Kubernetes Networking Best practices

Who am I?

- Senior Devops Engineer at Onmobile Global
- Part of Bangalore Docker Community
- One of the Contributors in Docker Labs
- Have travelled extensively from work.
- A Linux, Docker Enthusiast, Avid Runner and Cyclist

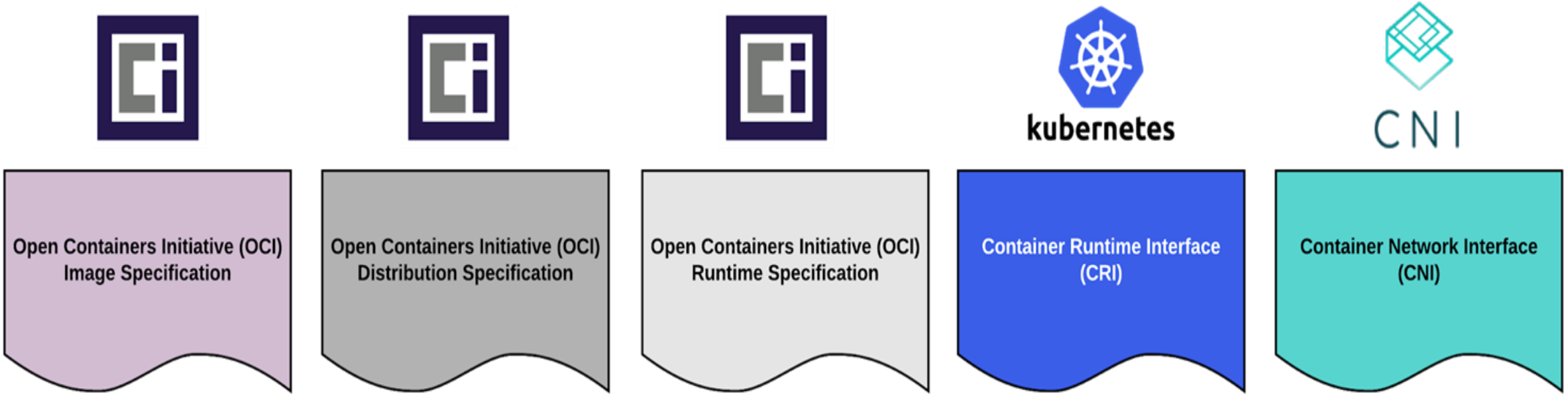
Twitter: @Bala35542522

Agenda

- Overview of Container Internals
- Docker CNM-Container Networking Model Dive
- Kubernetes CNI- Container Networking Interface Dive
- Kubernetes Networking Best Practices
- Kubernetes Networking Metrics

