Day 2

logistics

- Class Hours:
- Start time is 9:00am
- End time is ~4:00pm
- Class times may vary
- Breaks mid-morning and afternoon (15 minutes)

- Lunch:
- Lunch is 12:00pm to 1:15pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

- Telecommunication:
- Turn off or set electronic devices to vibrate / do not disturb
- Reading or attending to devices can be distracting to other students

- Miscellaneous:
- Courseware is available electronically
- Bathroom

Docker Security

Q: How important is security?

Containers Aren’t Very Secure

- VMs are considered safer than containers
- No host kernel separation

Container Security Best Practices

Get 80% of the way there with 20% of the effort

- Overlaps well with non-container or pre-container best practices

Keep Host OS Image Updated

- Like keeping the VM Host OS updated

- AWS: AMIs
- GCP: images (note COS: container-optimized OS)
- linuxkit
- Team-generated

Keep All Base Images Updated

- Like keeping the VM Guest OS updated

Examples:

- alpine
- distroless
- any well maintained base image you trust
  - redis
  - nginx
  - haproxy

Keep the Application Updated

- Follow security vulnerabilities for your stack
- Build on latest stack-specific base images
- Keep libraries and application code up to date

No Secrets in the Sauce

- Artifacts, not secrets
- Images not secret by convention

Where to Store Secrets

- Orchestration-layer secret apis
  - kubernetes secrets
  - docker secrets
  - etc…
- vault
- cloud keychains
- custom storage?

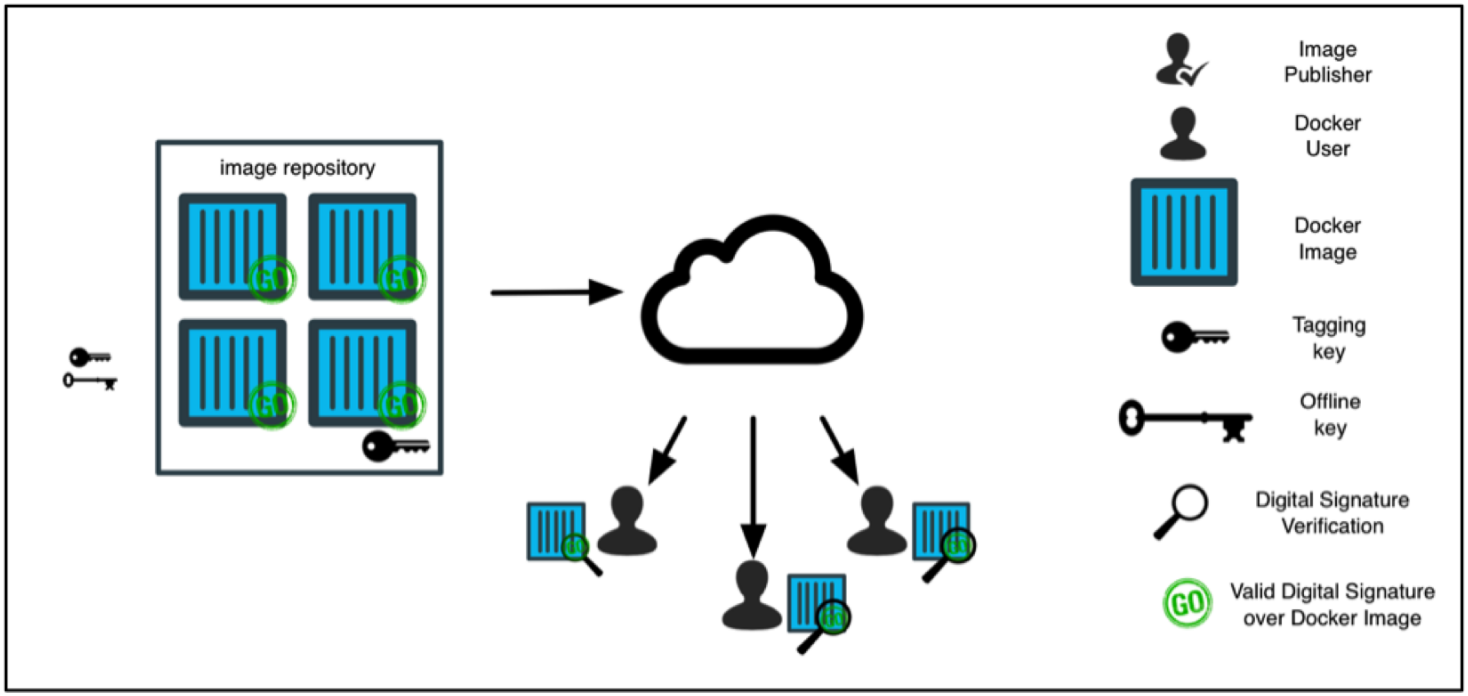
Create minimally sized images

- Less surface area to attack

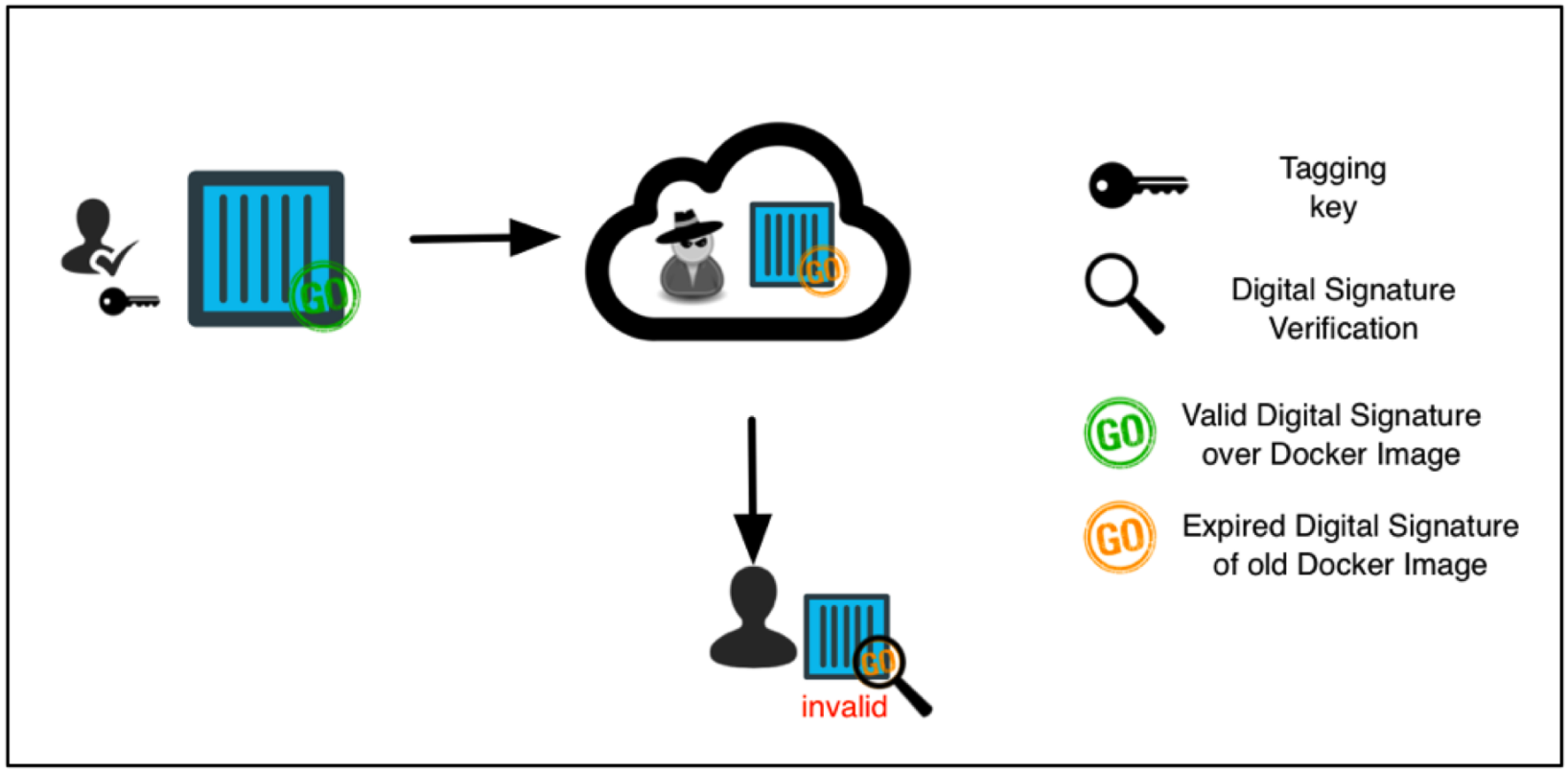
docker content trust = signed images

- Image author verification
- Image integrity

run on client: export DOCKER_CONTENT_TRUST=1



docker content trust (cont)



docker notary

- Notary is a tool for publishing and managing trusted collections of content.
- Publishers can digitally sign collections and consumers can verify integrity and origin of content.
- For example, a collection of container images

