THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Container and Kubernetes Networking 101

THE LINUX FOUNDATION

Before we begin

Poll

Arun Sriraman
(Software Engineer)
Platform9 Systems Inc
@arun_sriraman sarun87

Aditya Amar
Sr. Principal Software Engineer
@adityaCal

Agenda

Part I - Container Networking 101

- Need for container networking
- Linux networking constructs
  - Bridge drivers
  - Network Namespace
- Intro to docker networking - the CNM model
- Docker networking drivers and its comparison

Part II - Kubernetes Networking 101

- K8s networking fundamentals
- Kubernetes communication
  - Container-to-Container
  - Pod-to-Pod
  - Pod-to-Service
  - Service-to-external
- Container Network Interface
- CNI backend (Flannel, Calico)

Part I - Container Networking

The Need for Container Networking

Containers need to talk to:

- outside world and vice-versa
- the host machine (maybe)
- other containers running within and across hosts

We also need to be able to:

- load balance traffic between containers
- provide multi-tenancy
- automatically discover services provided by other containers

This sounds very similar to VMs and VM networking….

What’s different

Virtual Machines	Containers
Separate networking stack	Network namespaces used to achieve network isolation
Multiple services run inside a single VM; the VM gets an IP - services may or may not be addressed explicitly.	Service (typically) gets a separate IP; Service (typically) maps to multiple containers. With Kubernetes, services have their own IP
Service Discovery and Load balancing (typically) done outside the VM	Microservices implemented using Containers leads to more integrated Service Discovery
Scaling needs are not that high	Scaling needs at least an order of magnitude higher

Linux networking constructs

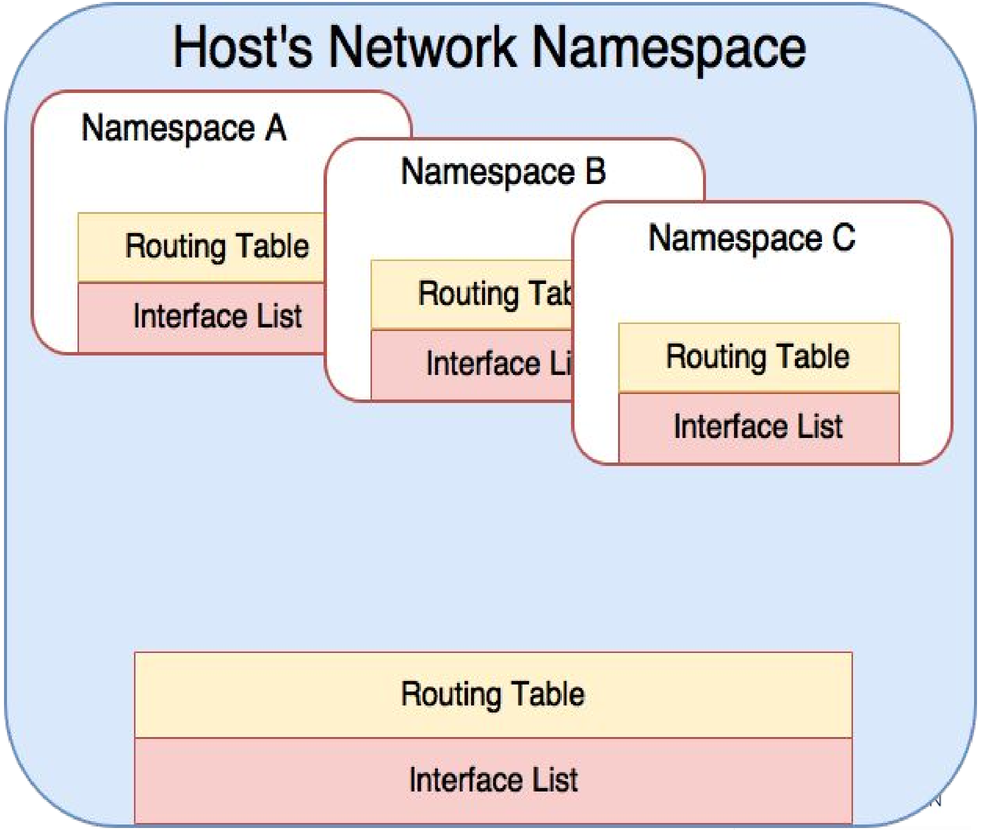
- The Linux Bridge device
- Network Namespaces
- Virtual Ethernet Devices
- iptables

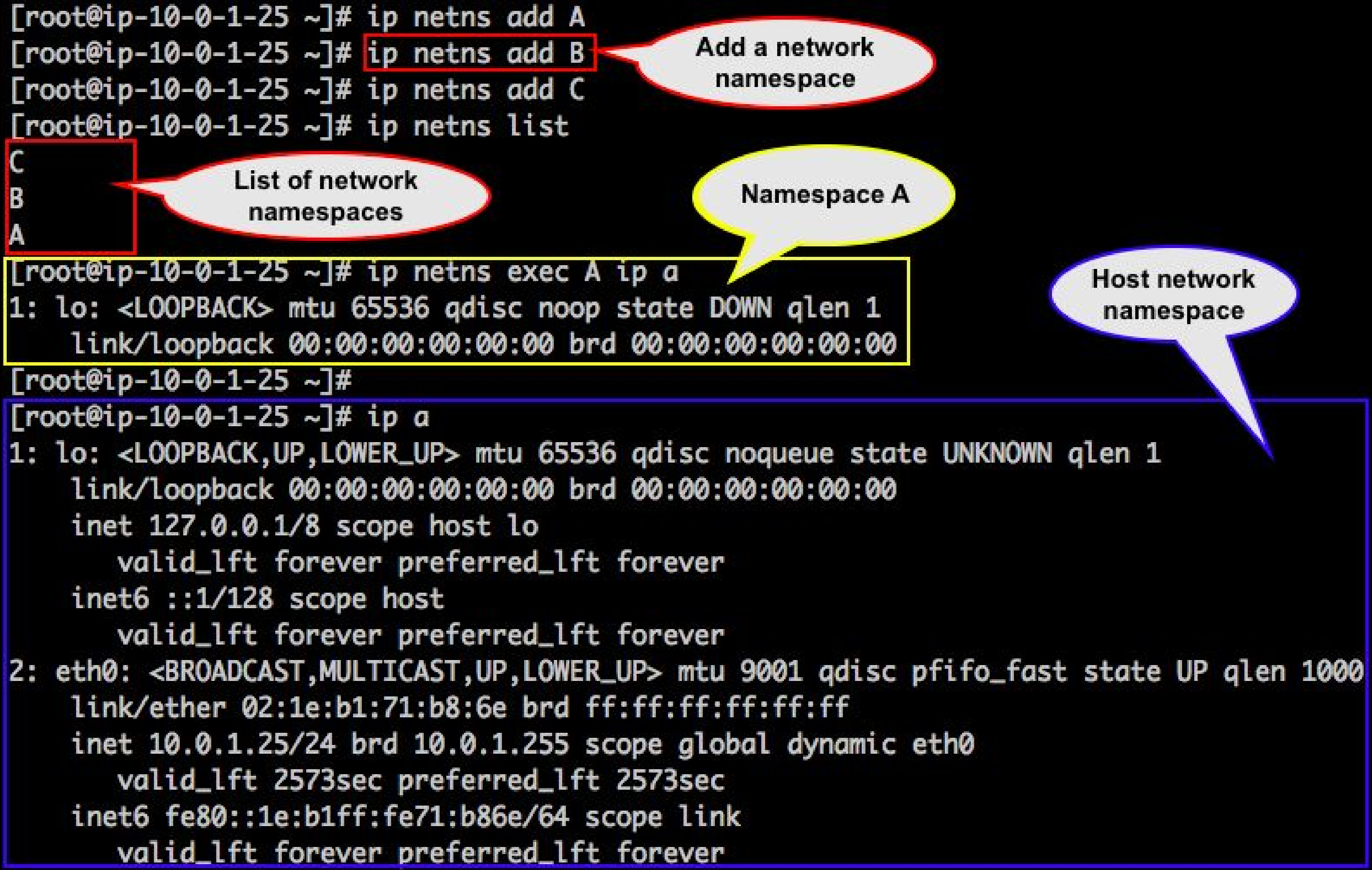
Network namespaces

Process started with a new network namespace gets its own private network stack with

- network interfaces (including lo)
- routing tables
- iptables rules
- sockets (ss, netstat)

```
flags = CLONE_NEWPID|
    CLONE_NEWNS|CLONE_NEWNET;
cpid = clone(child_function,
    childstack,
    flags, (void *)argv);
```





```
[root@ip-10-0-1-25 ~]# ip netns add A
[root@ip-10-0-1-25 ~]# ip netns add B
[root@ip-10-0-1-25 ~]# ip netns add C
[root@ip-10-0-1-25 ~]# ip netns list
C
B
A
[root@ip-10-0-1-25 ~]# ip netns exec A ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
[root@ip-10-0-1-25 ~]#
[root@ip-10-0-1-25 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 9001 qdisc pfifo_fast state UP qlen 1000
    link/ether 02:1e:b1:71:b8:6e brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.25/24 brd 10.0.1.255 scope global dynamic eth0
       valid_lft 2573sec preferred_lft 2573sec
    inet6 fe80::1e:b1ff:fe71:b86e/64 scope link
       valid_lft forever preferred_lft forever
```

