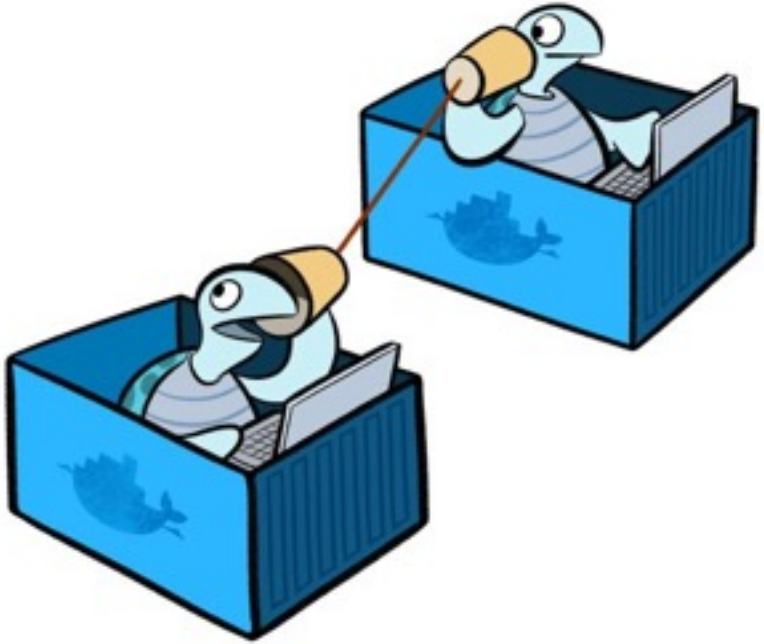


Docker Networking Deep Dive

online meetup 08/24/2016

@MadhuVenugopal

dockercon 16

Agenda

- What is libnetwork
- CNM
- 1.12 Features
  - Multihost networking
  - Secured Control plane & Data plane
  - Service Discovery
  - Native Loadbalacing
  - Routing Mesh
- Demo

Overview

What is libnetwork?

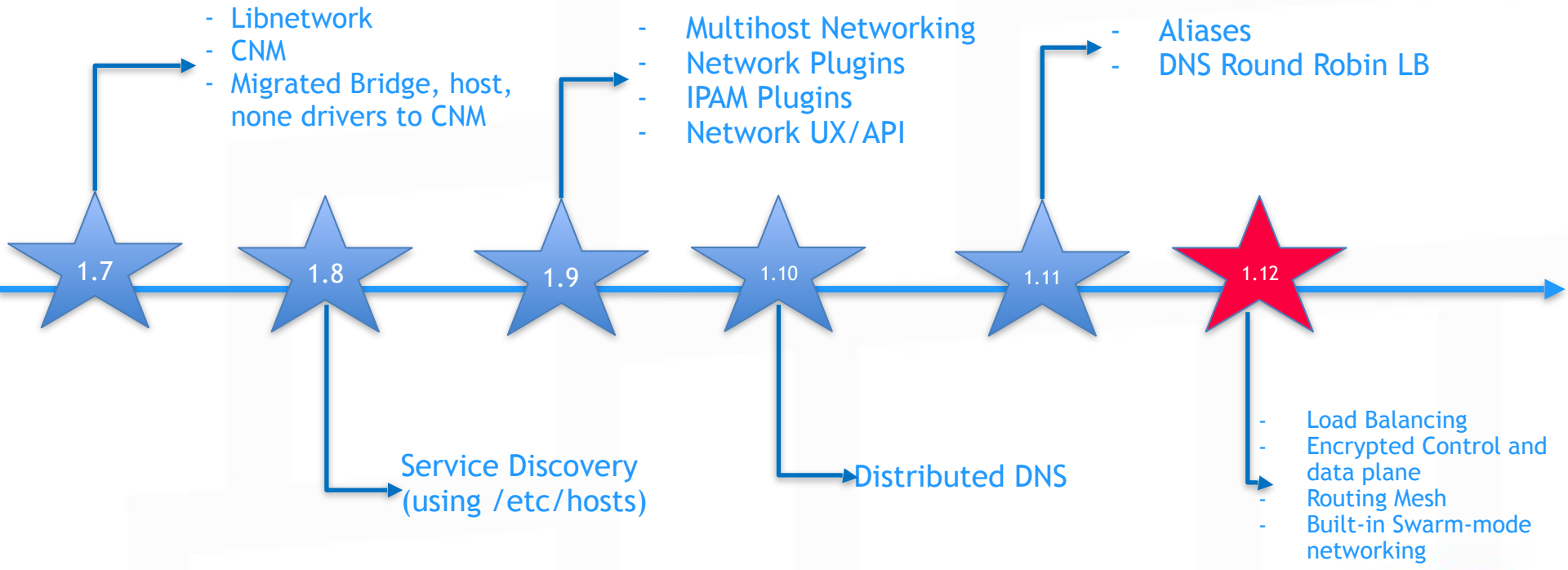
It is not just a driver interface

- Docker networking fabric
- Defines Container Networking Model
- Provides builtin IP address management
- Provides native multi-host networking
- Provides native Service Discovery and Load Balancing
- Allows for extensions by the ecosystem via plugins

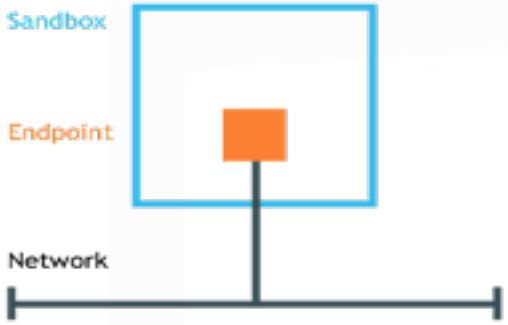
Design Philosophy

- Users First:
  - Application Developers
  - IT/Network Ops
- Plugin API Design
  - Batteries Included but Swappable

