VietKubers

# Deep Dive into Container Networking v1.0

Speakers:

- Nguyen Phuong An (AnNP@vn.fujitsu.com)
- Kim Bao Long (longkb@vn.fujitsu.com)
- Nguyen Hai Truong (truongnh@vn.fujitsu.com)

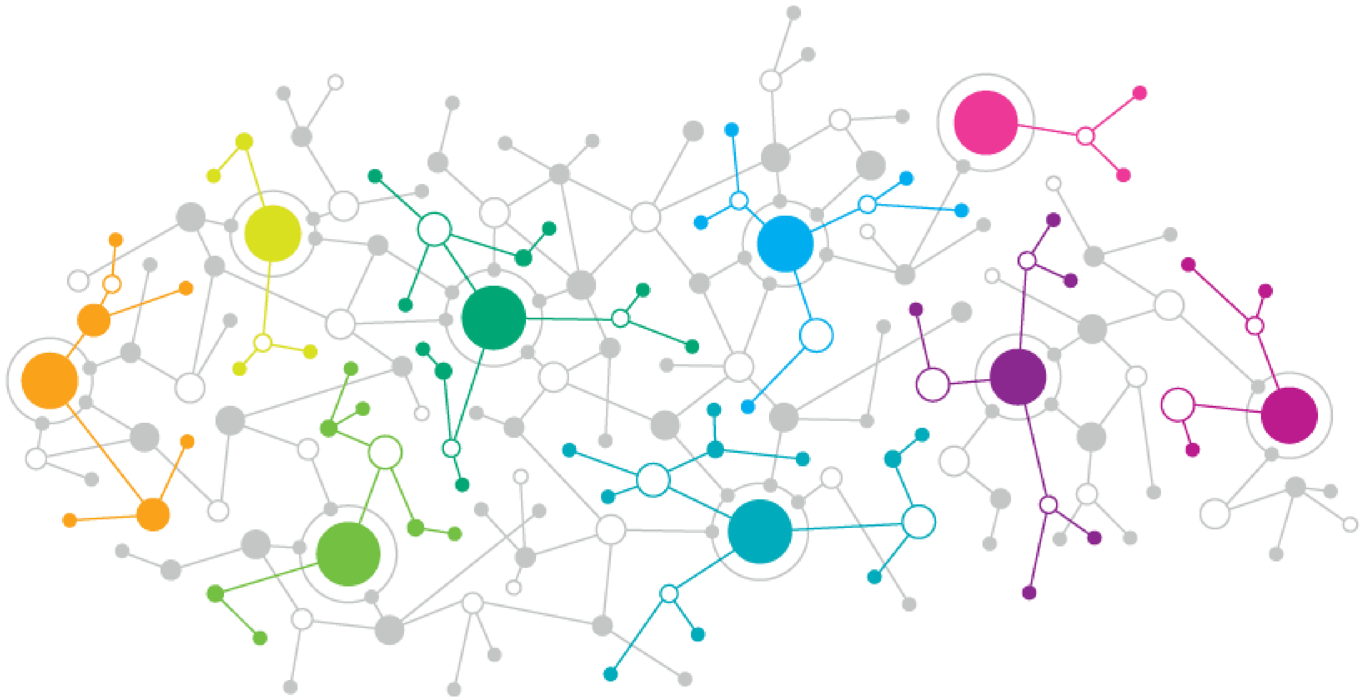
## Agenda

- $ whoarewe
- Container networking 101
  - Single host
  - Multiple host
  - Orchestration
  - Services discovery
  - Container network interface
- Kubernetes networking

## $ whoarewe

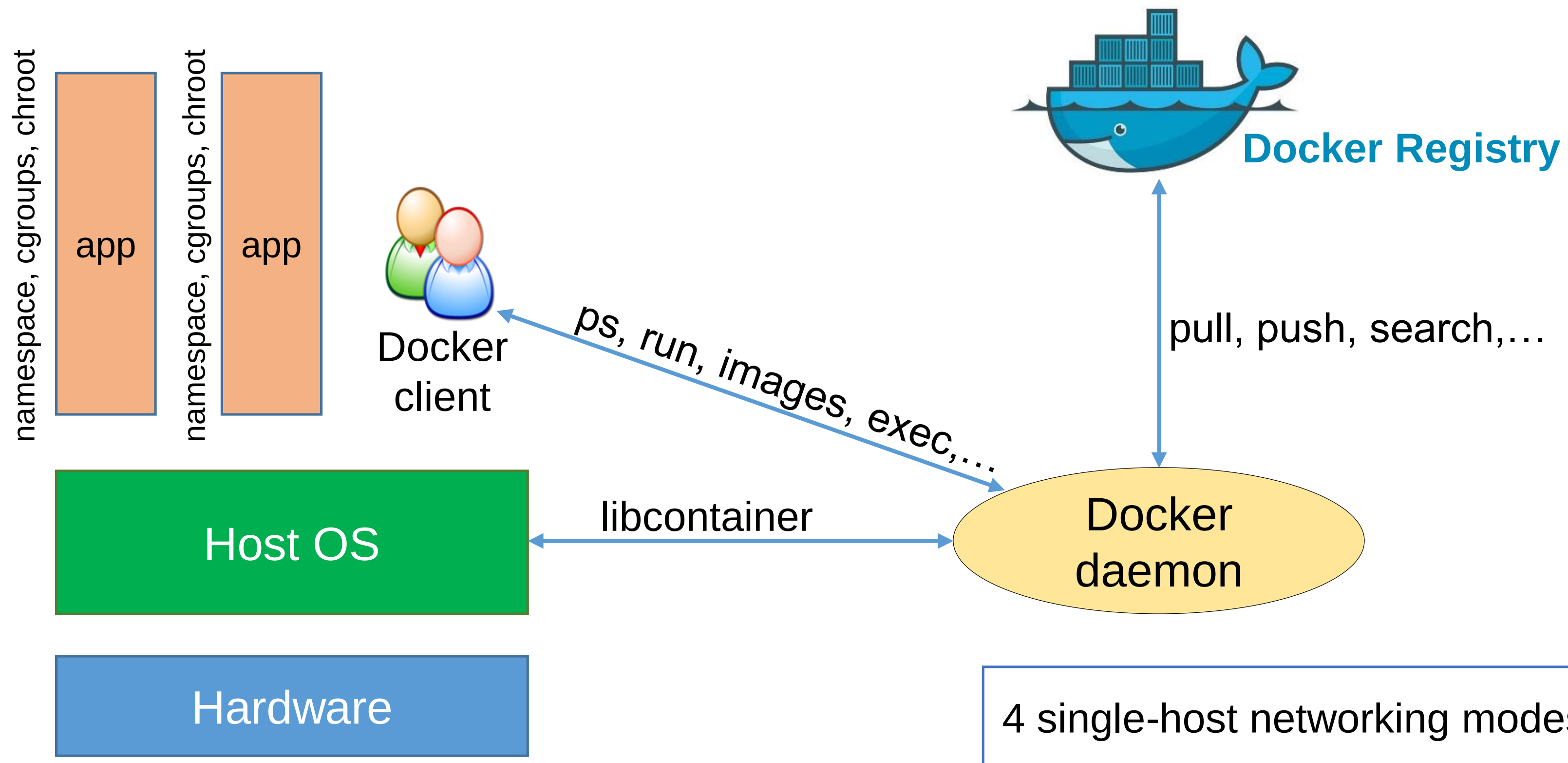
- Software engineers
- 3 years experience contribute to OpenStack Networking
  - 116 commits and 26960 LOCs
  - Neutron packet logging, firewall as a services v2.0, deploy Neutron-api under WSGI server,…
- Now, we’re moving to Kubernetes
- Twitter: @annp87, @long_kb, @truongnh92

VietKubers



# CONTAINER NETWORKING 101

# Single host

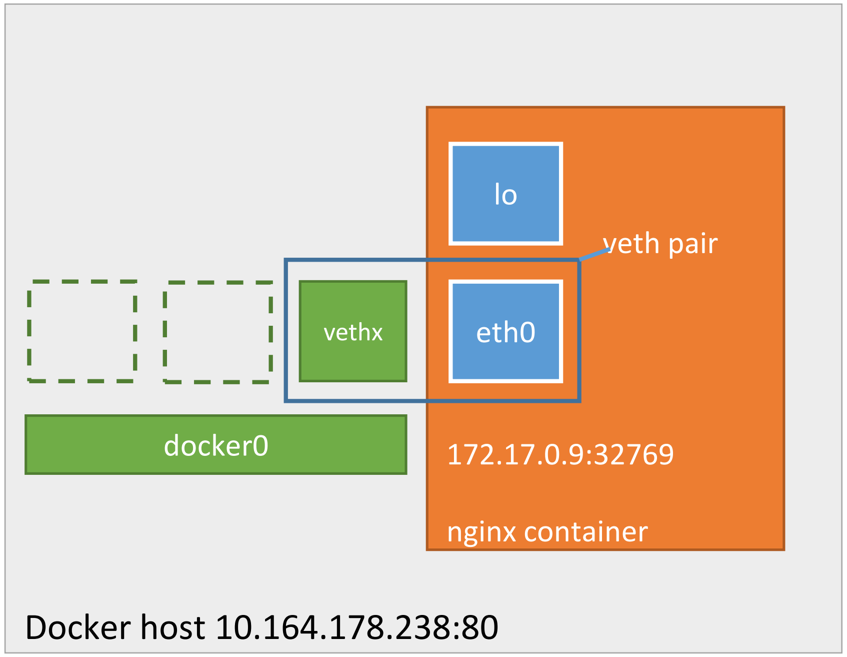


4 single-host networking modes:

- Bridge mode
- Host mode
- Container mode
- No networking

# Single host

## Bridge Mode Networking

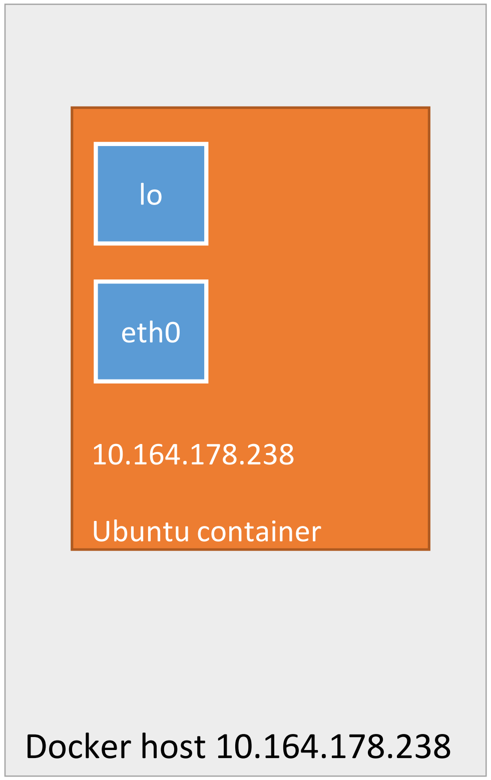


```
$ docker run -d -P --net=bridge nginx:1.9.1

$ docker ps
CONTAINER ID       IMAGE            COMMAND              CREATED
STATUS              PORTS            NAMES
cecf17dc9117      nginx:1.9.1      "nginx -g 'daemon ..."   5 minutes ago      Up 5 minutes
0.0.0.0:32769->80/tcp,
0.0.0.0:32768->443/tcp   loving_engelbart
```

# Single host

## Host Mode Networking



- Disable network isolation of a Docker container
- The container shares the network namespace of the host
- No routing overhead ➔ faster than bridge mode
- Expose the container directly to public network ➔ security implications

# Single host

## Container Mode Networking

Docker reuse the network namespace of another container

```
$ docker run -d -P --net=bridge nginx:1.9.1

$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS               NAMES
cecf17dc9117        nginx:1.9.1         "nginx -g 'daemon ..."   5 minutes ago       Up 5 minutes
0.0.0.0:32769->80/tcp, 0.0.0.0:32768->443/tcp   loving_engelbart

$ docker exec -it loving_engelbart ip addr
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:09 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.9/16 scope global eth0
       valid_lft forever preferred_lft forever

$ docker run -it --net=container:loving_engelbart ubuntu:14.04 ip addr
36: eth0@if37: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default
    link/ether 02:42:ac:11:00:09 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.9/16 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::42:acff:fe11:9/64 scope link
       valid_lft forever preferred_lft forever
```

# Single host

## No Networking

There is no networking configured

```
$ docker run -d -P --net=none nginx:1.9.1

$ docker ps
CONTAINER ID    IMAGE          COMMAND                  CREATED              STATUS              PORTS               NAMES
df7df16a84de    nginx:1.9.1    "nginx -g 'daemon ..."   About a minute ago   Up About a minute                       friendly_northcutt

$ docker inspect df7df16a84de | grep IPAddress
"SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "",
```

## Multiple host

When scaling out horizontally a cluster of machines:

- How do containers talk to each other on different hosts?
- Communication between containers and with the outside world?