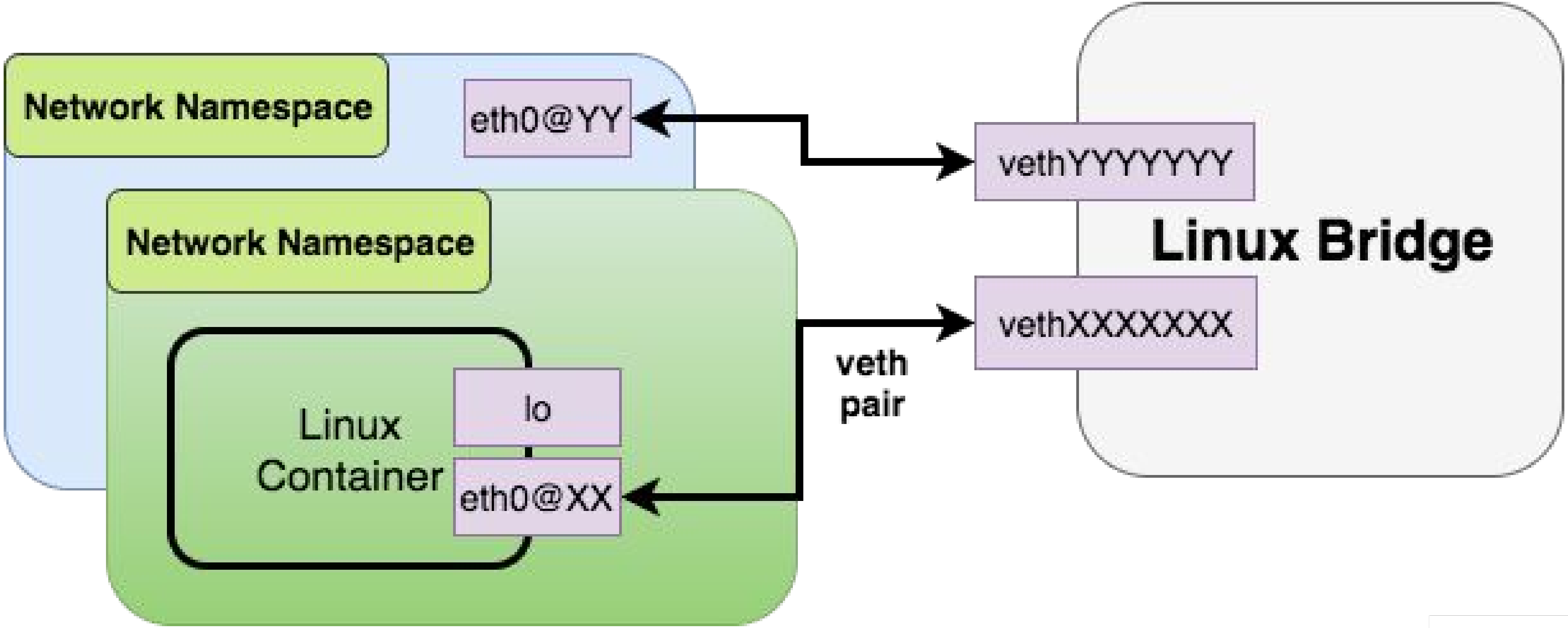
Add a network namespace

List of network namespaces

Namespace A

Host network namespace

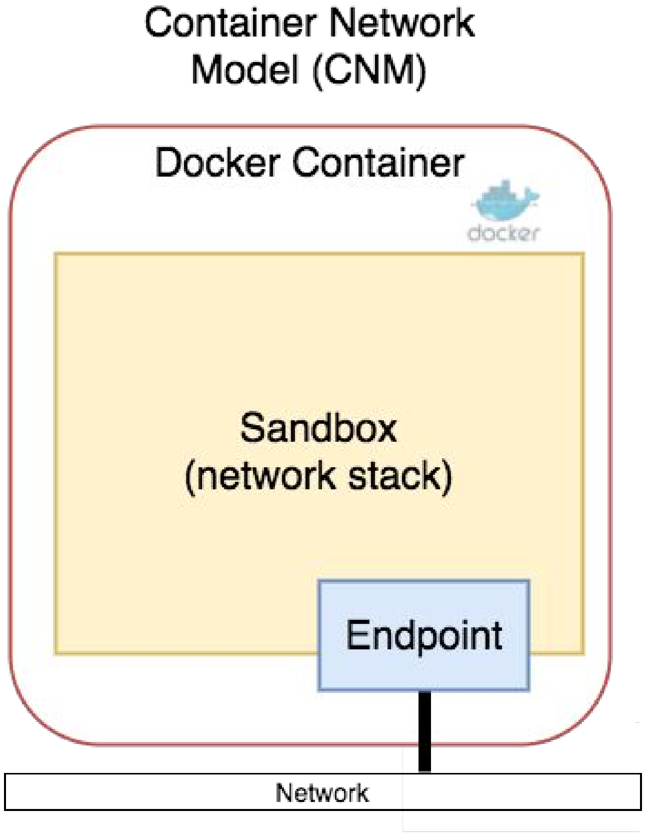
Linux bridge and veth interface



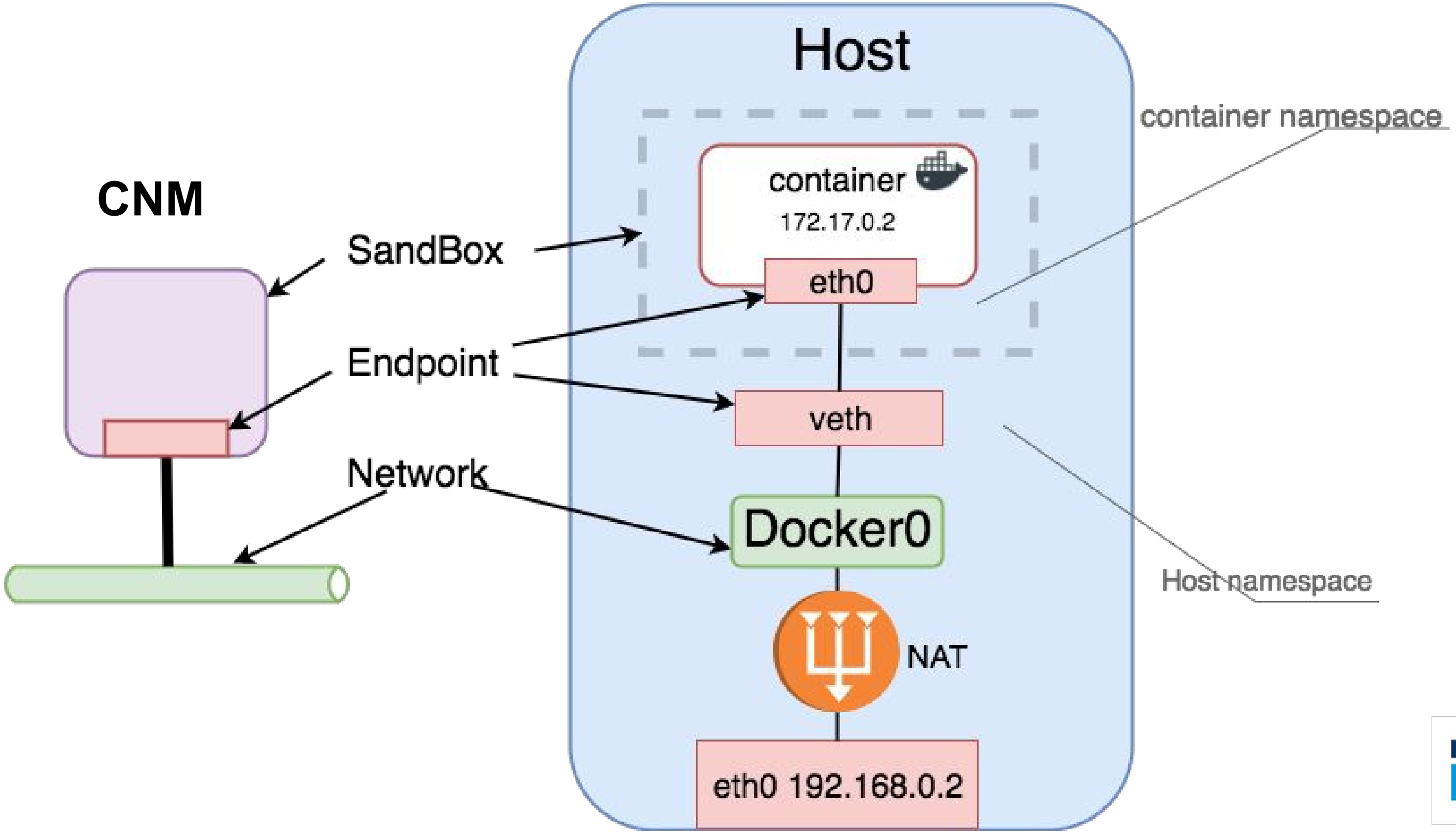
http://blog.arunsriraman.com/2017/01/container-namespaces-deep-dive-into.html

Container Network Model (CNM)

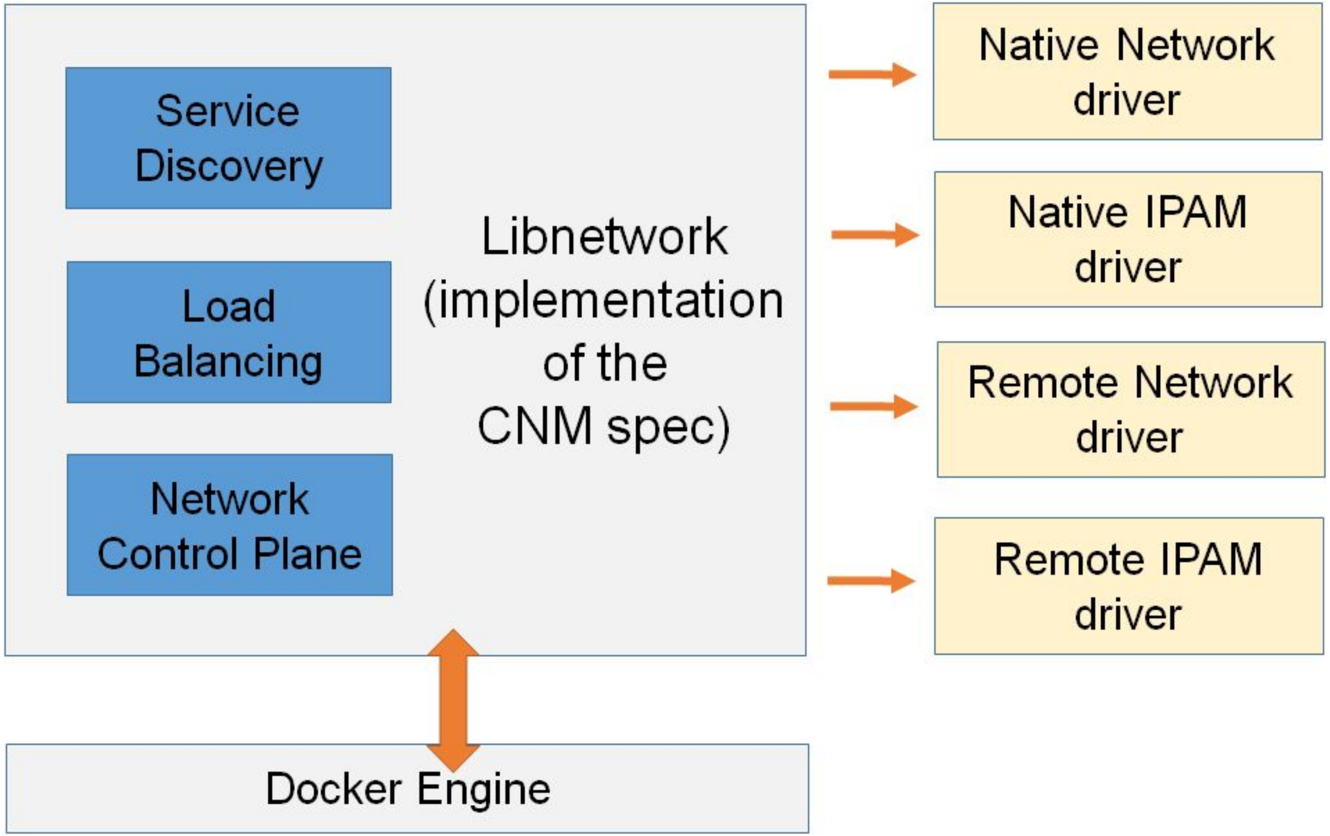
- Project started by Docker
- Separate networking from container runtime as a library
- Components
  - Sandbox
  - Endpoint
  - Network
- Implemented using libnetwork