Docker Networking



Container Networking Model



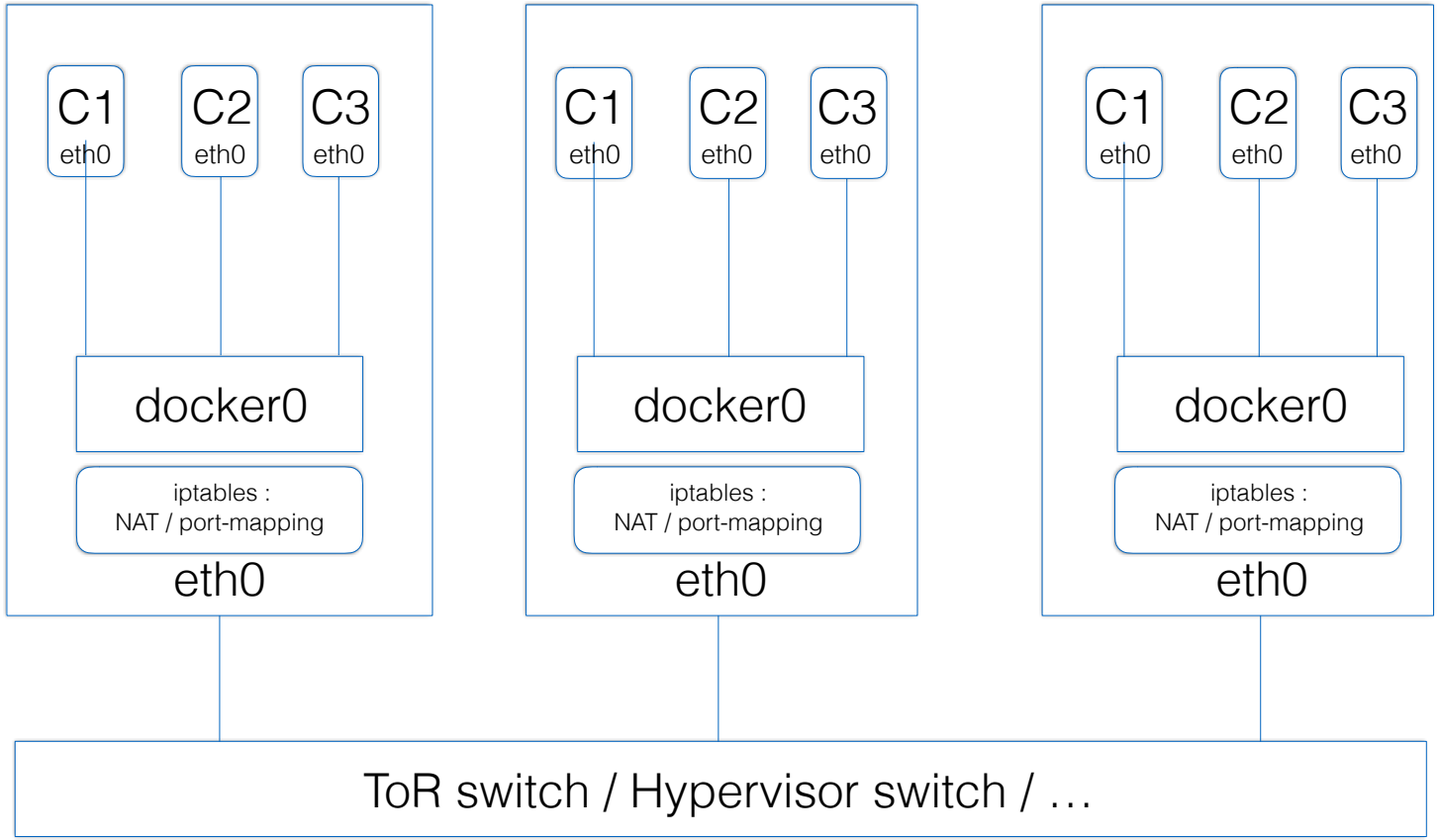
- Endpoint
- Network
- Sandbox
- Drivers & Plugins

https://github.com/docker/libnetwork/blob/master/docs/design.md

Network driver overview

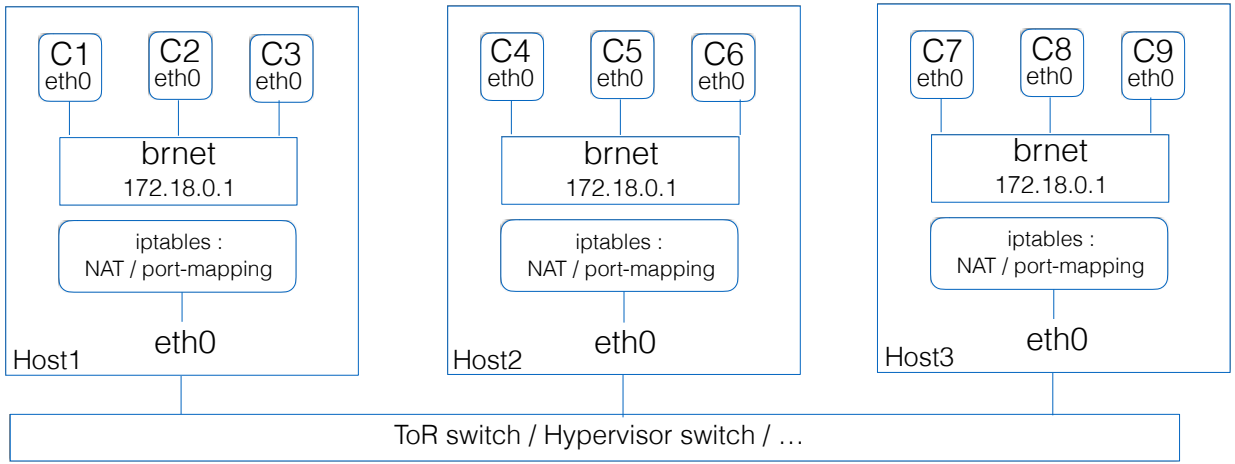
dockercon 16

Use-case1
Default Bridge Network
(docker0)


C1
eth0
C2
eth0
C3
eth0
docker0
iptables :
NAT / port-mapping
eth0
C1
eth0
C2
eth0
C3
eth0
docker0
iptables :
NAT / port-mapping
eth0
C1
eth0
C2
eth0
C3
eth0
docker0
iptables :
NAT / port-mapping
eth0
ToR switch / Hypervisor switch / …

Use-case2

User-Defined Bridge Network



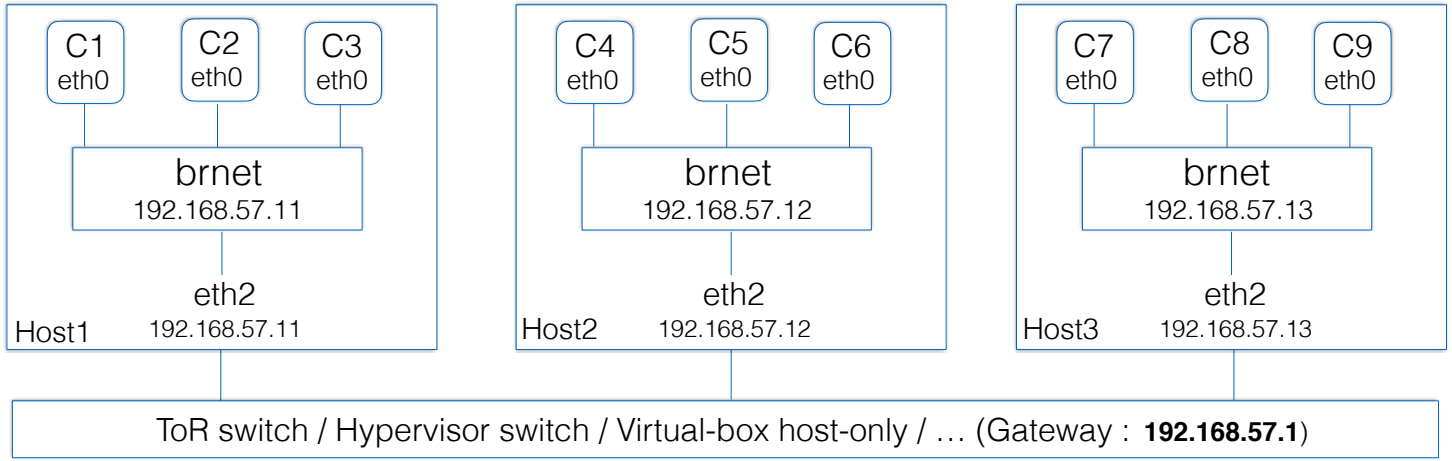
Host1 :
$ docker network create -d bridge -o com.docker.network.bridge.name=brnet brnet
$ docker run --net=brnet -it busybox ifconfig