Don’t run as root

- By default root in a container is also root on the host
- Most applications don’t need to run as root

details:

requires USER layer in dockerfile (or orchestration layer support) and coordination with base image, e.g.:

```
USER nobody
```

```
USER guest
```

container user mapping

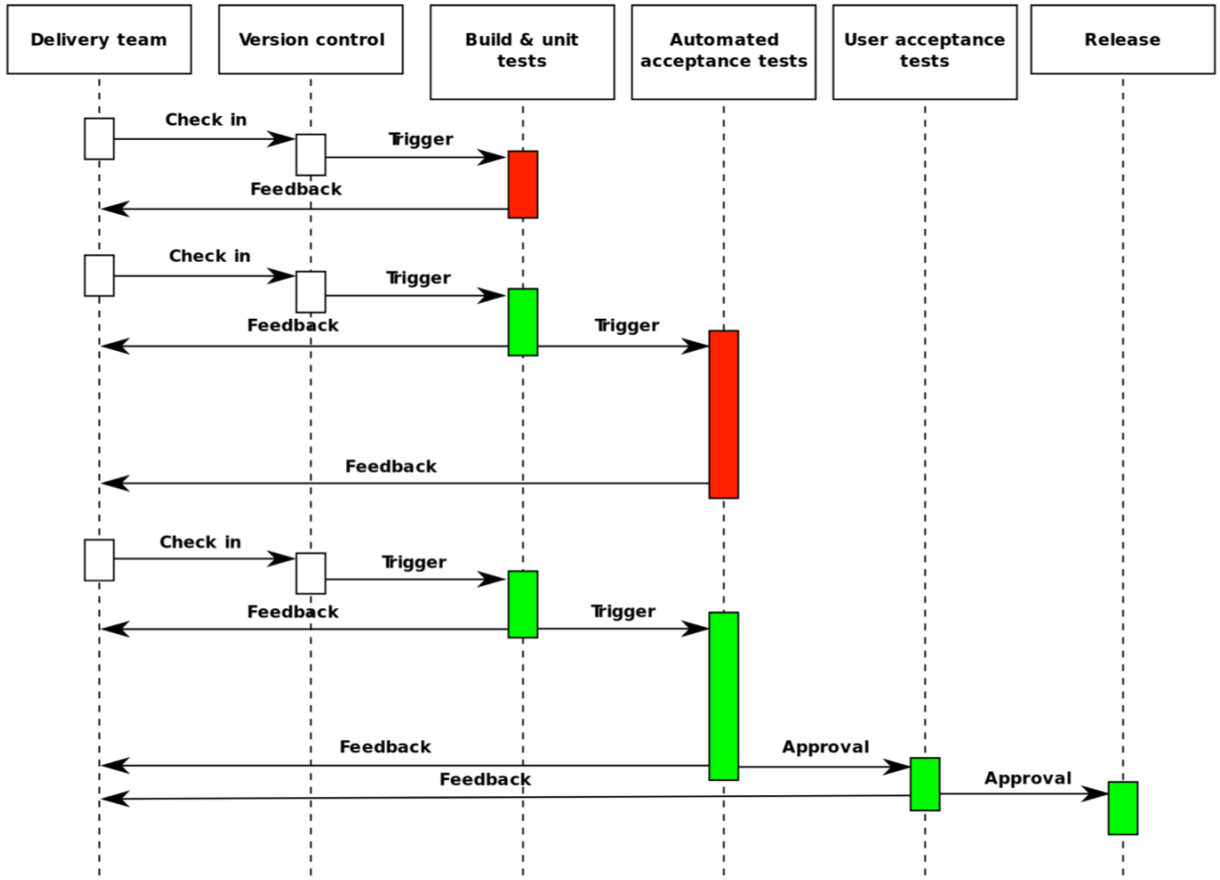
dockerd supports the new --userns-remap flag which will map users in a container to different UIDs on the host

details:

- incompatible with host namespace sharing: pid, net
- be aware of volume permissions
- it's a daemon-wide flag, not great for "cattle"
  - Disabling map requires --userns=host flag on a docker command
- requires writing map files on the container host:

/etc/subuid
/etc/subgid

IT Best Practices now with Containers - CI/CD



IT Best Practices now with Containers - Logging

- Orchestration may help
- Self-Hosting Example:
  - ELK
- Hosted Examples:
  - cloud-provider
  - papertrail
  - logz.io
  - logdna
  - loggly

IT Best Practices now with Containers - Monitoring

- Host stats
- Tools
  - Prometheus
  - InfluxDB
  - Heapster
  - ELK
  - Sensu
  - Grafana
- Container stats
  - docker engine api stats
  - cadvisor
- Third Party Services, e.g.
  - sysdig cloud
  - datadog
  - pingdom

Integrate with current tooling

- Network Monitors
- Intrusion Detection

Types of Threats

- escape
- cross-container-attack
- application vulns
- host attack
- host DoS
- orchestration layer, cloud, etc…

Linux (Root) Capabilities or CAP_*

- splits “root” up into granular privileges
- docker containers have root capabilities dropped by default
- sudo /sbin/capsh --print

CAP_*

A subset of docker default CAPs:

- CAP_CHOWN – change file gid/uid (ownership)
- CAP_NET_BIND_SERVICE – bind to privileged ports <1024
- CAP_SYS_CHROOT – chroot = change root path
- CAP_MKNOD – create device file
- …

for more information: man 7 capabilities

Seccomp = Secure Computing Model

By default docker uses a seccomp profile to block system calls unless you override it with the --security-opt option.

Some overlap with CAP_* functionality

Linux Security Modules (LSM)

LSM is a kernel framework using “hooks” when sensitive kernel calls are made.

SELinux: Complex and flexible. Hard to use but powerful.

- Modes: Enforce, Permissive, Disabled
- Based on labels (abstraction of many things: processes, files, resources)

AppArmor: Simpler, possibly as secure? **On by default in docker.**

- Modes: Enforce, Complain
- Based on file paths

GRSEC, PAX = kernel patches

You can run a kernel with GRSEC and PAX. This adds many safety checks, both at compile-time and run-time; it also defeats many exploits, thanks to techniques like address randomization. It doesn't require Docker-specific configuration, since those security features apply system-wide, independent of containers

Security Scanners and Platforms

- OSS
  - OpenSCAP
  - CoreOS Clair
  - Anchore
- Commercial
  - Anchore (commercial support)
  - Blackduck
  - IBM Vulnerability Advisor
  - Redhat Atomic Host
  - CoreOS Quay
  - Docker Cloud and Hub
  - Aqua
  - Tenable
  - Twistlock

Don’t Run as “root”

Most secure to least secure

1. Run as unprivileged USER (or gosu)
2. docker run --cap-drop=ALL --cap-add=<as needed>
3. docker run …
4. docker run --cap-add=ALL
5. docker run --privileged

Plan for an Attack

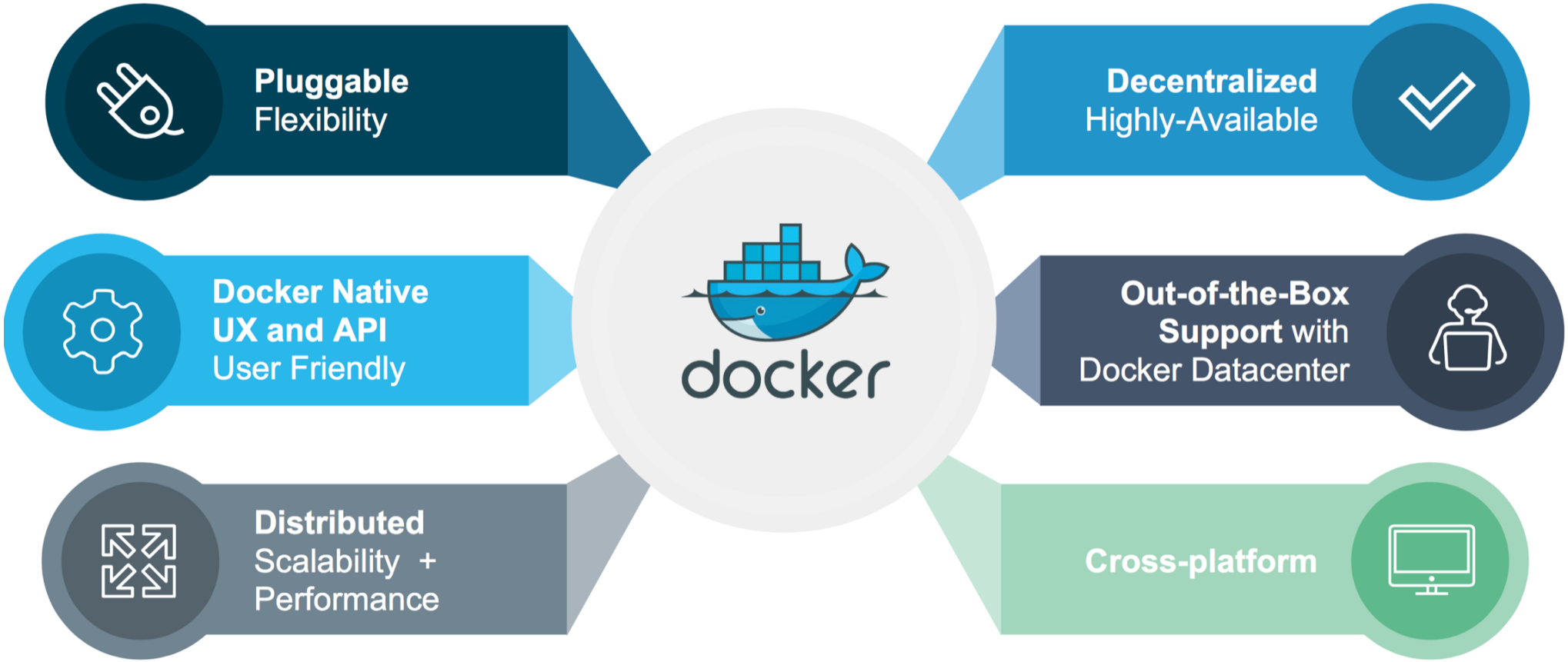
How would you investigate and respond to an attack?