Container Internals

Container Standards



Many different standards

Container Building Blocks

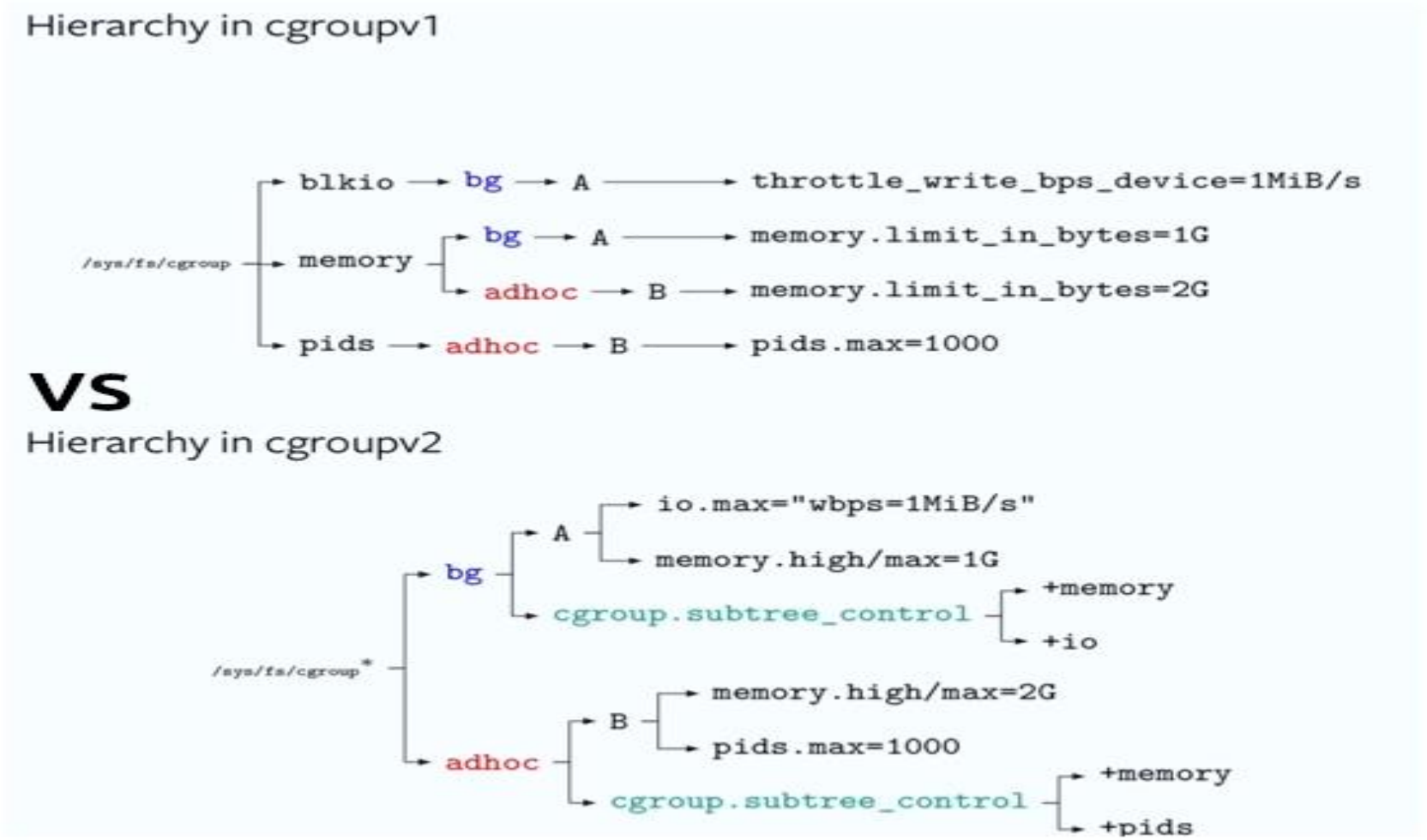
Namespace

- Linux provides seven different namespaces
  CLONE_NEWNET – isolate network devices
  CLONE_NEWUTS – host and domain names (UNIX Timesharing System)
  CLONE_NEWIPC – IPC objects
  CLONE_NEWPID – PIDs
  CLONE_NEWNS – mount points (file systems)
  CLONE_NEWUSER – users and groups