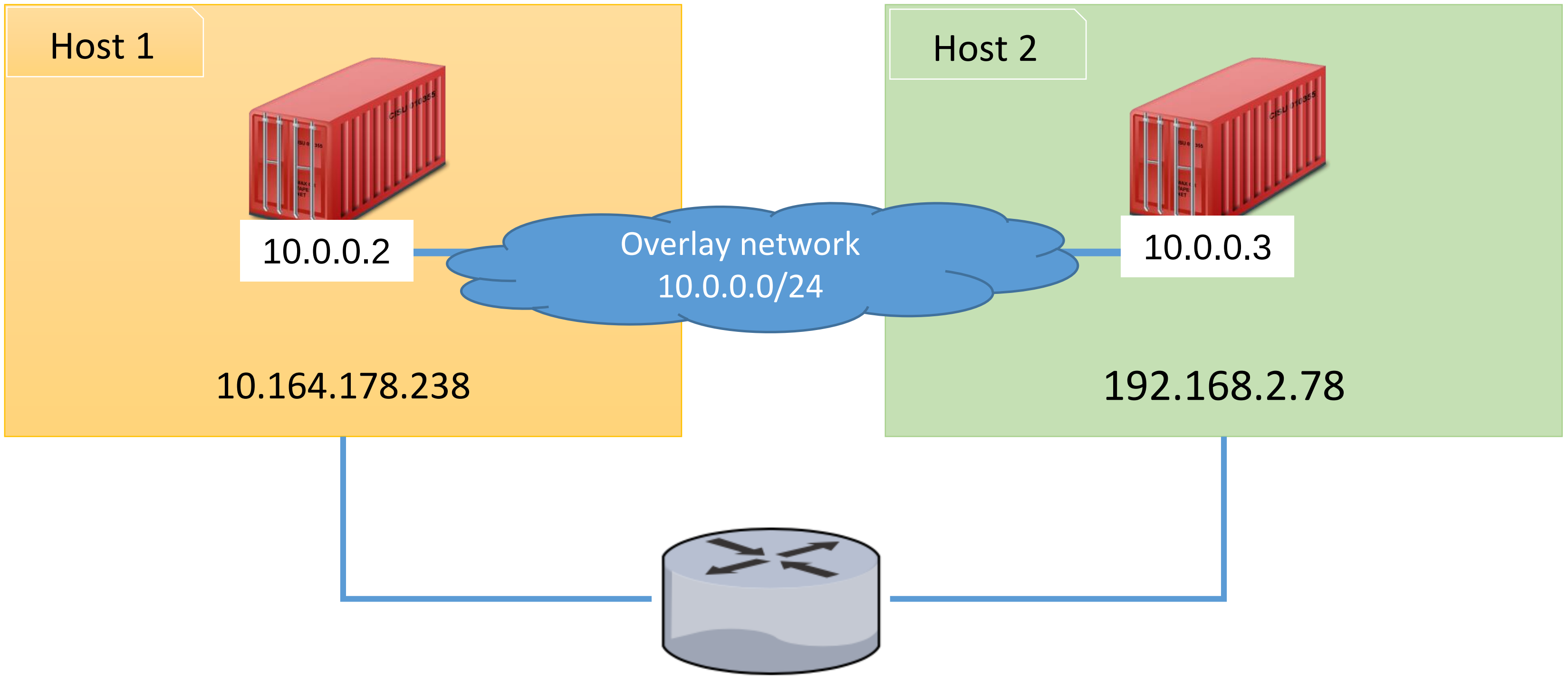
**Overlay Networks**

**Network plugins:**

- **Flannel**
- **Calico**
- **Weave Net**
- **…**

# Multiple host

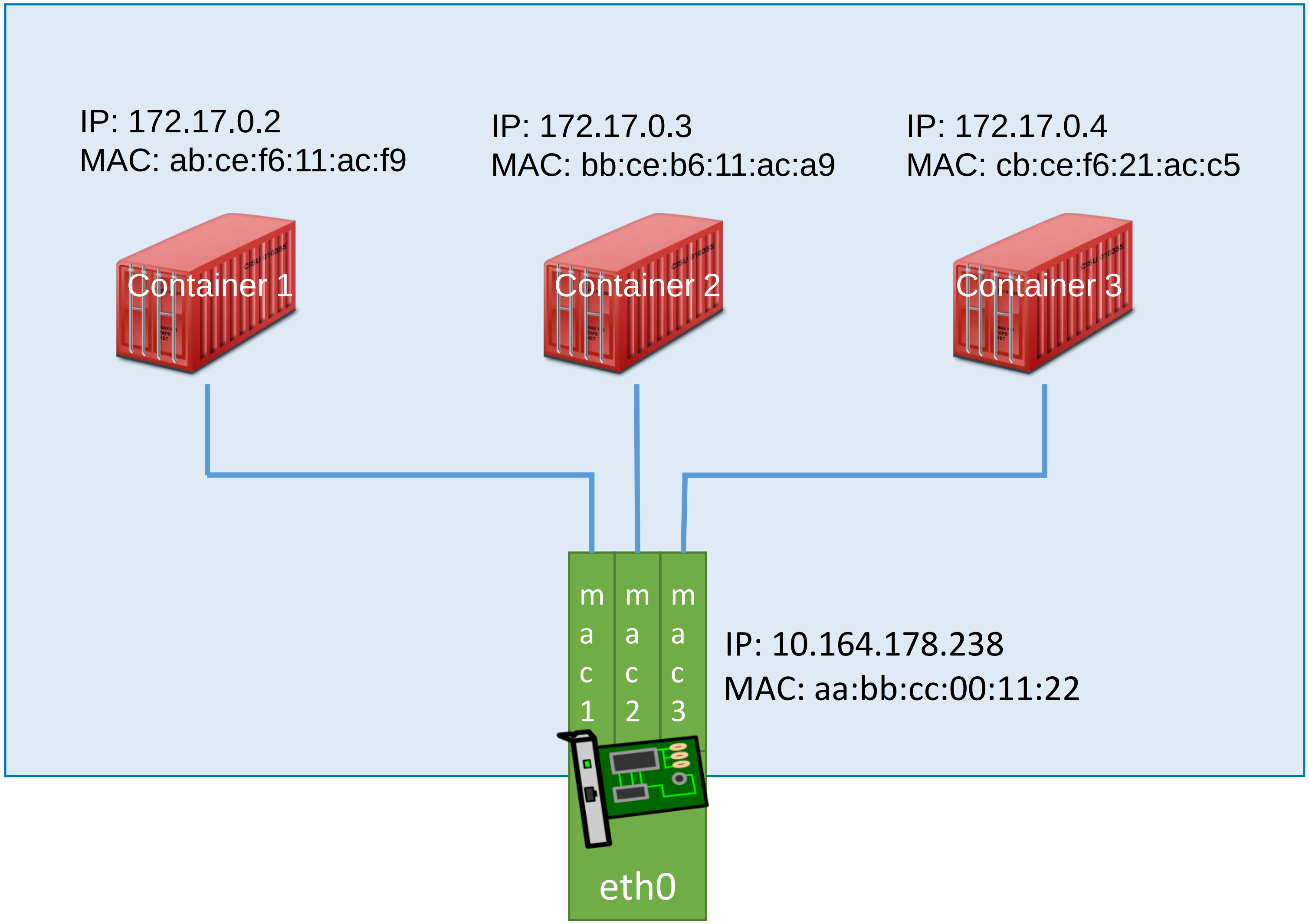
## Overlay Networking



Creating a distributed network across hosts on top of the host-specific network

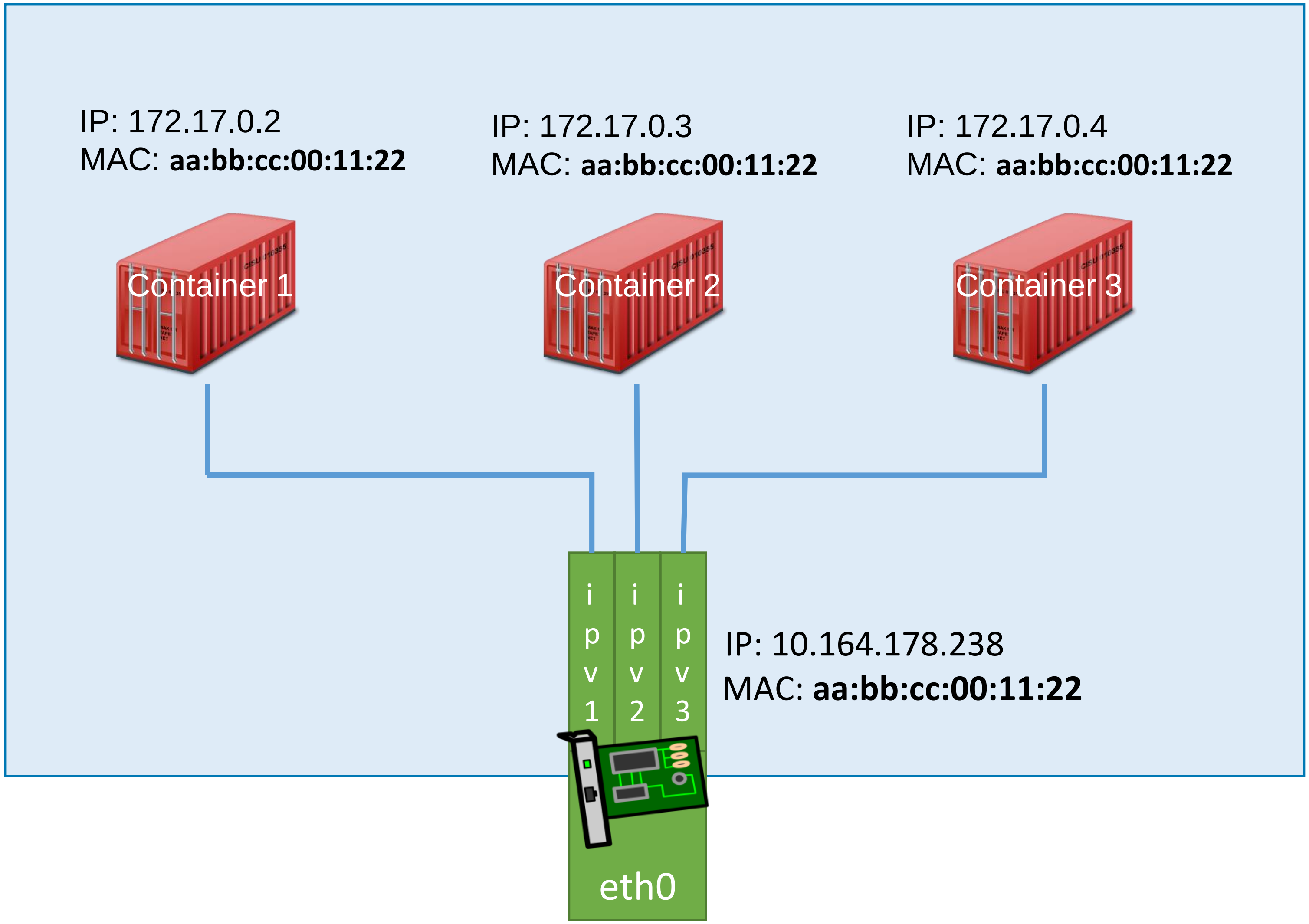
# Multiple host

## MACVLAN



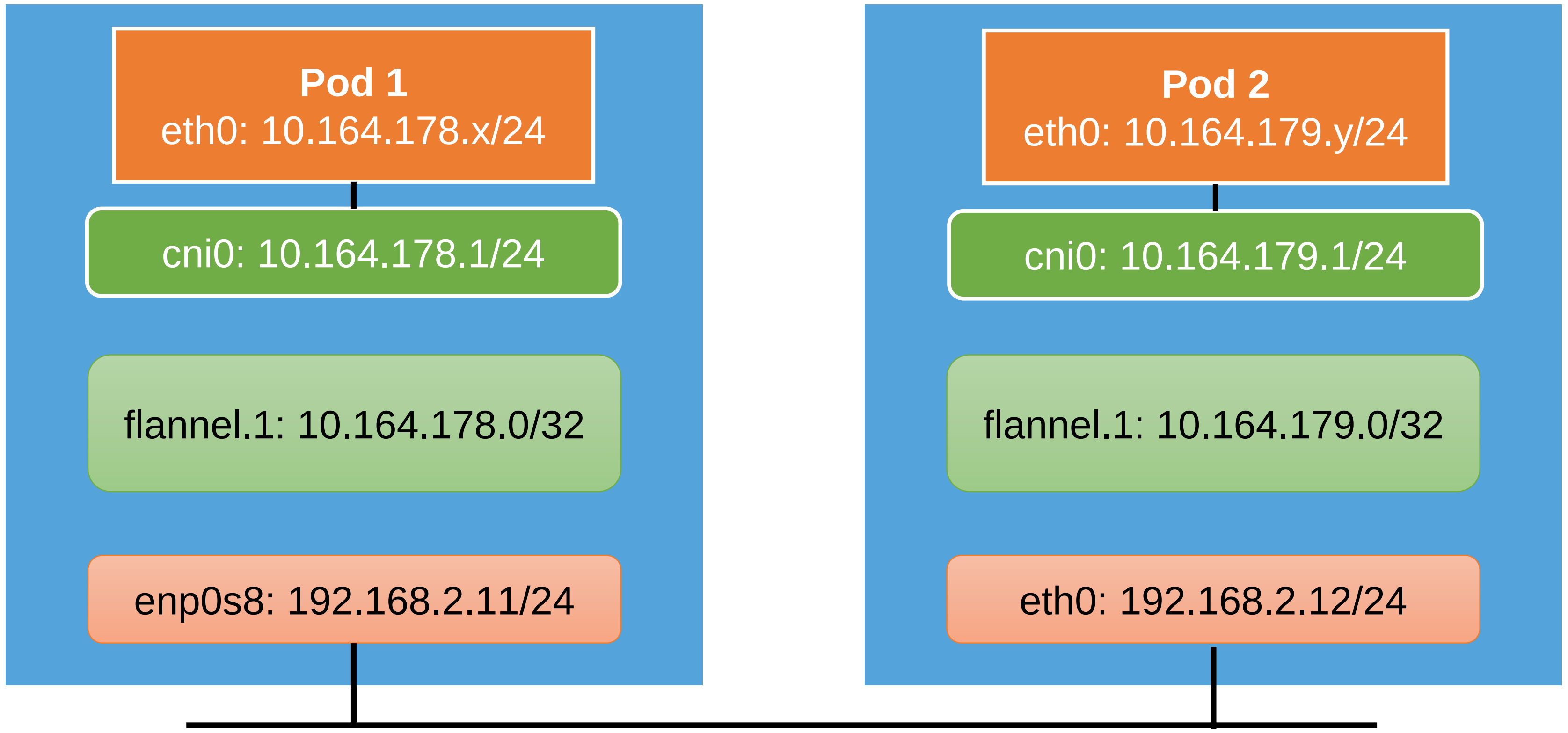
# Multiple host

## IPVLAN

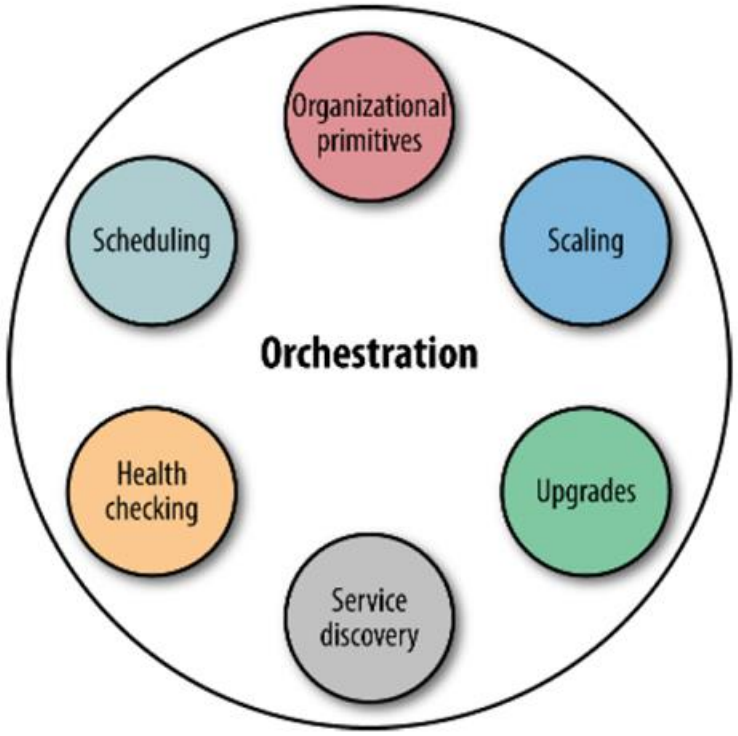


# Multiple host

flannel