Host2 :
$ docker network create -d bridge -o com.docker.network.bridge.name=brnet brnet
$ docker run --net=brnet -it busybox ifconfig

Host3 :
$ docker network create -d bridge -o com.docker.network.bridge.name=brnet brnet
$ docker run --net=brnet -it busybox ifconfig

Use-case3

Bridge Network plumbed to underlay with built-in IPAM
(no NAT / Port-mapping)



Host1 :
$ docker network create -d bridge --subnet=192.168.57.0/24 --ip-range=192.168.57.32/28 --gateway=192.168.57.11 --aux-address DefaultGatewayIPv4=192.168.57.1 -o com.docker.network.bridge.name=brnet brnet
$ brctl addif brnet eth2
$ docker run --net=brnet -it busybox ifconfig

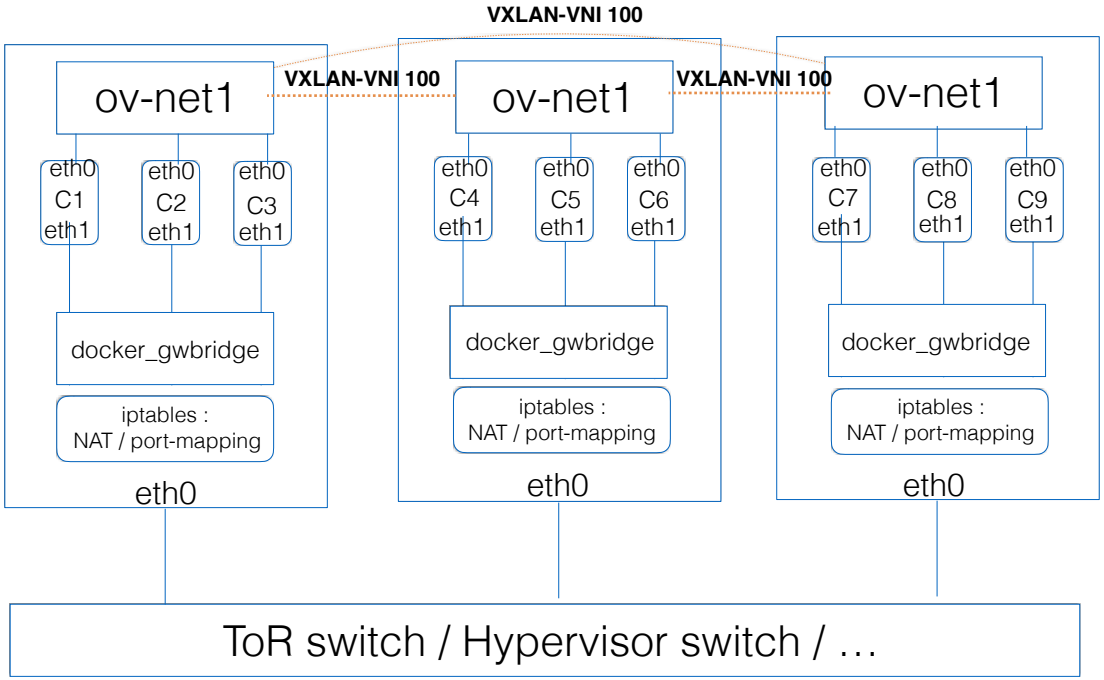
Host2 :
$ docker network create -d bridge --subnet=192.168.57.0/24 --ip-range=192.168.57.64/28 --gateway=192.168.57.12 --aux-address DefaultGatewayIPv4=192.168.57.1 -o com.docker.network.bridge.name=brnet brnet
$ brctl addif brnet eth2
$ docker run --net=brnet -it busybox ifconfig

Host3 :
$ docker network create -d bridge --subnet=192.168.57.0/24 --ip-range=192.168.57.128/28 --gateway=192.168.57.13 --aux-address DefaultGatewayIPv4=192.168.57.1 -o com.docker.network.bridge.name=brnet brnet
$ brctl addif brnet eth2
$ docker run --net=brnet -it busybox ifconfig

Use-case4

Docker Overlay Network

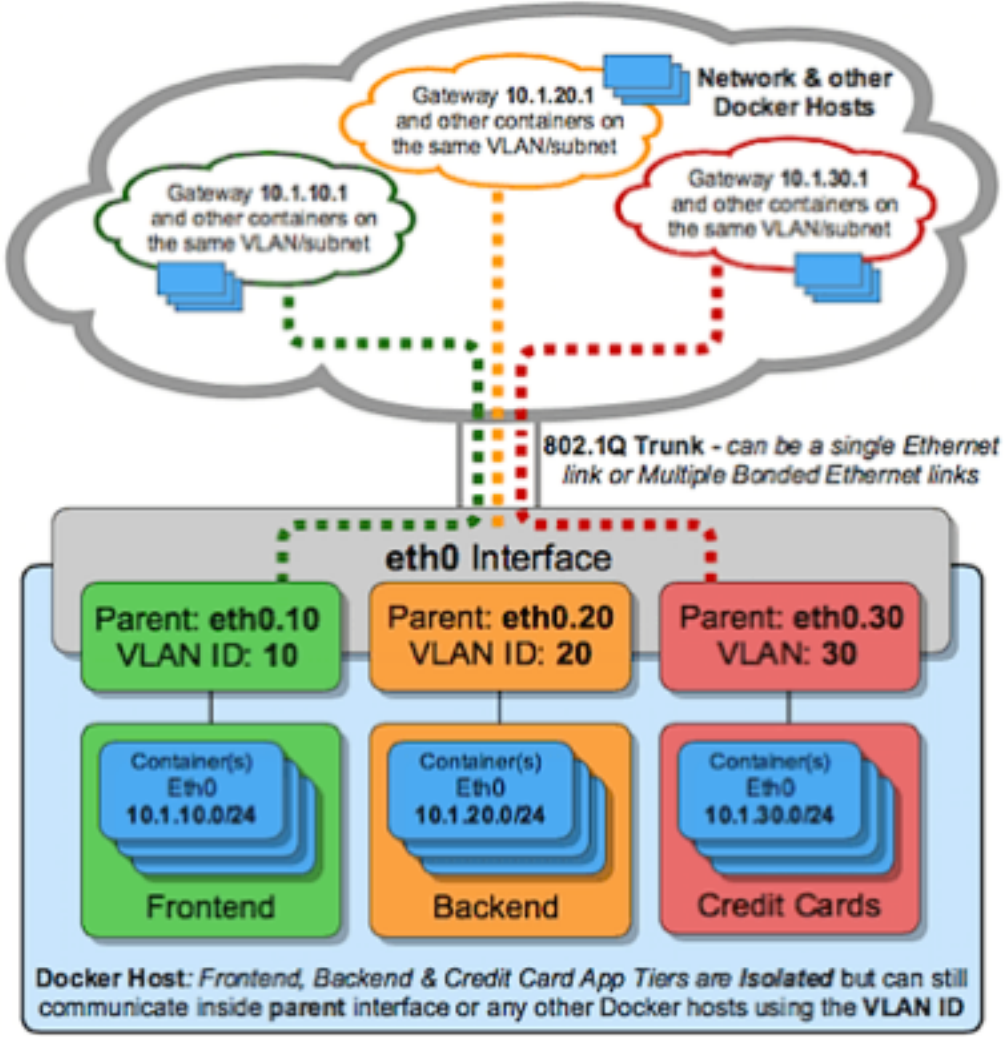
Docker overlay networking



Use-case5

Plumbed to underlay vlan with built-in IPAM

macvlan driver (& experimental ipvlan)
https://github.com/docker/docker/blob/master/experimental/vlan-networks.md