- Network namespaces (CLONE_NEWNET) determine the network resources that are available to a process,
- Each network namespace has its own network devices, IP addresses, IP routing tables, /proc/net directory, port numbers, and so on.

cgroups:

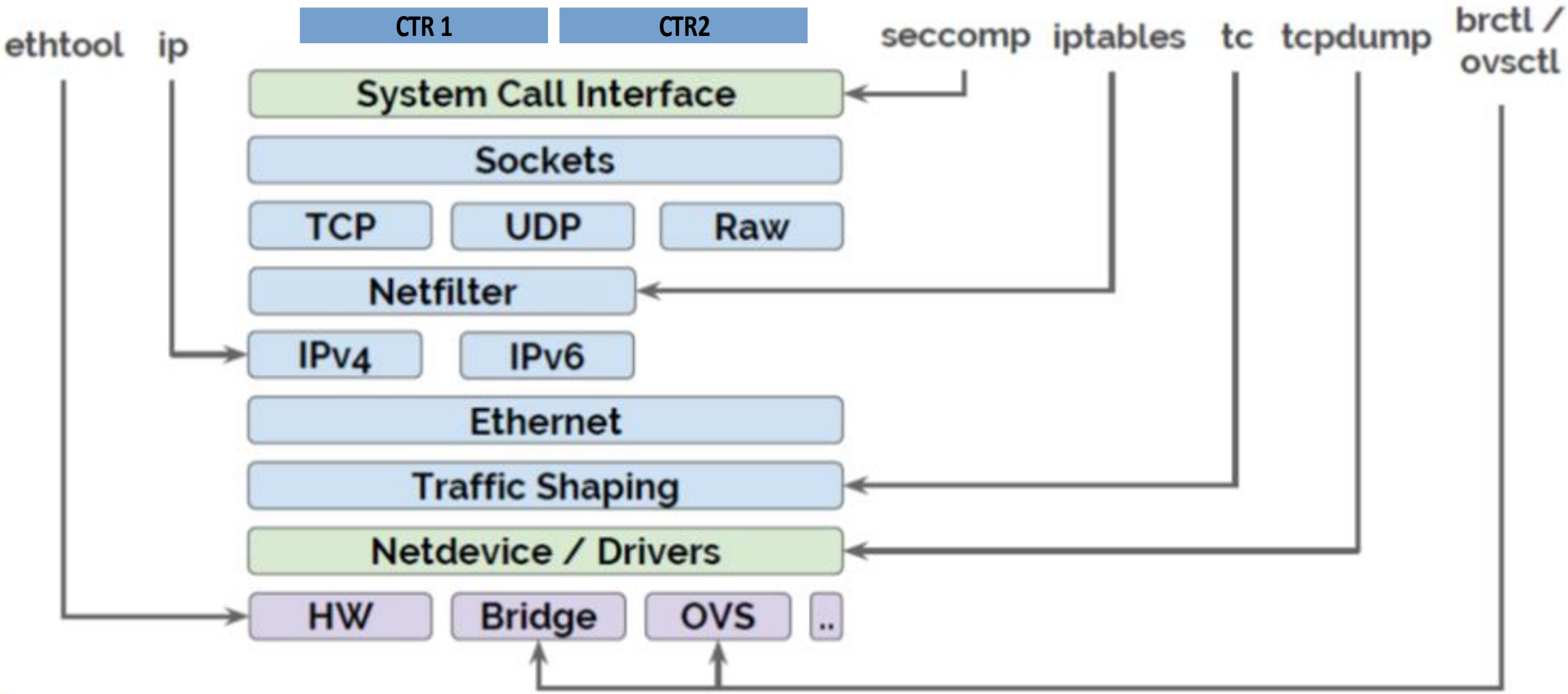
- blkio, cpu, cpuacct, cpuset, devices, hugetlb, memory,
- **net_cls,net_prio**, pids, Freezer,Perf_events,ns
- **xt_cgroup**(cgroupv2)

