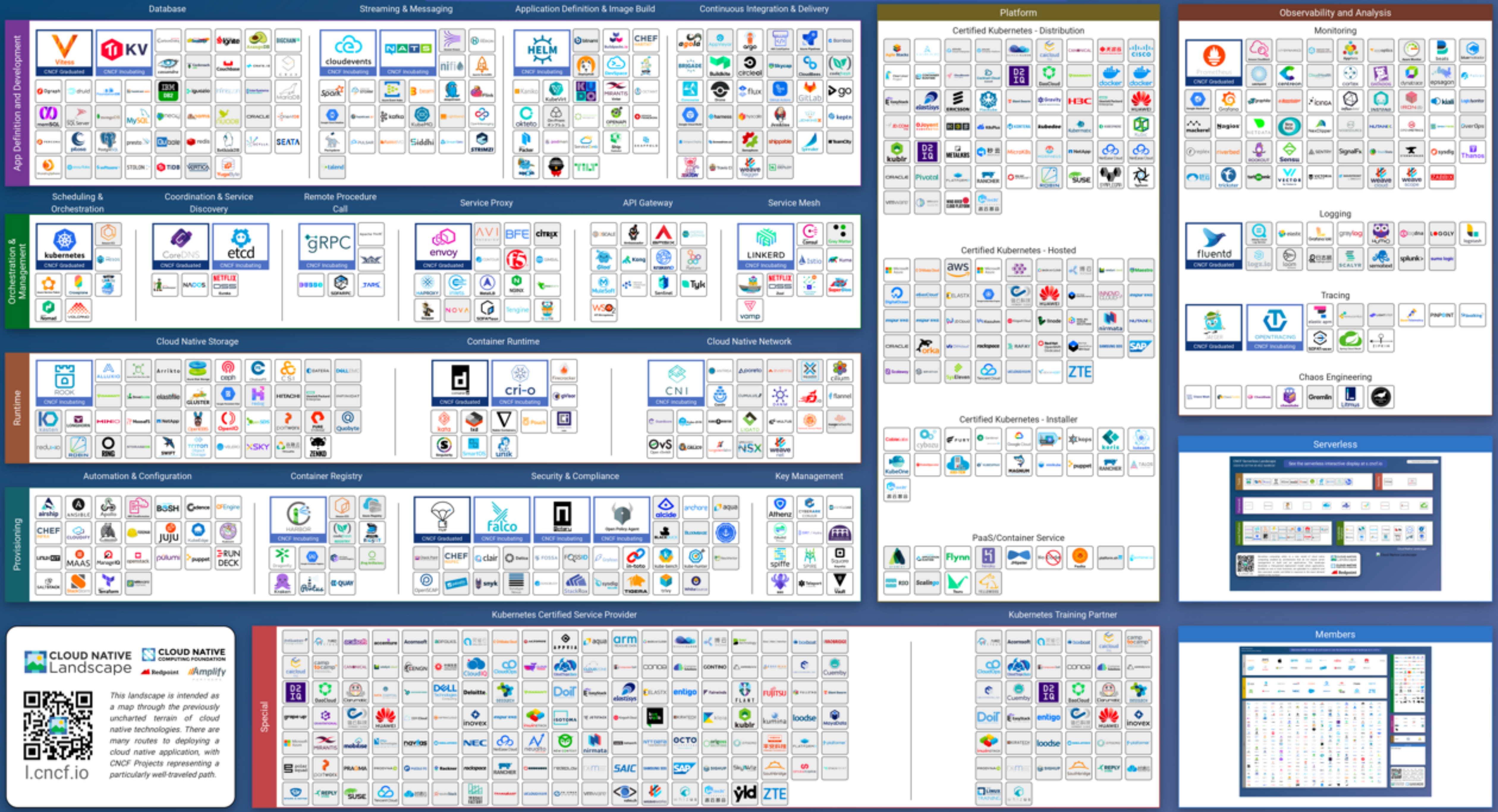

Xiang Li
Alibaba


Jeff Brewer
Intuit

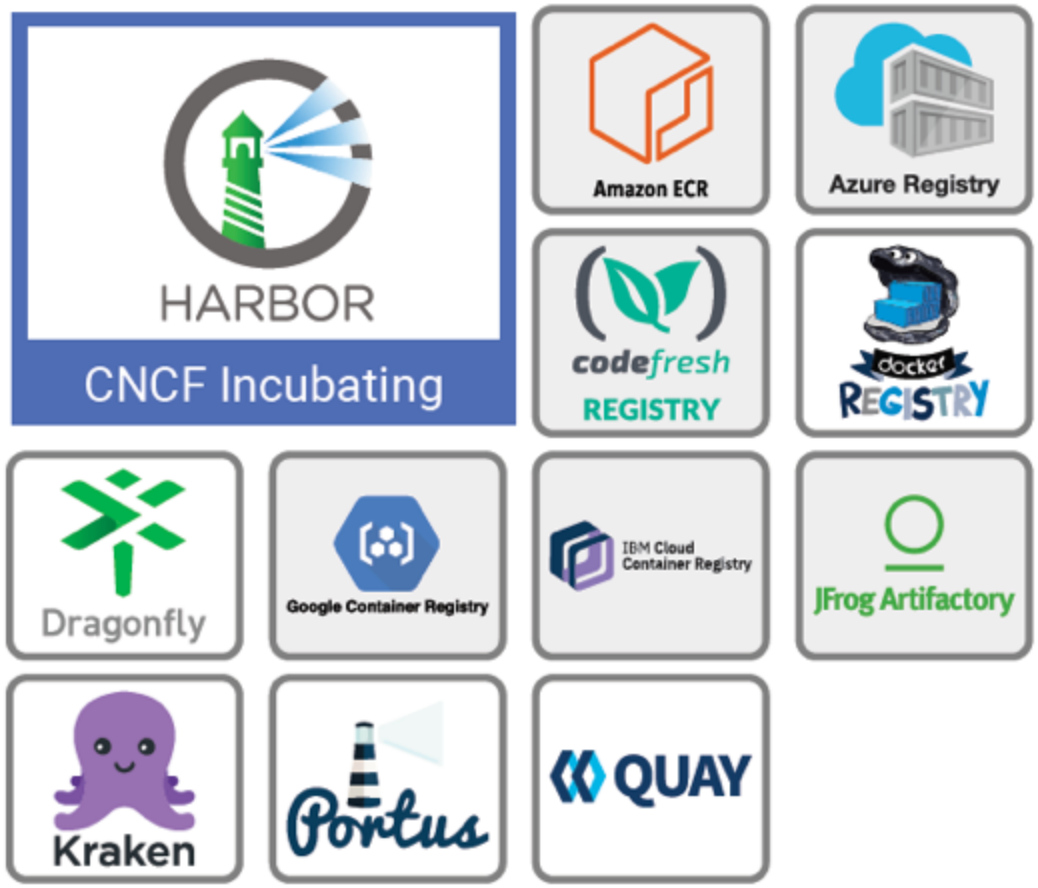

Sheng Liang
Rancher

CNCF Cloud Native Landscape
2020-02-20T04:30:45Z 3a6802d

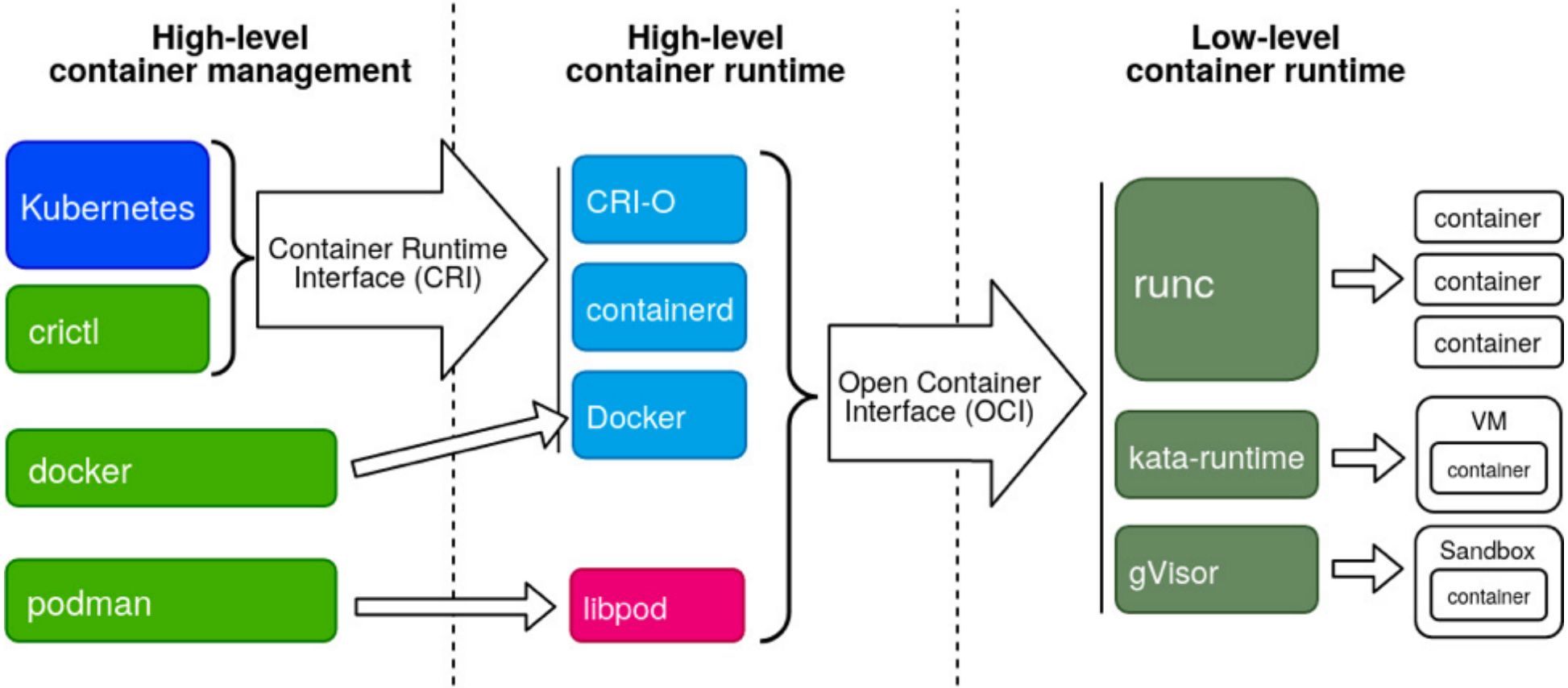
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io