- Logs
- Monitors
- Snapshots
- Data backups and recovery

Lab 4 - Security

Docker Networking

Docker Networking Goals

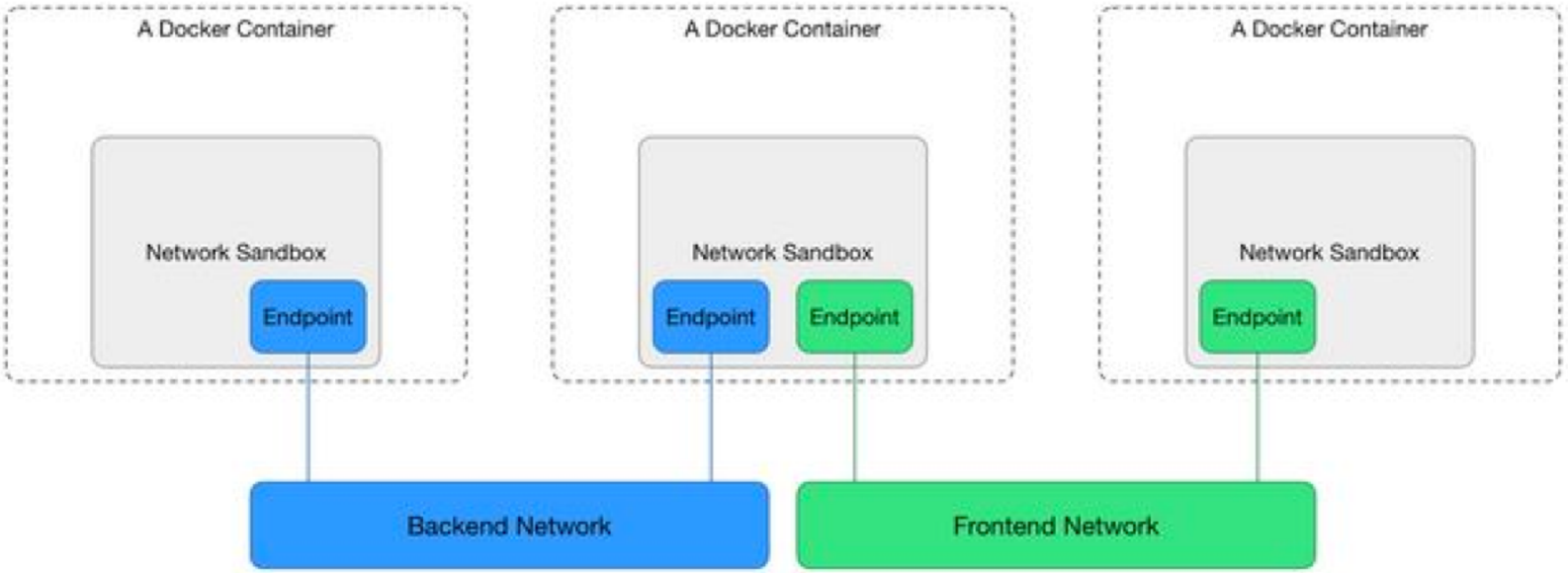


Container Network Model (CNM)

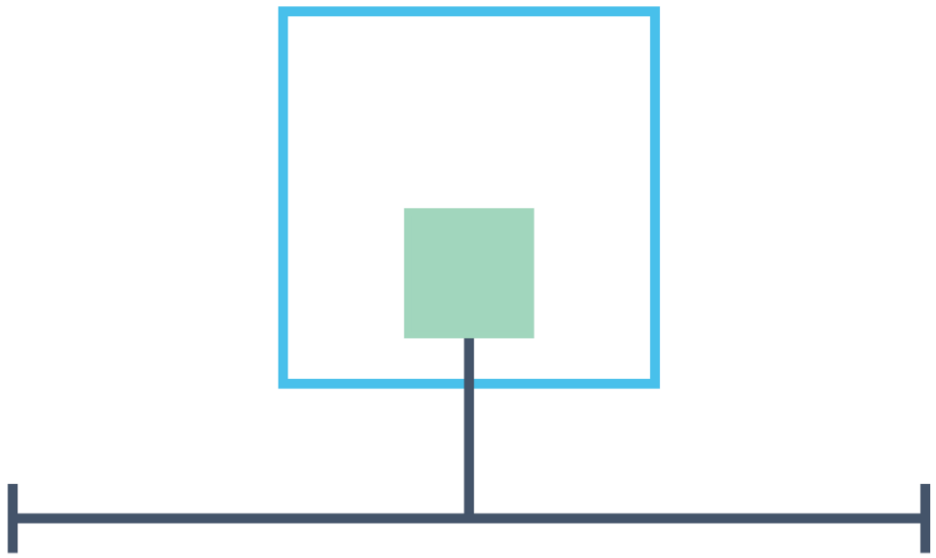
Open source network specification

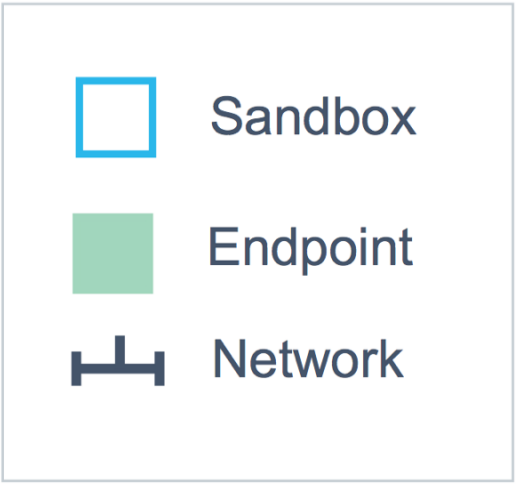
3 primary components:

- Sandbox
- Endpoint
- Network



Container Network Model (CNM)