Container Building Blocks

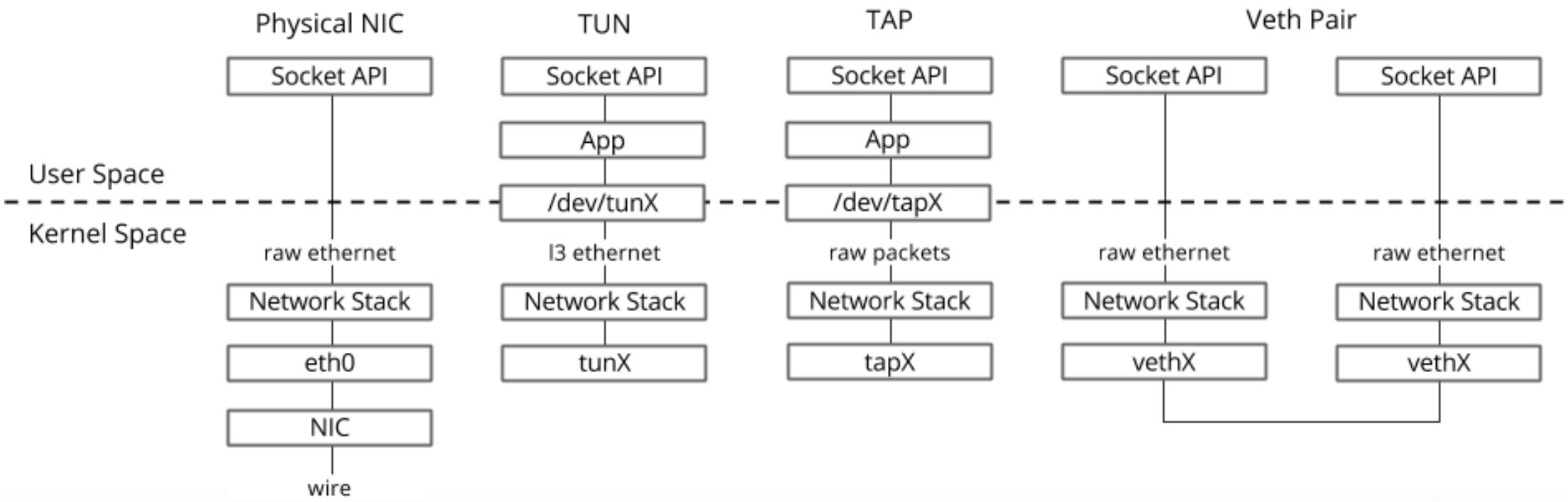


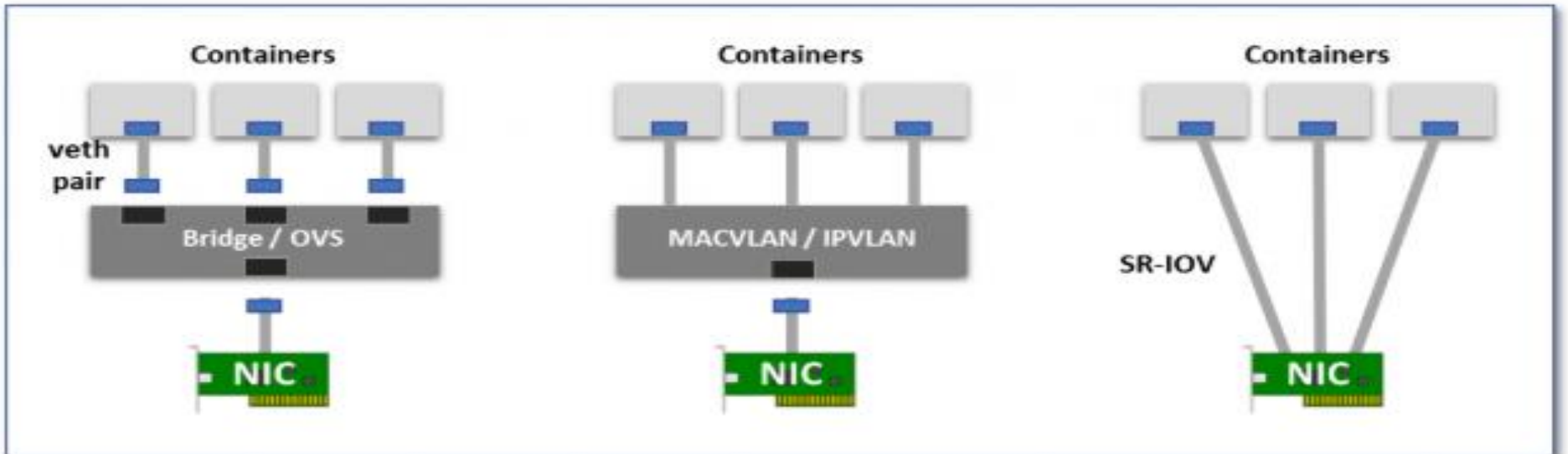
In cgroups v1, you could assign threads of the same process to different cgroups.But in Cgroup v2, this is not possible. Rhel8 by default comes up with cgroupv2.
Note: Kernel version has to be 4.5 and above

High Level Abstractions



Container Networking

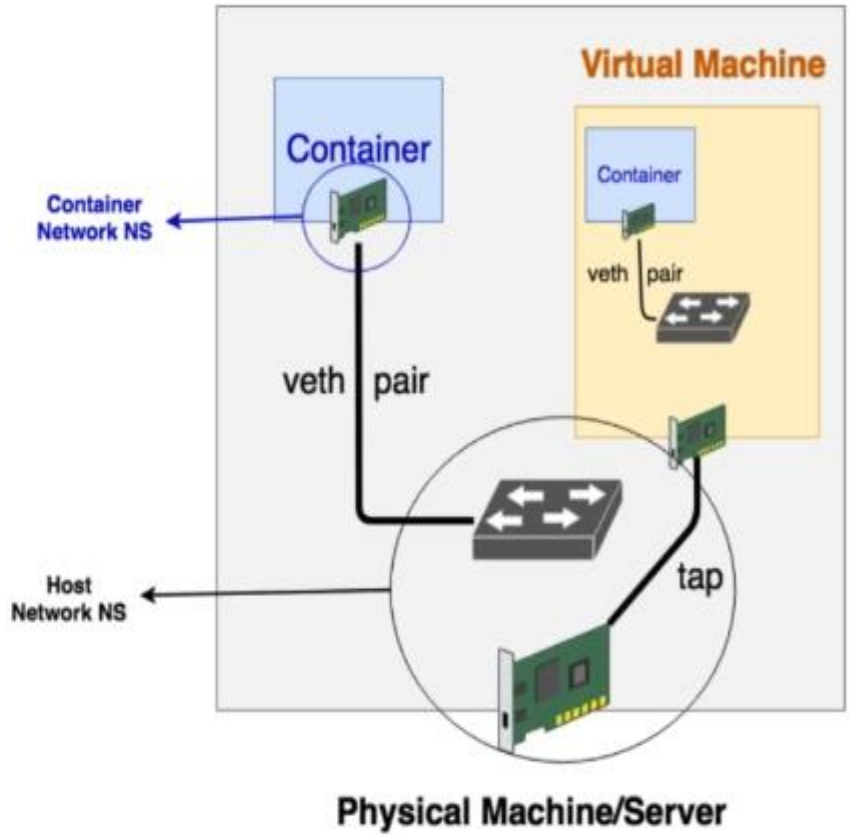




Linux Virtual Networking

- Virtual Switch/Bridge
  - Example: docker0 bridge

- Virtual Interfaces
  - Example: veth, tap



- Namespaces
  - A network namespace is logically another copy of the network stack, with its own routes, firewall rules, and network devices.