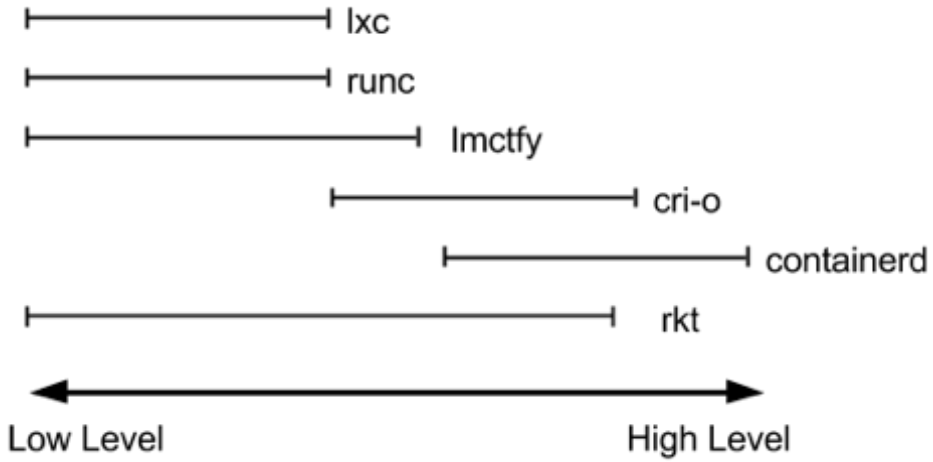
Container Registry

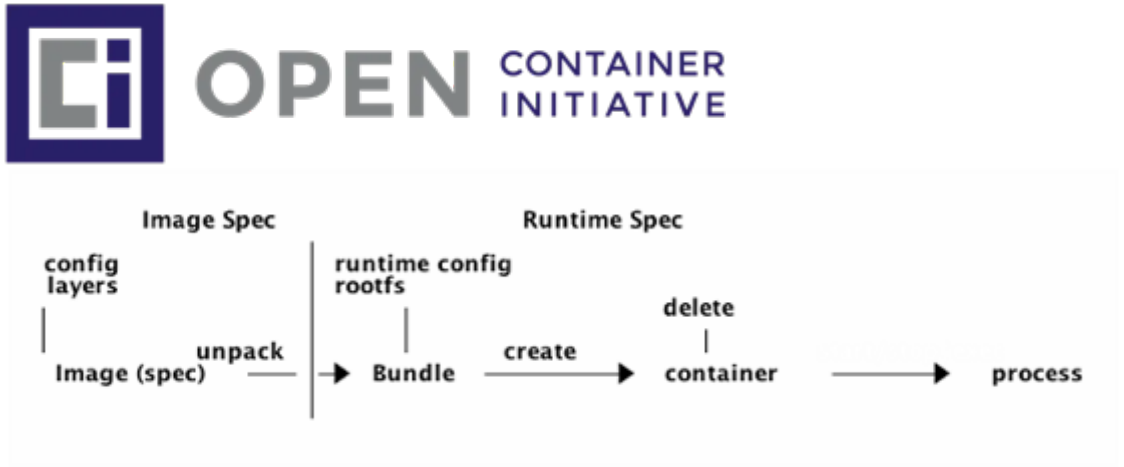


Container Runtimes & Interfaces



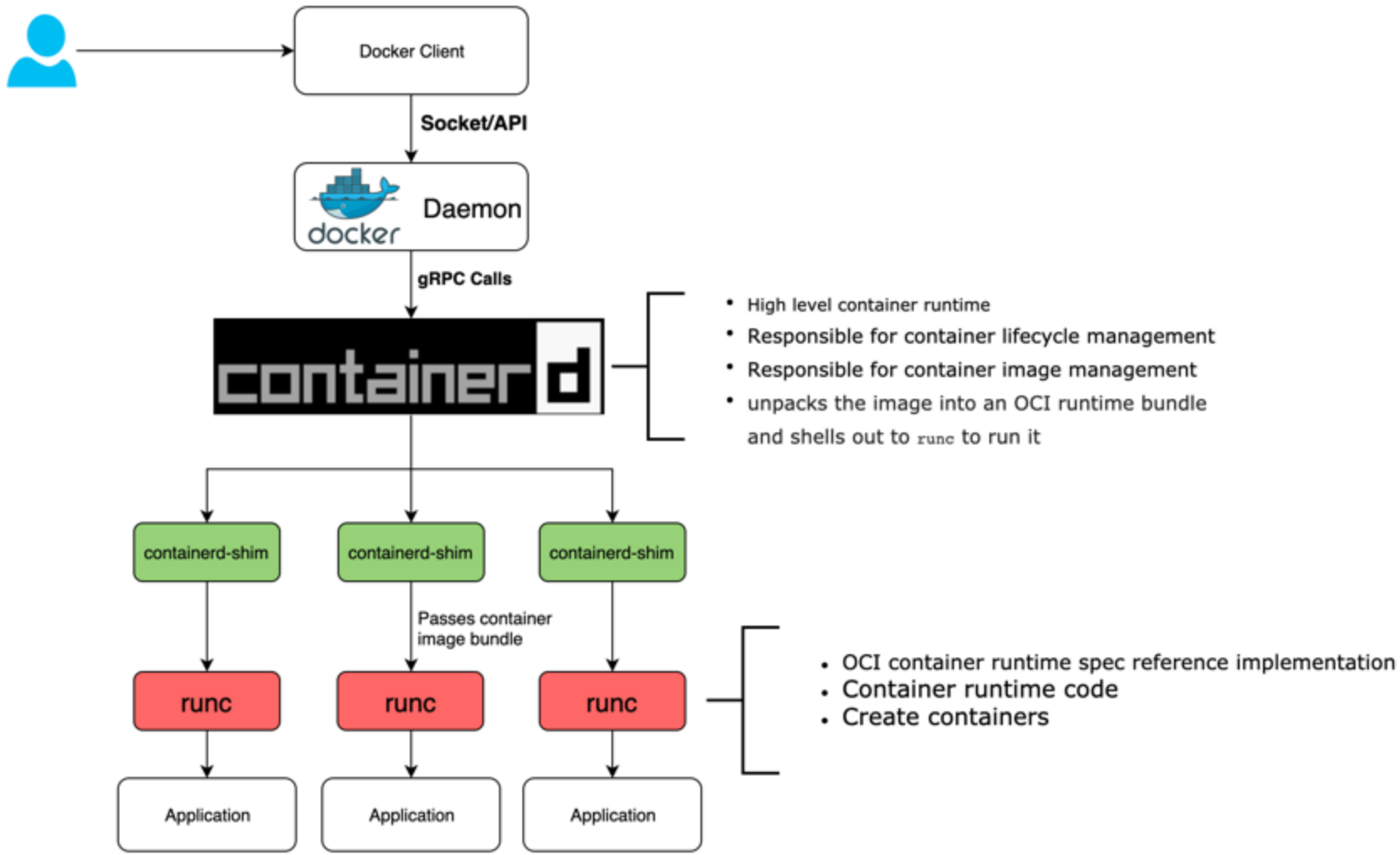
Runtime Level & OCI





low-level container runtimes: focus on just running containers
high-level container runtimes: support more high-level features, like image management and gRPC/Web APIs

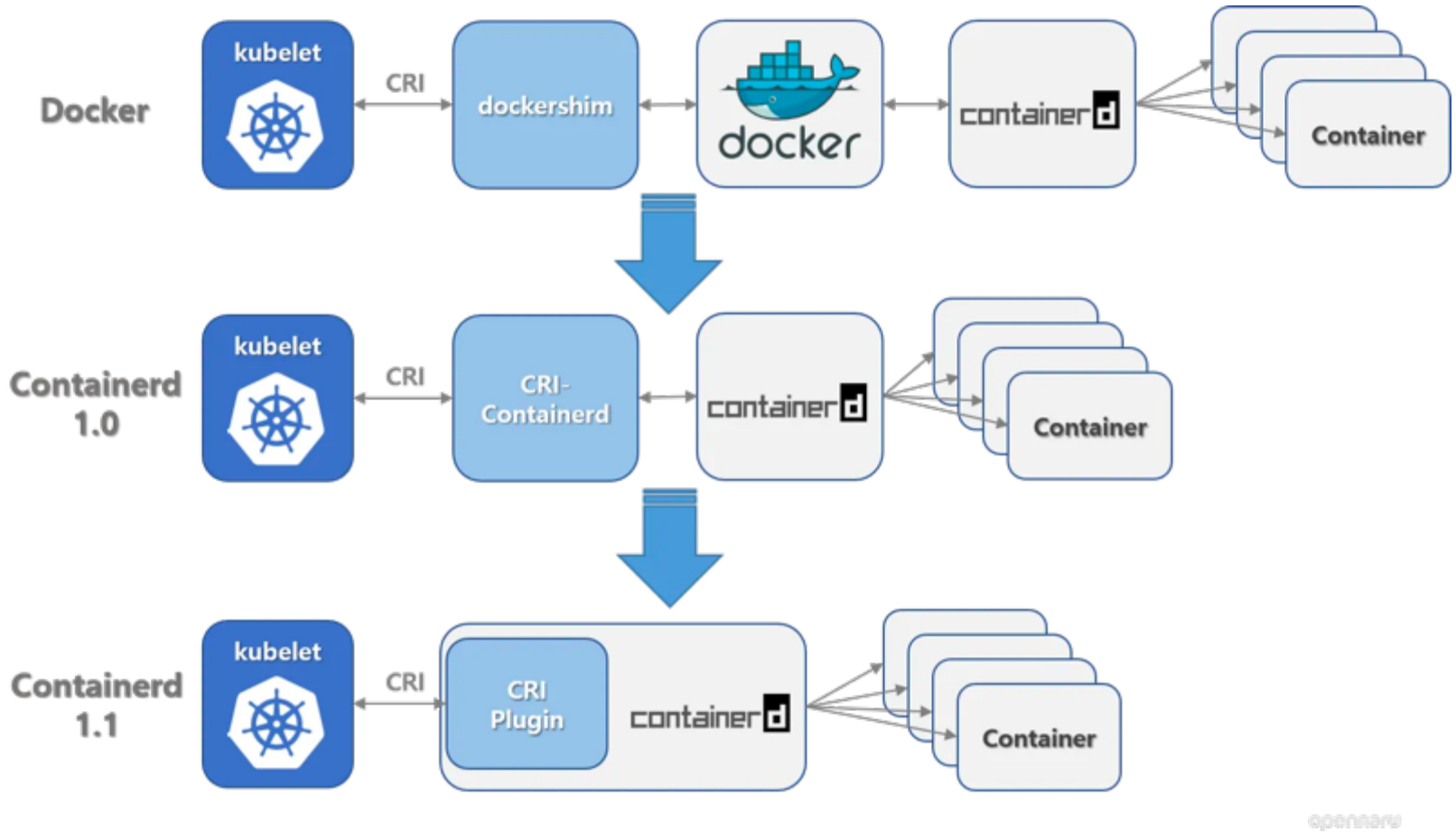
Docker Runtimes



Low-Level Runtime

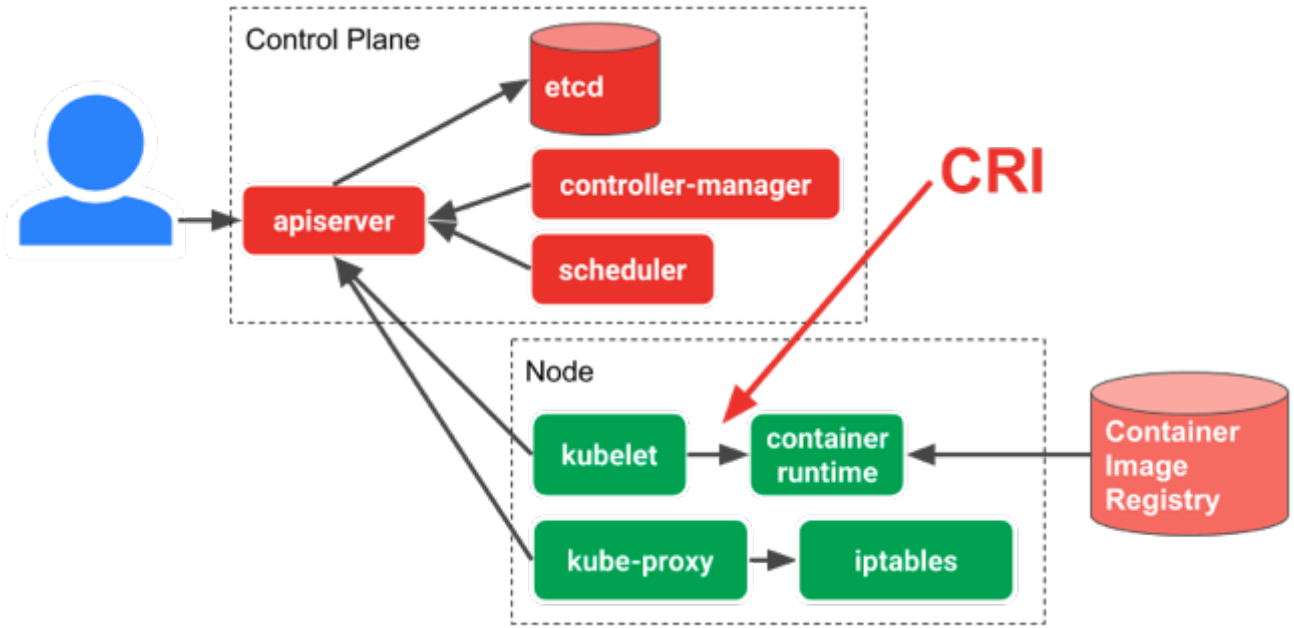
- Simple example of a container runtime (https://github.com/ianlewis/exec)
  - $ CID=**$(**docker create busybox**)**
  - $ ROOTFS=**$(**mktemp -d**)**
  - $ docker export $CID | tar -xf - -C $ROOTFS
  - $ UUID=**$(**uuidgen**)**
  - $ cgcreate -g cpu,memory:$UUID
  - $ cgset -r memory.limit_in_bytes=100000000 $UUID
  - $ cgset -r cpu.shares=512 $UUID
  - $ cgexec -g cpu,memory:$UUID ****
  - > unshare -uinpUrf --mount-proc ****
  - > sh -c "/bin/hostname $UUID && chroot $ROOTFS /bin/sh"
  - *# echo "Hello from in a container"* Hello from in a container
  - *# exit*

Docker & CRI


Docker
kubelet
CRI
dockershim
docker
containerd
Container
Containerd 1.0
kubelet
CRI
CRI-Containerd
containerd
Container
Containerd 1.1
kubelet
CRI
CRI Plugin
containerd
Container

Container Runtime Interface (CRI)

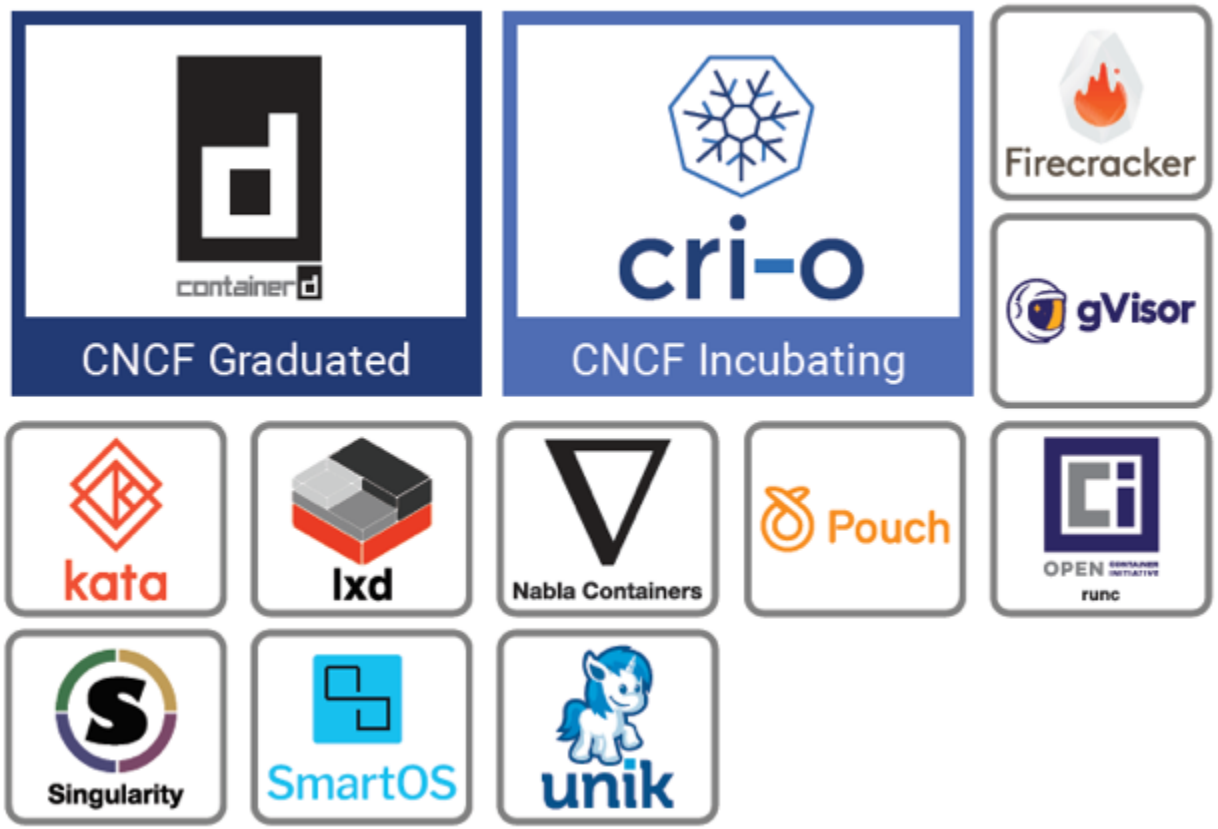
- A gRPC interface and a group of libraries (protobuf)
- Enables Kubernetes to use a wide variety of container runtimes
- Introduced in Kubernetes 1.5
- GA in Kubernetes 1.7



Kubernetes, CRI, Runtime

CRI Implementations

Container Runtimes



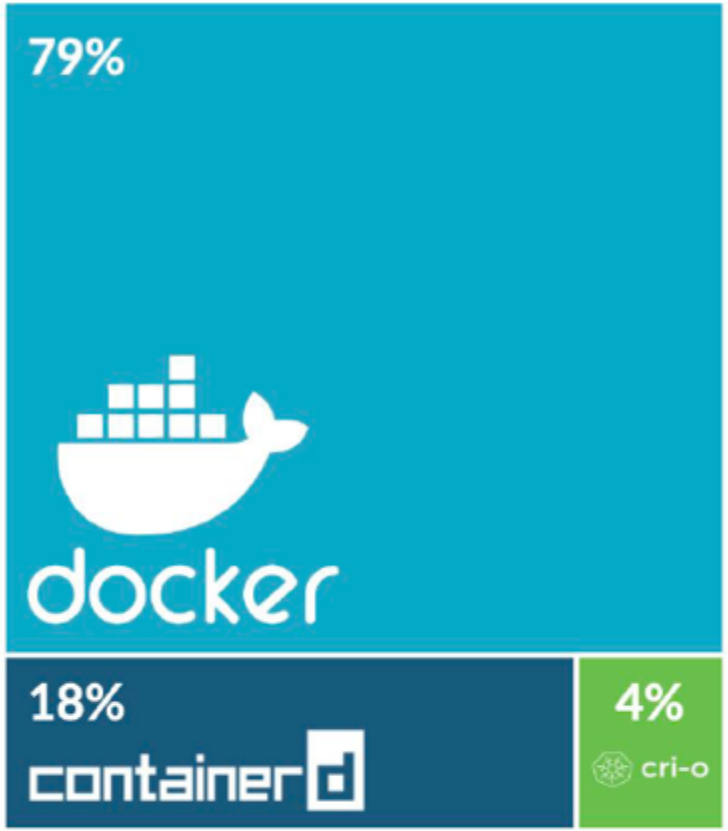
Why Different Runtimes?