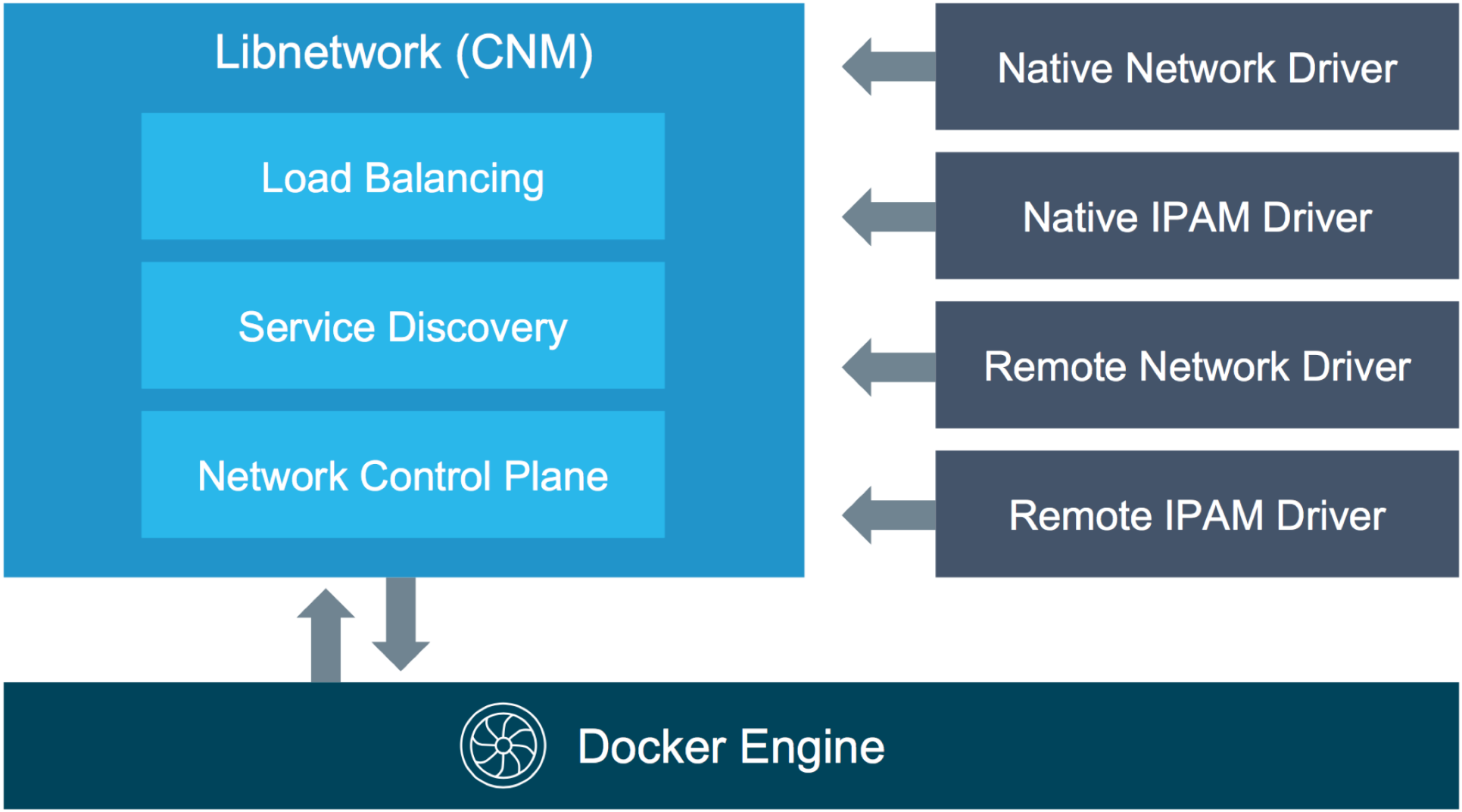




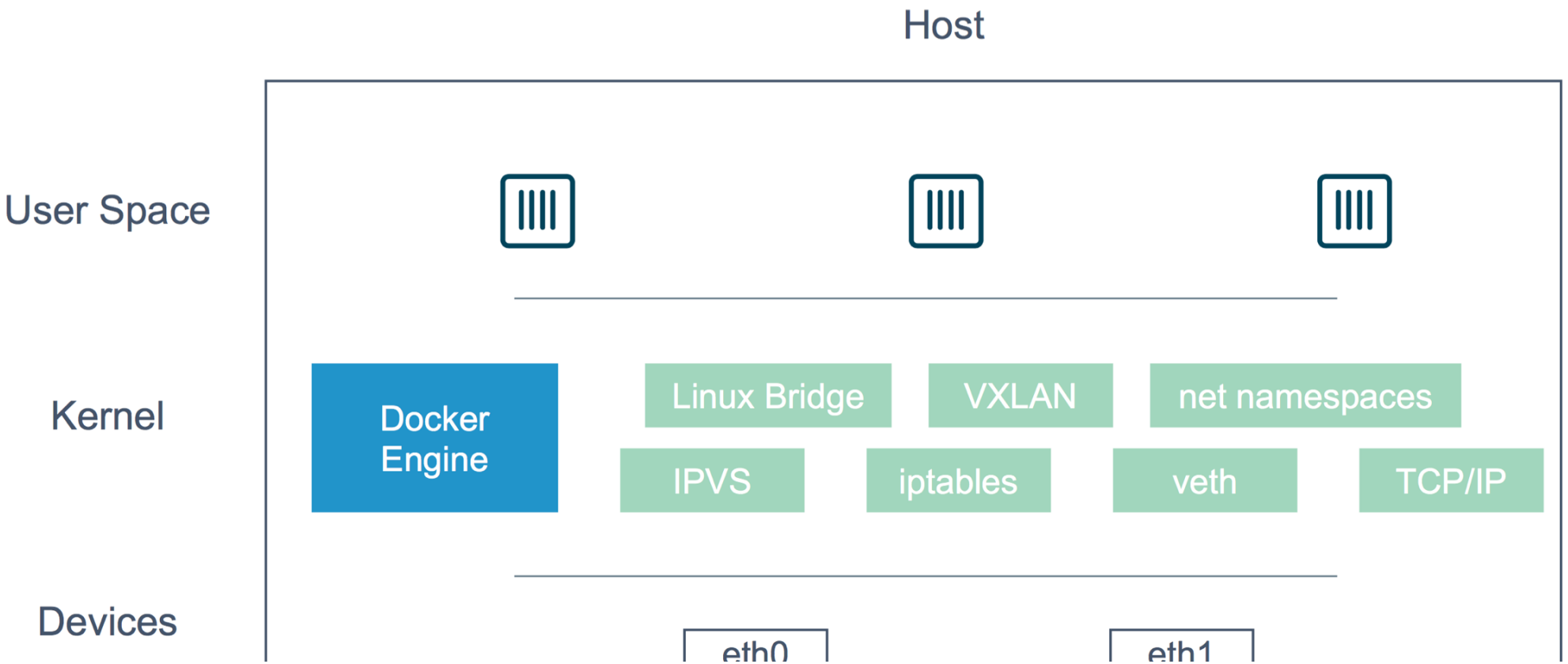
libnetwork

extensible via pluggable drivers
drivers allow support for many network technologies

Libnetwork implements Container Network Model (CNM) which formalizes the steps required to provide networking for containers while providing an abstraction that can be used to support multiple network drivers


Libnetwork (CNM)
Load Balancing
Service Discovery
Network Control Plane
Native Network Driver
Native IPAM Driver
Remote Network Driver
Remote IPAM Driver
Docker Engine

Docker Networking is Linux/Windows Networking



- native driver implements overlay and bridge
- libnetwork handles allocation of IP addresses
- remote plugins can also handle allocation of IP addresses

network drivers

Local/built-in drivers

- bridge
- host
- macvlan
- null
- overlay

Remote/3rd party drivers

- Calico
- Contiv
- Kuryr
- Weave

Networking on Linux

- Uses linux networking capabilities (tcp/ip, vxlan, dns, …)
- networking features (net namespaces, bridges, iptables, veth pairs, …)
- linux bridge: L2 virtual switch implemented in the kernel
- network namespace: isolating the network stack
- veth pairs: Connect containers to container networks
- iptables: used for port mapping, load balancing, network isolation, …

Networking on Windows

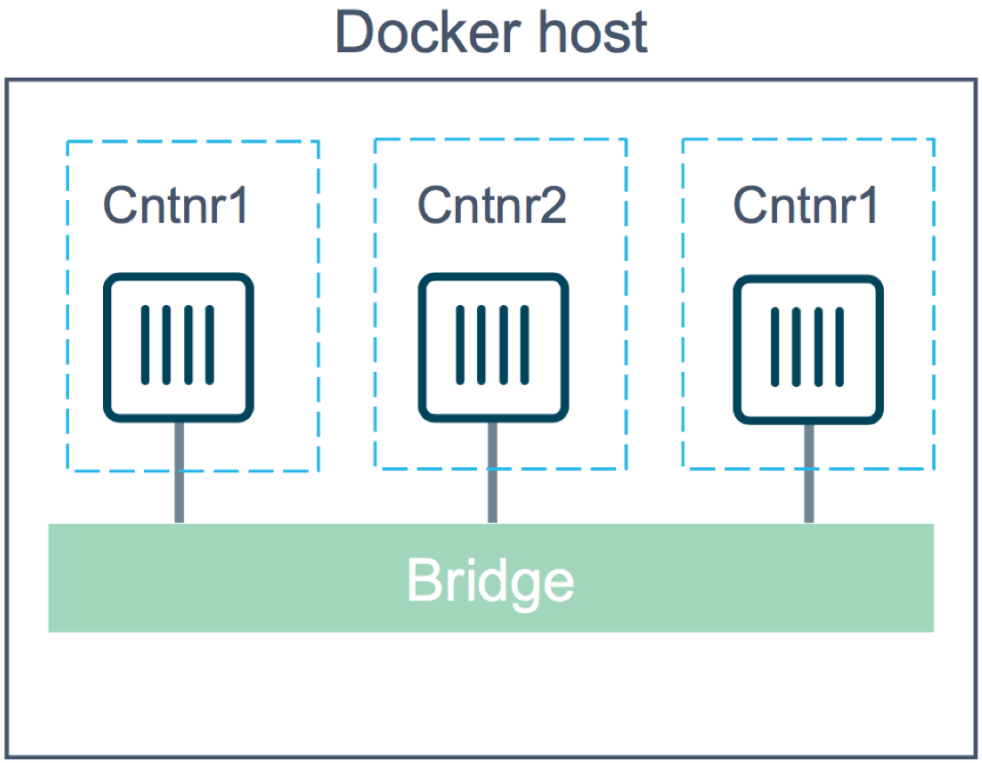
- Network Compartments (like linux namespaces)
- Vswitch (like linux bridge)
- Virtual nics (like linux virtual ethernet, veth)
- Firewall & VFP Rules (like linux IPTables)