Mapping CNM to Libnetwork (Docker)



Libnetwork contd.



bridge, host, none, overlay

libnetwork built-in IPAM driver

weave

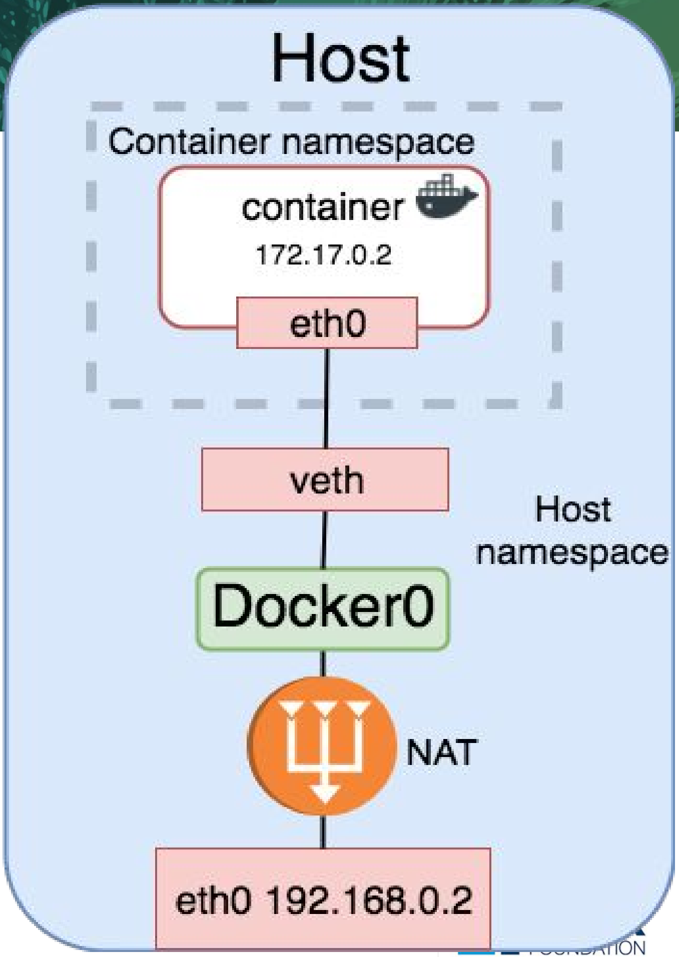
Contiv

Infoblox

Default Bridge Driver

- Responsible for creating the docker0 bridge.
- Connects docker containers to the network using a veth pair
- Provides out-of-the-box support for bridge based container networking
- Allows creation of user-defined bridges

```
docker network create --driver bridge
<name>
```