- Performance/Stability
- Extensibility/Custom Uses
- Compatibility to K8s
- Additional Isolation

https://www.youtube.com/watch?v=92MQcV0GSyk

Slides: https://sched.co/Uaaq

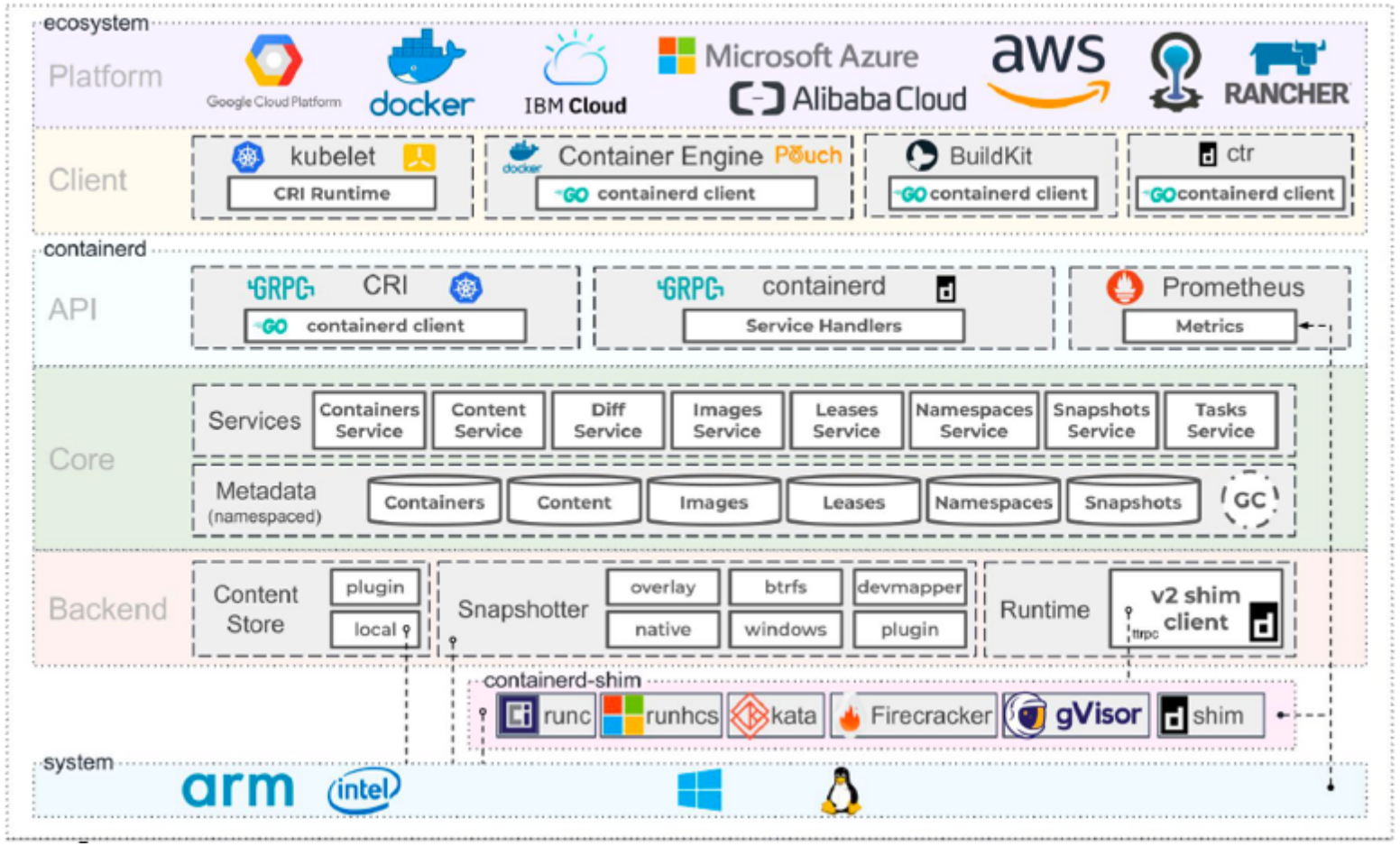
High-Level Runtimes Usage



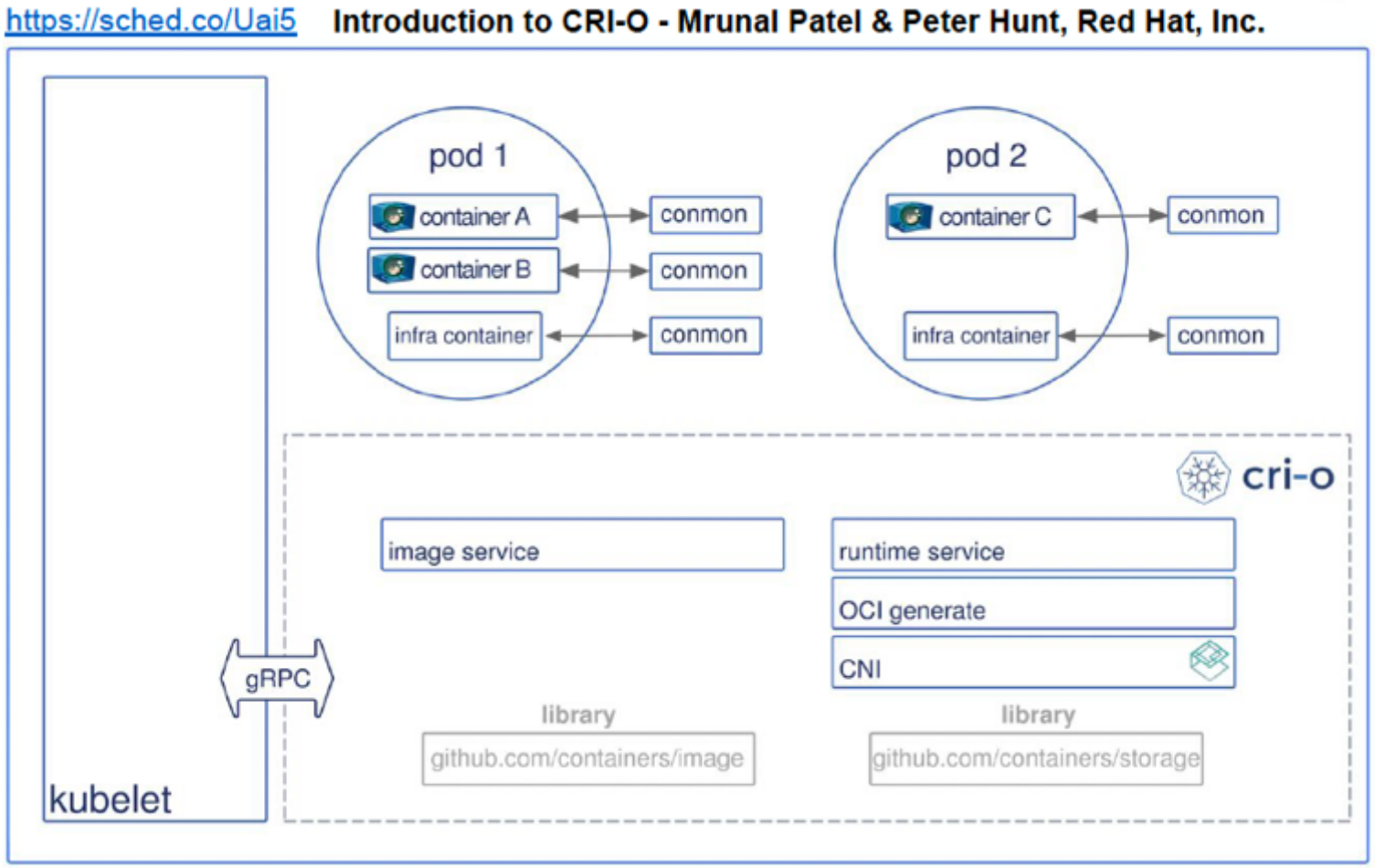
Source: Sysdig 2019 Container Usage Report

https://sysdig.com/blog/sysdig-2019-container-usage-report/

Containerd


ecosystem
Platform
Google Cloud Platform
docker
IBM Cloud
Microsoft Azure
Alibaba Cloud
aws
RANCHER
Client
kubelet
CRI Runtime
Container Engine
containerd client
BuildKit
containerd client
ctr
containerd client
containerd
API
GRPC
CRI
containerd client
GRPC
containerd
Service Handlers
Prometheus
Metrics
Core
Services
Containers Service
Content Service
Diff Service
Images Service
Leases Service
Namespaces Service
Snapshots Service
Tasks Service
Metadata
(namespaced)
Containers
Content
Images
Leases
Namespaces
Snapshots
GC
Backend
Content Store
plugin
local
Snapshotter
overlay
btrfs
devmapper
native
windows
plugin
Runtime
v2 shim
client
ttrpc
containerd-shim
runc
runhcs
kata
Firecracker
gVisor
shim
system
arm
intel

CRI-O Architecture

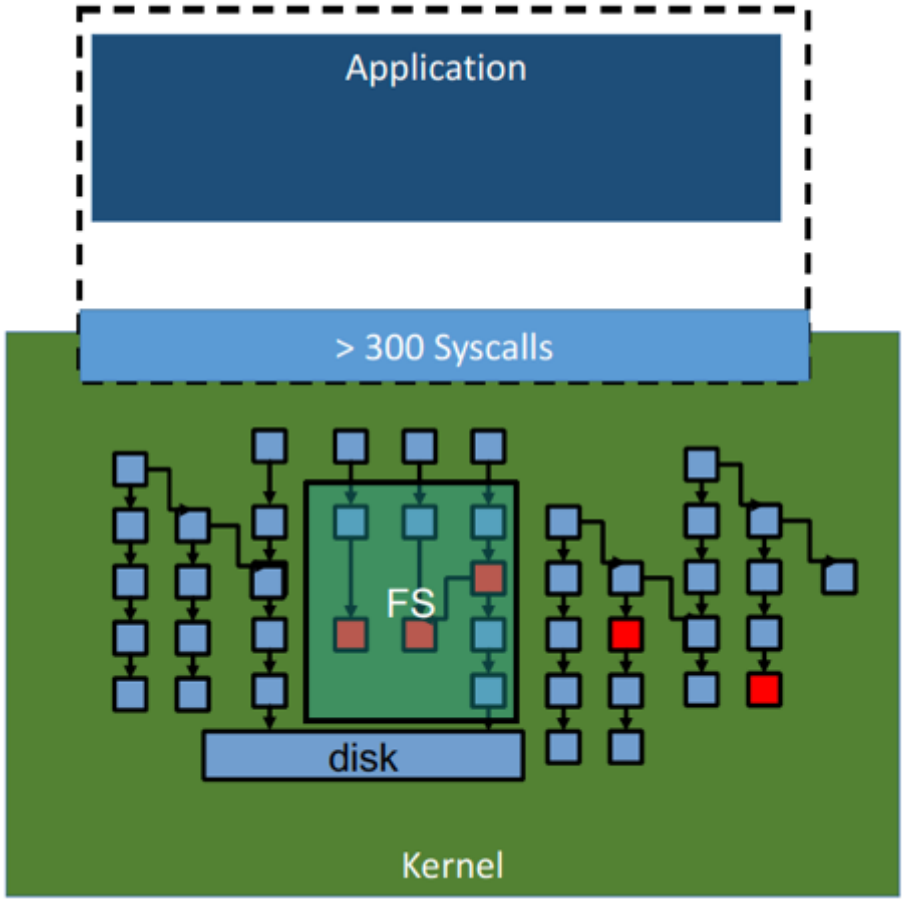


- Lightweight CRI runtime made as a Kubernetes specific high-level runtime
- Supports other OCI compatible low-level runtimes in theory, but relies on compatibility with the runc

Recent Container Trend

New Isolations

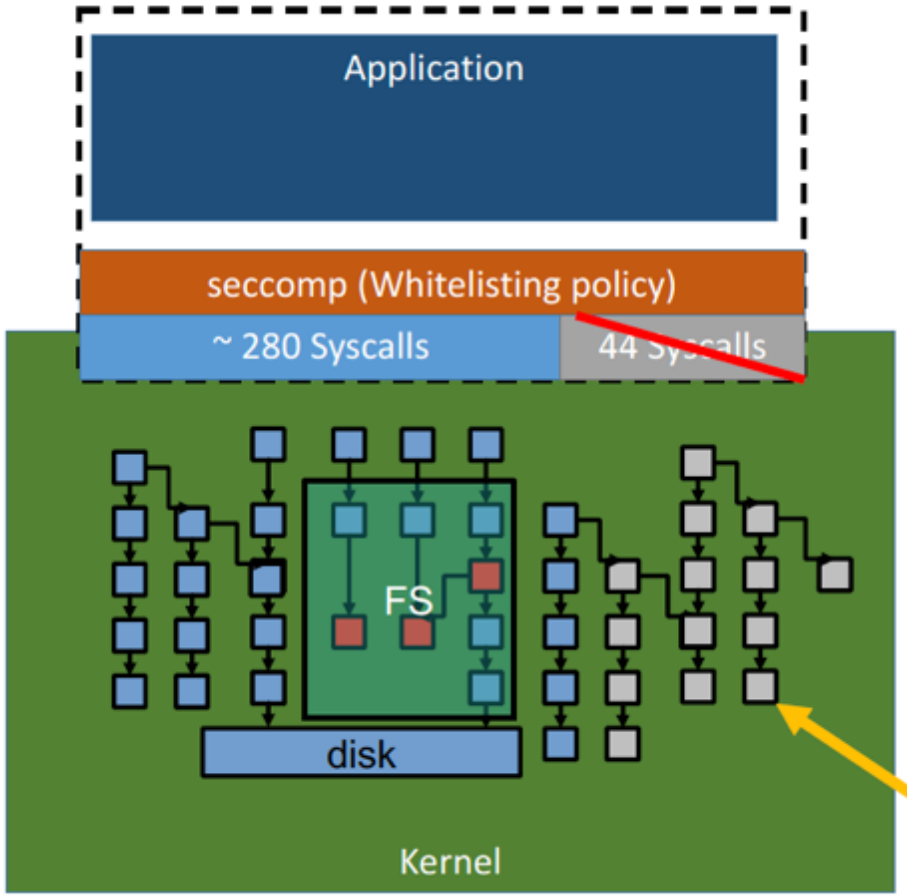
Kernel Footprint



- Exploits target vulnerable part of kernel via syscalls.