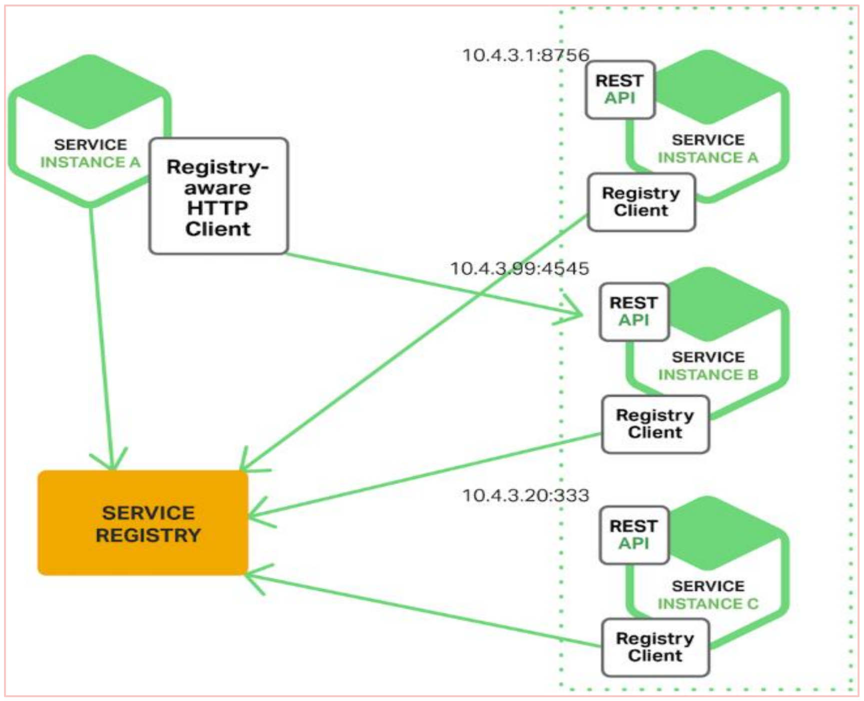
## Orchestration



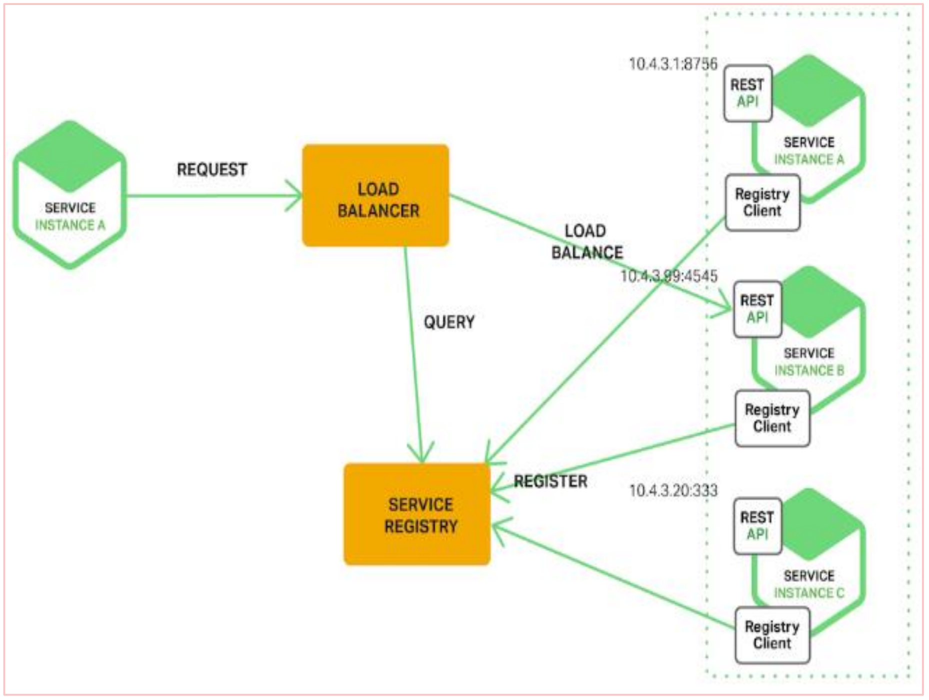
## Service discovery

- How do you determine which host your container ended up being scheduled on so that you can connect to it?
  - maintaining a mapping between a running container and its location
  - 2 operations must be supported by a container service discovery solution
    - Registration: Establishes the container -> location mapping
    - Lookup: Enables other services or applications to look up the mapping we stored during registration