Host

Container namespace

container

172.17.0.2

eth0

veth

Host namespace

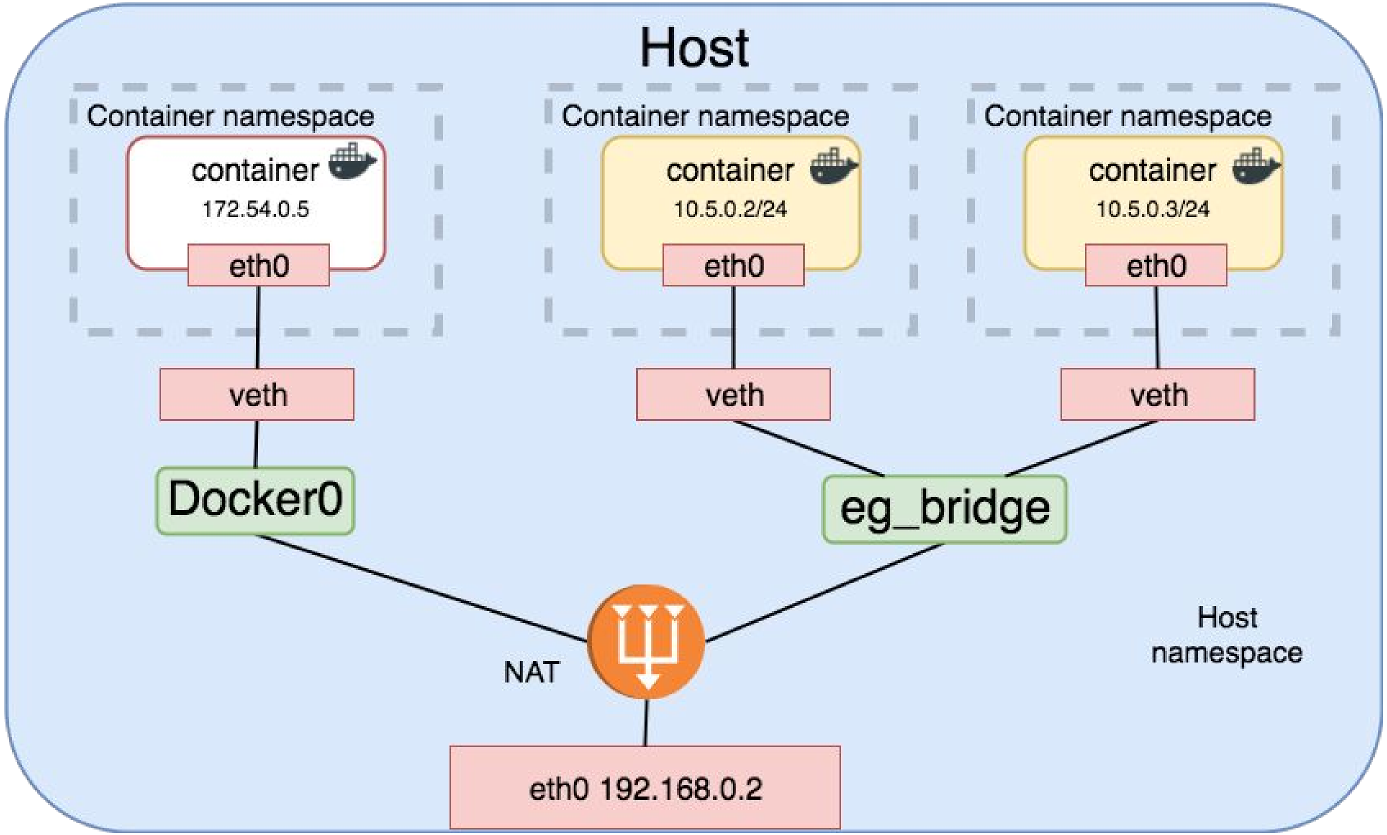
Docker0

NAT

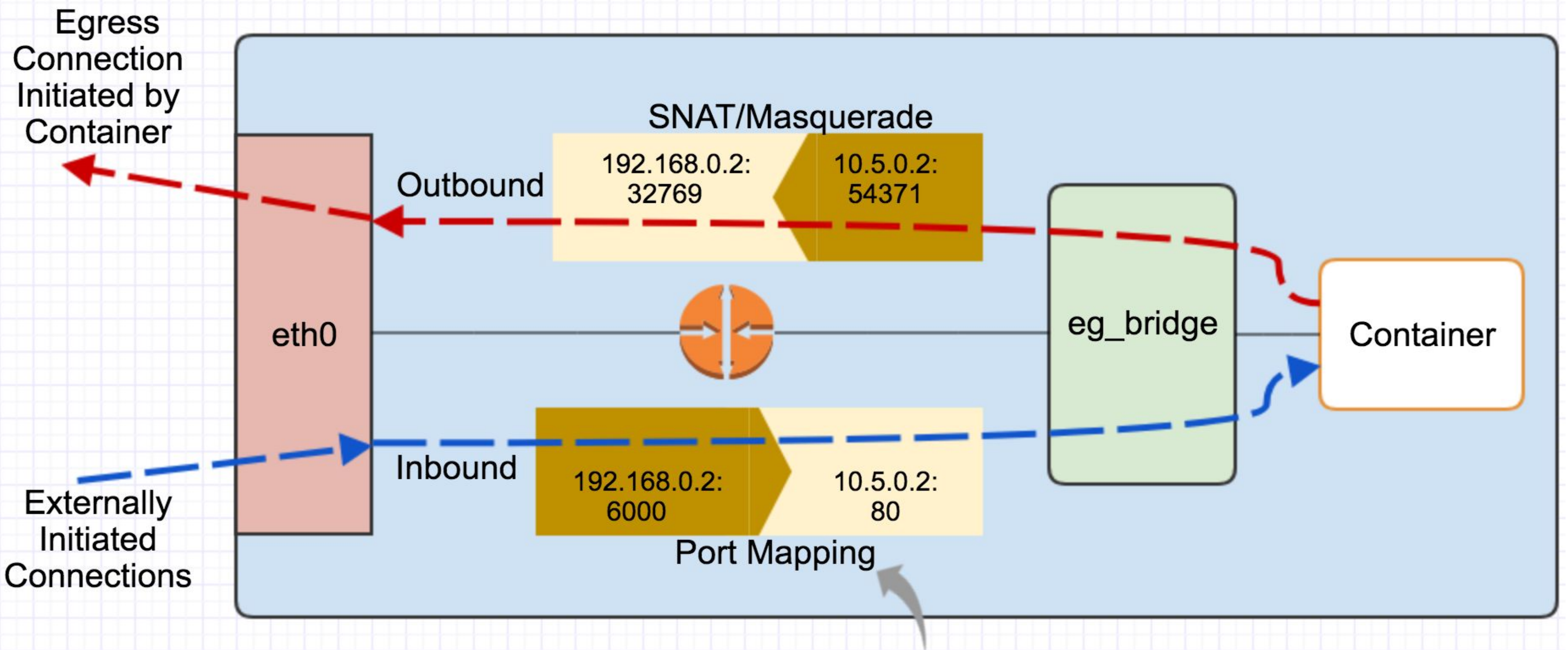
eth0 192.168.0.2

https://github.com/sarun87/examples/blob/master/images/Docker_Bridge_driver.png

User Defined Bridge



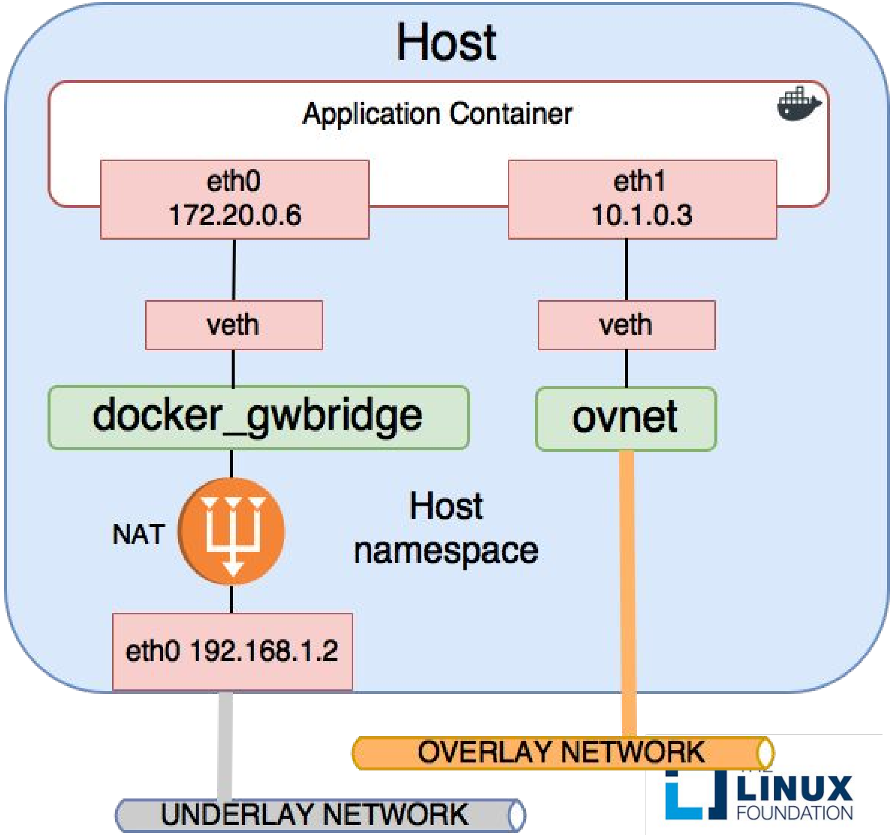
External Access for Containers



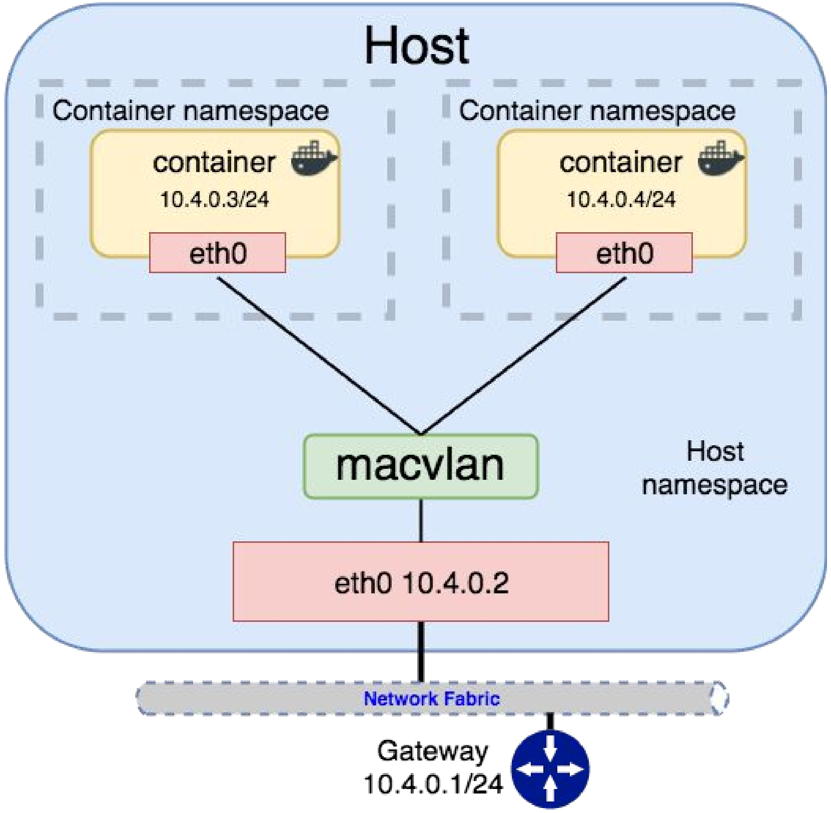
```
$ docker run -d --net eg_bridge -p 6000:80 nginx
```

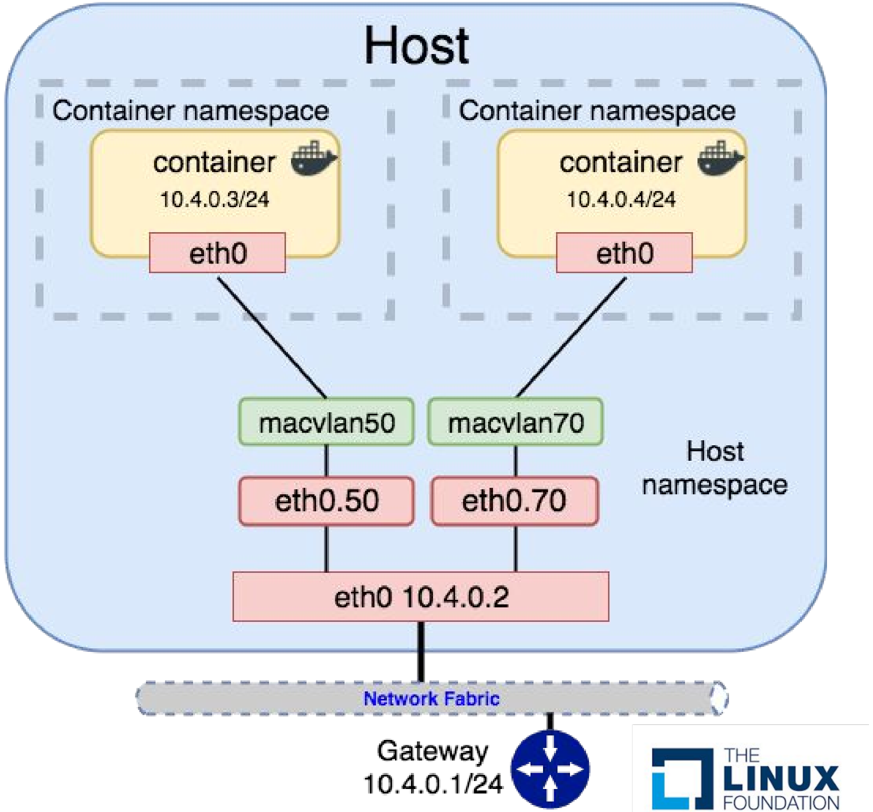
Overlay Driver

- Multi-host networking
- First-class citizen in docker networking
- Uses swarm-distributed control plane for centralized mgmt, stability & security
- Uses VXLAN encap (decouples container n/w from physical n/w)
- Overlay datapath entirely in kernel space



Macvlan Driver





Default Networks Created by Docker

‘bridge’ using **bridge** driver, ‘none’ using **null** driver, ‘host’ using **host** driver

```
arun-neotrekker:~ arunsriraman$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
544fd2b5b674        bridge              bridge              local
790b79d68240        host                host                local
6aaec591a006        none                null                local
```

Don’t want the bridge driver? Remove it by specifying OPTIONS

```
/etc/sysconfig/docker
OPTIONS="--bridge=none --log-driver=json-file"
```

Compare Docker Network driver types

Driver Features	Bridge / User defined bridge	Host	Overlay	Macvlan / ipvlan
Connectivity	Same host	Same host	Multi-host	Multi-host
Namespace	Separate	Same as host	Separate	Separate
External connectivity	NAT	Use Host gateway	No external connectivity	Uses underlay gateway
Encapsulation	No double encap	No double encap	Double encap using Vxlan	No double encap
Application	North, South external access	Need full networking control, isolation not needed	Container connectivity across hosts	Containers needing direct underlay networking

Part II - Kubernetes Networking

Fundamental requirements

All containers can communicate with all other containers without NAT

All nodes can communicate with all containers (and vice-versa) without NAT

The IP that a container sees itself as is the same IP that others see it as

Quoted from K8s docs

Kubernetes networking

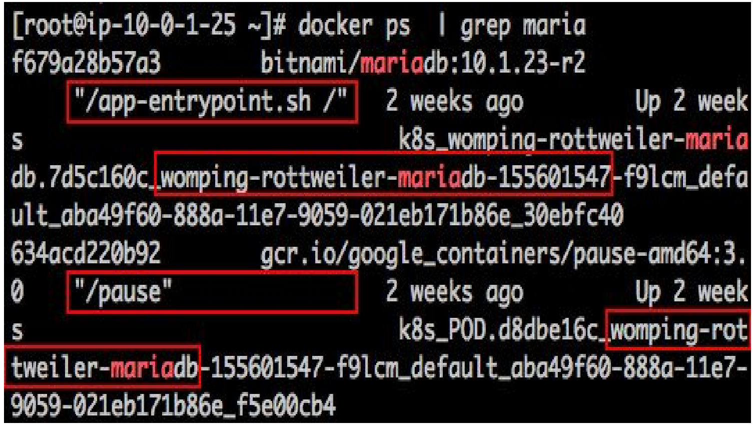
- Container-to-Container communication
- Pod-to-Pod communication
- Pod-to-Service (cluster internal) communication
- External-to-Service (cluster external) communication