- If we restrict the number of syscalls
- → Less reachable kernel functions
- → Less potential vulnerabilities
- → Less possible exploits

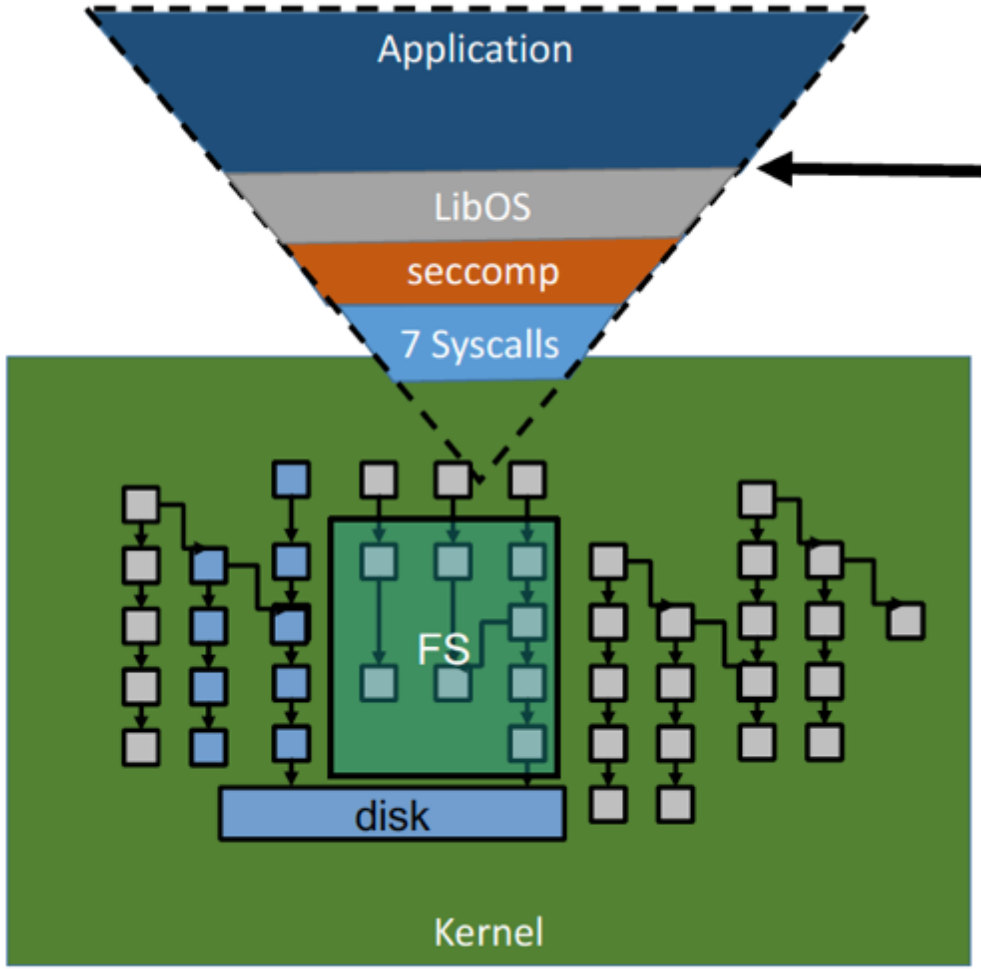
Docker Default Seccomp Policy



- Docker default seccomp policy
  - disables around 44 system calls out of 300+.
- Generic seccomp policies – hard to create s.t. it is secure
- Syscall profiling is mostly heuristic based

Greyed – unreachable functions

Nabla (IBM)



Original 300+ Syscall interface

- **Deterministic** and **generic** seccomp policy
- Only 7 syscalls!
- Uses LibOS techniques

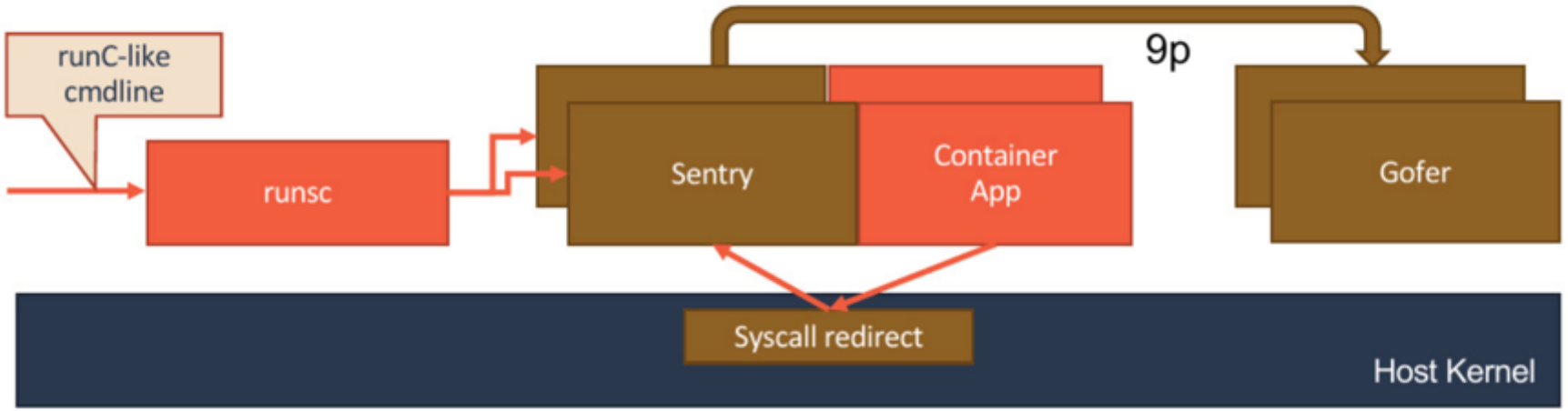
"Unikernels as Processes"
(ACM SoCC '18)

gVisor (Google)

All in one binary (runsc + Sentry +Gofer)

Sentry as a user-space kernel, and per-container process stub

Gofer for 9p IO



Detailed Architecture of gVisor (1)

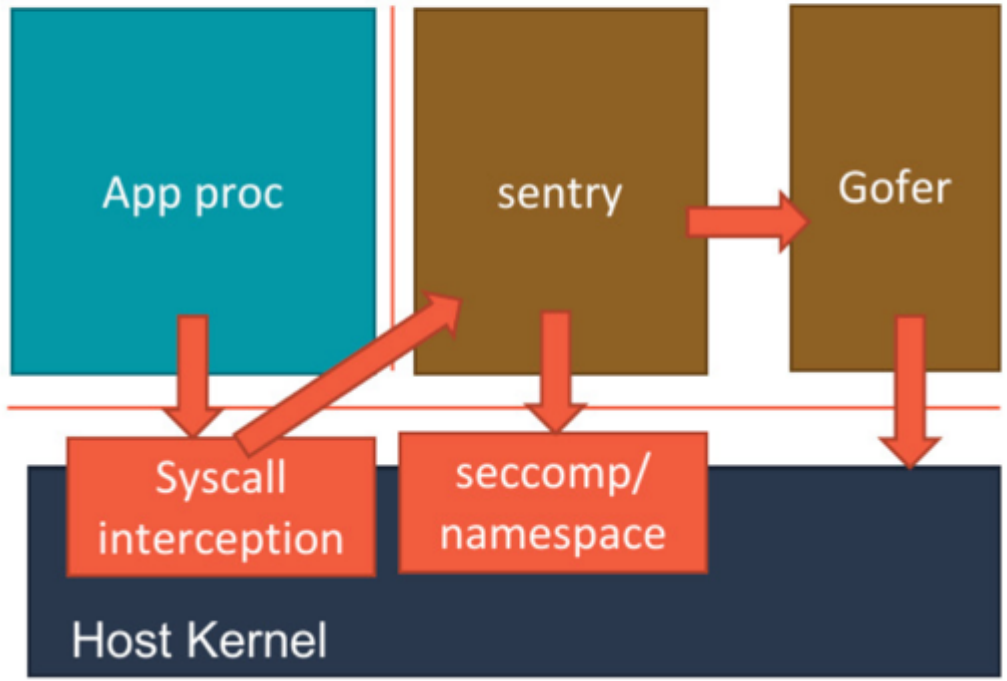
2 layers of isolations

- Minimal kernel and written in Golang
  - Safer but the compatibility…
- Ring protection and small set of syscalls
  - Avoid access more buggy syscalls

File operations are proceeded in separated gofer process

Syscall interception by ptrace or kvm

- The performance impactions…



Detailed Architecture of gVisor (2)

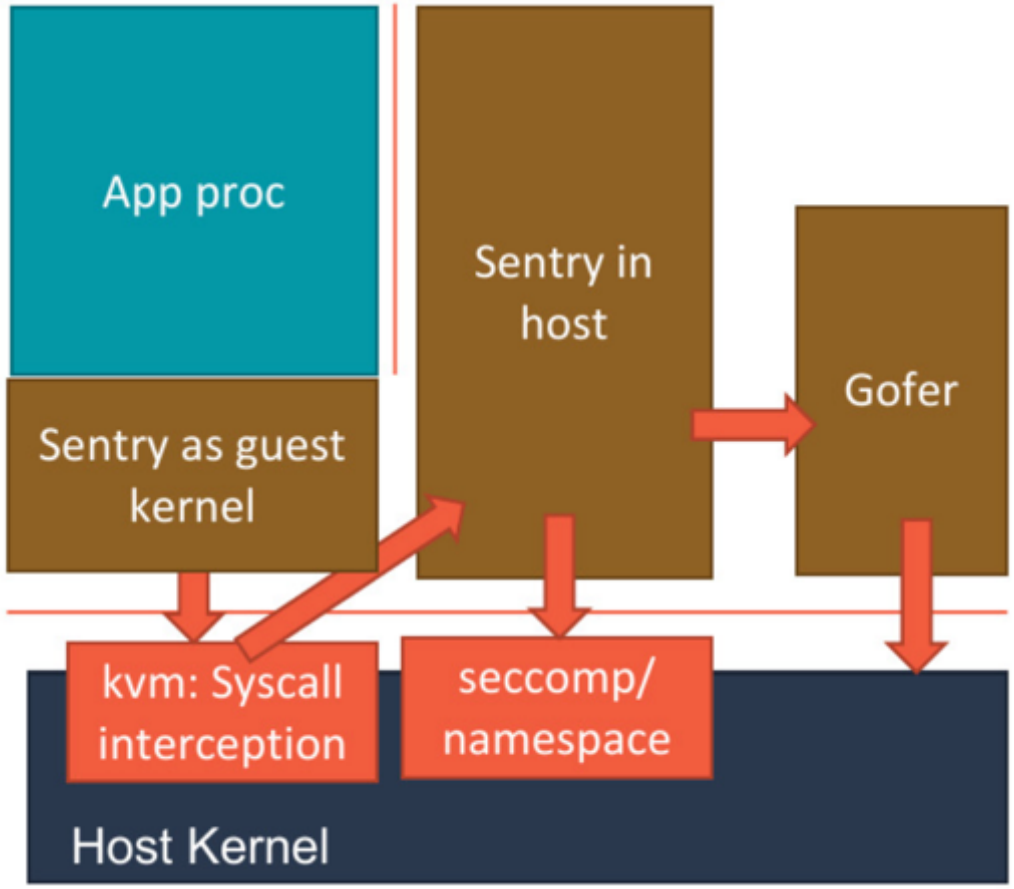
In gVisor-kvm:

Sentry is both host process and guest kernel

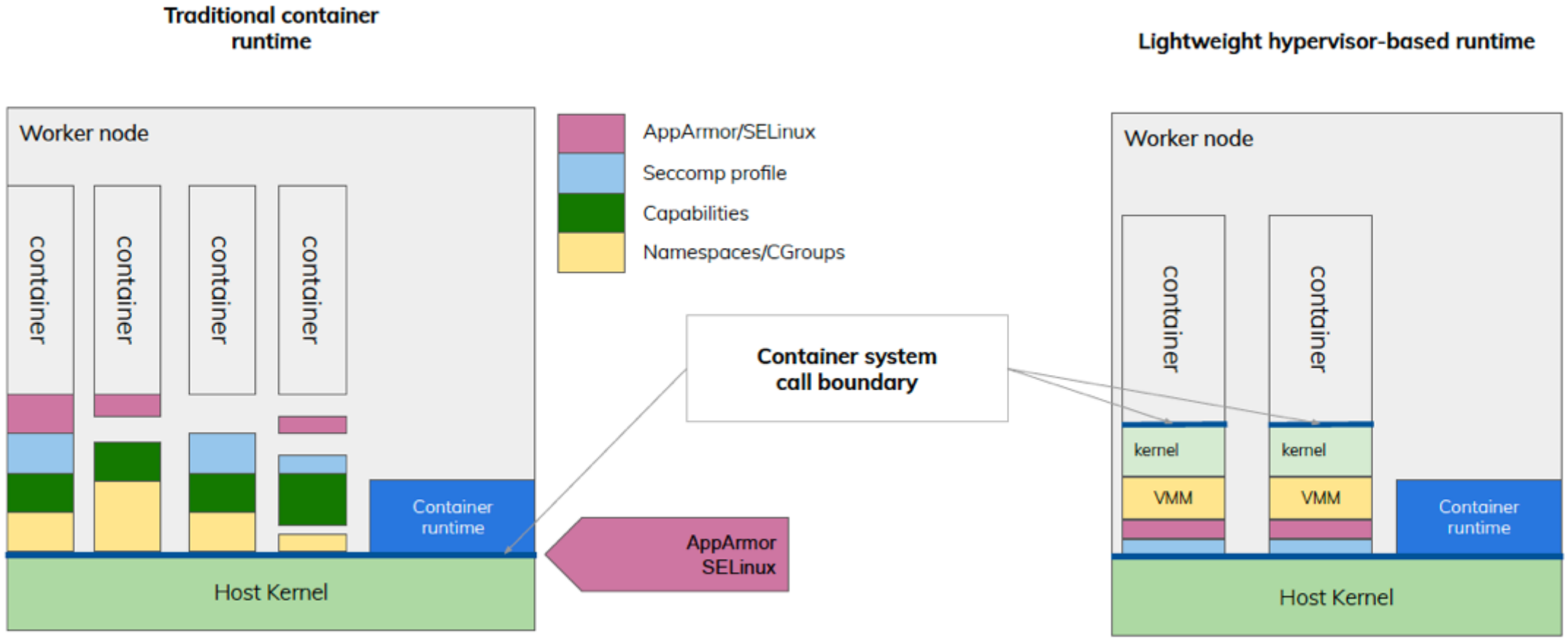
- Ring0 lower in guest
- Upper in VM
- In Host