# Service Discovery pattern



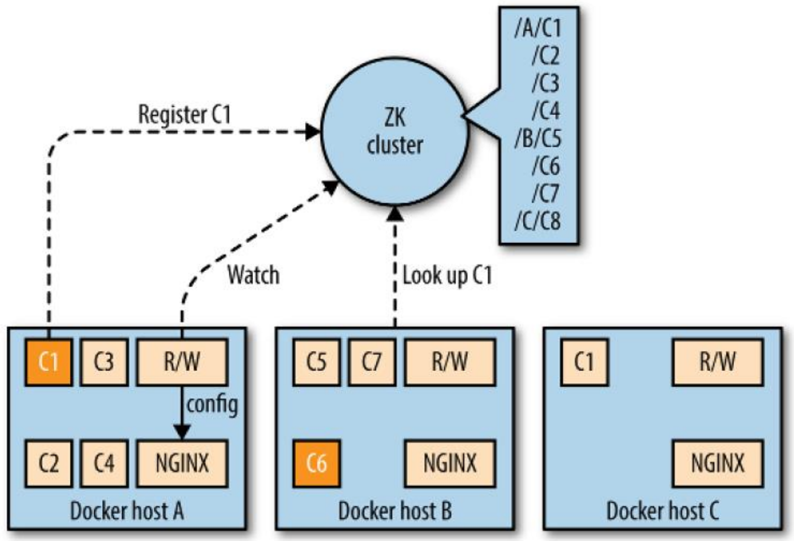
Client-side discovery pattern

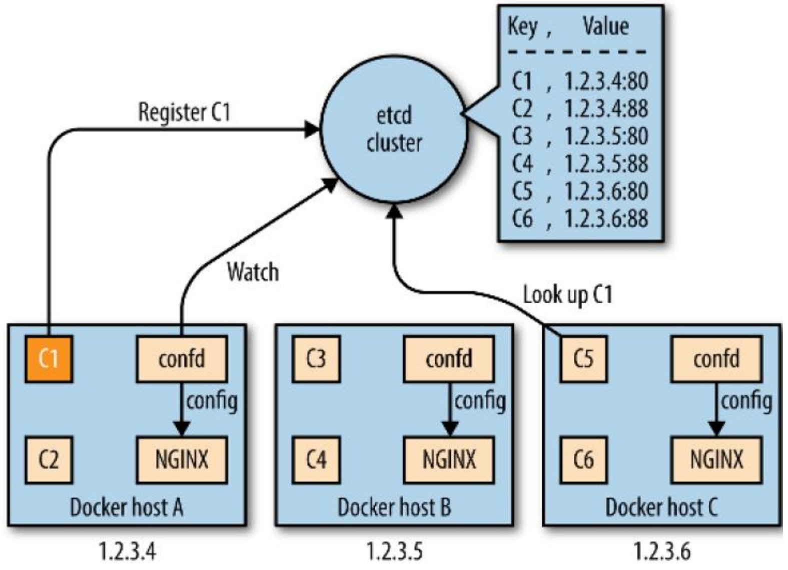


Server-side discovery pattern

# Services discovery technology

- Zookeeper
- Etcd
- Consul,
- …



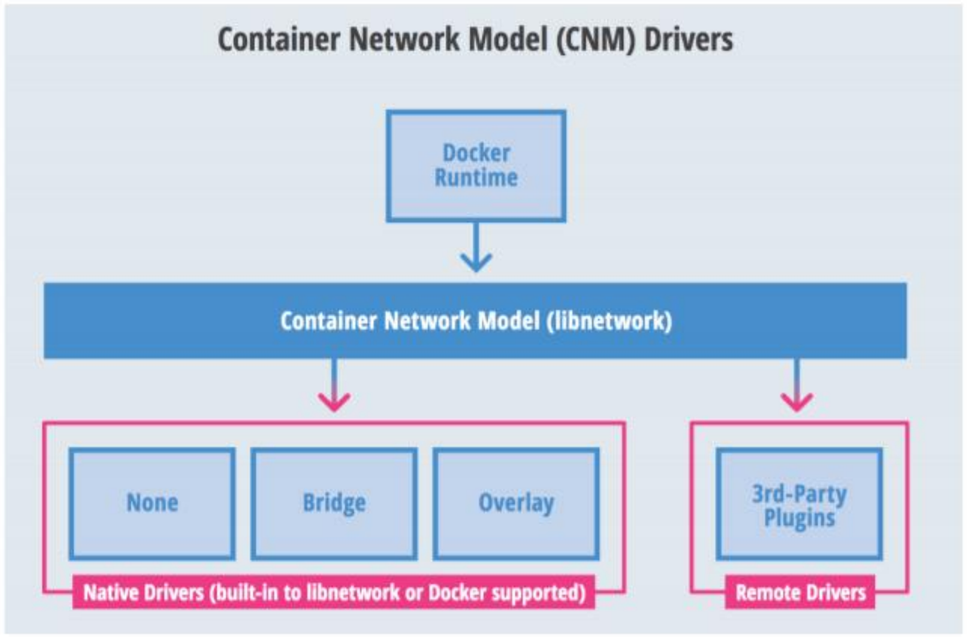


## Container network interface

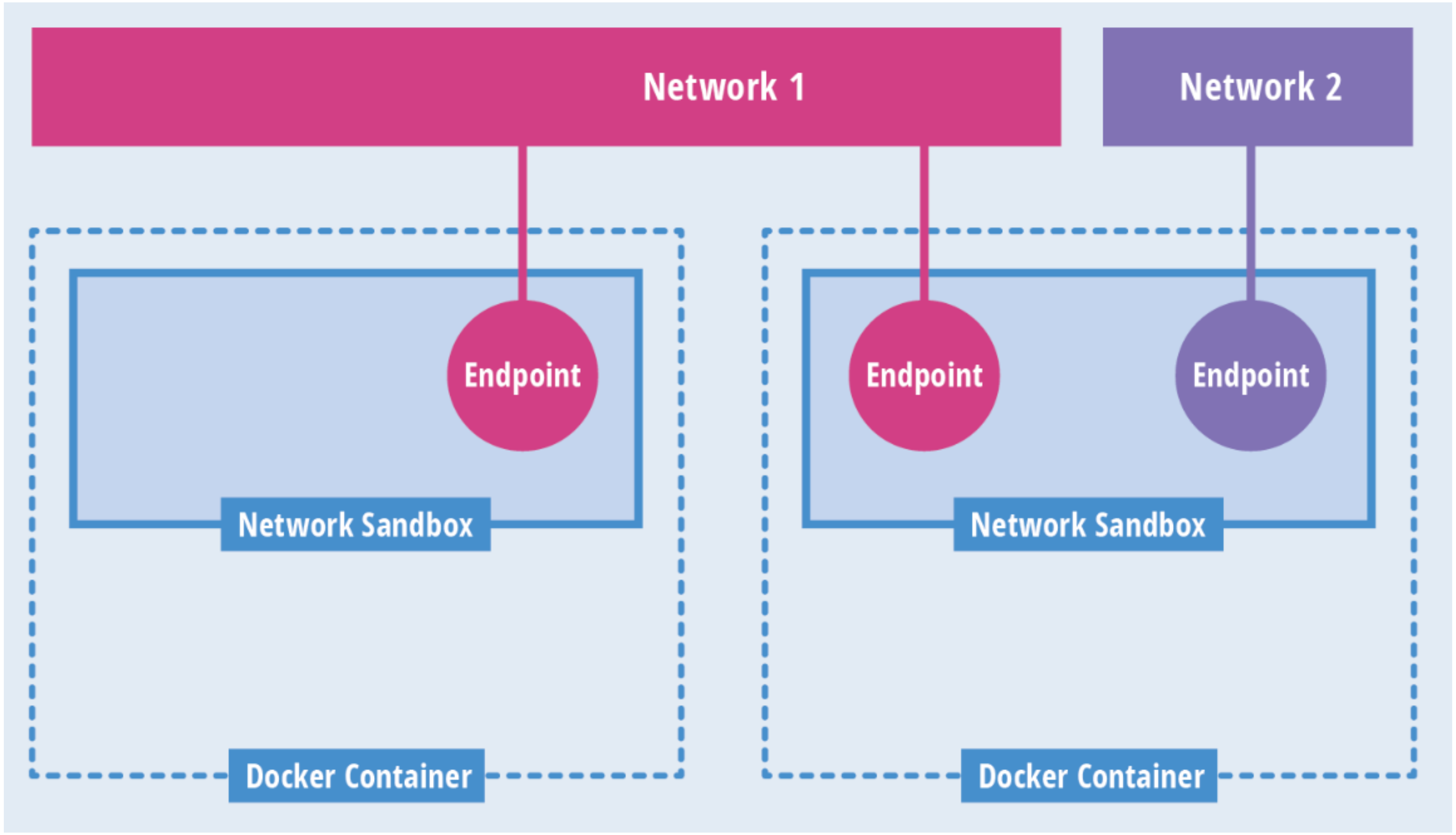
- There are two proposed standards for configuring network interfaces for Linux containers: CNM & CNI

## Container network model (CNM)

- Specification proposed by Docker and adopted by libnetwork
- Supports only Docker runtime
- Integrate with any kind of networking technology to connect and discover containers

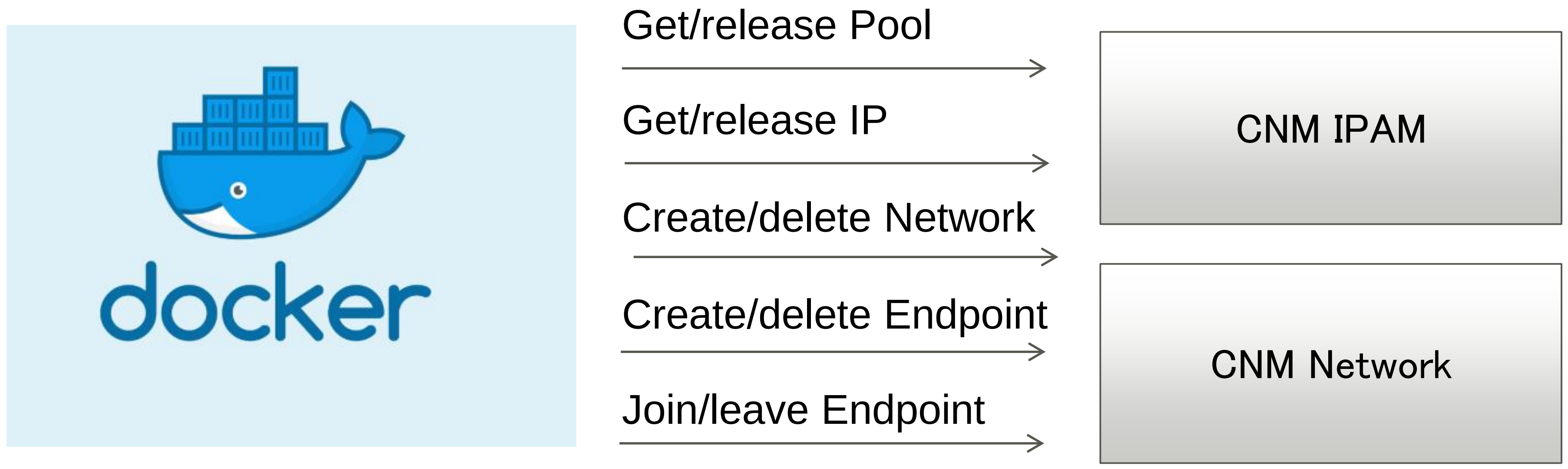

Container Network Model (CNM) Drivers
Docker Runtime
Container Network Model (libnetwork)
None
Bridge
Overlay
Native Drivers (built-in to libnetwork or Docker supported)
3rd-Party Plugins
Remote Drivers

# Container network model



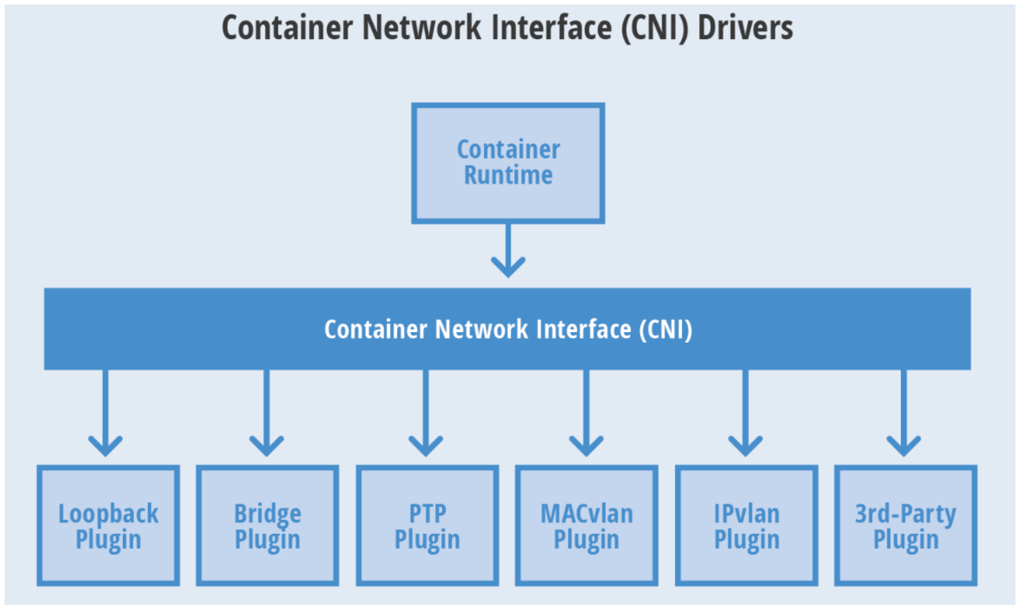
CNM topology

# Container network model



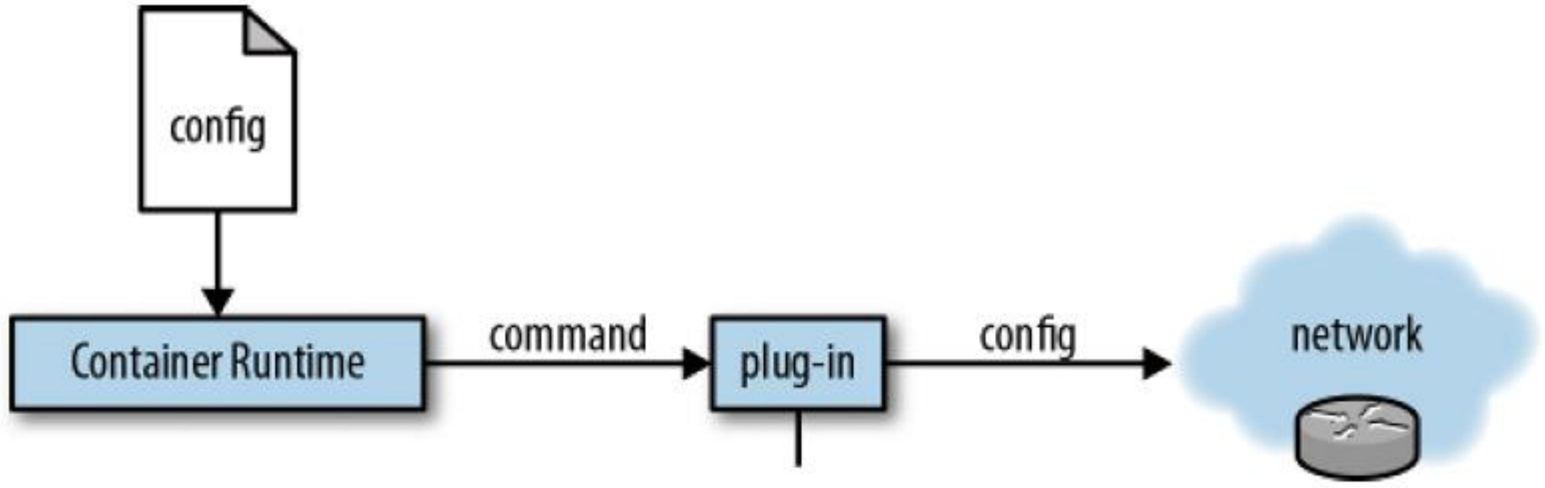
## Container network interface

- Specification proposed by CoreOS and adopted by projects such as Kubernetes
- Supports any container runtime
- Only dealing with connectivity between containers



## Container network interface

- CNI enable you to do add/remove network interfaces
- CNI defines the following operations
  - Add container to one or more networks
  - Delete container from network
  - Report CNI version