Container CIDR
Service CIDR

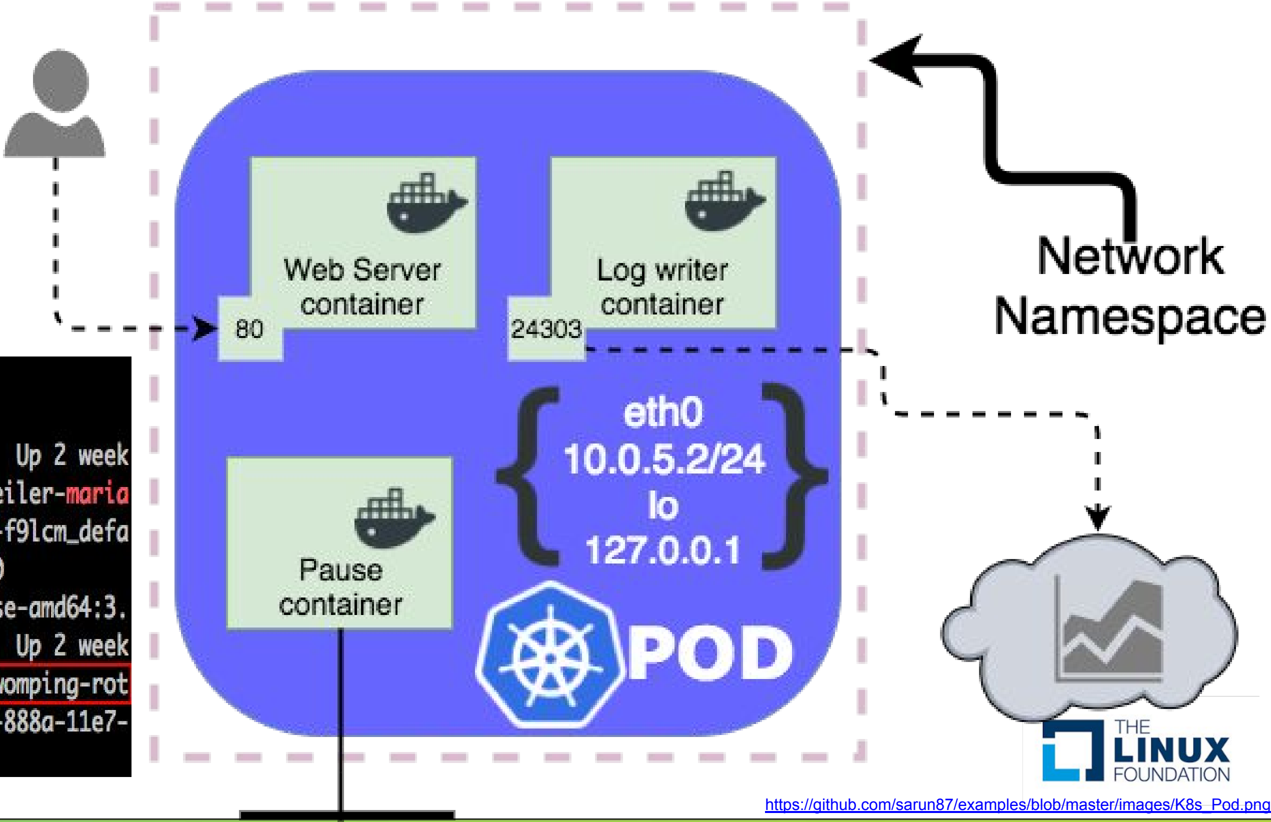
Container-to-Container

Pod

Group of one or more containers with shared storage/network



```
[root@ip-10-0-1-25 ~]# docker ps  | grep maria
f679a28b57a3        bitnami/mariadb:10.1.23-r2
    "/app-entrypoint.sh /"   2 weeks ago         Up 2 week
s                            k8s_womping-rottweiler-maria
db.7d5c160c_womping-rottweiler-mariadb-155601547-f9lcm_defa
ult_aba49f60-888a-11e7-9059-021eb171b86e_30ebfc40
634acd220b92        gcr.io/google_containers/pause-amd64:3.
0   "/pause"                 2 weeks ago         Up 2 week
s                            k8s_POD.d8dbe16c_womping-rot
tweiler-mariadb-155601547-f9lcm_default_aba49f60-888a-11e7-
9059-021eb171b86e_f5e00cb4
```

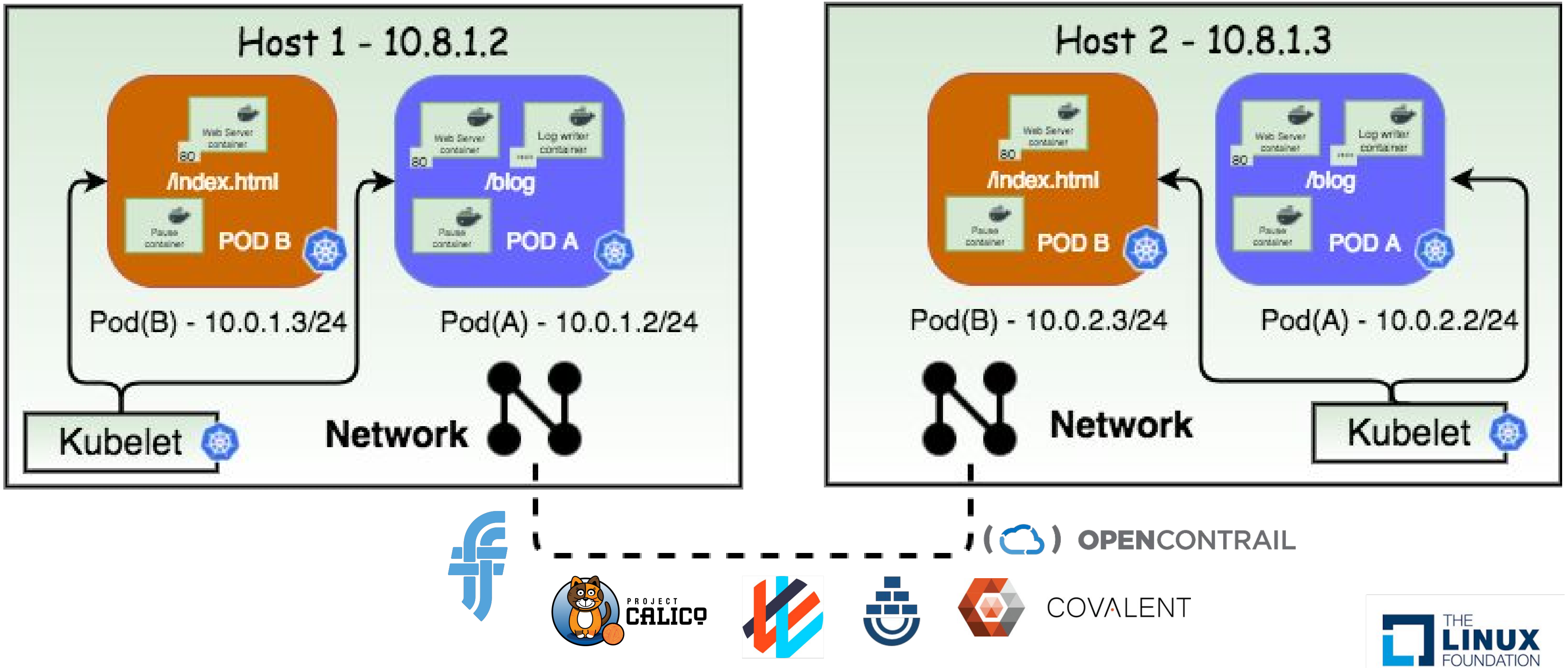


https://github.com/sarun87/examples/blob/master/images/K8s_Pod.png

Container-to-Container takeaways

- Containers in a pod run on the same host.
- A pod generally represents a service unit of an application.
- Uses localhost (127.0.0.1) within the pod’s network namespace to communicate with each other
- Containers in the same Pod cannot reuse ports ⚠
- Pause container - Keeps the networking alive
- New concepts: Pod, Pause container

Pod-to-Pod



https://github.com/sarun87/examples/blob/master/images/K8s_Pod_to_Pod.png

Pod-to-Pod takeaways

Currently supported networking models -

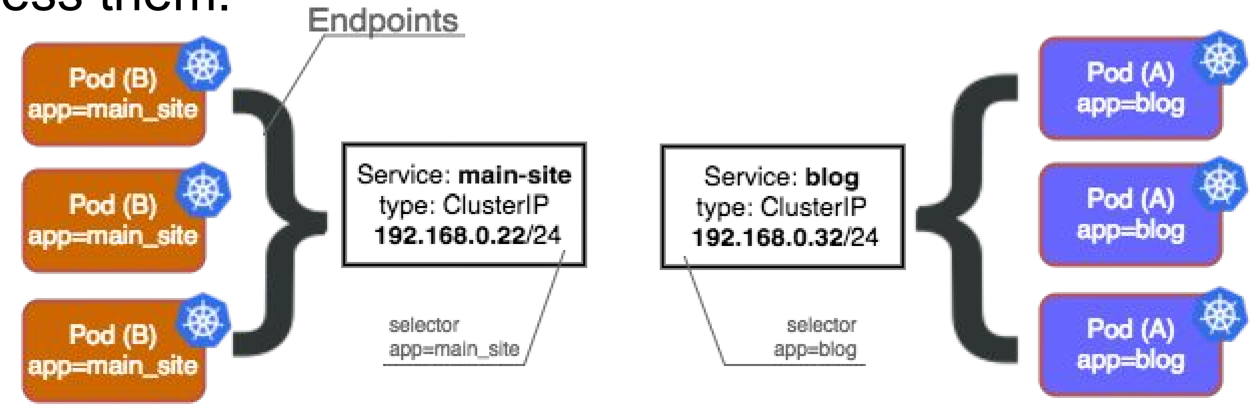
- Kubenet via kubelet (will be moved out to CNI)
- Multiple network backends via CNI (We'll discuss this in depth later)

Network backend responsible for -

- Pod networking setup
- Pod-to-Pod networking setup (uses L3 BGP like Calico, network overlay like weave, flannel)
- New concepts: Kubelet, CNI, network backend

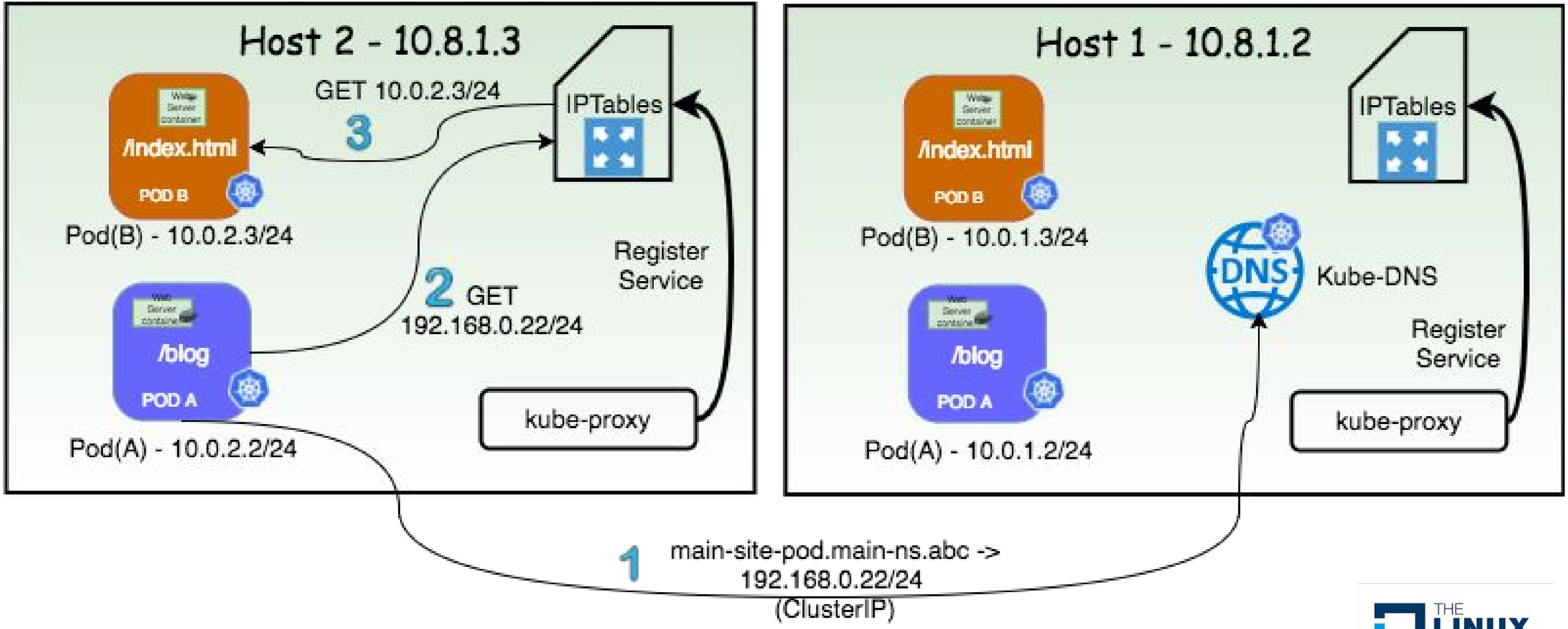
Kubernetes “Service” Primer

Service - an abstraction which defines a logical set of Pods and a policy by which to access them.