Kvm is for higher syscall interception performance rather than isolation

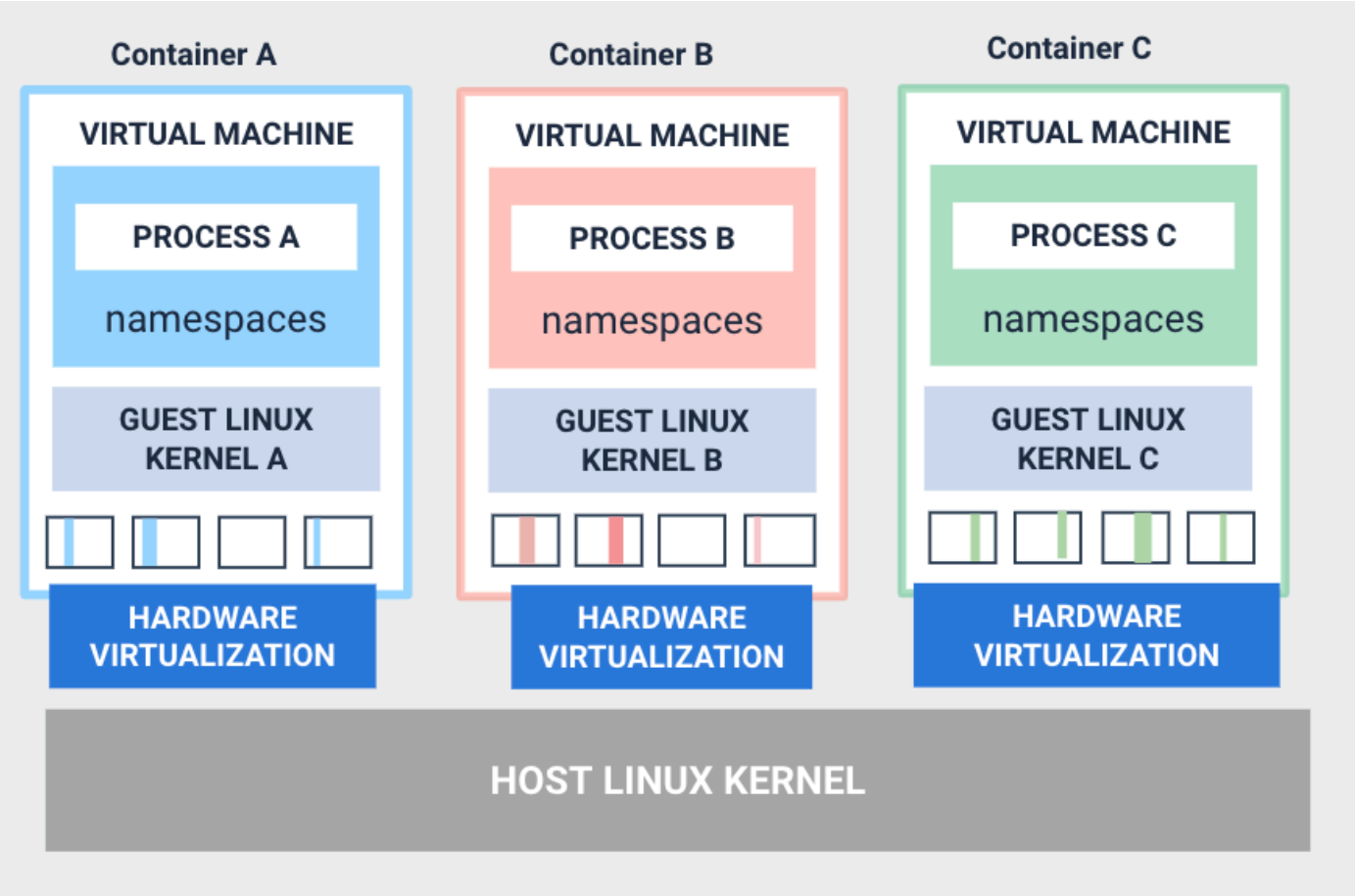
- No insolation between different modes of sentry itself



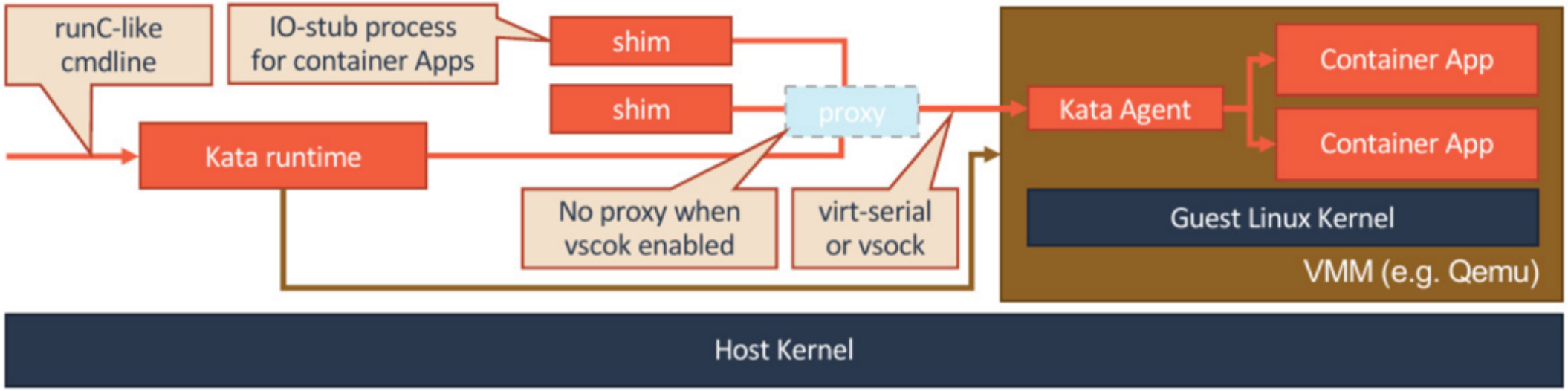
Hypervisor-based Runtime Isolation



Kata Container



Kata Container Architecture



Detailed Architecture of Kata

Classic hardware-assisted virtualization

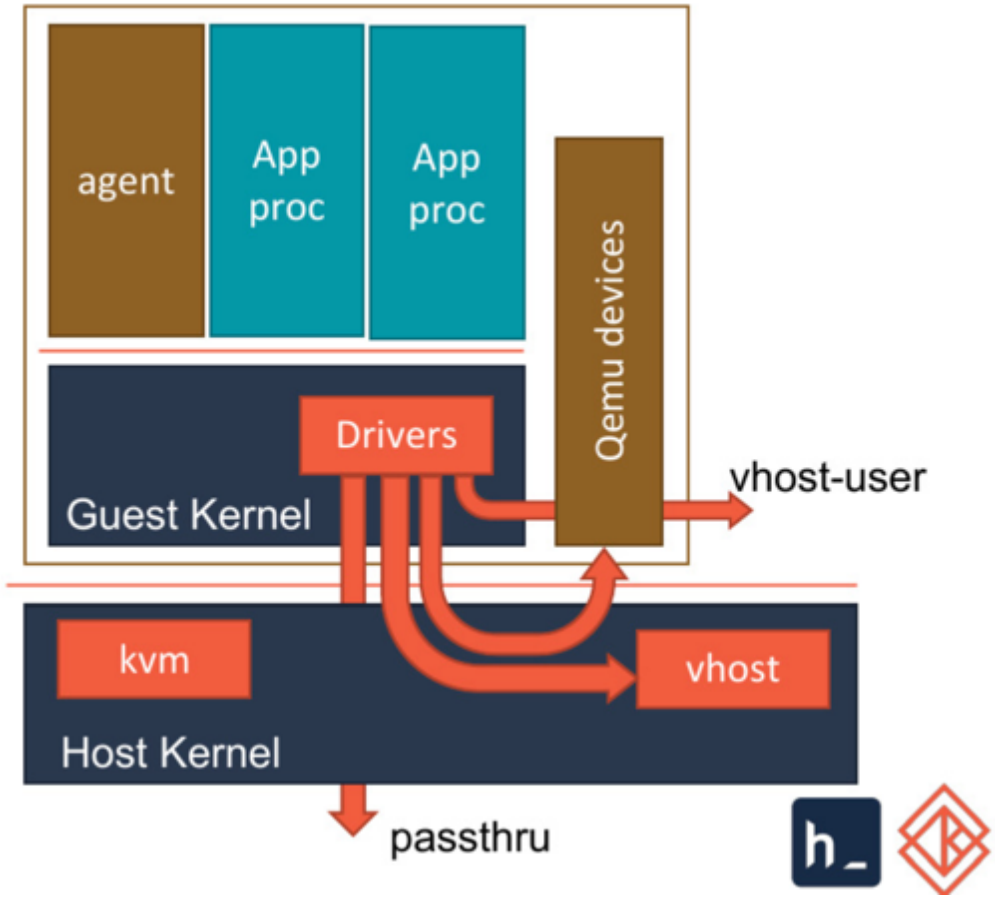
- Provide virtual hardware interface to Guest

2 layers of isolation

- kvm and CPU ring protection

Many IO optimization ways

- Pass through
- Vhost and vhost-user



Comparison

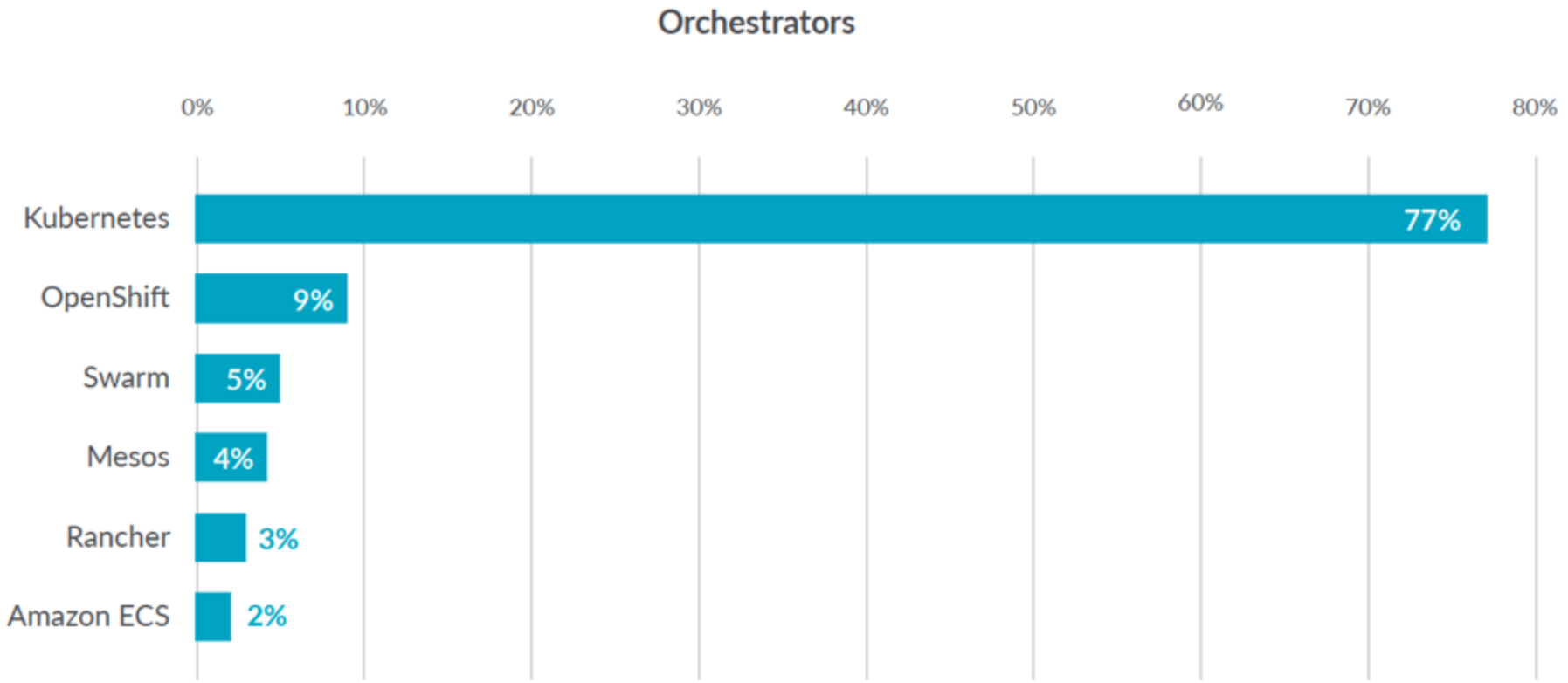
	Supported container platforms	Dedicated guest kernel	Support different guest kernels	Open source	Hot-plug	Direct access to HW	Required hypervisors	Backed by
Nabla	Docker, K8s	Yes	Yes	Yes	No	No	None	IBM
gVisor	Docker, K8s	Yes	No	Yes	No	No	None	Google
Firecracker	Not yet	Yes	Yes	Yes	No	No	KVM	Amazon
Kata	Docker, K8s	Yes	Yes	Yes	Yes	Yes	KVM or Xen	OpenStack

	CRI-O	Containerd CRI plugin	Docker Engine (native)	gVisor CRI plugin	CRI-O Kata Containers
sponsors	CNCF	CNCF	Docker Inc	Google	Intel
started	2016	2015	Mar 2013	2015	2017
version	1.12	1.2	17.03	runc	1.3
runtime	runc (default)	containerd managing runc	runc	runsc	kata-runtime
kernel	shared	shared	shared	partially shared	isolated
syscall filtering	no	no	no	yes	no
kernel blobs	no	no	no	no	yes
footprint	-	-	-	-	30mb
start time	<10ms	<10ms	<10ms	<10ms	<100ms
io performance	host performance	host performance	host performance	slow	host performance
network performance	host performance	host performance	host performance	slow (see comment)	close to host performance
Docs	https://github.com/kubernetes-sigs/cri-o/	https://github.com/containerd/cri	https://github.com/moby/moby	https://github.com/google/gvisor	https://github.com/kata-containers/runtime
Why?	Lightweight Kubernetes specific. No need for the Docker daemon. Default on OpenShift. Probably the best container based runtime.	Installed by default with the latest Docker Engine. Kubernetes can now use ContainerD directly. Docker can also be used on the same hosts directly. No need to run the DockerD daemon. Beta on Google GKE.	Most mature runtime that has been tested and iterated on by a massive number of users. Can use seccomp, SELinux and AppArmor to harden. Fastest start times. Lowest memory usage.	Used by gcloud appengine as the isolation layer between customers. Good for stateless web apps. Adds two additional layers of security over standard containers.	Arguably the most secure option. The major trade-offs for security don't actually seem that bad. Will anyone notice 100ms startup time increase on their micro service? Or an extra 30mb per container? Doubtful.
Why not?	Same security issues as native Docker Engine. Still need to manage a bunch of security policy stuff that nobody ever does.	This is slightly newer as it has been through a few iterations of being installed differently.	Kubernetes is moving to the CRI plugin architecture. Hardening is too complex for most to manage. DockerD is quite bloated and running it as root is bad.	Not versioned and shouldn't be used in production yet on Kubernetes. Not good for applications that make lots of syscalls. Not all 400 Linux syscalls implemented causing some apps to not work (e.g. postgres).	The kata-runtime itself is v1 however I'm not sure how this translates to Kubernetes readiness. Less efficient binpacking due to 30mb memory overhead. Slower start times.