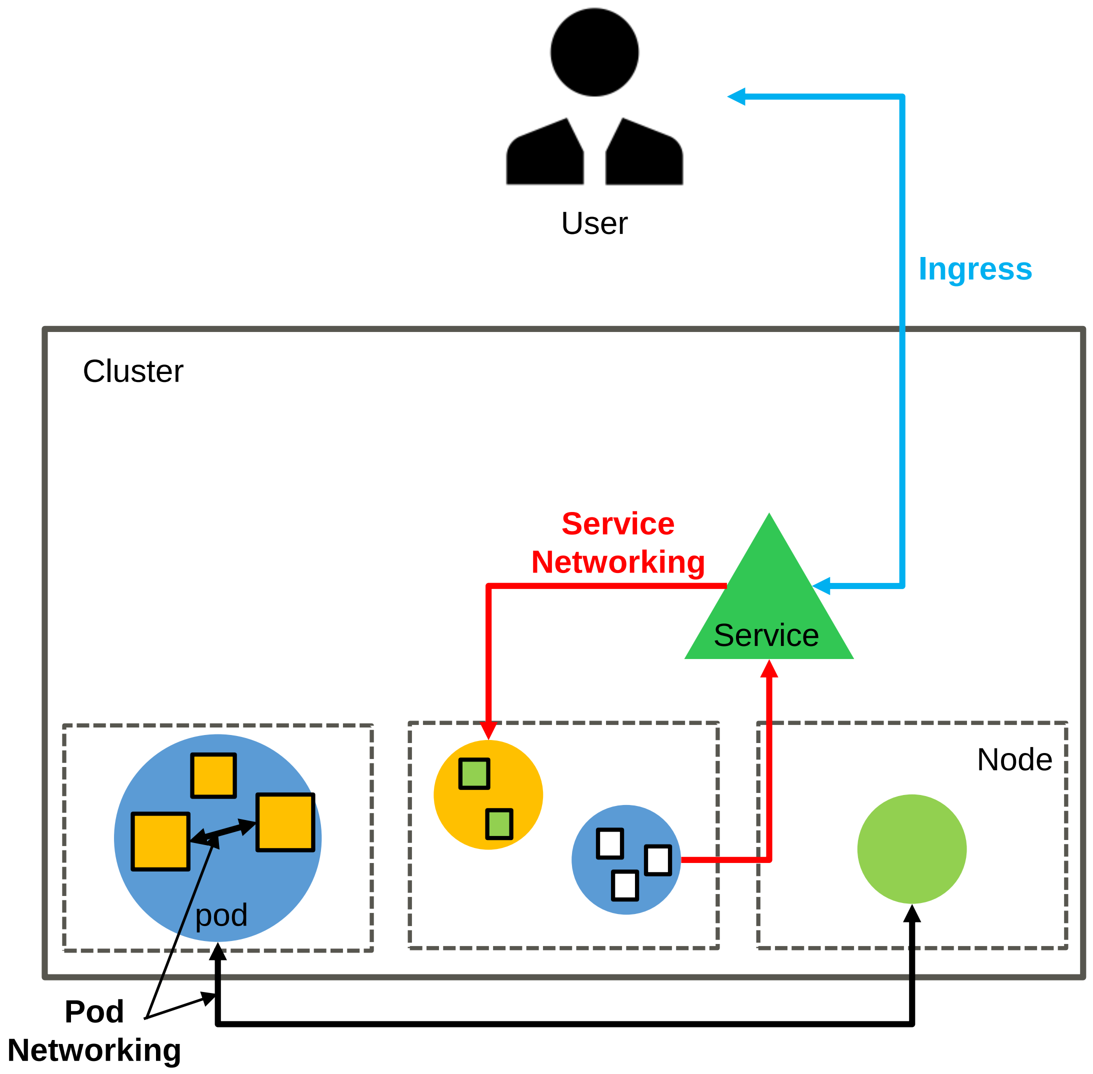


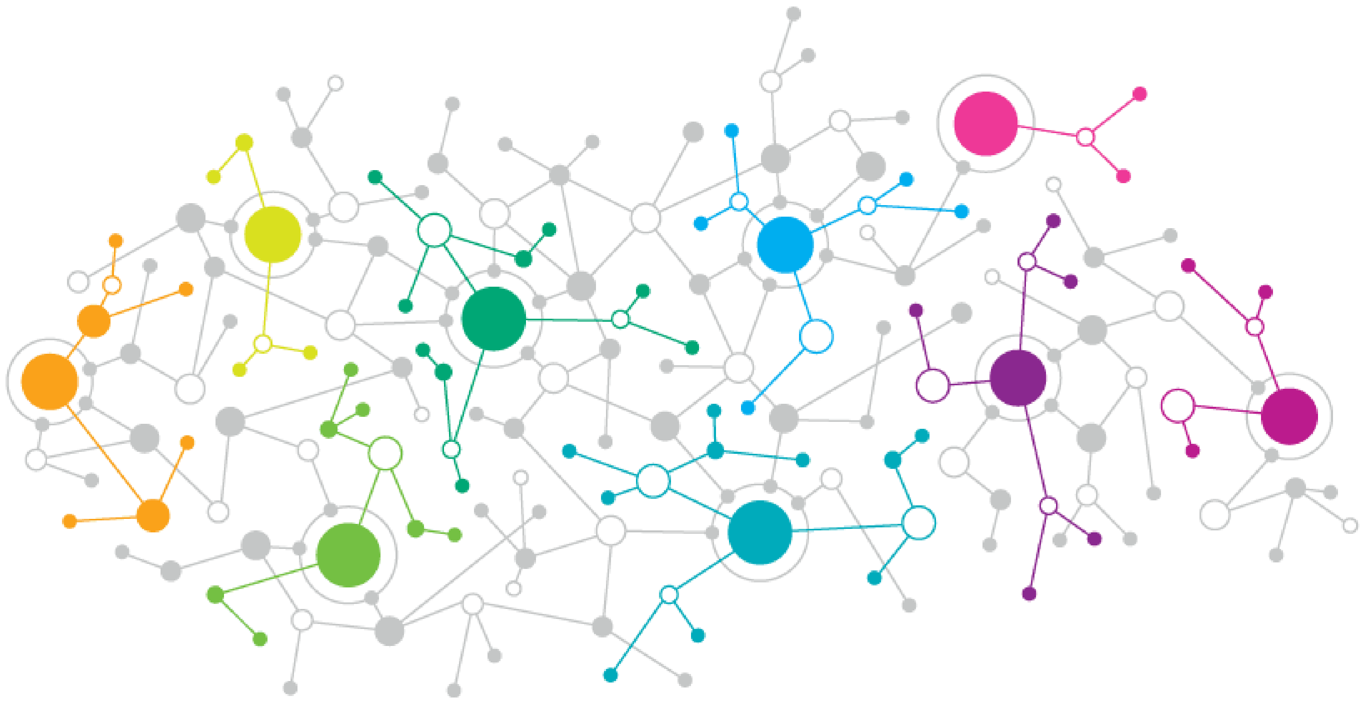
CNI flow

## KUBERNETES NETWORKING

# Kubernetes Networking

- Pod Networking
- Service Networking
- Ingress

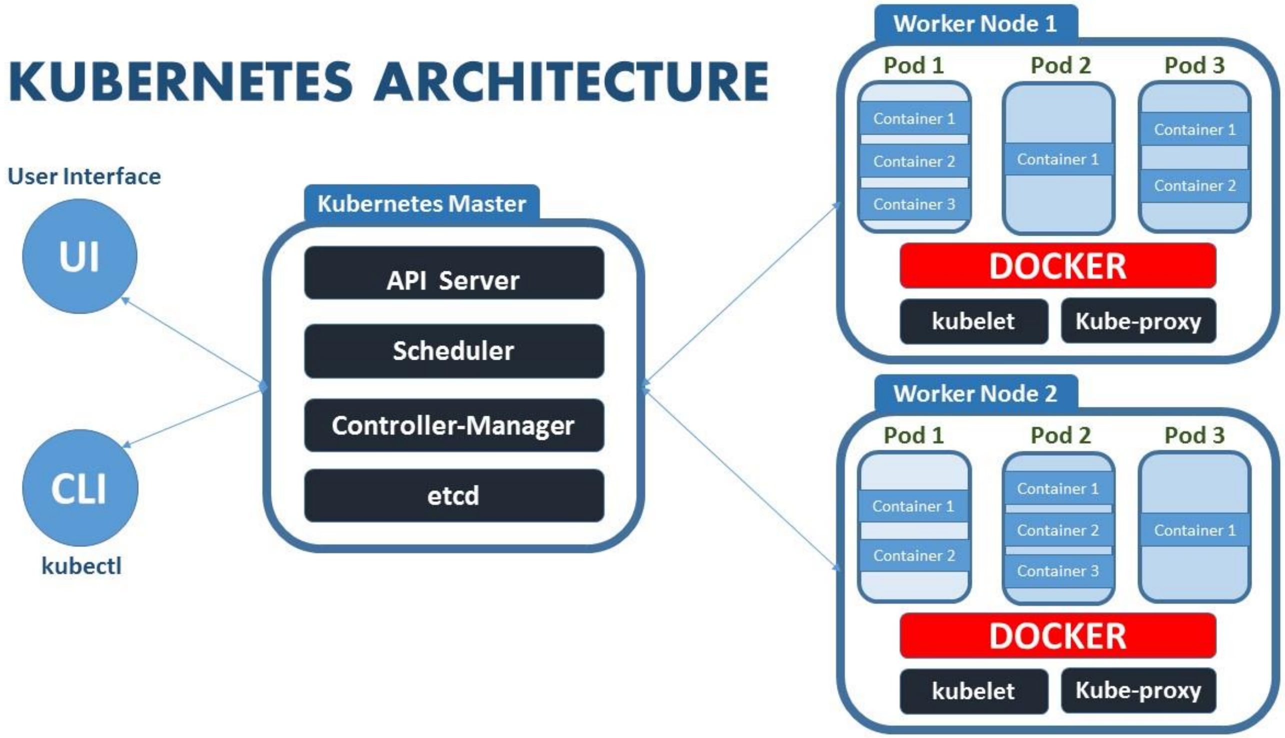




# KUBERNETES NETWORKING

## Fundamental Requirements

- Kubernetes only states three fundamental requirements
  - Containers can communicate with all others without NAT
  - Nodes can communicate with all containers without NAT
  - The IP a container sees itself is the same IP as others see it

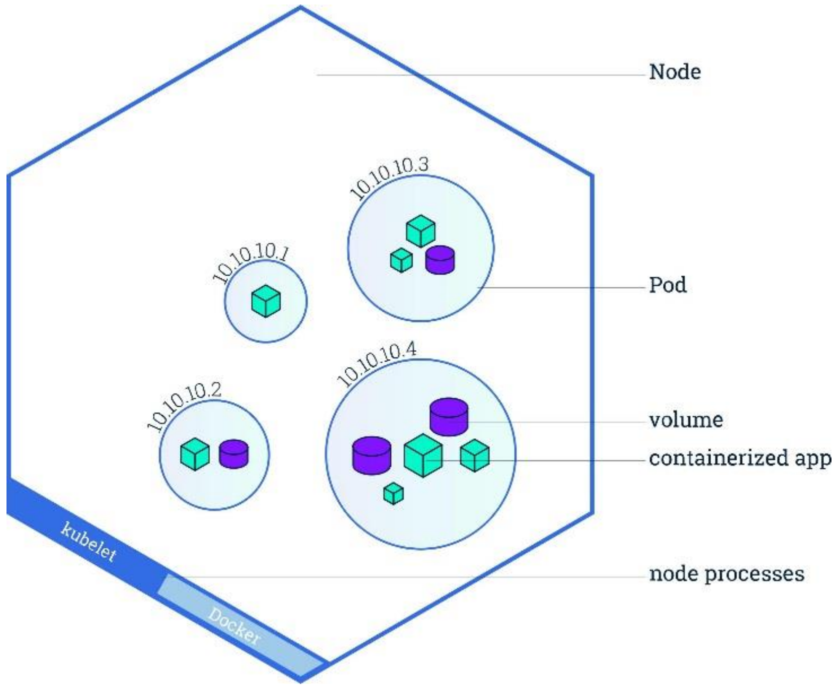


The network that enables pods to connect to each other across nodes in a kubernetes cluster

## POD NETWORKING

# Kubernetes Pod

- What is Pod?
  - **Basic unit** in K8s
  - One or more containers that are collocated on the same host, and are configured to **share a network stack** and **other resources** (volume)