Host Driver

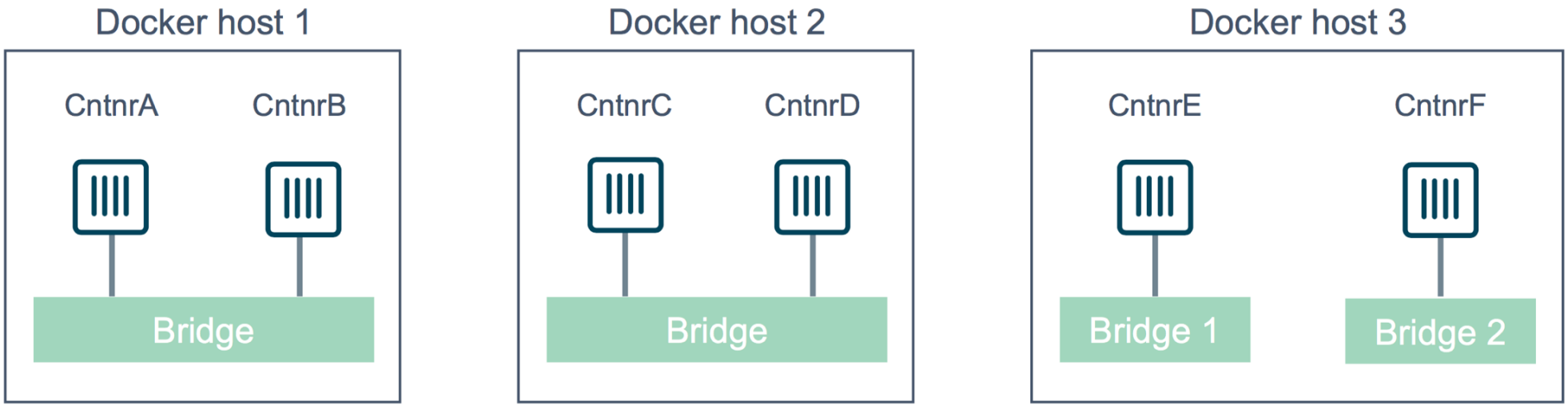
- Just use the docker host’s network namespace, IP, etc.

Bridge Driver

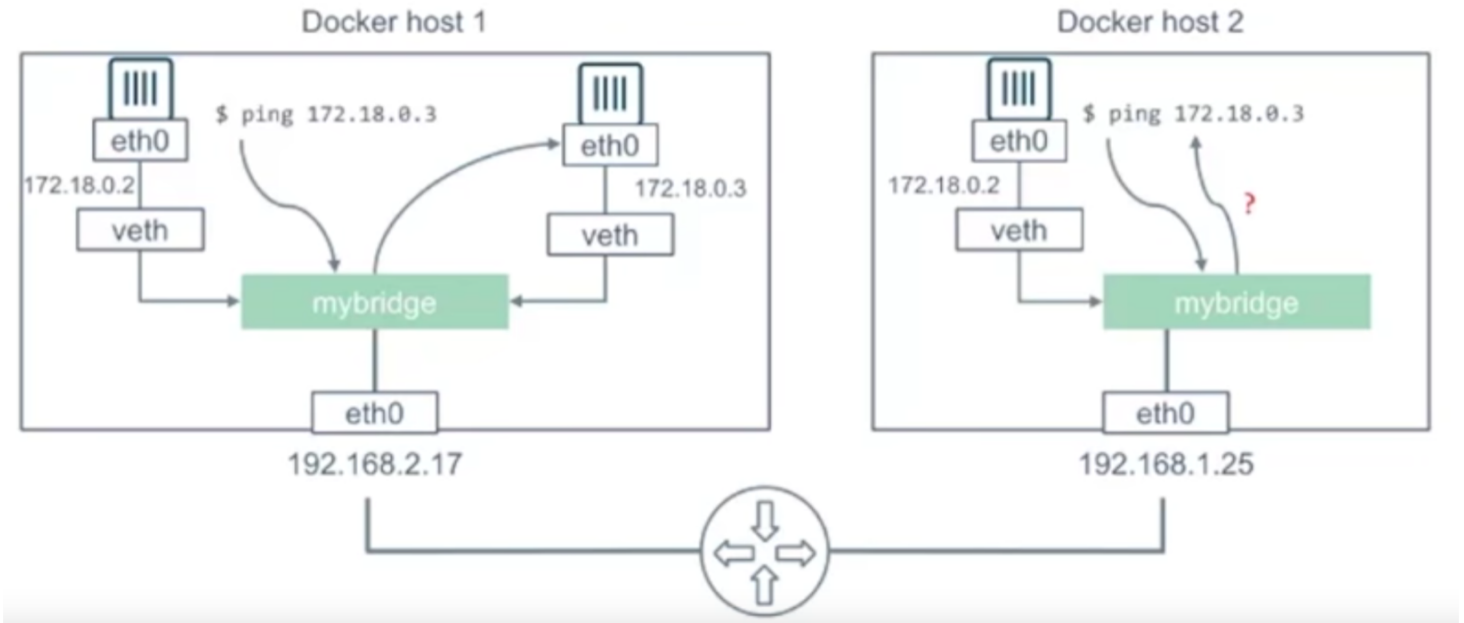
- Each container in own namespace
- bridge driver creates a bridge (virtual switch) on docker host
- All containers on bridge can communicate
- The bridge is a private network restricted to a single Docker host



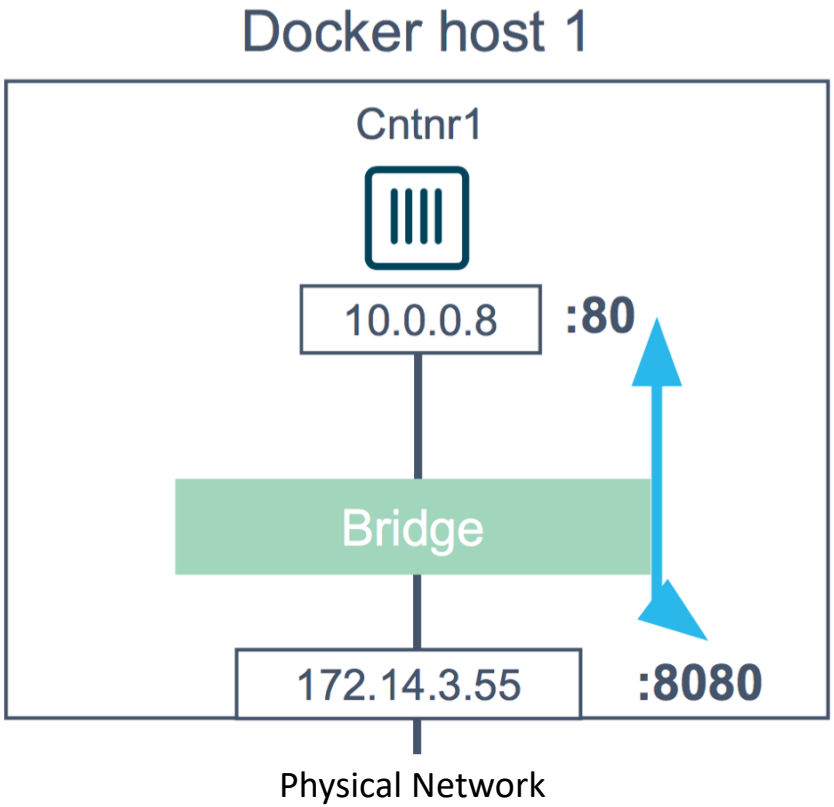
Containers on separate bridges cannot communicate



Bridge Networks Don’t Span Hosts



Port Mapping



Overlay Driver

- Docker Engine can create overlay networks on a single host. Docker Swarm can create overlay networks that span hosts in the cluster
- A container can be assigned an IP on an overlay network. Containers that use the same overlay network can communicate even if they are running on different hosts
- By default, nodes in the swarm encrypt traffic between themselves and other nodes. Connections between nodes are automatically secured through TLS authentication with certificates