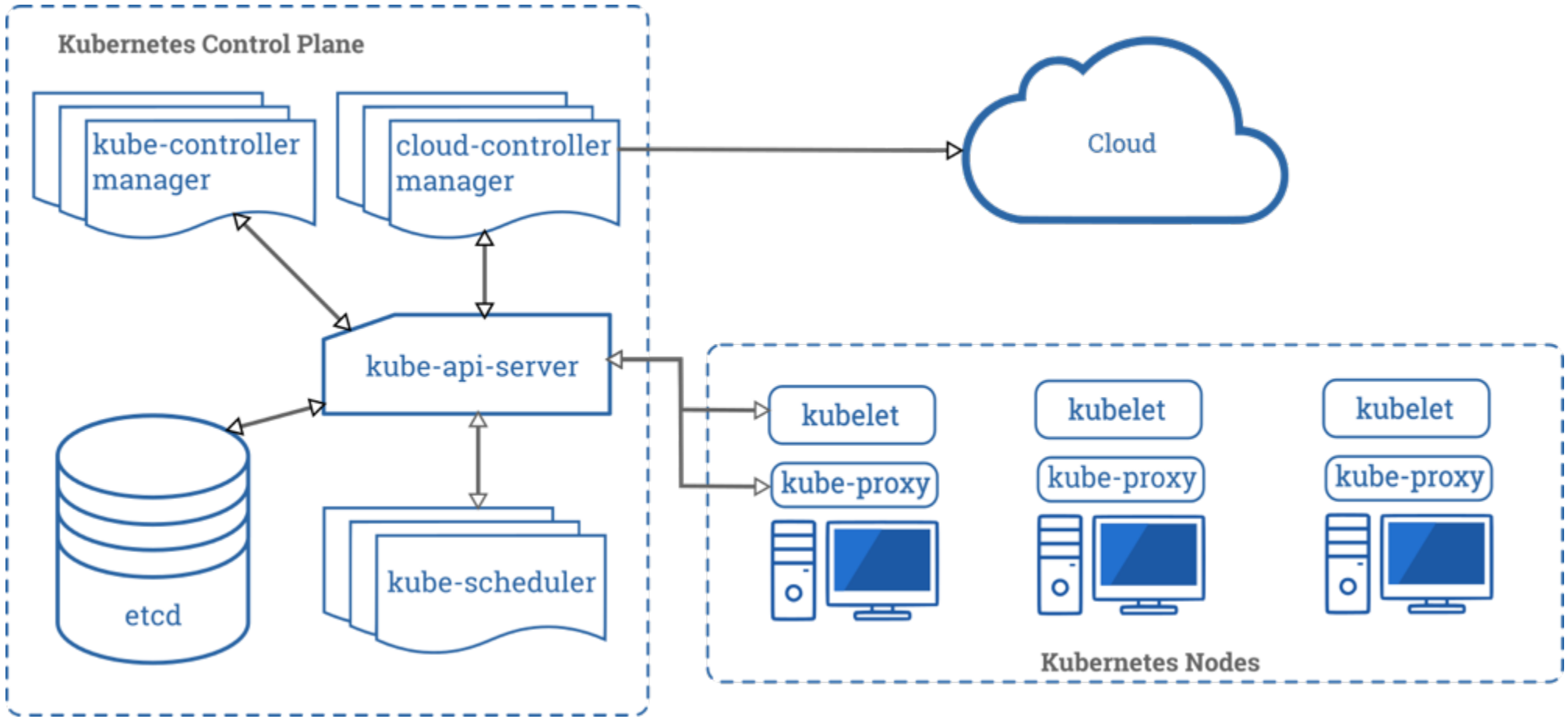
Kubernetes

Container Delivery Pipeline

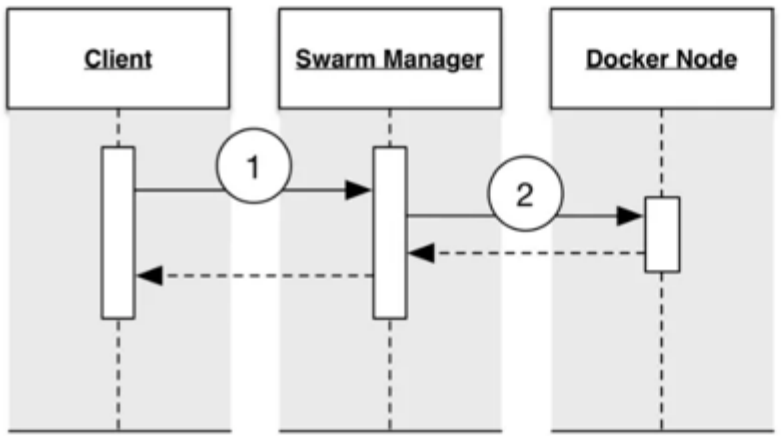
Orchestrator Usage



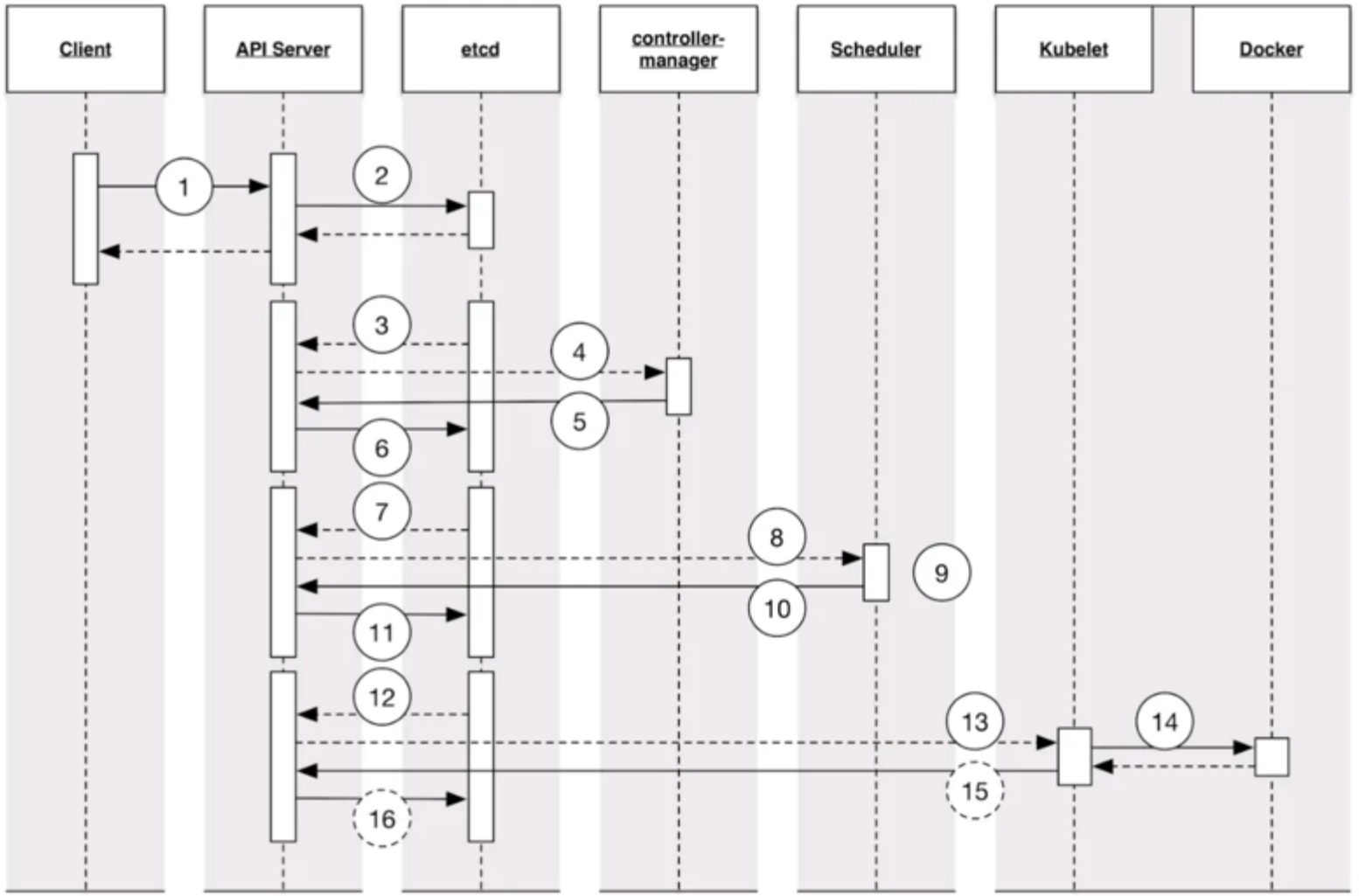
Kubernetes


Kubernetes Control Plane
kube-controller manager
cloud-controller manager
Cloud
kube-api-server
kubelet
kubelet
kubelet
kube-proxy
kube-proxy
kube-proxy
etcd
kube-scheduler
Kubernetes Nodes