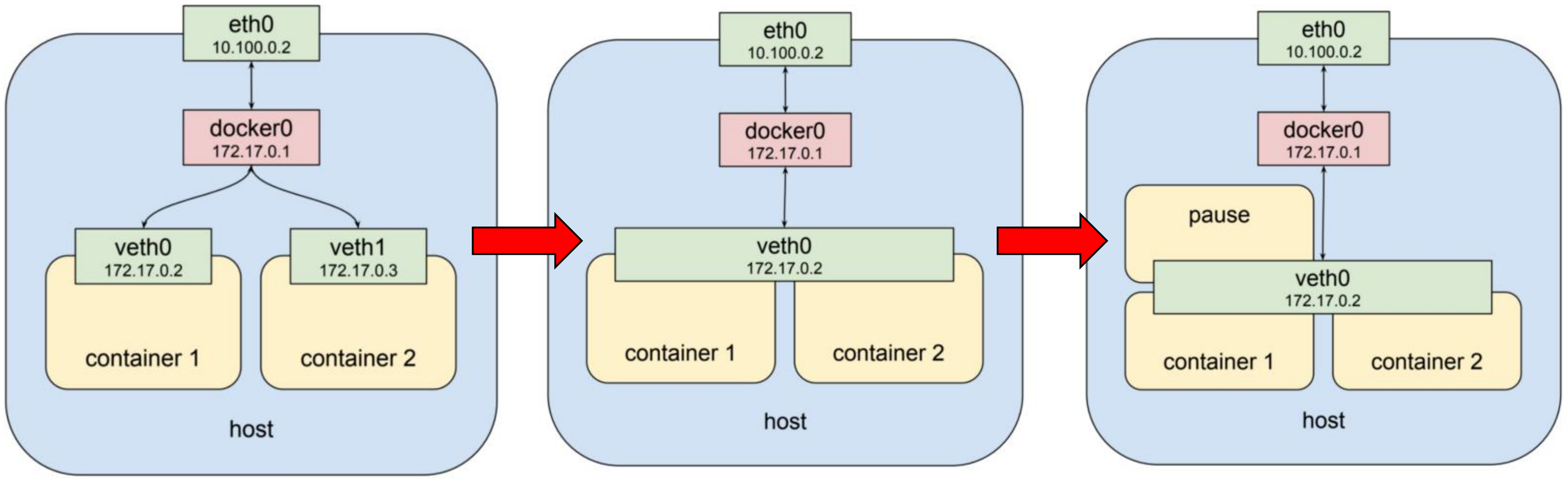


# From Docker to K8s networking

- Docker containers on local machine
  - Communicate via **docker0** bridge
  - Not **“share network stack”**

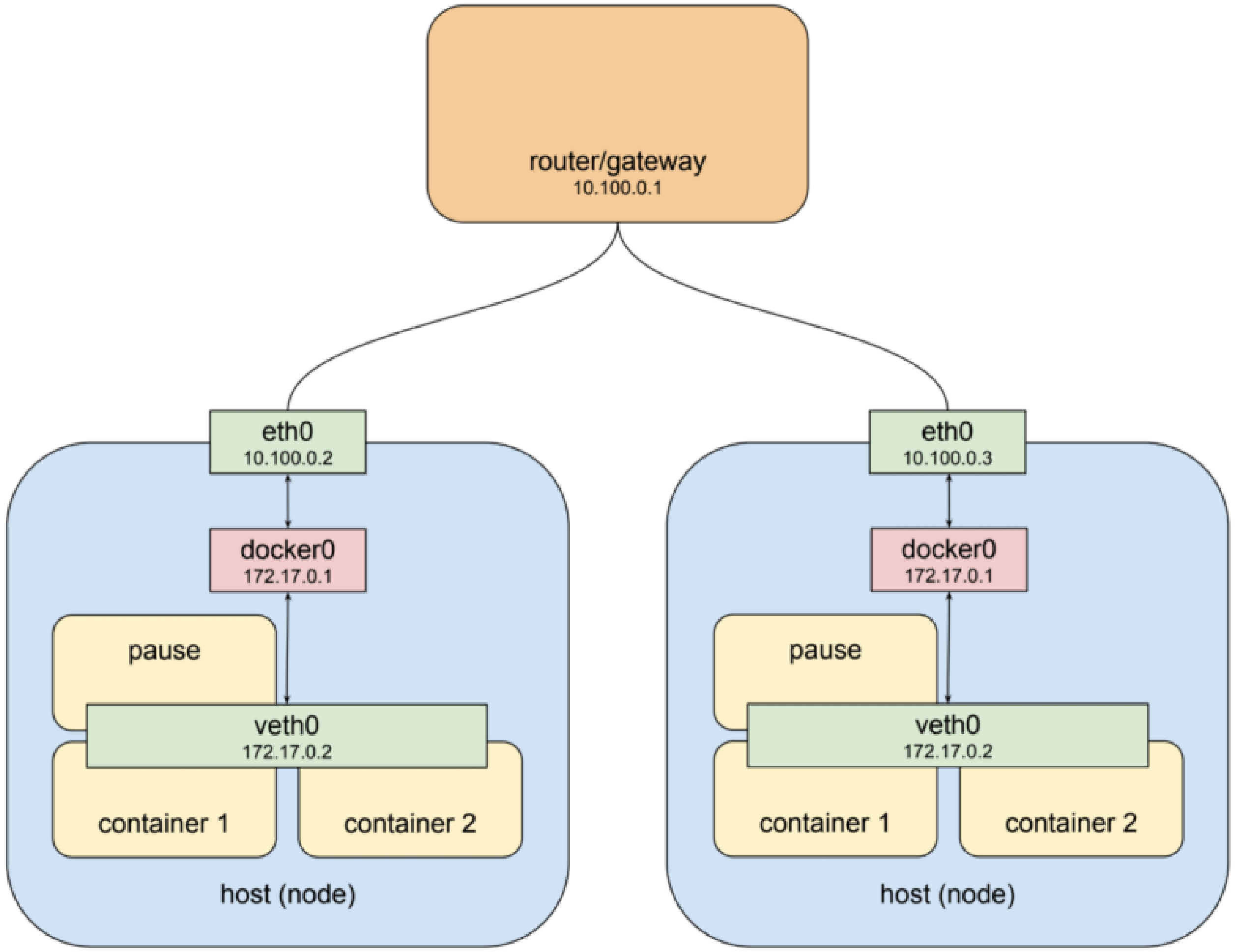
=> Share an existing interface
  - Both containers are addressable from the outside (172.17.0.2)
  - The inside each can hit ports opened by the other on **localhost**
  - **Restriction:** cannot open the same port (same with multiple processes)

- In Kubernetes
  - Using **pause** container as a heart of the pod



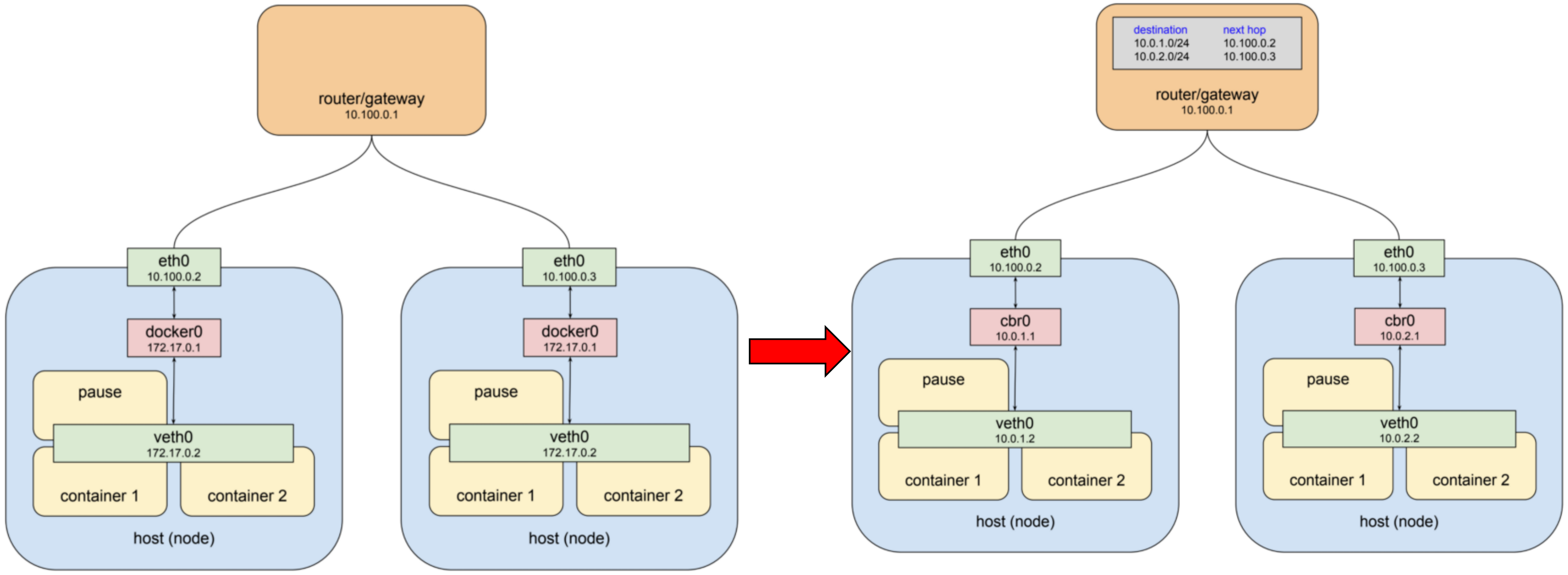
# Pod networking

- Requirement: Pods are able to communicate with the others in both single host and multi-host **without NAT**



# Pod networking

- Each pod has a routable IP
- Kubernetes built an **overlay network** by:
  - Assigns an overall address space for the bridges on each node
  - Assigns the bridges addresses within that space, based on the node the bridge is built on
  - Adds routing rules to the gateway telling it how packets destined for each bridge should be routed
- pods can communicate **without NAT**



How the service network provides load balancing for pods so that clients inside the cluster can communicate with them reliably

## SERVICE NETWORKING

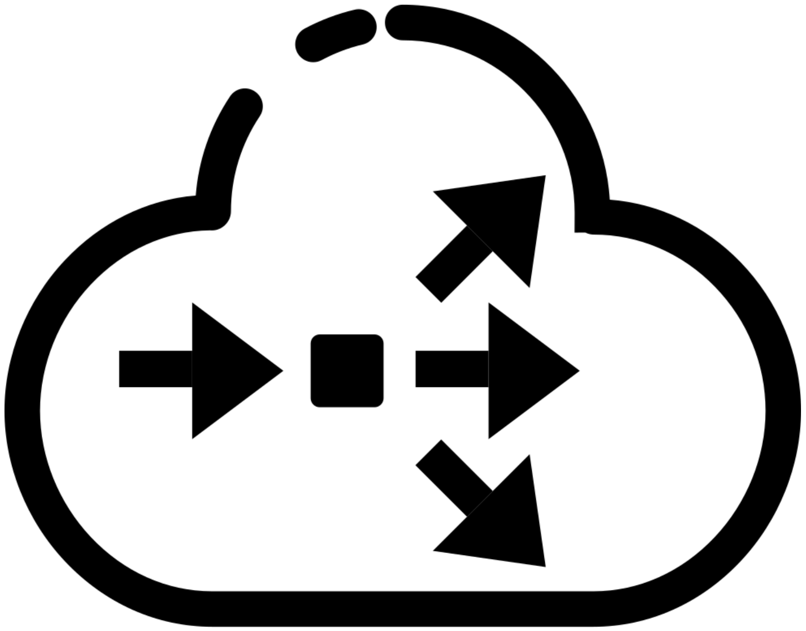
# Kubernetes Service

- **Problem**: pods in kubernetes are ephemeral
  - Auto-scaling
  - Auto-healing ➡ **Address changing**
  - Migration

=> Standard solution

- Clients connect to the proxy
- The proxy responsibilities:
  - Durable and resistant to failure
  - Maintain a list of healthy servers
  - Having a keep alive mechanism

=> Service



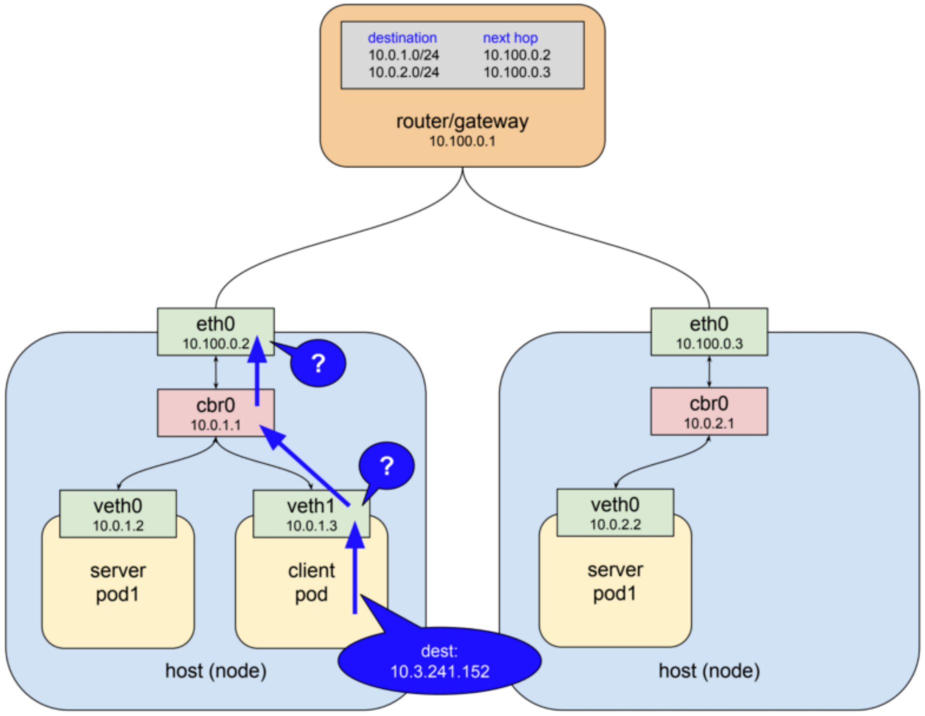
## Service Networking

- A service
  - A type of kubernetes resource
  - Having a configured proxy that can forward requests to a set of pods
- Service’s IP
  - Belong to “Service Network”
  - Different with pod’s network and node’s network
    - Services needed their own, stable, non-conflicting network address space
    - A system of virtual IPs is not stable
  - The service network **does not exist**

??? How could the request reach the running pod via service’s IP ???