 # vlan 10 (eth0.10)

```
$ docker network create -d macvlan –subnet=10.1.10.0/24 –
gateway=10.1.10.1 -o parent=eth0.10 mcvlan10

$ docker run --net=mcvlan10 -it --rm alpine /bin/sh
```

 # vlan 20 (eth0.20)

```
$ docker network create -d macvlan –subnet=10.1.20.0/24 –
gateway=10.1.20.1 -o parent=eth0.20 mcvlan20

$ docker run --net=mcvlan20 -it --rm alpine /bin/sh
```

 # vlan 30 (eth0.30)

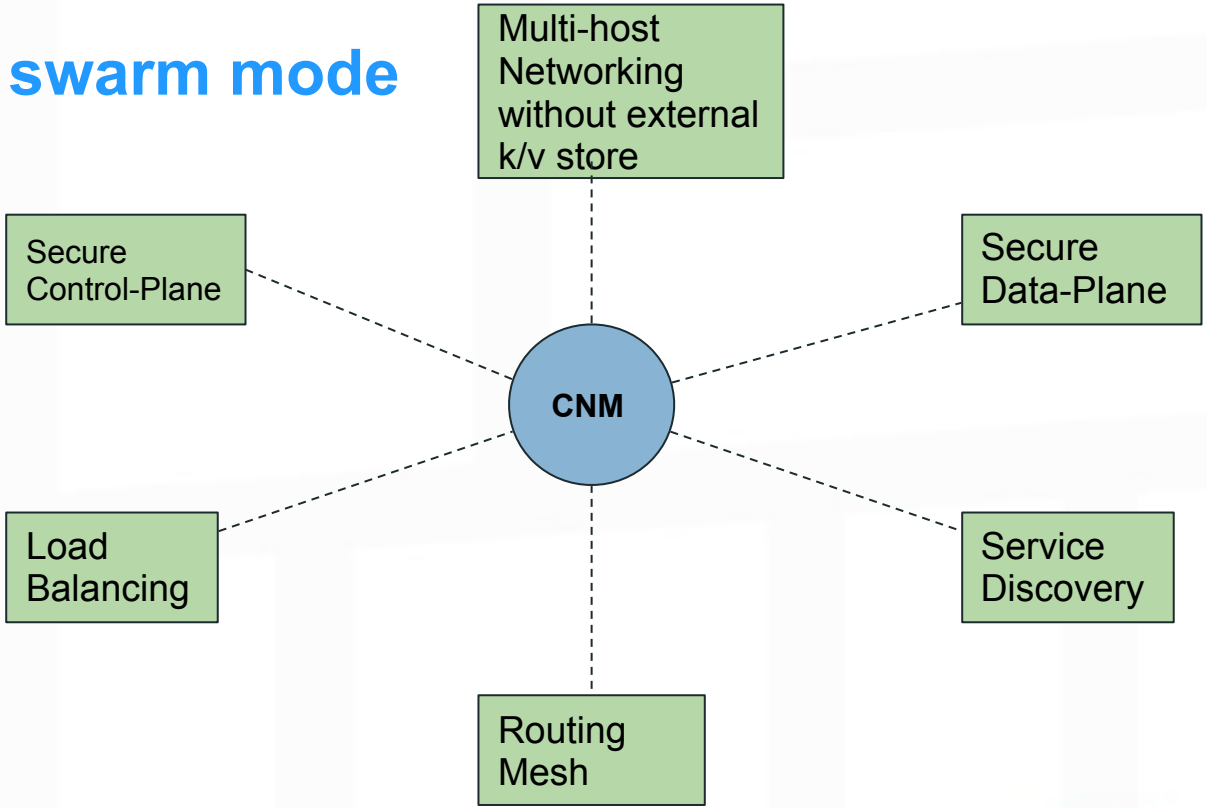
```
$ docker network create -d macvlan –subnet=10.1.30.0/24 –
gateway=10.1.30.1 -o parent=eth0.30 mcvlan30

$ docker run --net=mcvlan30 -it --rm alpine /bin/sh
```

Docker 1.12 Networking

dockercon 16

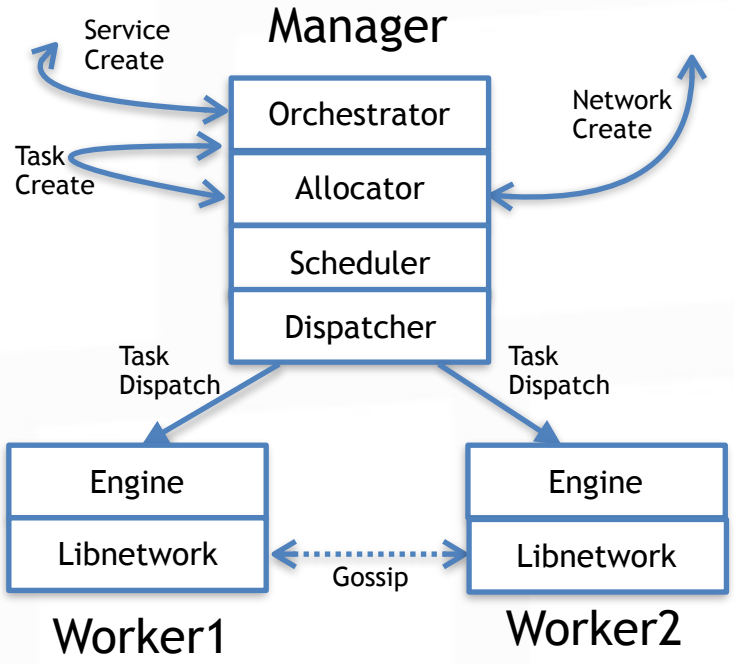
New features in 1.12 swarm mode

- Cluster aware
- De-centralized control plane
- Highly scalable



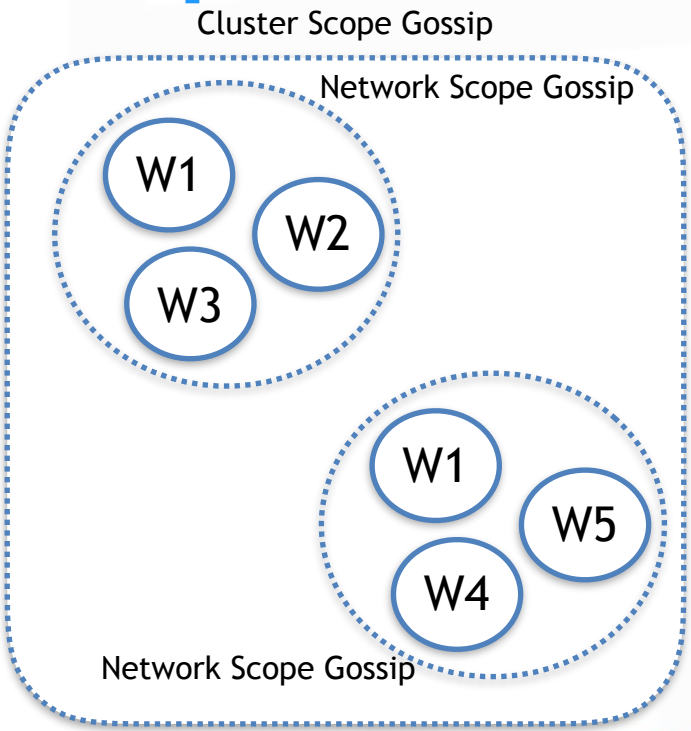
Swarm-mode Multi-host networking

- VXLAN based data path
- No external key-value store
- Central resource allocation
- Improved performance
- Highly scalable