- A service is “generally” backed by pods (endpoints) using a “label selector”.
- Users can explicitly define an endpoint that isn’t backed by pods
- K8s defines many types of services
    - Internal: ClusterIP
    - External: NodePort, LoadBalancer, Ingress

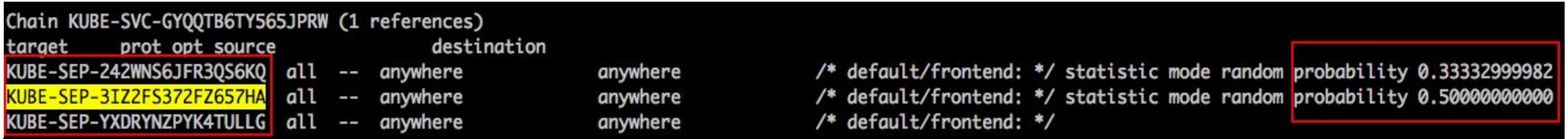
Pod-to-Service (Cluster Internal)



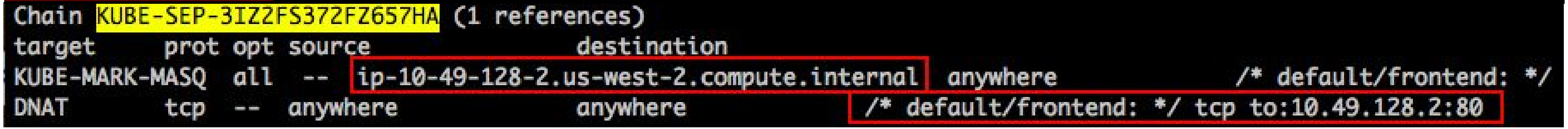
https://github.com/sarun87/examples/blob/master/images/K8s_Pod_to_Service.png

Pod-to-Service takeaways

- Service is a logical definition/collection of pods.
- ClusterIP is allocated from the Services CIDR
- kube-proxy modes
  - userspace
  - iptables (our discussed example)
- New concepts: kube-proxy, kube-dns, Service, clusterIP, iptables

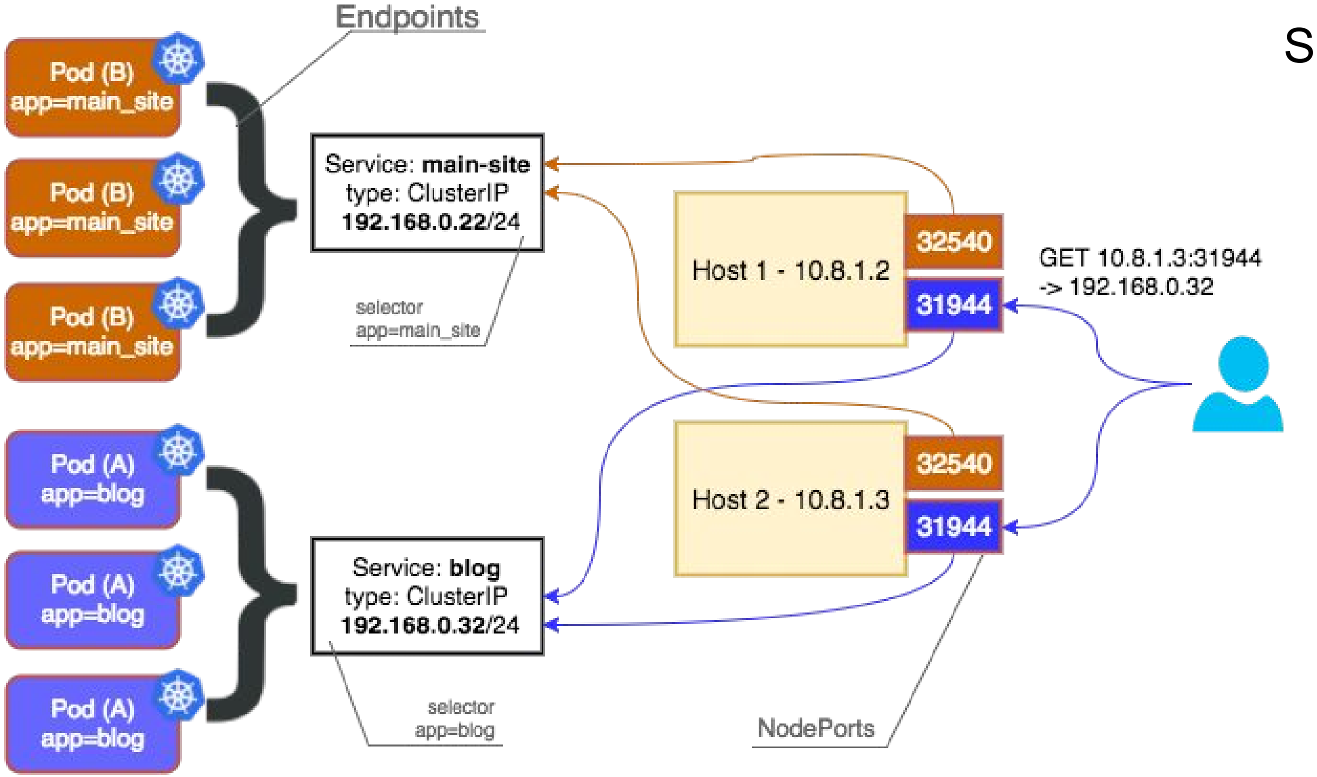


```
Chain KUBE-SVC-GYQQTB6TY565JPRW (1 references)
target     prot opt source               destination
KUBE-SEP-242WNS6JFR3QS6KQ  all  --  anywhere             anywhere             /* default/frontend: */ statistic mode random probability 0.33332999982
KUBE-SEP-3IZ2FS372FZ657HA  all  --  anywhere             anywhere             /* default/frontend: */ statistic mode random probability 0.50000000000
KUBE-SEP-YXDRYNZPYK4TULLG  all  --  anywhere             anywhere             /* default/frontend: */
```



```
Chain KUBE-SEP-3IZ2FS372FZ657HA (1 references)
target     prot opt source               destination
KUBE-MARK-MASQ  all  --  ip-10-49-128-2.us-west-2.compute.internal  anywhere             /* default/frontend: */
DNAT       tcp  --  anywhere             anywhere             /* default/frontend: */ tcp to:10.49.128.2:80
```

External-to-Service

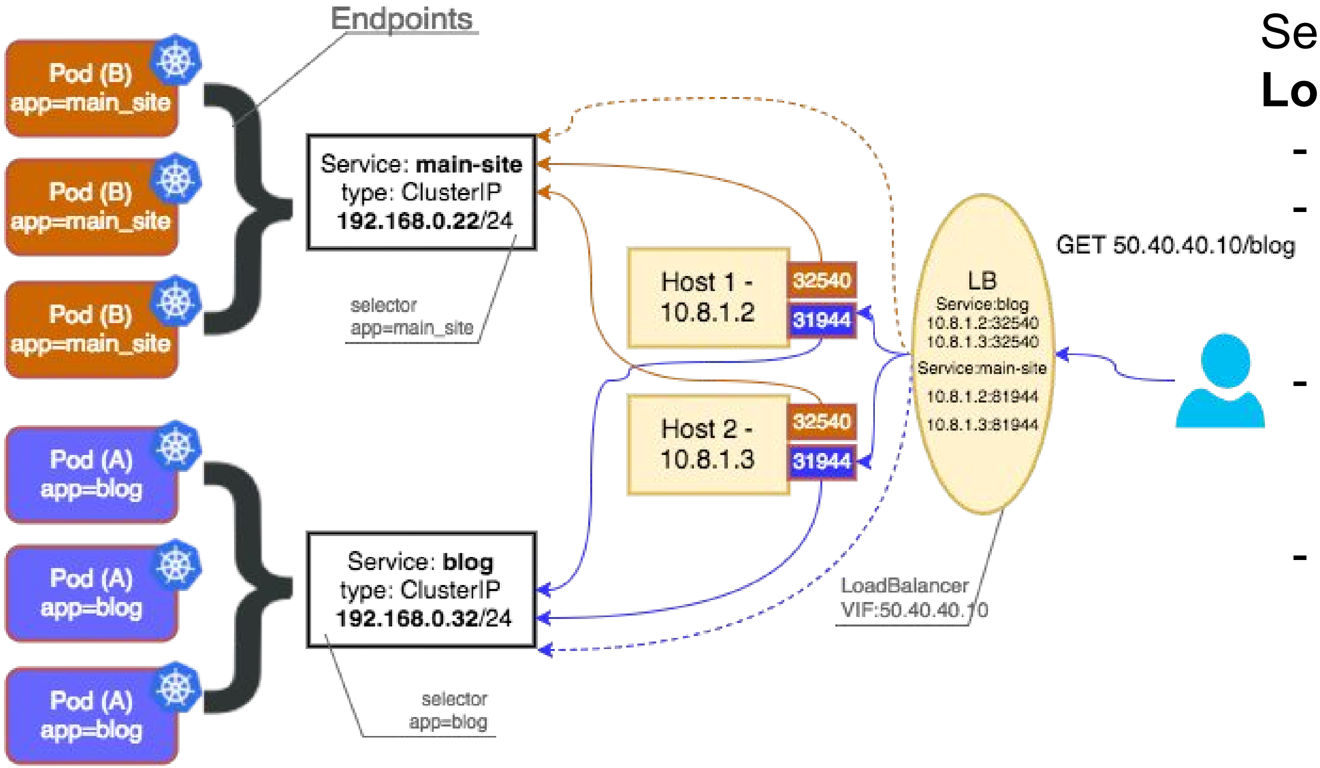


Service type: **NodePort**

- Kubernetes master allocates a port from a flag-configured range (default: 30000-32767).
- Each Node will proxy that port (the same port number on every Node) into your Service

https://github.com/sarun87/examples/blob/master/images/K8s_NodePort.png

External-to-Service - II



Service type:
LoadBalancer

- Fronts the K8s Service
- Traffic from load balancer is directed to backend Pods
- Exactly how that works depends on the cloud provider
- NodePort and ClusterIP to which LB will route are created automatically

External-to-Service III

Ingress

- An Ingress is a collection of rules that allow inbound connections to reach the cluster services.
- Ingress is useful since services typically have internal IPs/endpoints
- All traffic that ends up at an edge router is either dropped or forwarded elsewhere
- Gives services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting

External IPs

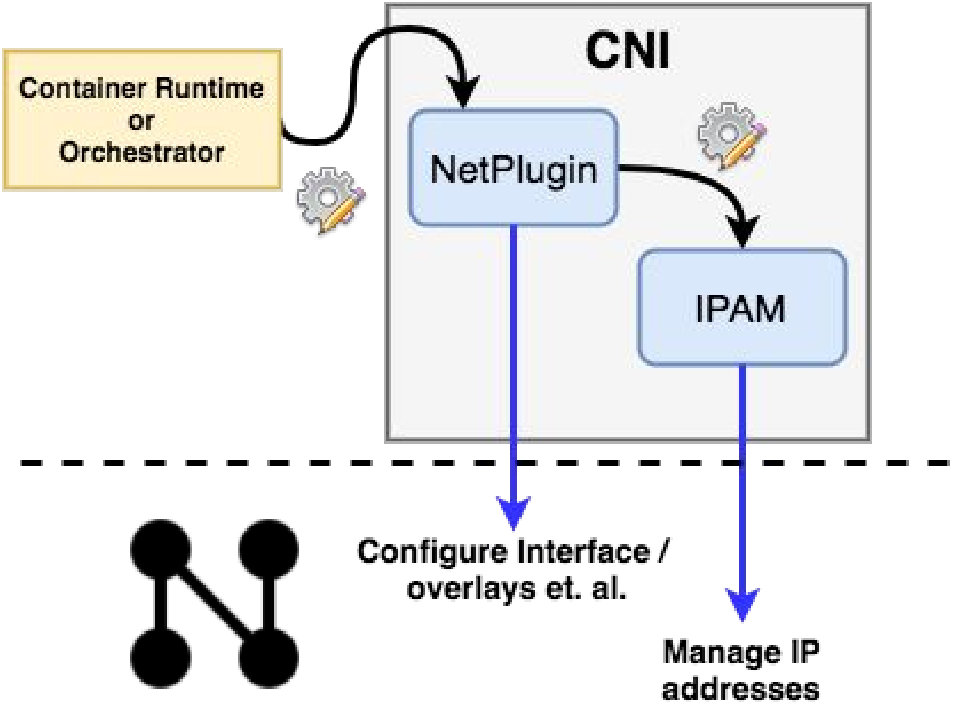
- A public/external IP points to a node of the cluster
- Service ingresses the requests from the external IP
- Are not managed by K8s

Note: If you came here to understand ingress specifically, let's chat offline. I will cover this if time permits

CNI - Container Network Interface

- Simple interface between container runtime & network
- **CNCF** project. Started by CoreOS for the **rkt** runtime

- Config passed to the NetPlugin by runtime then passed to IPAM
- CNI Interfaces - ADD, DEL



CNI - plugins

CNI Maintained

Plugins that create/delete interfaces

- bridge
- ipvlan
- lo
- macvlan
- vlan
- ptp

IPAM - IP address management

- dhcp
- host-local

3rd party/others

- flannel (now under CNI)
- calico
- canal
- weave
- Cilium
- Contrail
- Contiv
- Infoblox
- Romana
- Nuage
- ….

Github repo - https://github.com/containernetworking/cni

Using CNI with individual containers

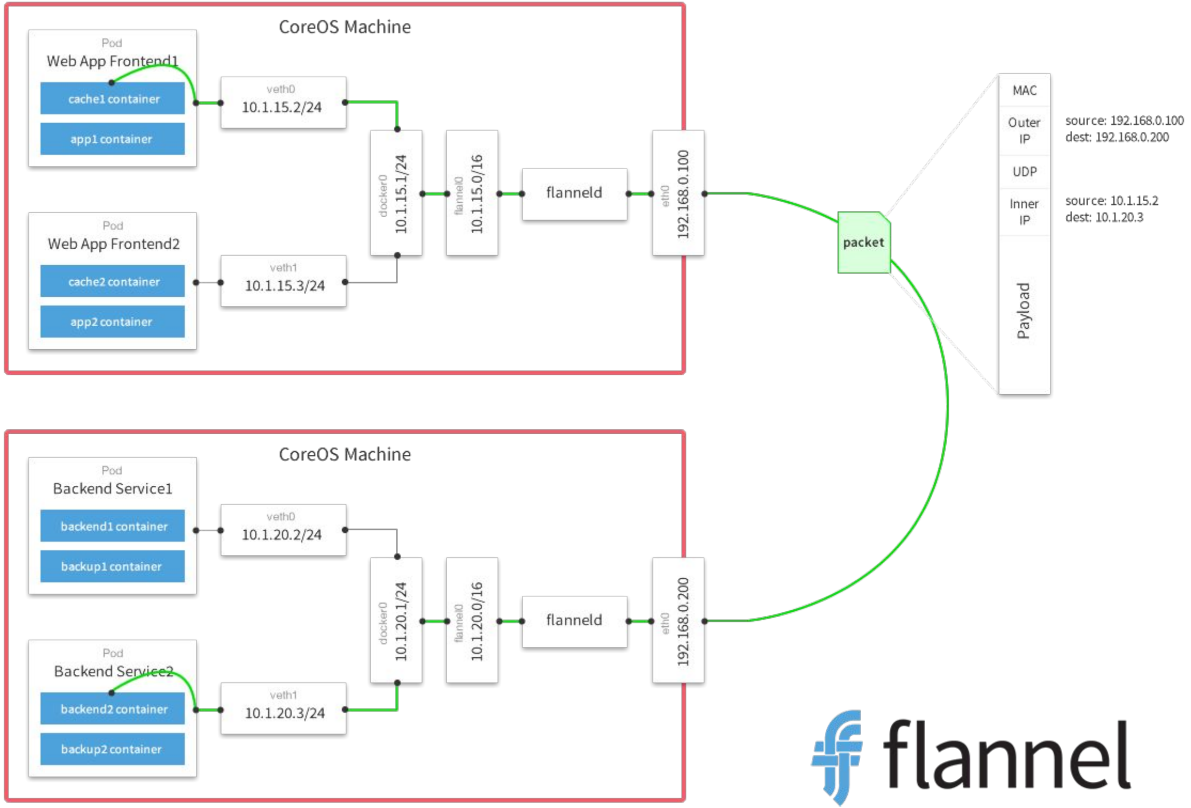
Eg: host-local IPAM. To ADD n/w to a container

```
$ CNI_COMMAND=ADD \
CNI_CONTAINERID=arun_container_01 \
CNI_NETNS=/var/run/netns/cni_ipam_eg \
CNI_IFNAME=eth0 \
CNI_PATH=/home/ubuntu/cni/bin \
./host-local < sample_ipam_config

{
        "cniVersion": "0.3.1",
        "ips": [{
                  "version": "4",
                  "address": "10.10.10.2/24",
                  "gateway": "10.10.10.1"
    }],
        "dns": {}
```

```
$ cat sample_ipam_config
{
  "cniVersion": "0.3.1",
  "name": "example-network",
  "ipam": {
    "type": "host-local",
    "subnet": "10.10.10.0/24”,
    "dataDir":
"/home/ubuntu/sample_ipam_datadir
"
  }
}
```

Flannel network backend



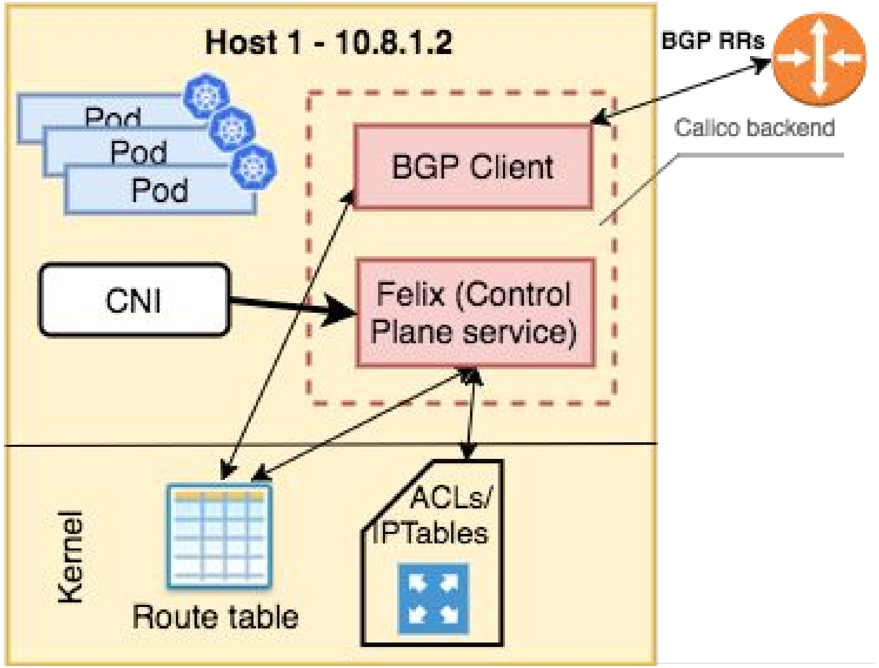
flannel

- Uses overlay network for host-host connectivity
- Backends - UDP, vxlan
- flanneld binary runs on every host
- Does **not** perform host - container networking.
- Via CNI, flannel delegates interface operations to bridge driver.