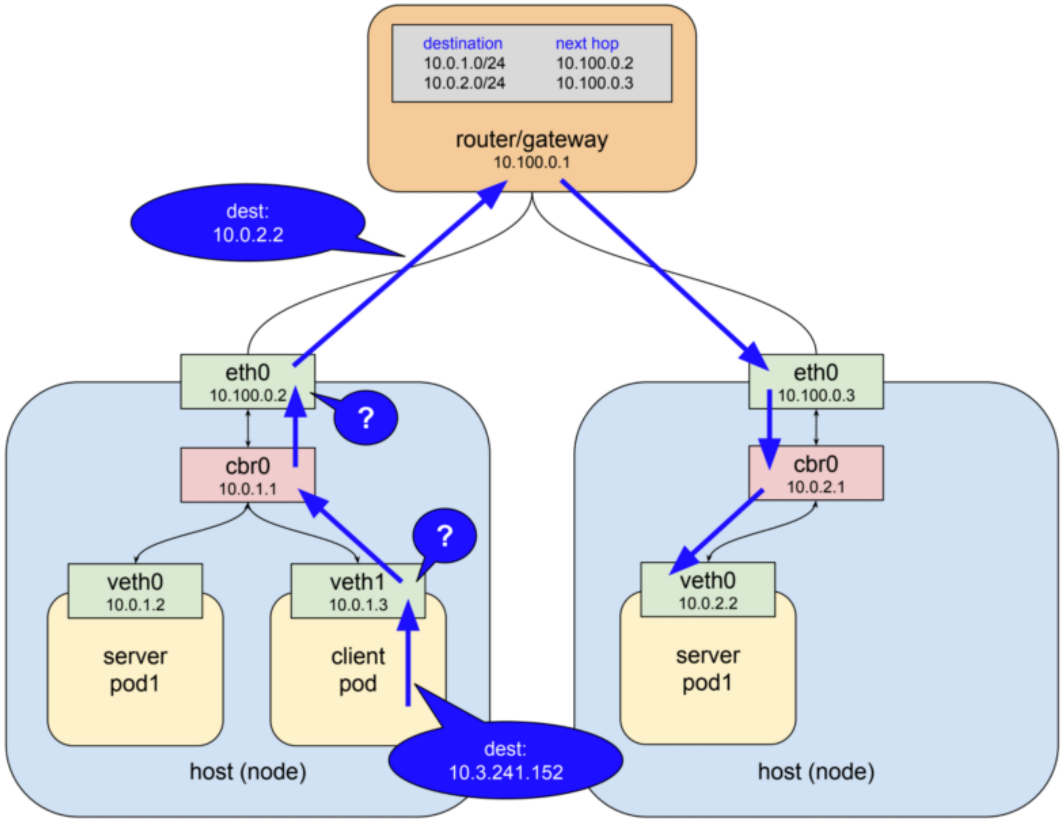
# Service Networking

- The client make an HTTP request to the service via DNS name
- The DNS system resolves service's name to cluster IP
- The client creates HTTP request to cluster IP
- The router/gateway does not know about cluster IP
- ???



# Service Networking

- Strategy: upstream gateway for forwarding unknown address



- How do those packets popped out at the right place in cluster

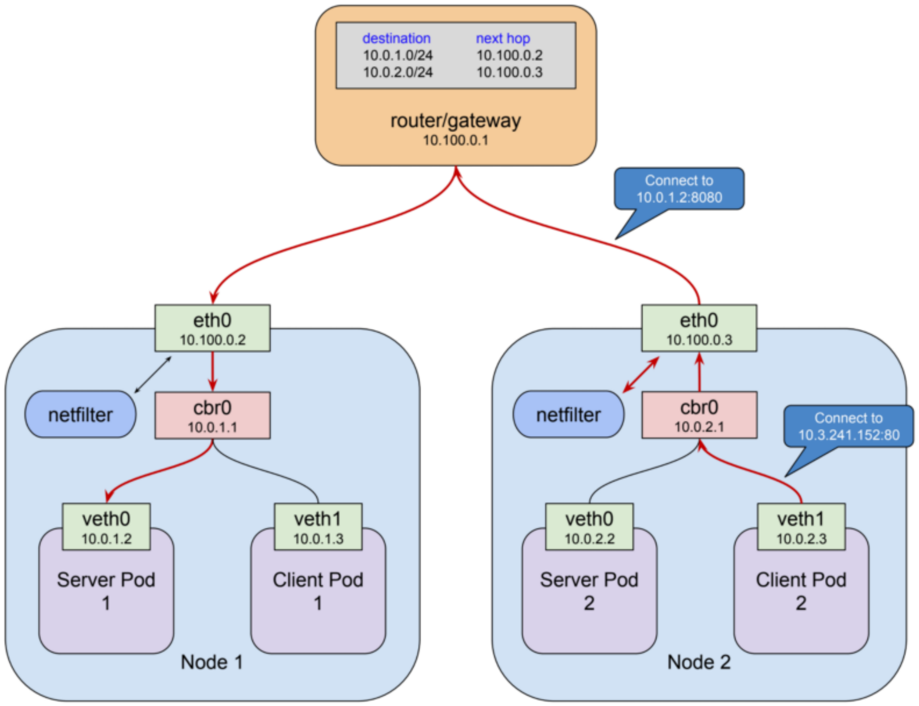
=> The magician here called **kube-proxy**

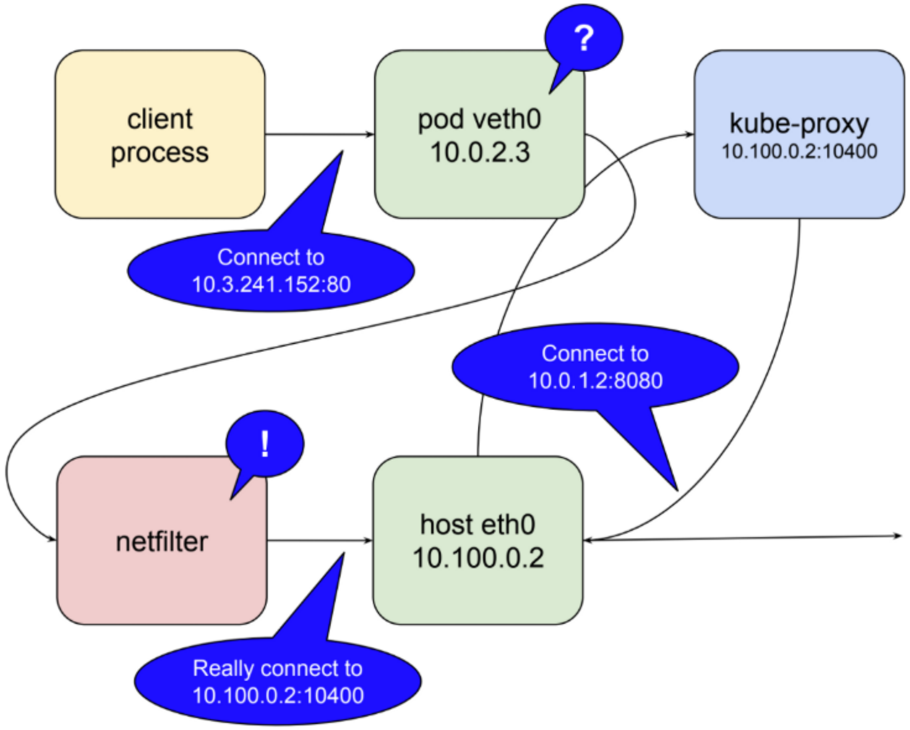
## Service Networking

- What is **kube-proxy**
  - A kubernetes resource
  - A record in a central database
  - Quite different from a typical reverse-proxy
- **Issue:** How can we listen on a port or open a connection through an interface that doesn't exist???
  - The host's ethernet interface
  - The virtual ethernet interfaces on the pod network
- **Solution**
  - Using **netfilter** as a **kernel space proxy**
    - Running in kernel space and gets a look at every packet at various points
    - Redirecting the packet to another destination

# Service Networking

- Kube-proxy
  - Open a port (10400) on the node interface to listen for requests to the registered **service**
  - Inserts netfilter rules to reroute packets destined for the service IP to its own port
  - Forwards those requests to a pod on a service port

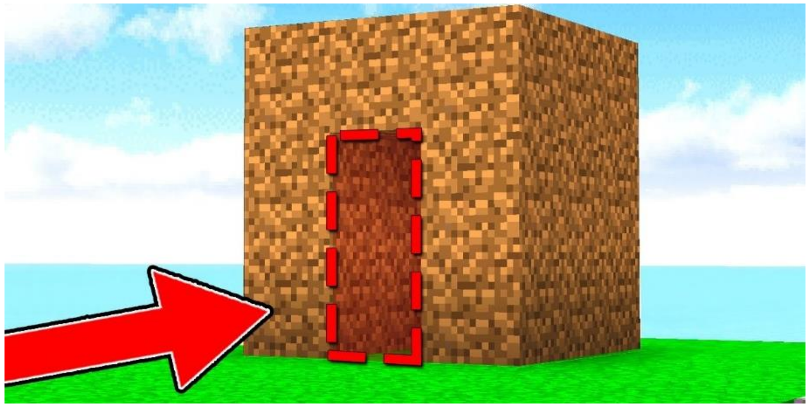




client process
pod veth0 10.0.2.3
?
kube-proxy 10.100.0.2:10400
Connect to 10.3.241.152:80
Connect to 10.0.1.2:8080
netfilter
!
host eth0 10.100.0.2
Really connect to 10.100.0.2:10400

## Service Networking

- Is the services proxy system **durable**?
  - Running as a system unit => it will be restarted if it fails
- Is the service proxy aware of **healthy server pods**?
  - kube-proxy listens to the master **apiserver** for changes in the cluster (service, endpoint)
    - Service creation => kube-proxy => create the necessary rules
    - Serivce deletion => kube-proxy => remove the related rules
  - Health-check are performed by kubelet
    - Unhealthy endpoints => apiserver => kube-proxy => netfilter rules edition
- **Only working for request that originate from inside the cluster**

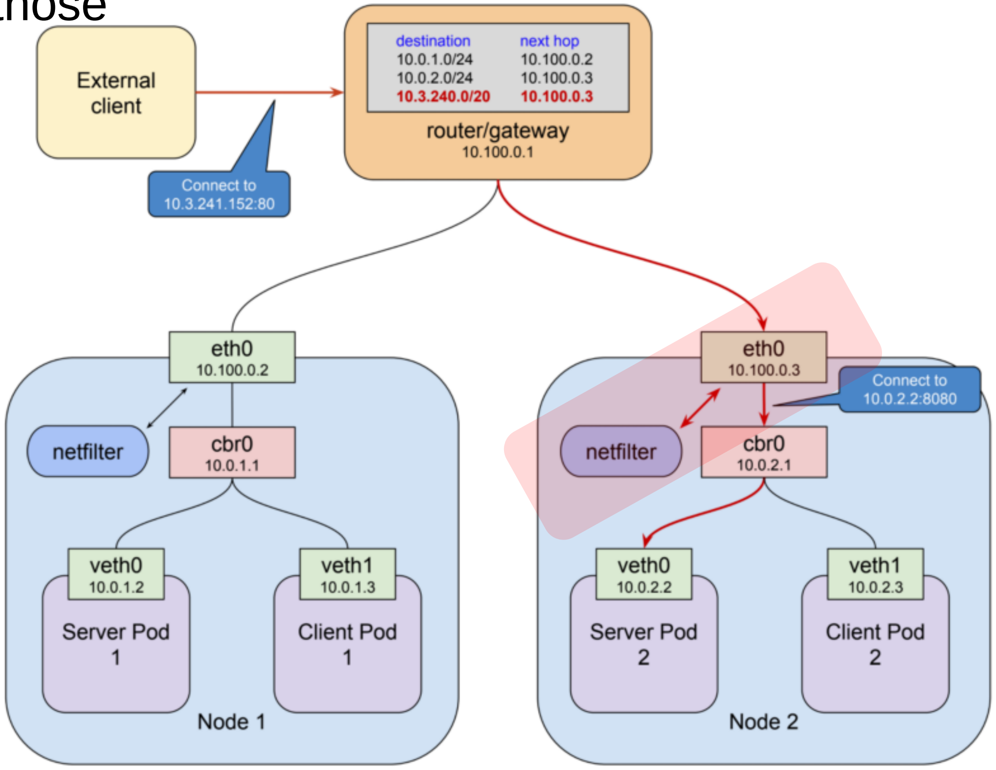


How clients *outside* the cluster can connect to pods using the same service network