Overlay

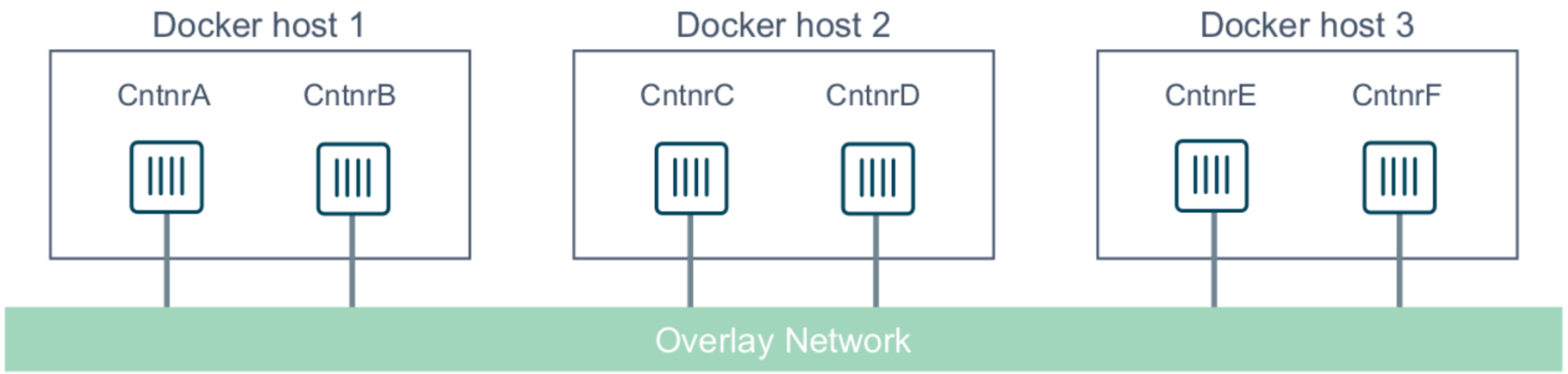
Pros:

- Portable: works in any cloud/on-prem network with little or no reconfiguration
- Docker handles control plane configuration and management.
- Control plane encrypted by default
- Data plane can be optionally encrypted

Cons:

- Performance may suffer from additional overhead

Overlay Driver

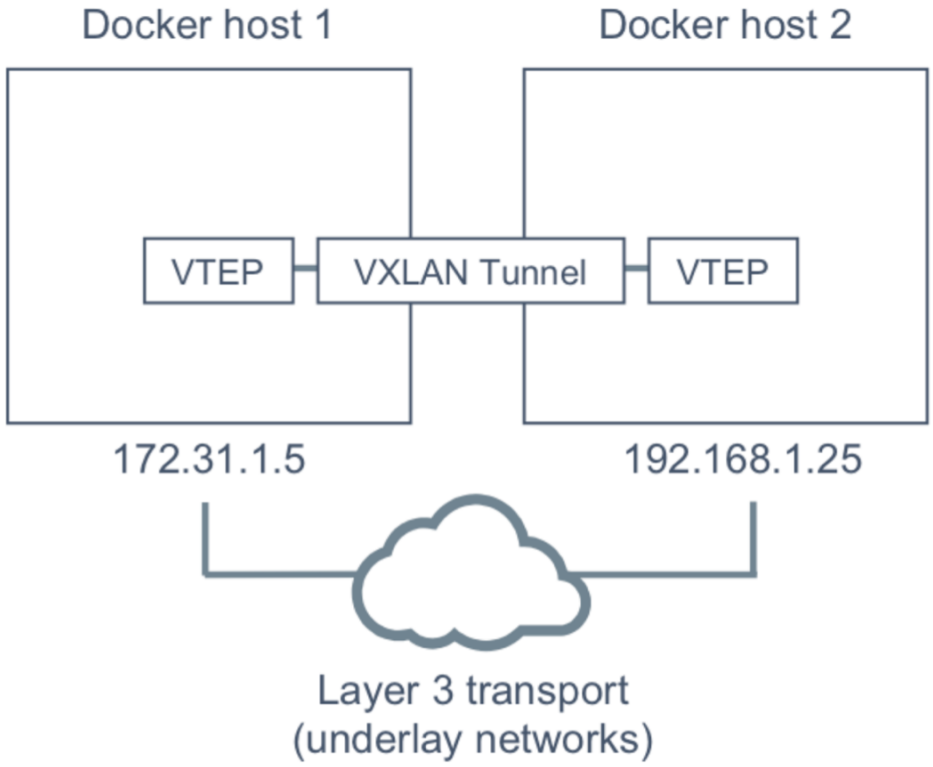


Overlay Details

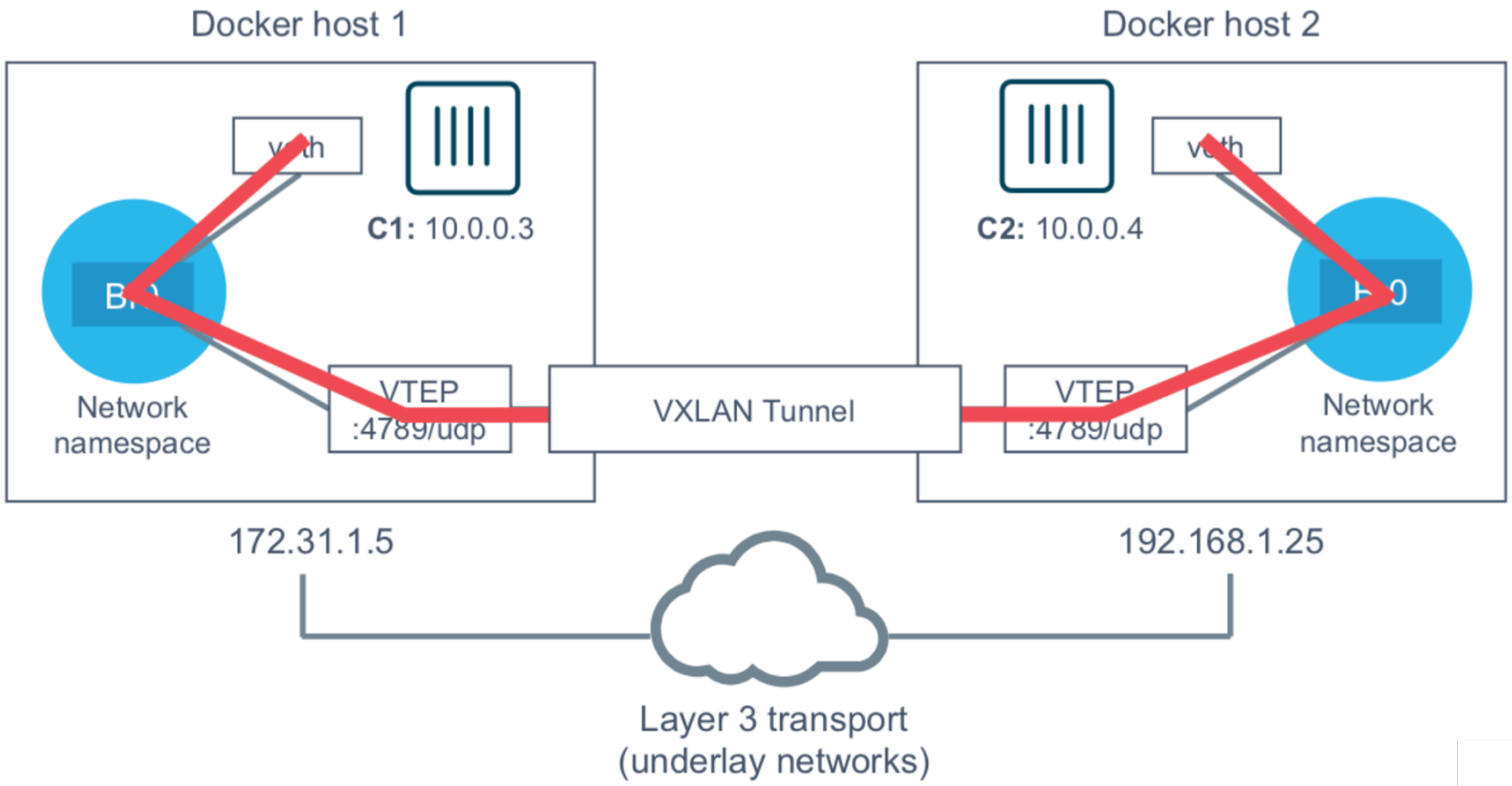
- Creates a new L2 network over an L3 transport network
- Leverages the distributed KV store created by Swarm

Overlay and VXLAN

- Uses VXLAN technology to build the network
- A VXLAN tunnel is created through the underlay network(s)
- At each end of the tunnel is a VXLAN tunnel end point (VTEP)
- The VTEP performs encapsulation and de-encapsulation
- The VTEP exists in the Docker Host's network namespace