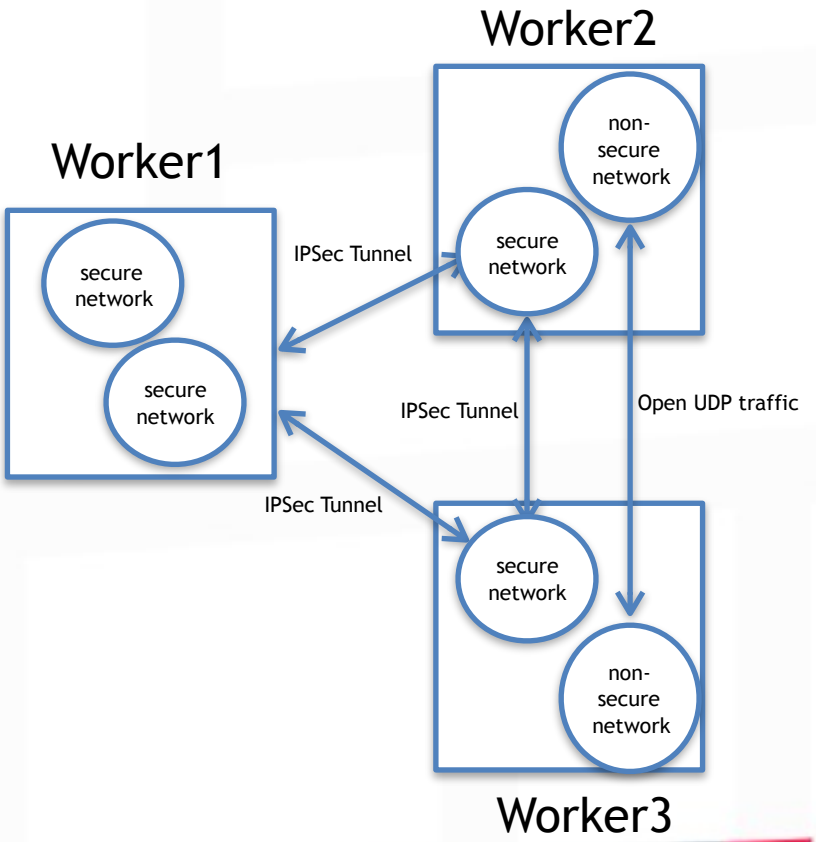
Secured network control plane

- Gossip based protocol
- Network scoped
- Fast convergence
- Secure by default
  - periodic key rotations
  - swarm native key-exchange
- Gossips control messages
  - Routing-states
  - Service-discovery
  - Plugin-data
- Highly scalable



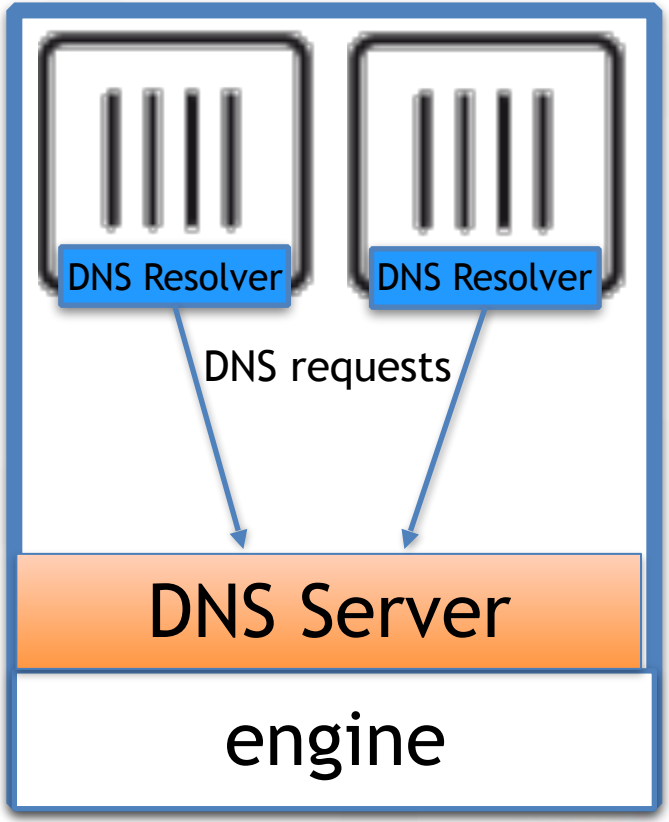
Secure dataplane

- Available as an option during overlay network creation
- Uses kernel IPSec modules
- On-demand tunnel setup
- Swarm native key-exchange
- Periodic key rotations



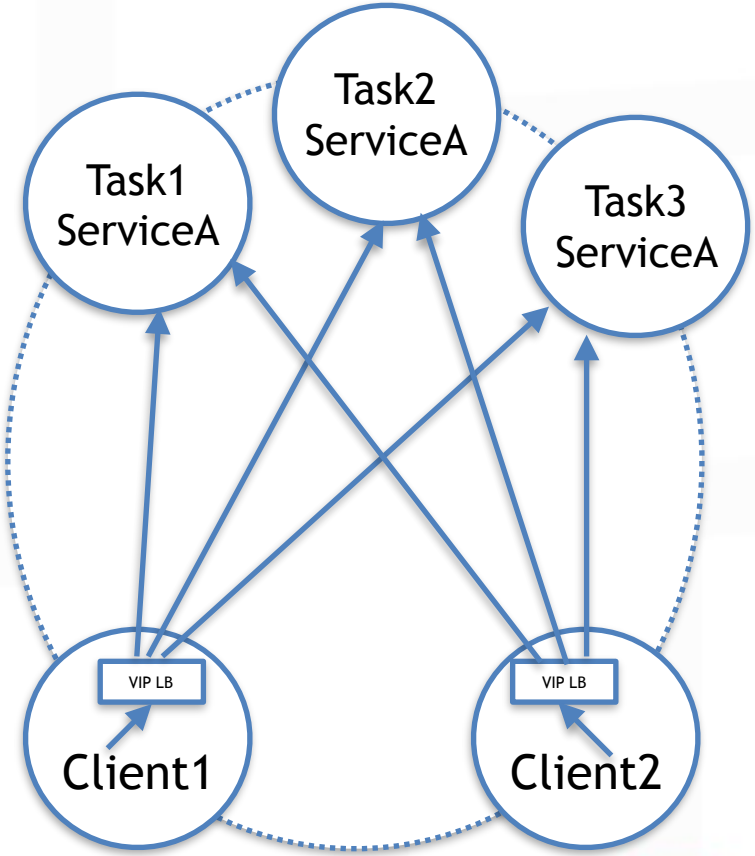
Service Discovery

- Provided by embedded DNS
- Highly available
- Uses Network Control Plane to learn state
- Can be used to discover both tasks and services