## INGRESS

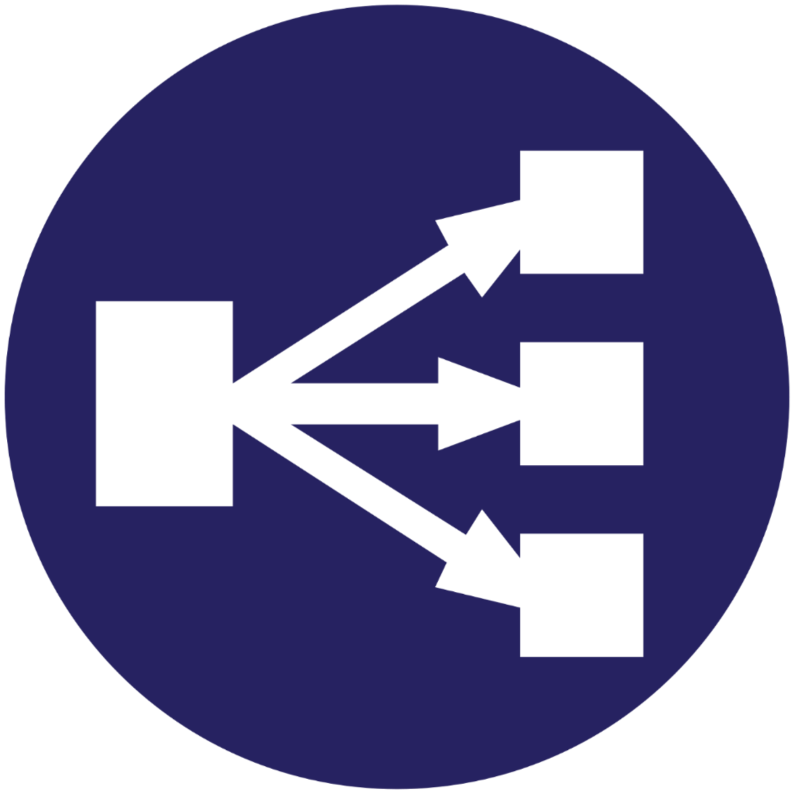
# Ingress

- Try to make use of the routing infrastructure
  - Calling ClusterIP and Port
  - Problem: The ClusterIP only reachable from node interface
  - ⇒How can we forward traffic from a public IP to an IP that is only reachable from node?
    - ⇒Should we add a route to get those packets to one of the nodes?
      - - It works
      - - Not stable
      - - Not optimal



# Ingress

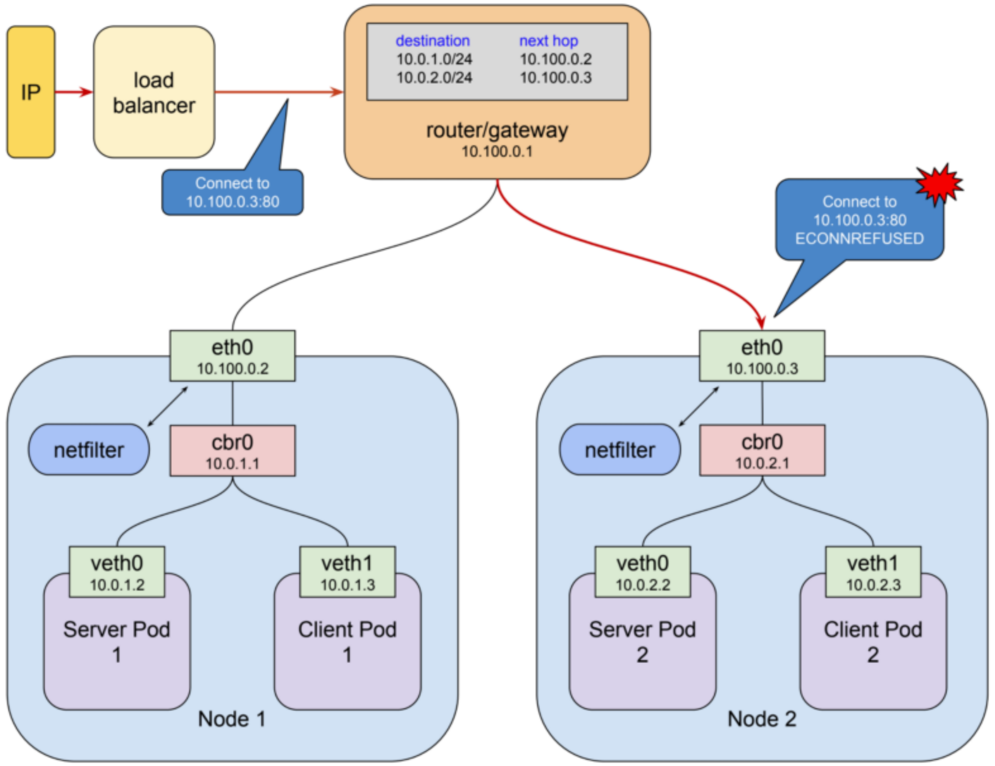
- Solution



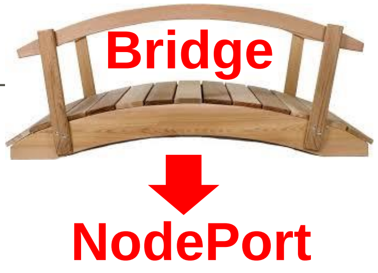
**Load Balancer**
$\left\{ \begin{array}{l} \text{- Public IP} \\ \text{- Node's IP} \end{array} \right.$

# Ingress

- No process listening on **10.100.0.3:80**

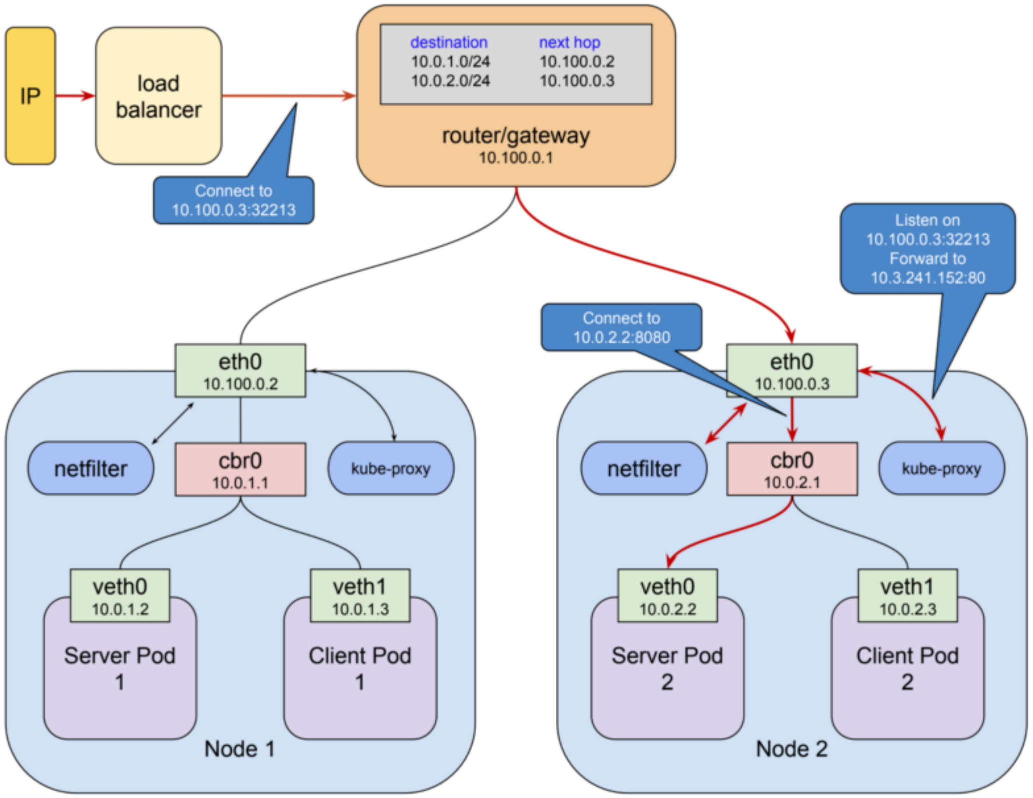


- netfilter rules don't match destination address
  - Service network: match rule, but not routable
  - Node network: routable, but not match rule



# Ingress

- NodePort (ClusterIP extension)
  - Reachable at IP address of Node
  - Reachable in Service Network
  - How it works?
    - Kube-proxy allocates a port in **30000-32767**
    - Open this port on the **eth0** of every node
    - Connection to this port are forwarded to the service's cluster IP



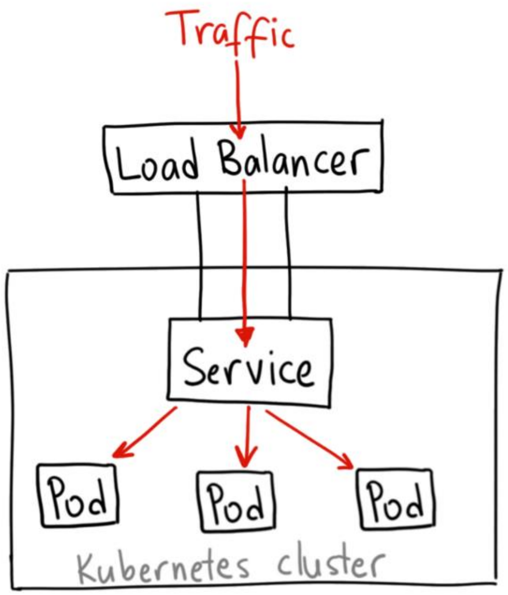
  - Issues:
    - Exposing Port is a non-standard port
    - Limited resource (2768 ports)

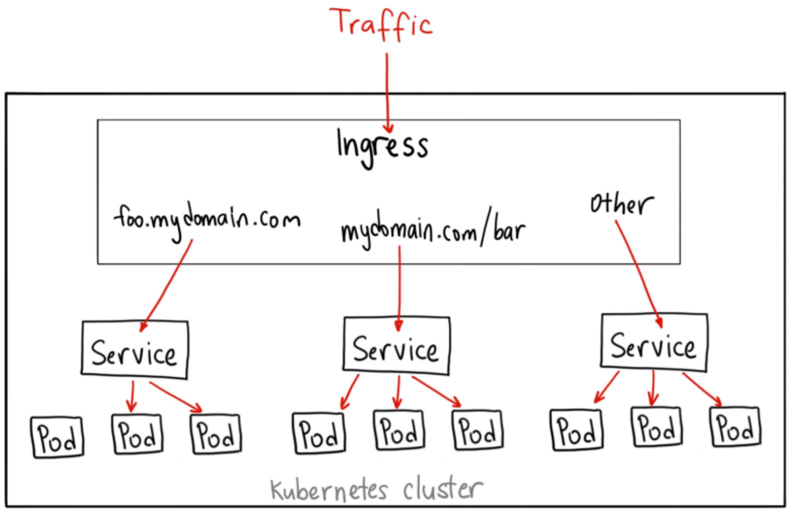
# Ingress

- LoadBalancer Service (NodePort Extension)
  - Build out a complete ingress path with external IP

```
$ kubectl get svc service-test
NAME         CLUSTER-IP       EXTERNAL-IP      PORT(S)       AGE
openvpn      10.3.241.52      35.184.97.156    80:32213/TCP  5m
```

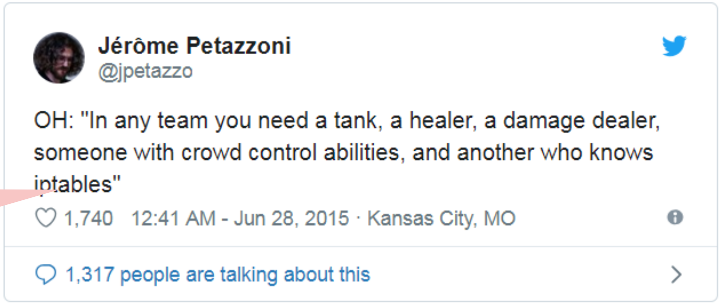
  - If service isn't destroyed and recreated, the IP won't change.
  - Limitation
    - Support single service (cannot use single load balancer to multiple service)
- Ingress
  - Sit in front of multiple services
  - Act as an entrypoint into the cluster.





# v2.0 is coming

20 years ago



Jérôme Petazzoni
@jpetazzo

OH: "In any team you need a tank, a healer, a damage dealer, someone with crowd control abilities, and another who knows iptables"

1,740 12:41 AM - Jun 28, 2015 · Kansas City, MO

1,317 people are talking about this

***"Kubernetes networking with Istio, Envoy, Cilium"***

## *Q & A*