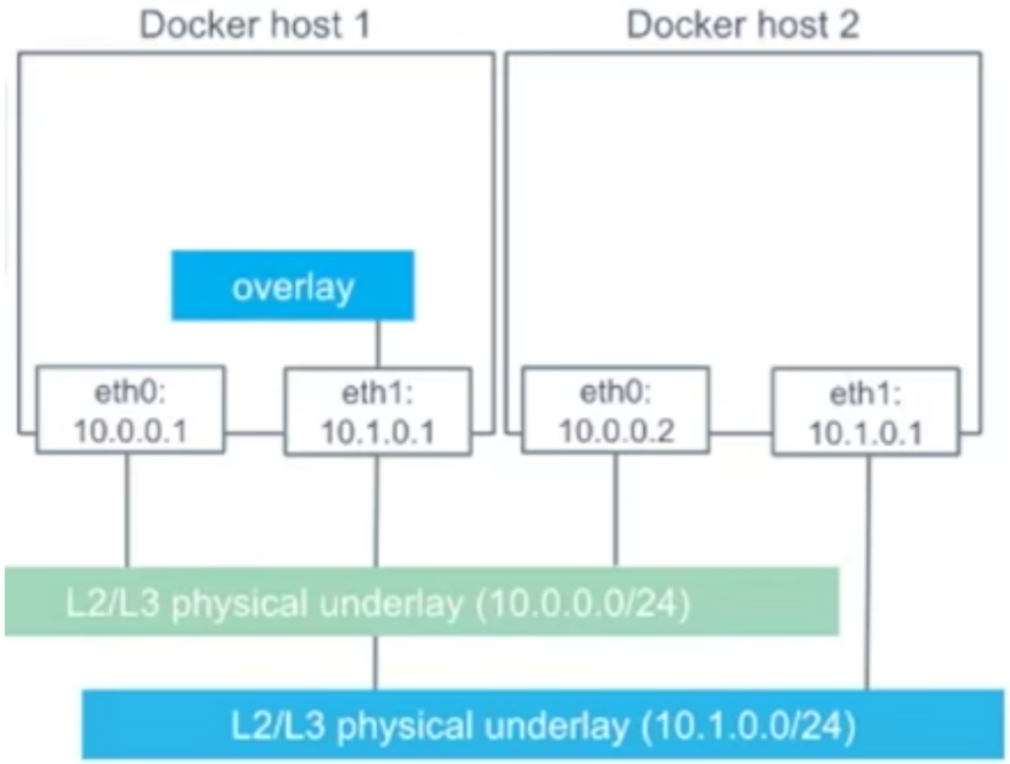


Overlay driver details



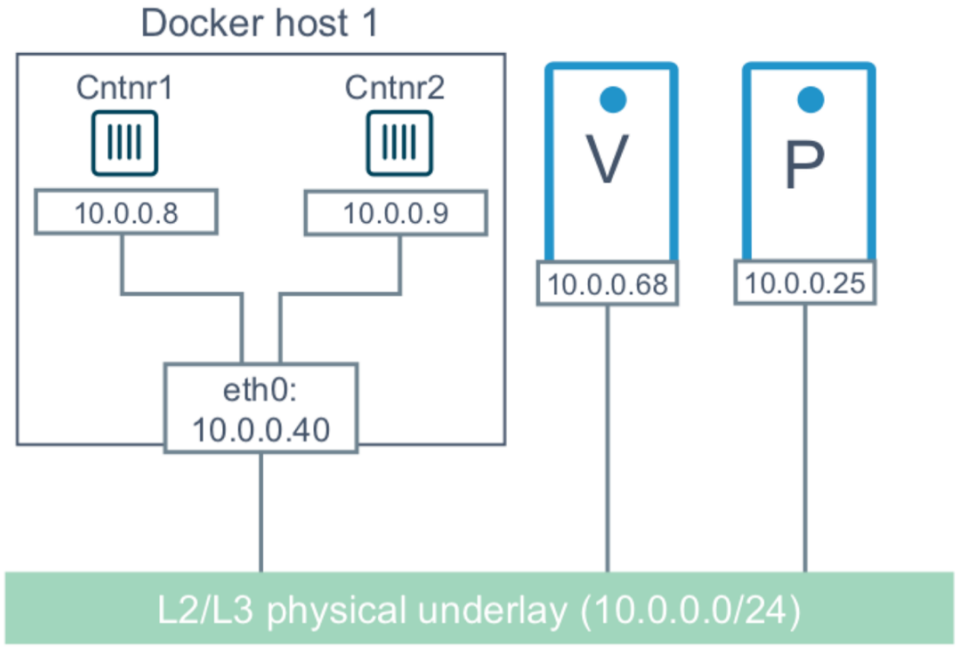
Separate control/data networks

If you have 2 physical interfaces, you can configure swarm to use one for data and one for control.



MACVLAN Driver

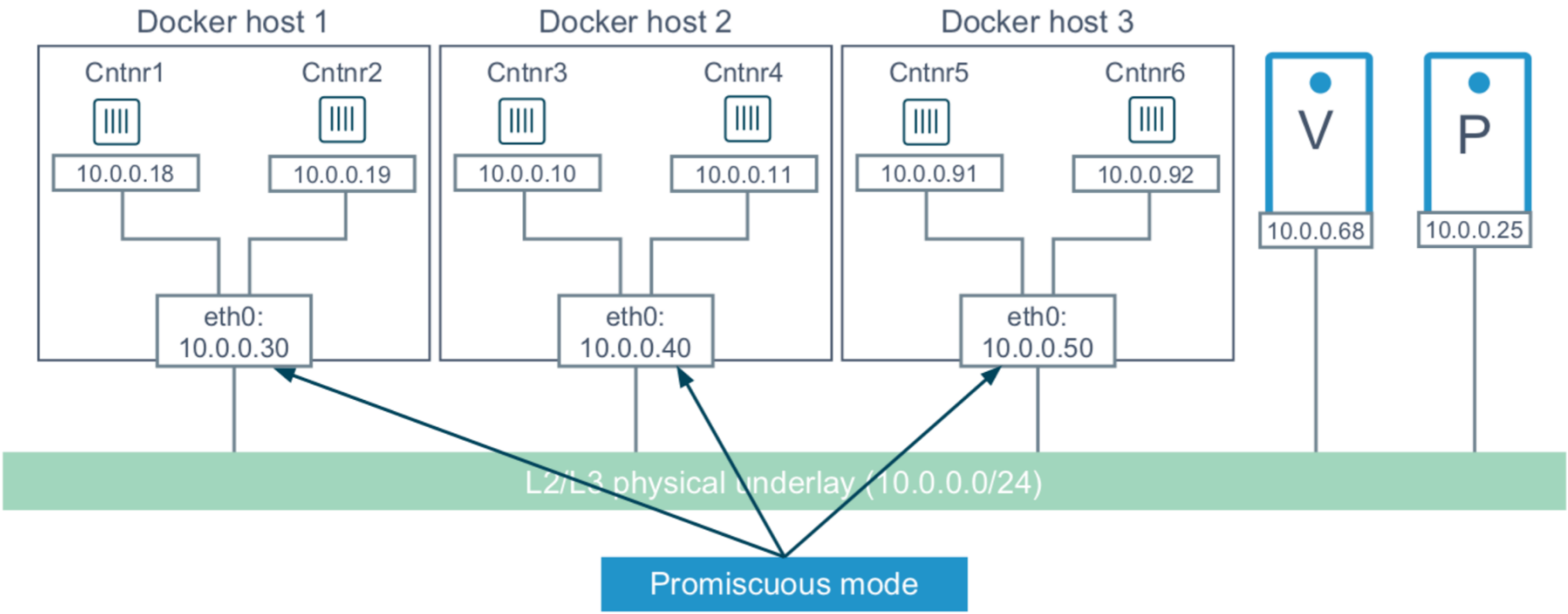
- A way to attach containers to virtual and physical machines on existing networks and VLANs
- Good for apps that are not ready to be fully containerized



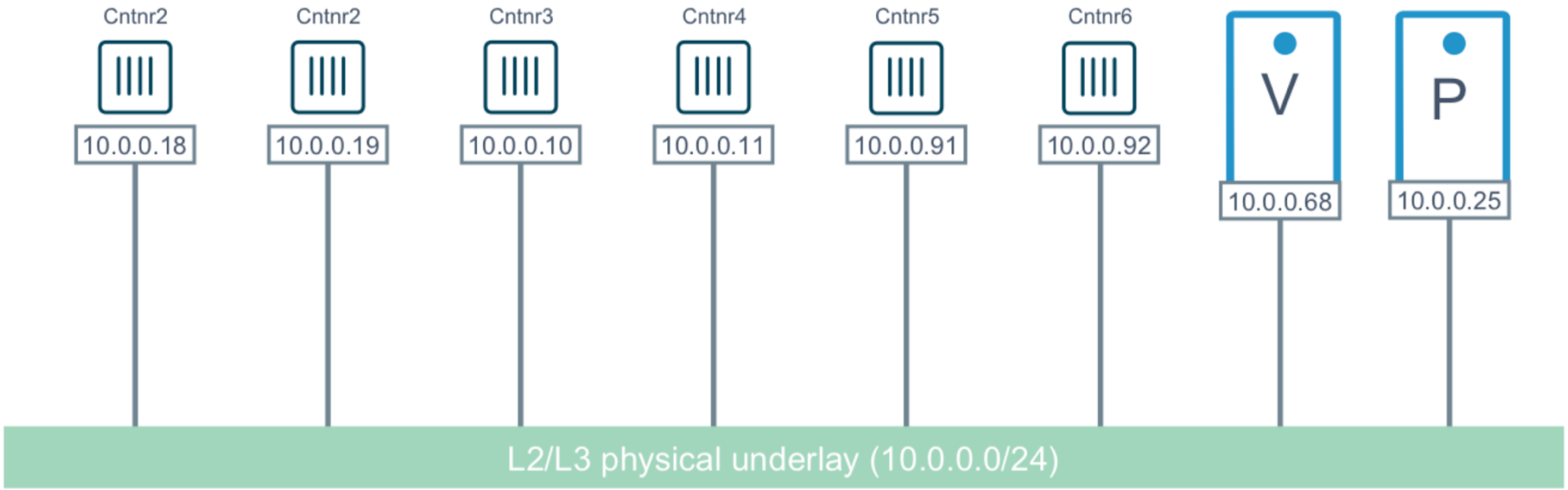
MACVLAN Driver Details

- Parent interfaces have to be connected to physical underlay
- Each container gets its own MAC and routable IP on the underlay
- Each container is visible on the physical underlay network
- Gives containers direct access to the underlay network without port mapping and without a Linux bridge
- Requires promiscuous mode
- High Performance (no NAT, no bridge, no encapsulation)