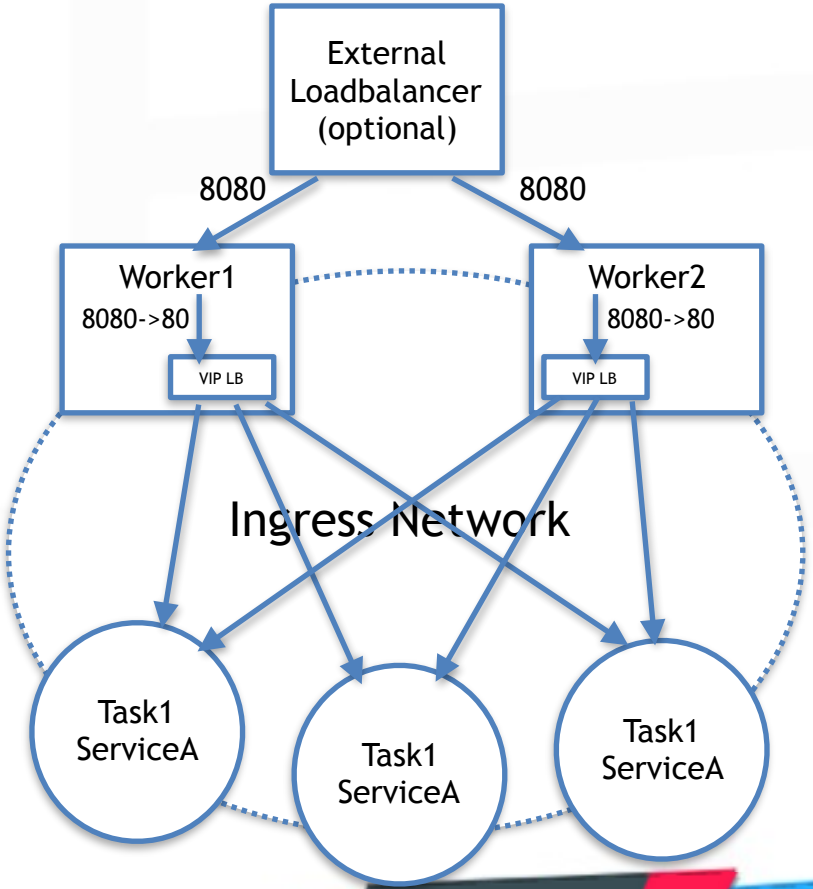
Load balancer

- Internal & Ingress load-balancing
- Supports VIP & DNS-RR
- Highly available
- Uses Network Control Plane to learn state
- Minimal Overhead

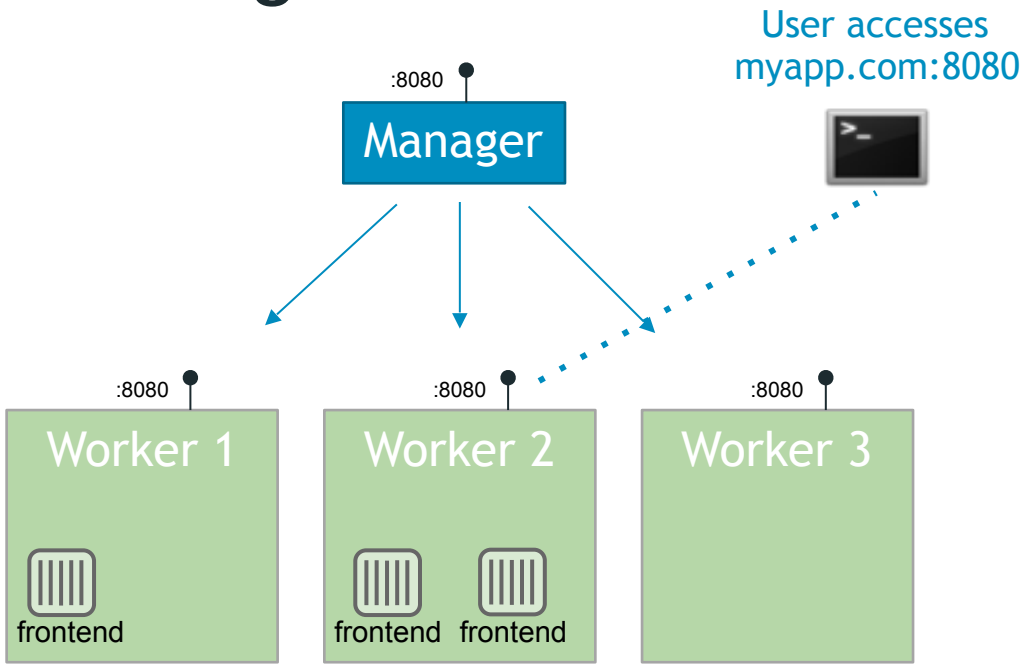


Routing mesh

- Builtin routing mesh for edge routing
- Worker nodes themselves participate in ingress routing mesh
- All worker nodes accept connection requests on PublishedPort
- Port translation happens at the worker node
- Same internal load balancing mechanism used to load balance external requests



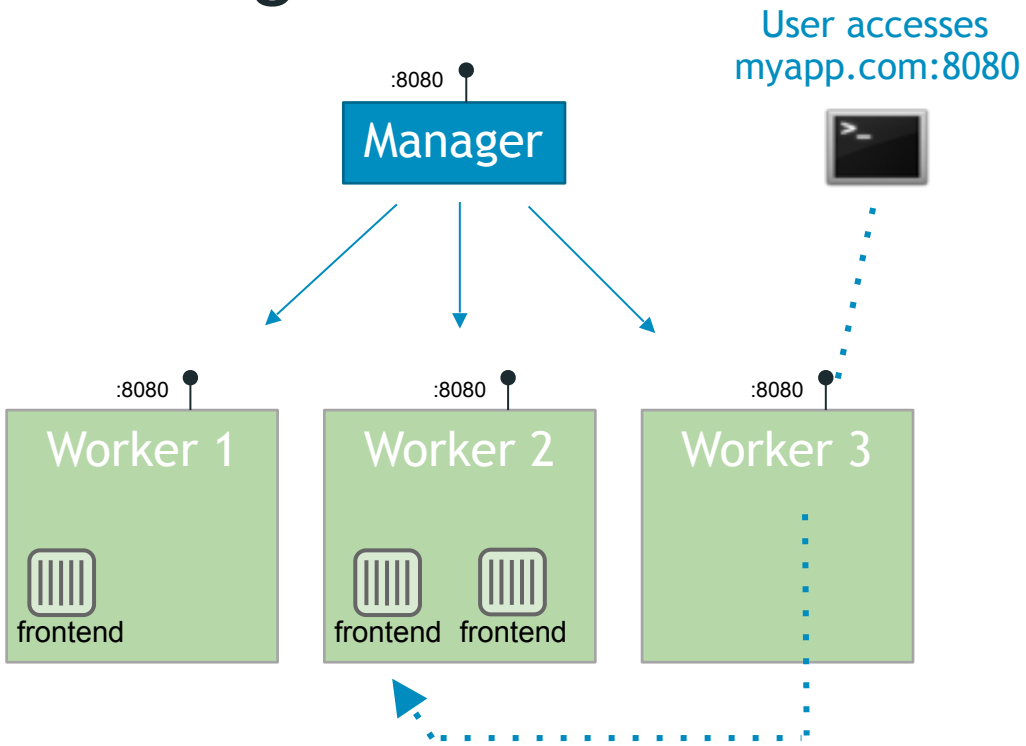
Routing Mesh



- Operator reserves a swarm-wide ingress port (8080) for `myapp`
- Every node listens on 8080
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

```
$ docker service create --replicas 3 --name frontend --network mynet
 --publish 8080:80/tcp frontend_image:latest
```