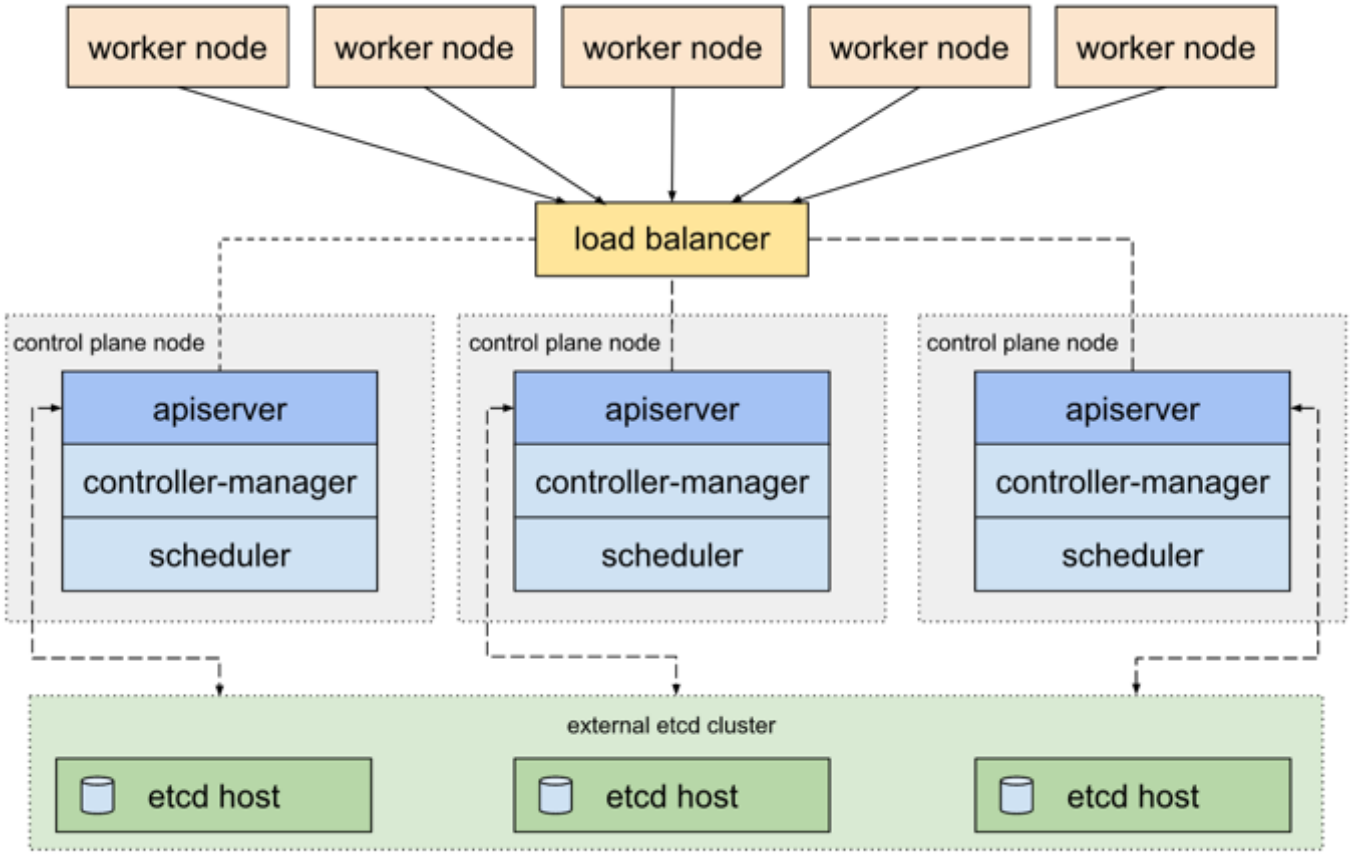
Docker Swarm



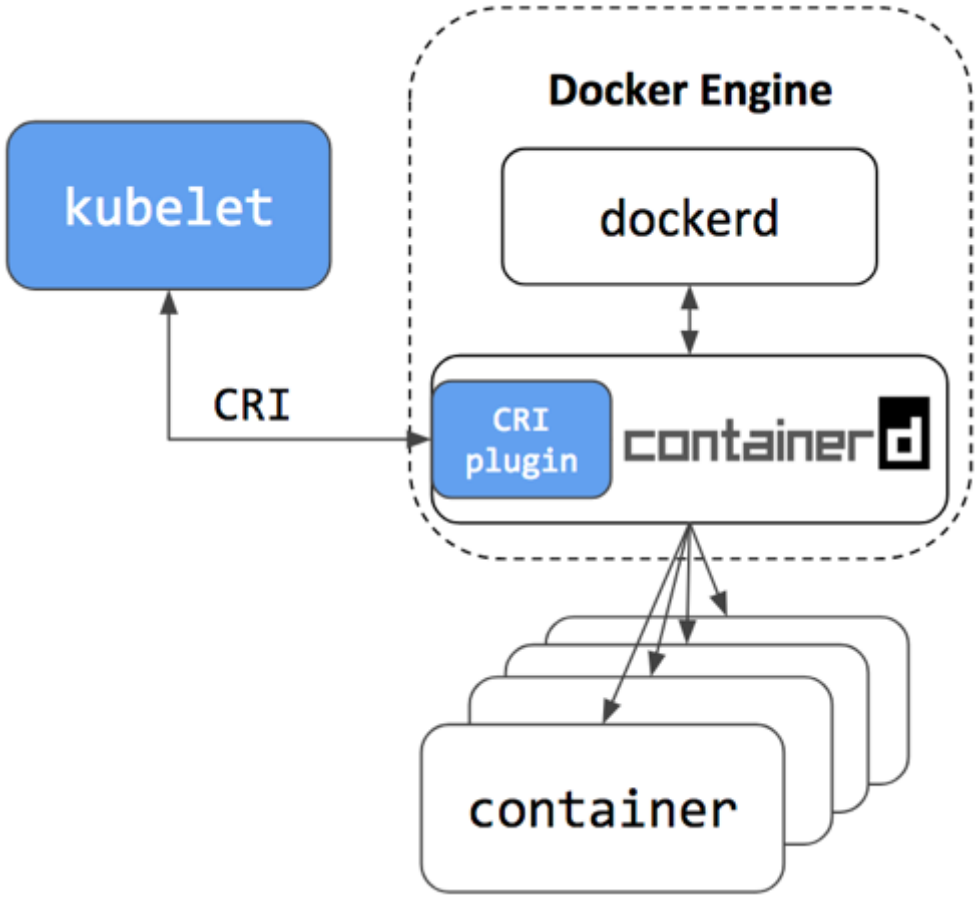
Google Kubernetes



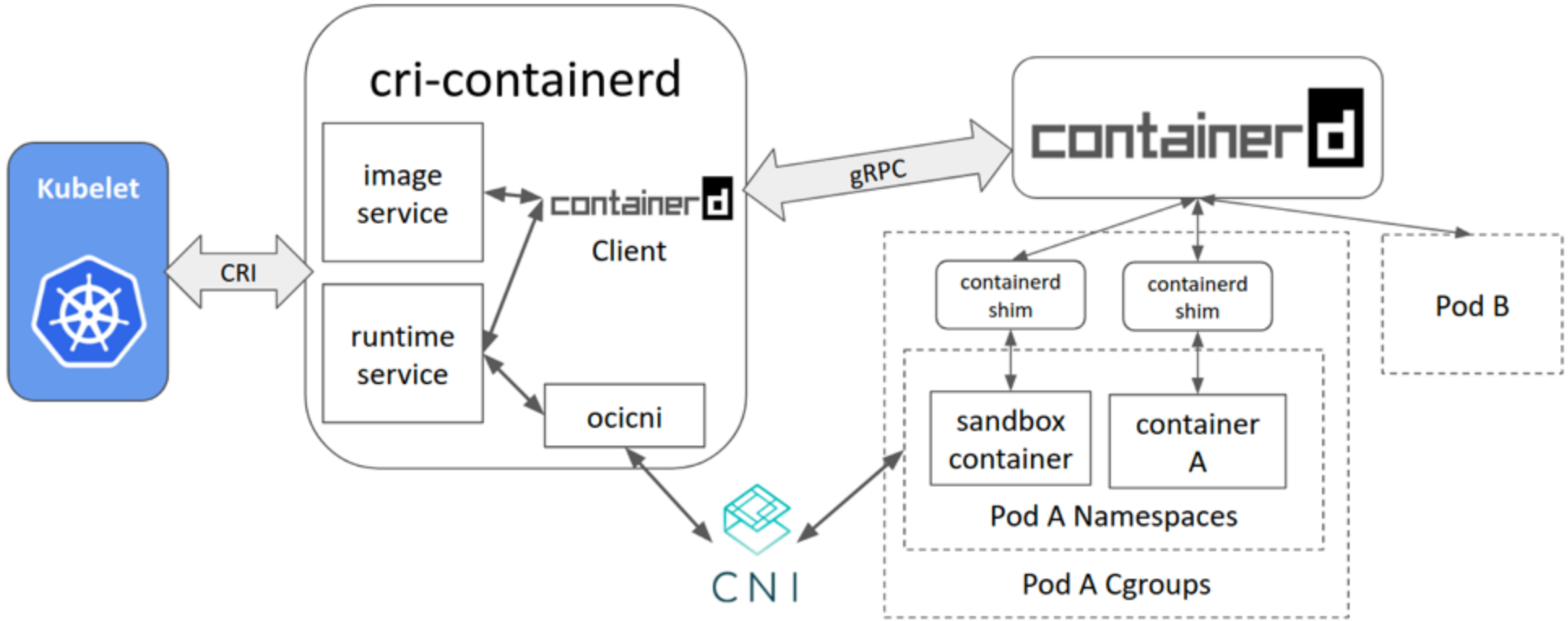
External etcd Cluster



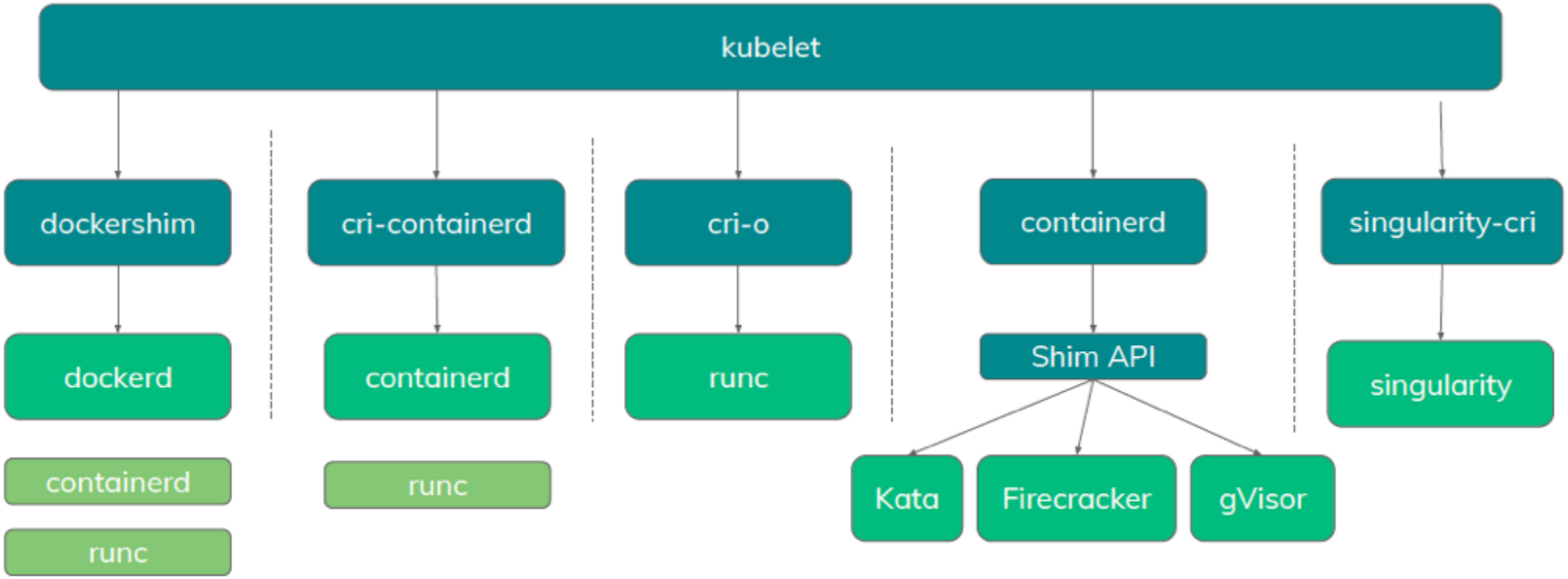
Kubernetes with Docker



Kubernetes with cri-containerd, containerd



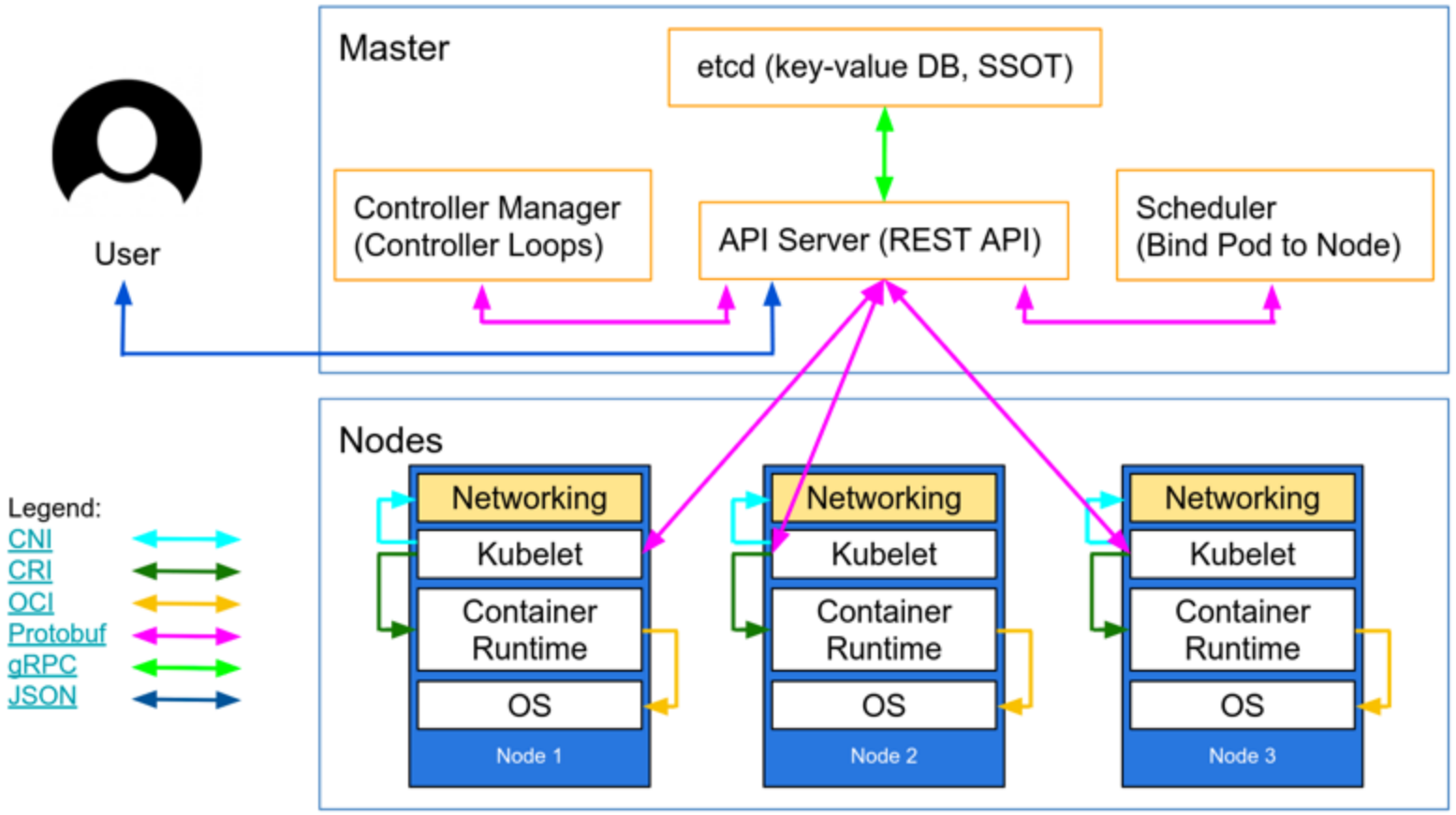
Kubernetes with CRIs



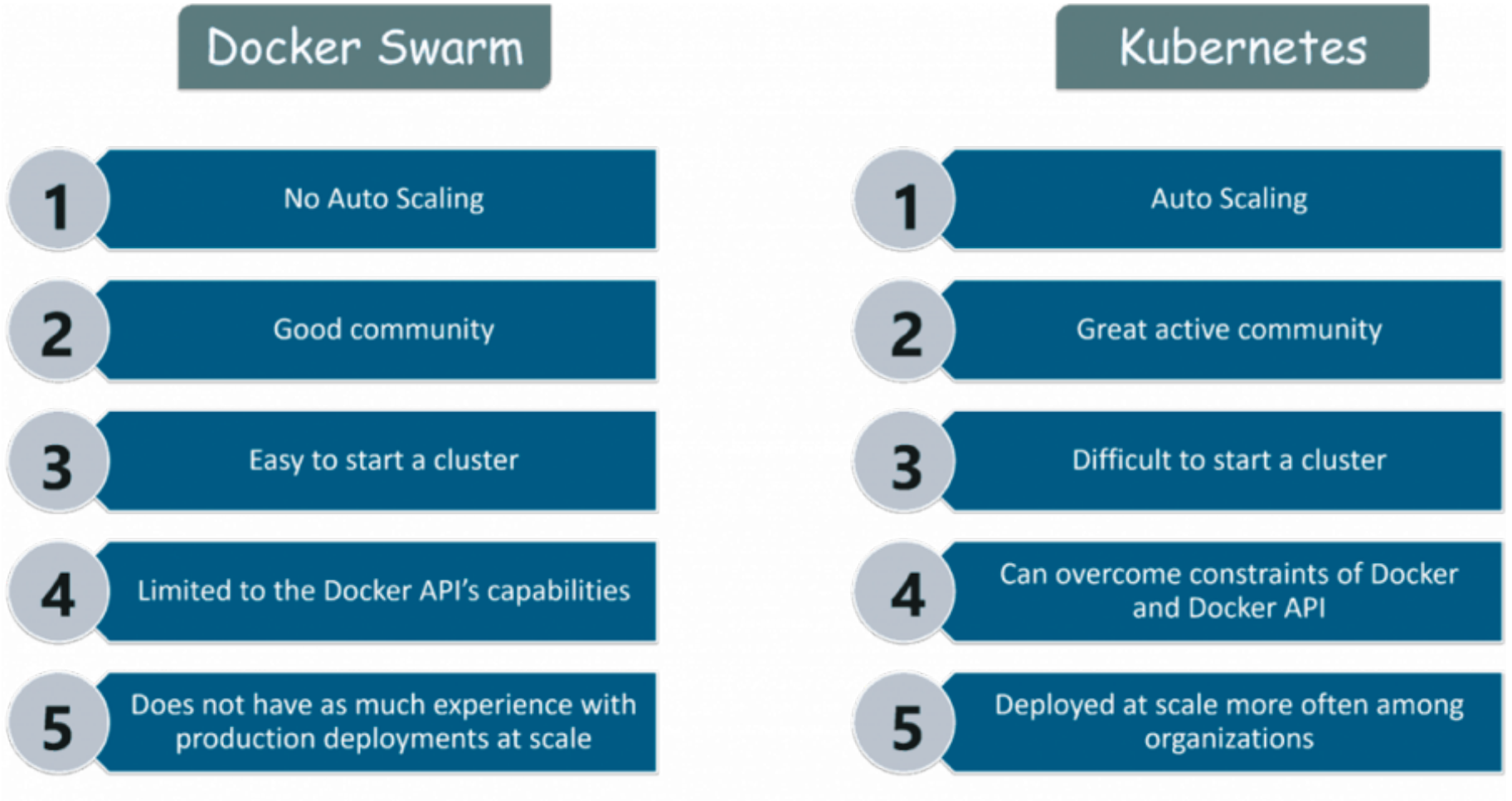

```
kubelet --container-runtime {string}
        --container-runtime-endpoint {string}
```