MACVLAN


Docker host 1
Docker host 2
Docker host 3
Cntnr1
Cntnr2
Cntnr3
Cntnr4
Cntnr5
Cntnr6
10.0.0.18
10.0.0.19
10.0.0.10
10.0.0.11
10.0.0.91
10.0.0.92
eth0:
10.0.0.30
eth0:
10.0.0.40
eth0:
10.0.0.50
V
P
10.0.0.68
10.0.0.25
L2/L3 physical underlay (10.0.0.0/24)
Promiscuous mode

MACVLAN


Cntnr2
10.0.0.18
Cntnr2
10.0.0.19
Cntnr3
10.0.0.10
Cntnr4
10.0.0.11
Cntnr5
10.0.0.91
Cntnr6
10.0.0.92
V
10.0.0.68
P
10.0.0.25
L2/L3 physical underlay (10.0.0.0/24)

Competing Network Models

Container Networking Model (CNM)

- Proposed by Docker, adopted by libnetwork
- Supports only docker runtime

Container Networking Interface (CNI)

- Proposed by CoreOS, adopted by Kubernetes, Cloud Foundry, and Apache Mesos, AWS ECS
- Supports any container (oci) runtime

Competing Network Models

CNM and CNI commonalities:

- both driver based
- multiple network drivers can be in use concurrently
- containers may join one or more networks
- runtime can launch network in its own namespace

Lab 5 - Networking

Docker APIs, Events, States, Plugins

Miscellaneous collection of things API related

Docker Engine API

RESTful API

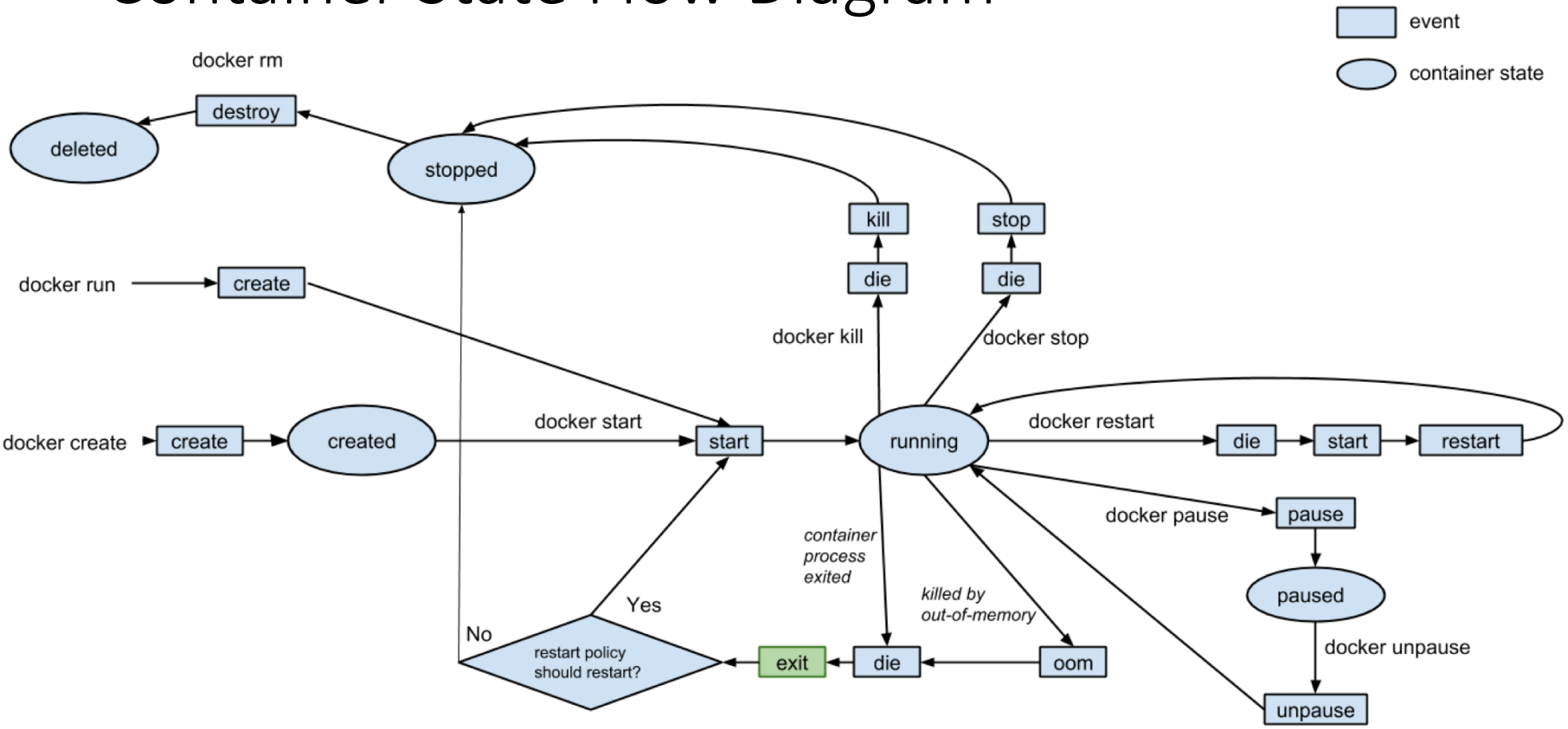
Official SDKs:

- Go
- Python

Docker Events Object Types and Events

- Containers
  - attach
  - commit
  - copy
  - create
  - destroy
  - detach
  - die
  - exec_create
  - …
- Images
  - delete
  - import
  - load
  - pull
  - push
  - save
  - tag
  - untag
- Plugins
  - install
  - enable
  - disable
  - remove
- Volumes
  - create
  - mount
  - unmount
  - destroy
- Networks
  - create
  - connect
  - disconnect
  - destroy
- Daemons
  - reload

Container State Flow Diagram



Plugins

- Docker plugins are out-of-process extensions which add capabilities to the Docker Engine.
- Plugins can run inside or outside containers. Currently running them outside containers is recommended.
- Docker discovers plugins by looking for them in the plugin directory whenever a user or container tries to use one by name.
- Currently Docker supports authorization, volume and network driver plugins

A Tale of 2 Logs

How should we collect logs? Use docker containers log api or log files?

Log files are simpler, no plugins required

Log files are a natural buffer

I've had experiences where docker daemon hiccups caused container log api data loss (but the log files were still written to correctly)

With logs you still have to rotate log files

Lab 6 – Docker API

Orchestration

Container Problems, Node Count > 1

- Volume management
- Container scheduling
- Service discovery
- Secret management
- Host redundancy
- Health monitoring
- Traffic routing
- Cluster state management

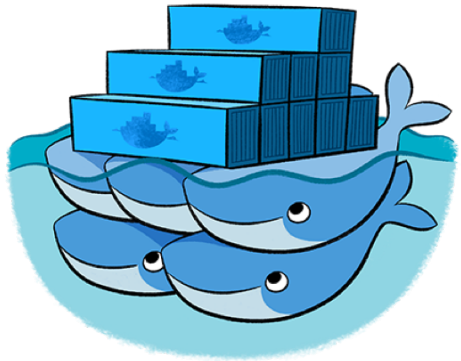
Docker Cloud

2 modes: legacy and swarm

+ Meant to supercede docker-machine for cloud providers
+ Centralized way to manage simple docker hosts or docker swarms
+ CI/CD – link to github, bitbucket
+ Supports teams/orgs

- Not cloud feature-rich; metrics, logs, auto-scaling, etc. handled elsewhere

Docker Swarm



+ built into docker
+ feature rich

- built into docker (poor separation of concerns)
- Learning Curve for developers
- Self-hosting is operationally complex

Apache Mesos

+ not container specific, good for legacy workloads
+ feature rich
+ hosted or self-hosted options

- complicated for operators
- Learning Curve for developers

Kubernetes

kubernetes

+ Opinionated based on years of experience
+ Flexible
+ Scalable (5k nodes) https://kubernetes.io/docs/admin/cluster-large/
+ Feature-rich
+ Abstracts away cloud provider management
+ Role based permissions
+ Hosted, self-hosted options

- Opinionated
- Learning Curve for developers
- Self-hosting is operationally complex

Lab 7 - Orchestration

Thank You VMWare!