Routing Mesh: Published Ports



- Operator reserves a swarm-wide ingress port (8080) for `myapp`
- Every node listens on 8080
- Container-aware routing mesh can transparently reroute traffic from Worker3 to a node that is running container
- Built in load balancing into the Engine
- DNS-based service discovery

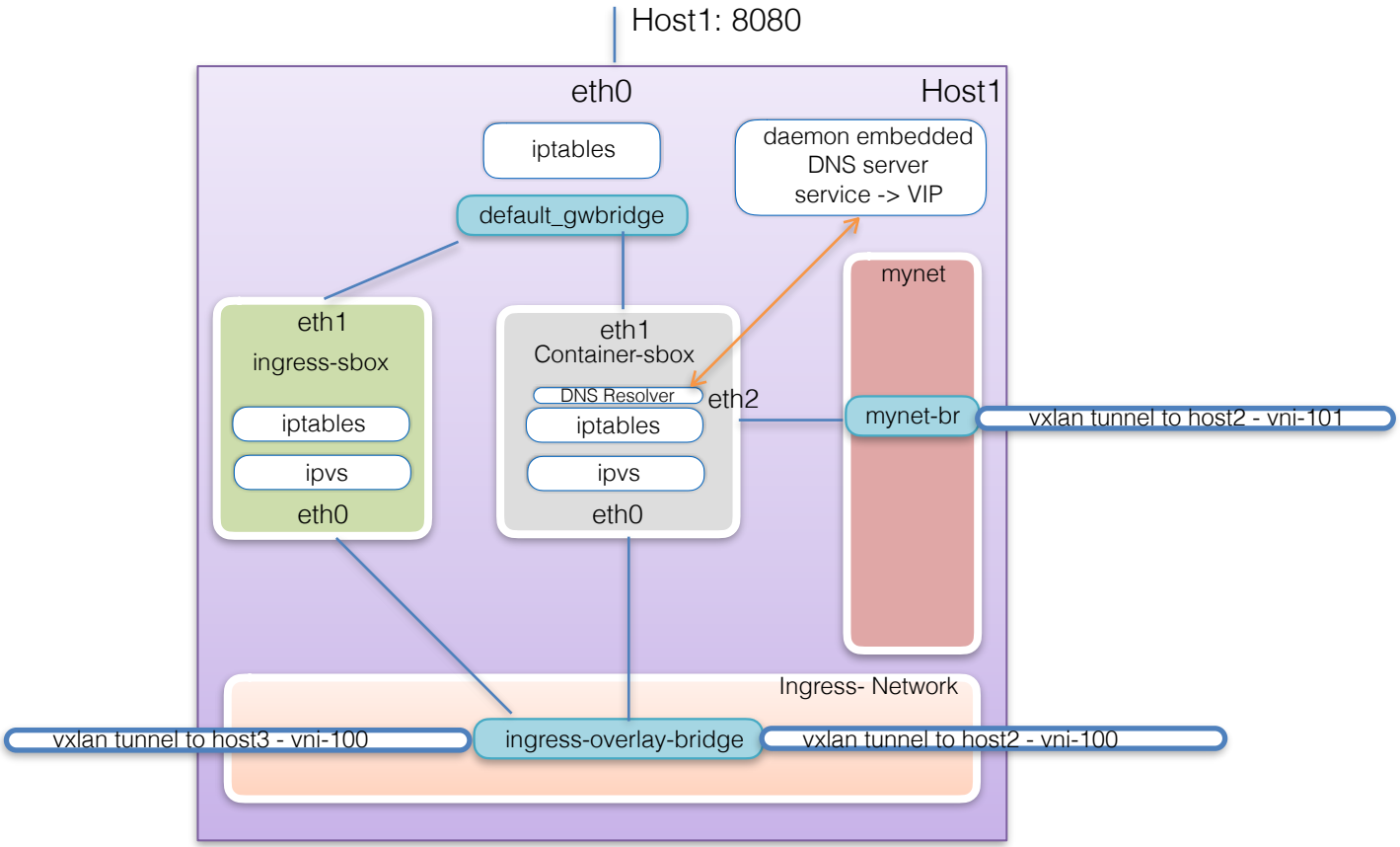
```
$ docker service create --replicas 3 --name frontend --network mynet
  --publish 8080:80/tcp frontend_image:latest
```