https://github.com/coreos/flannel/blob/master/packet-01.png

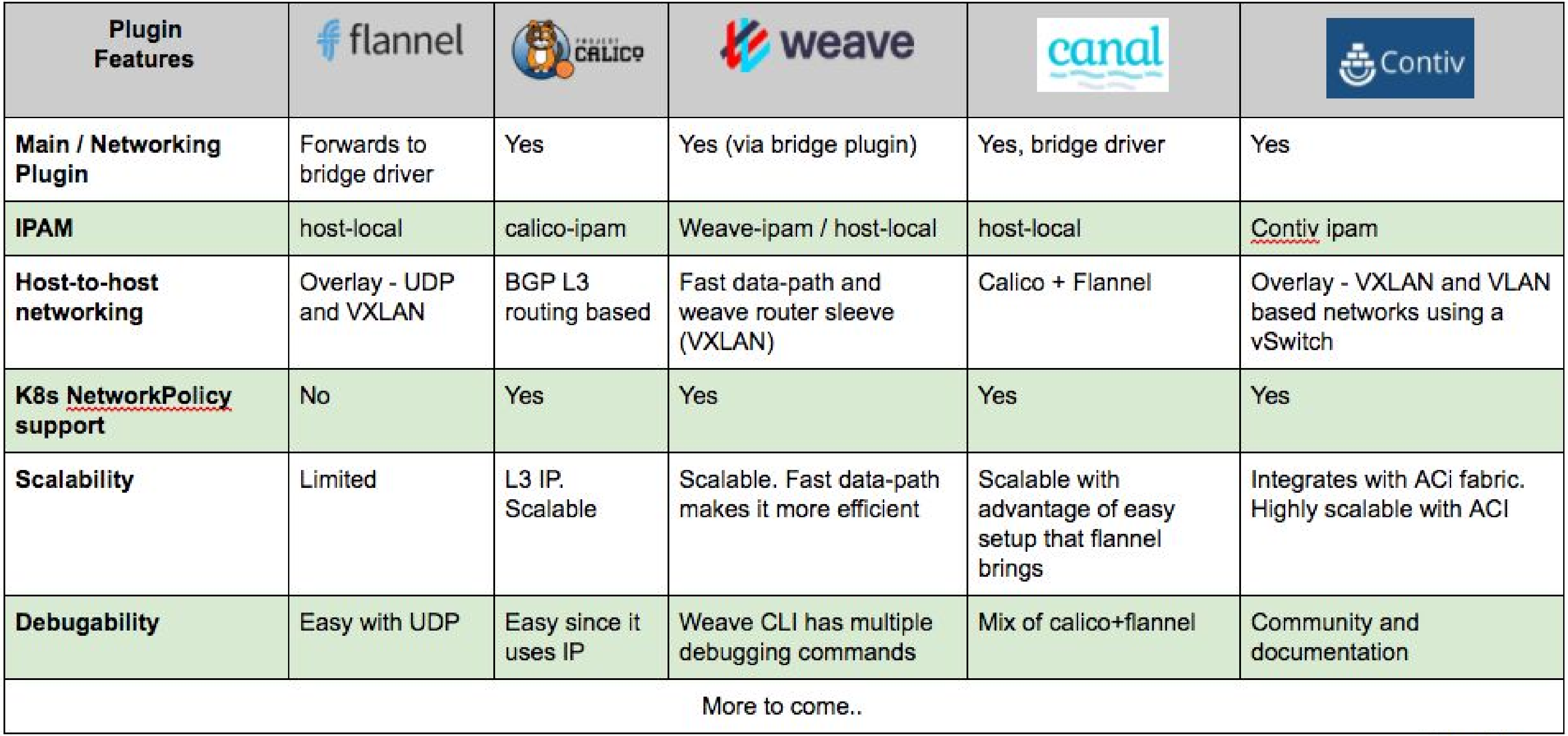
Calico network backend

- Pure L3 based network solution
- Router per node
- Uses BGP
- via CNI plugin - has its own IPAM driver as well
- Supports Kubernetes NetworkPolicy constructs
- BIRD protocol (BGP stack)
- ACL and L3 forwarding performed in the linux kernel
- Ease of debugging
- Scalable



https://github.com/sarun87/examples/blob/master/images/Calico_Architecture.png

CNI backends summarized



Plugin Features	flannel	PROJECT CALICO	weave	canal	Contiv
Main / Networking Plugin	Forwards to bridge driver	Yes	Yes (via bridge plugin)	Yes, bridge driver	Yes
IPAM	host-local	calico-ipam	Weave-ipam / host-local	host-local	Contiv ipam
Host-to-host networking	Overlay - UDP and VXLAN	BGP L3 routing based	Fast data-path and weave router sleeve (VXLAN)	Calico + Flannel	Overlay - VXLAN and VLAN based networks using a vSwitch
K8s NetworkPolicy support	No	Yes	Yes	Yes	Yes
Scalability	Limited	L3 IP. Scalable	Scalable. Fast data-path makes it more efficient	Scalable with advantage of easy setup that flannel brings	Integrates with ACi fabric. Highly scalable with ACI
Debugability	Easy with UDP	Easy since it uses IP	Weave CLI has multiple debugging commands	Mix of calico+flannel	Community and documentation
More to come..					

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
NORTH AMERICA

Thank You

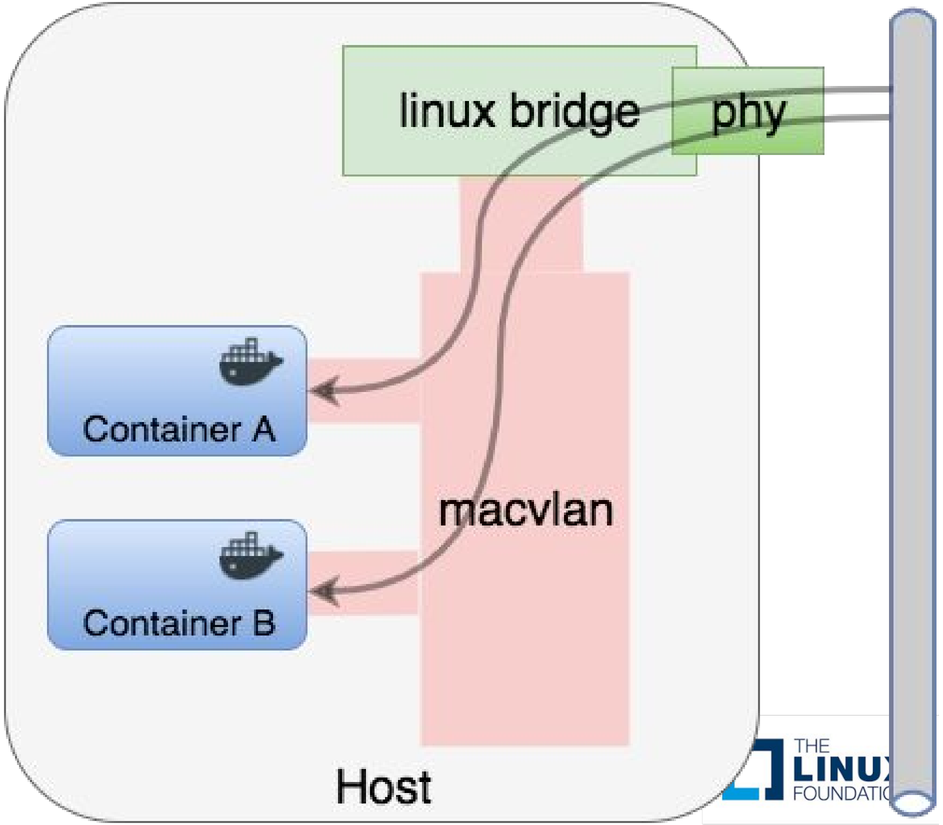
Help me to better help you next time.
Questions/Feedback:

🐦 @arun_sriraman

THE LINUX FOUNDATION

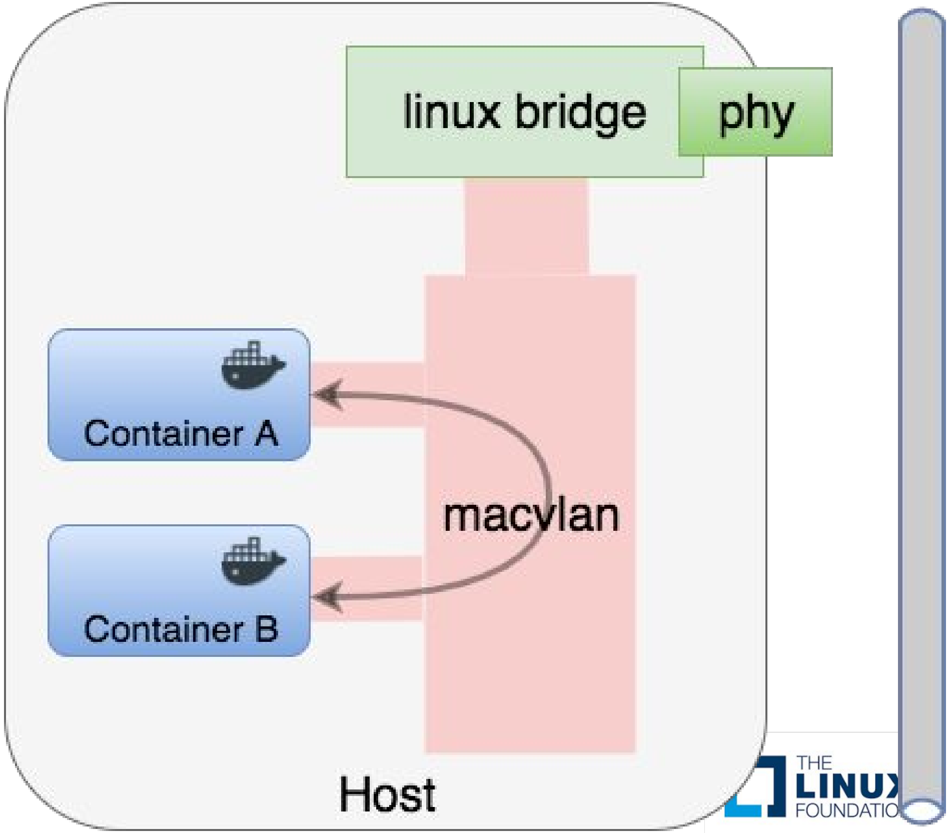
Macvlan - VEPA mode

- Virtual Ethernet Port Aggregator is the default macvlan mode
- Data sent directly via ethernet card
- External devices should support hairpin/reflective relay
- Container traffic can be seen at phy switch



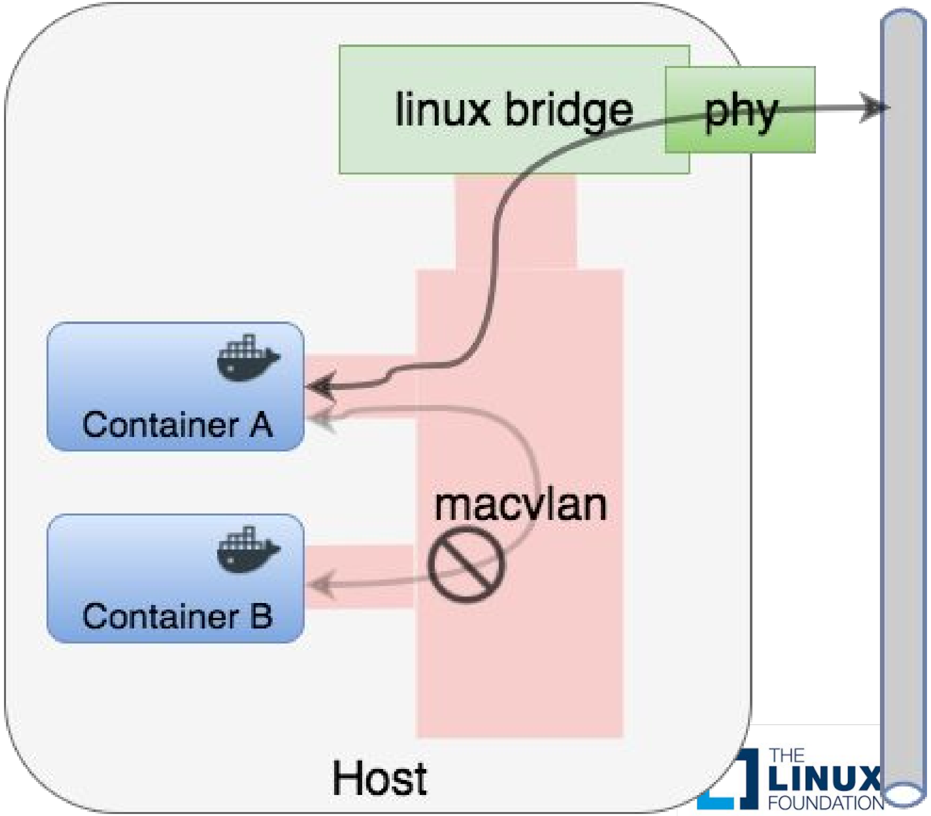
Macvlan - Bridge mode

- Containers on the same macvlan device are bridged
- No need to send traffic outside if target is on another macvlan device
- Trivial bridge with no learning required
- Simple & fast



Macvlan - Private mode

- Containers on the same macvlan device cannot talk to each other
- Container isolation
- External access allowed for all containers


linux bridge
phy
Container A
Container B
macvlan
Host