RuntimeClass
and/or annotations

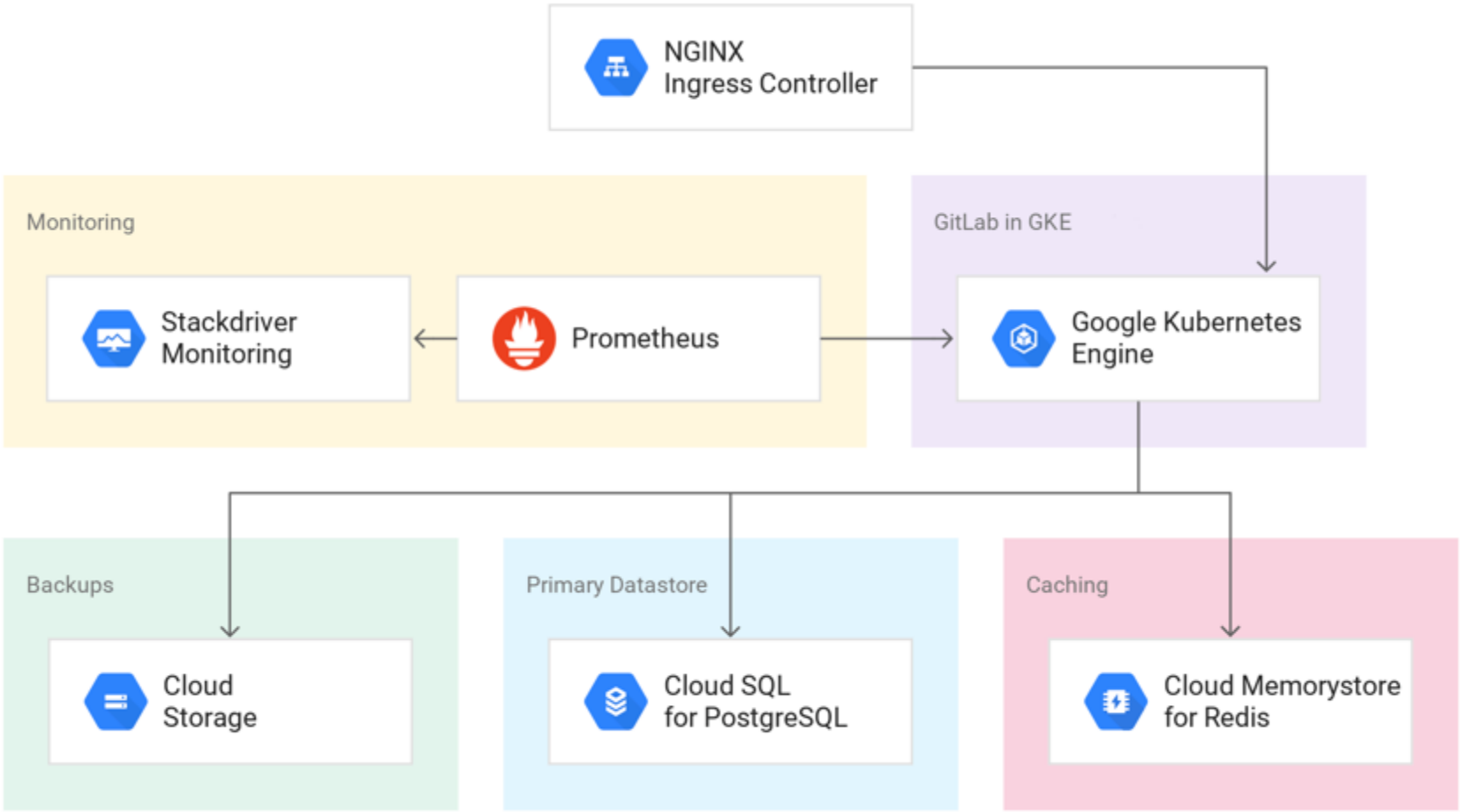
Kubernetes Communication



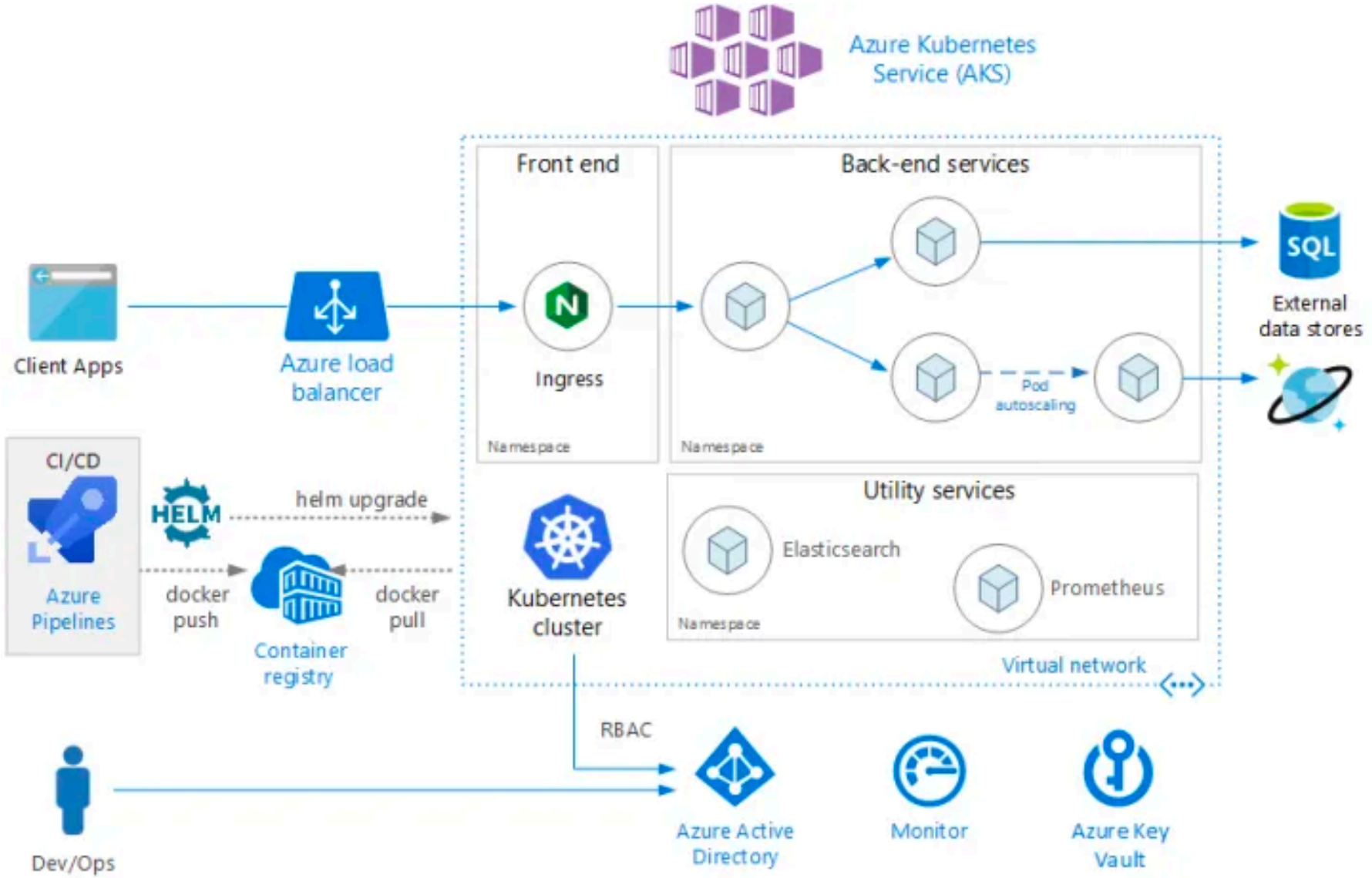
Docker Swarm vs Kubernetes



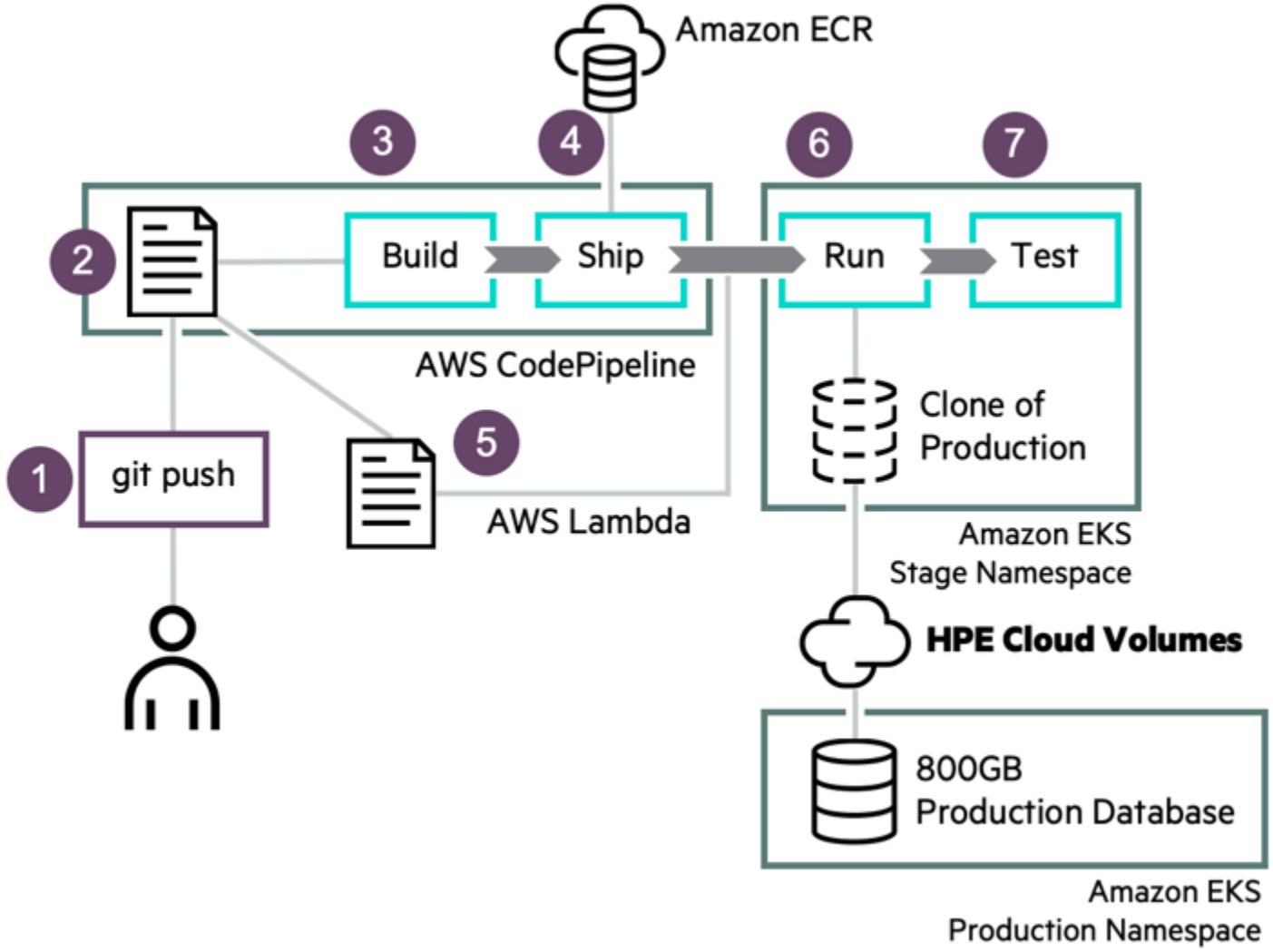
DevOps in GCP Kubernetes (GKE)



DevOps in Azure Kubernetes (AKS)



DevOps in AWS Kubernetes (EKS)



Hewlett Packard
Enterprise

GKE vs AKS vs EKS

	Google GKE	Microsoft AKS	Amazon EKS
Year started	2014	2017	2018
Kubernetes Versions	1.11 - 1.12 - 1.13	1.10 - 1.11 - 1.12 - 1.13 -1.14	1.11 - 1,12 - 11.13
Regions Supported	Worldwide	Worldwide	Almost Worldwide
SLA(master)	99.95 (regional), 99.5 (zonal)	99.5	99.9+ or 99.9 to below 95.0 based on payment
Managed Worker Nodes	Yes	Yes	No
New worker start time	< 3 mins	< 5 mins	< 5 mins
Autoscaling Worker Nodes	Yes	Yes (preview)	Yes (not managed)
Autoscaling with Container Instances	No	Yes	No
Kubernetes Upgrades	Automatic or On Demand	On Demand	Automatic security, On Demand upgrades
Control Plane Costs	Free	Free	20 cents per hour per master
Compliance	PCI DSS, ISO, SOC, HIPAA	PCI DSS, ISO, SOC, HIPAA	PCI DSS, ISO, SOC, HIPAA
Auto Repair	Yes	No(In development)	No
Cross region load balancing	Yes	Yes (with Azure Traffic Manager)	Yes(wit ALB ingress controller, not auto deletable)
Container as a Service integration	No	Virtual Kubelet with ACI	No
Dev UX	Mediocre with UI, good with external tools(gcloud code)	Very good with UI, superb with VS Code	Bad
Virtual Pod Autoscaling	Yes (beta)	No	No
Services Mesh network Integration	Istio(beta)	No(needs to be implemented by hand)	Yes (AWS App mesh)
Load Balancer SLA	99.9+ or 99.9 to below 95.0 based on payment	99.9 (with standart SKU)	99.9+ or 99.9 to below 95.0 based on payment
Kubernetes Service UI	Good, simple	Extensive, mediocre design	Bad, not in one screen(across entirely different pages)
CLI tool	Simple, security enchanted	Extensive, documentation is not good	eksctl (open source-official), basic
Documentation	Community driven, official documentation is not extensive	Official documentation is extensive	Official documentation is bad, workshop site is good
DNS service	Not integrated for creations, able to use other dns	Integrated Azure DNS service for free	Ingrated with Route 53, initial costs
Read-Write Many for PV	No	Yes	Yes
Addons provided	Very extensive list	Basic needs	None

Kubeadm Default Setup

- Users can create pods with wild permissions by default
- Scheduling is not a security boundary
- Namespace isolation is not always enough
- Mitigations: encrypt etcd secrets at rest, and don’t run a kubelet on control plane nodes

Kubernetes Security

Area of Concern for Workload Security	Recommendation
RBAC Authorization (Access to the Kubernetes API)	https://kubernetes.io/docs/reference/access-authn-authz/rbac/
Authentication	https://kubernetes.io/docs/reference/access-authn-authz/controlling-access/
Application secrets management (and encrypting them in etcd at rest)	https://kubernetes.io/docs/concepts/configuration/secret/ https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/
Pod Security Policies	https://kubernetes.io/docs/concepts/policy/pod-security-policy/
Quality of Service (and Cluster resource management)	https://kubernetes.io/docs/tasks/configure-pod-container/quality-service-pod/
Network Policies	https://kubernetes.io/docs/concepts/services-networking/network-policies/
TLS For Kubernetes Ingress	https://kubernetes.io/docs/concepts/services-networking/ingress/#tls