Deep Dive

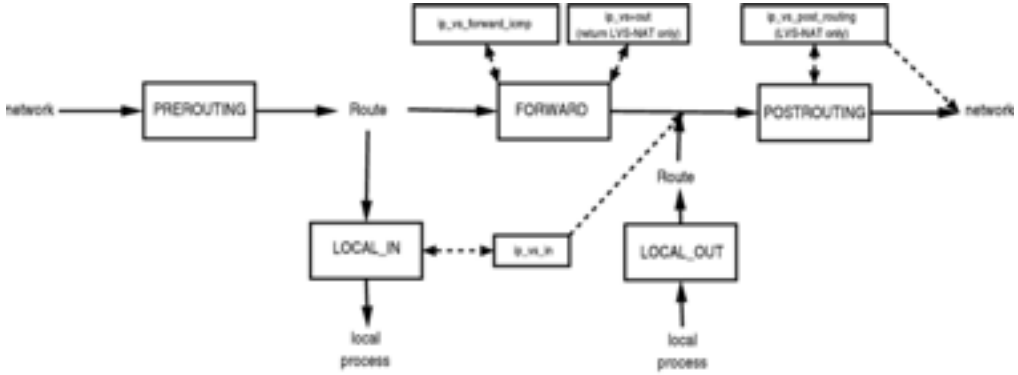
dockercon 16

Service , Port-Publish & Network

docker service create —name=test —network=mynet -p 8080:80 —replicas=2 xxx

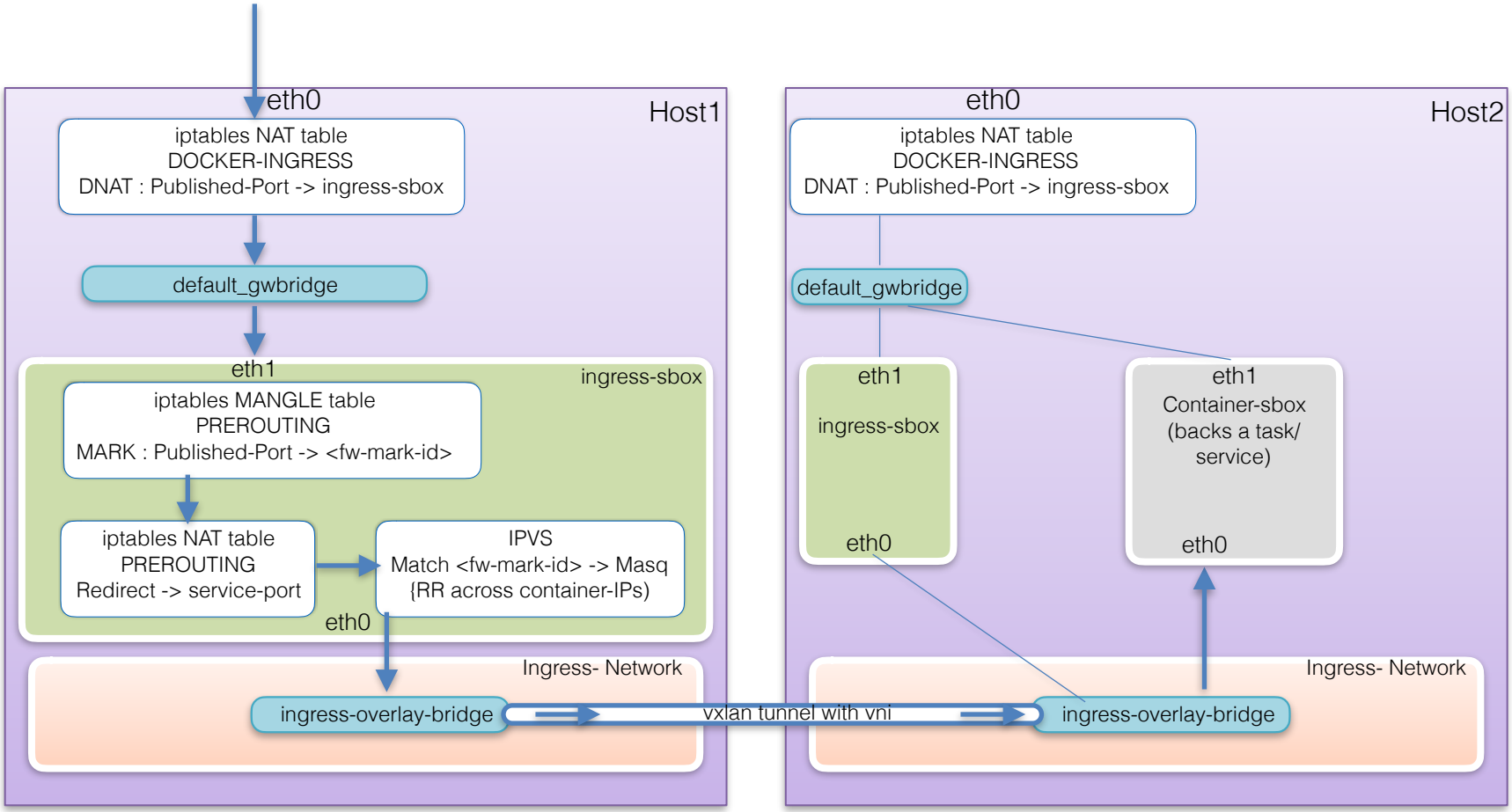


Day in life of a packet - IPTables & IPVS

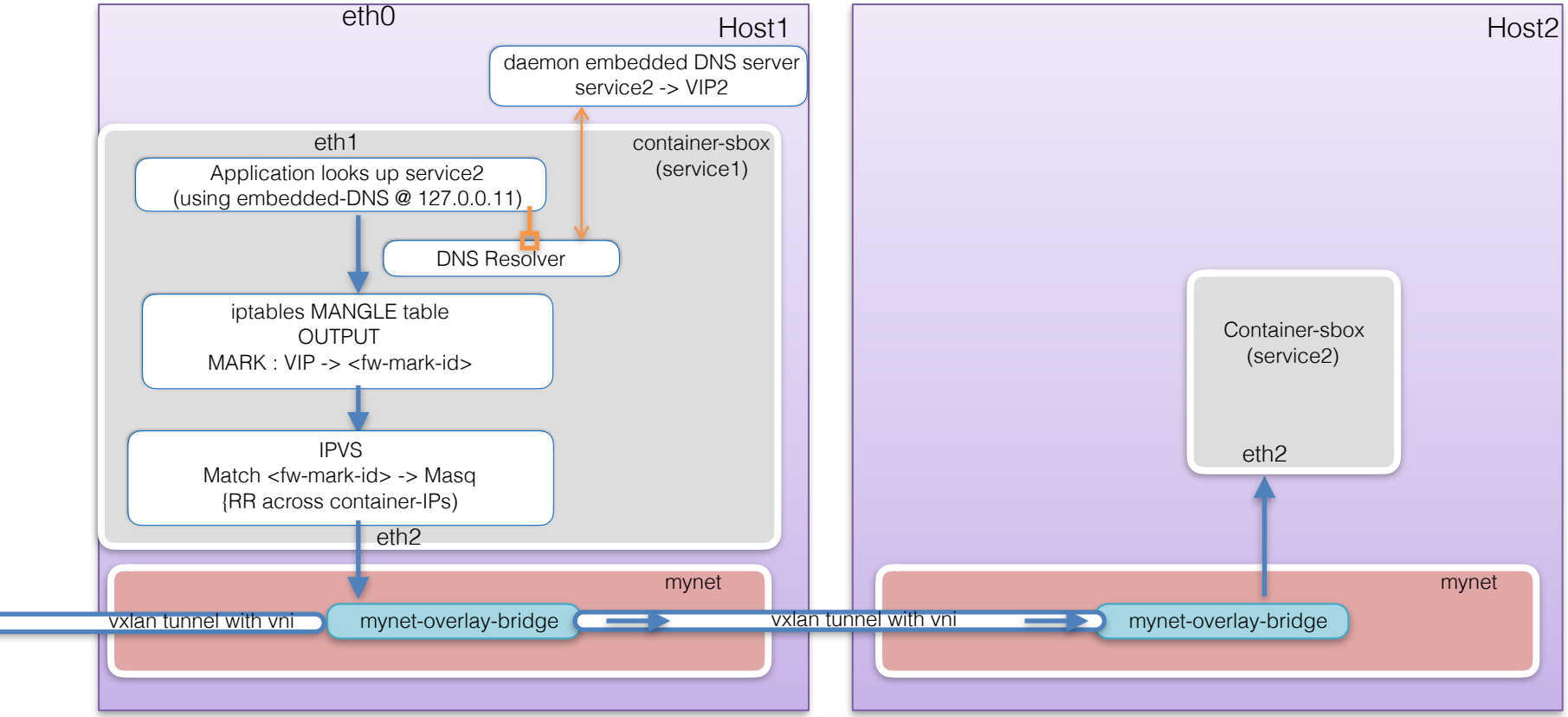


Linux Kernel Netfilter Hooks and LVS
Horms <horms@verge.net.au>, v0.1.9-1, October 2003

Day in life of a packet - Routing Mesh & Ingress LB



Day in life of a packet - Internal LB



Thank you!