iptables

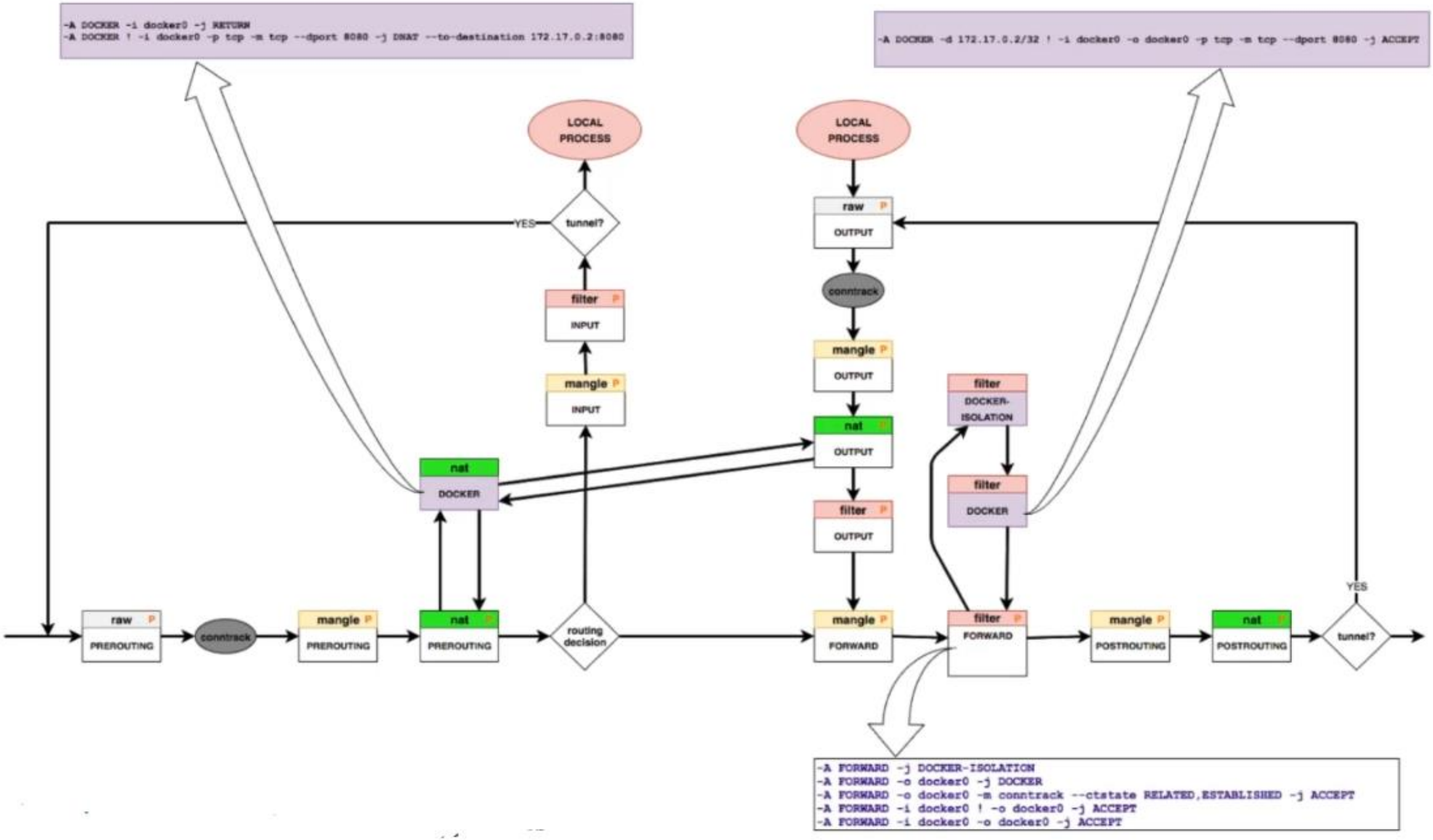
It is a user-space utility program used to configure the tables provided by the Linux kernel firewall (netfilter) and the chains and rules it stores.

Tables:

- raw
- mangle
- nat
- filter

Chains:

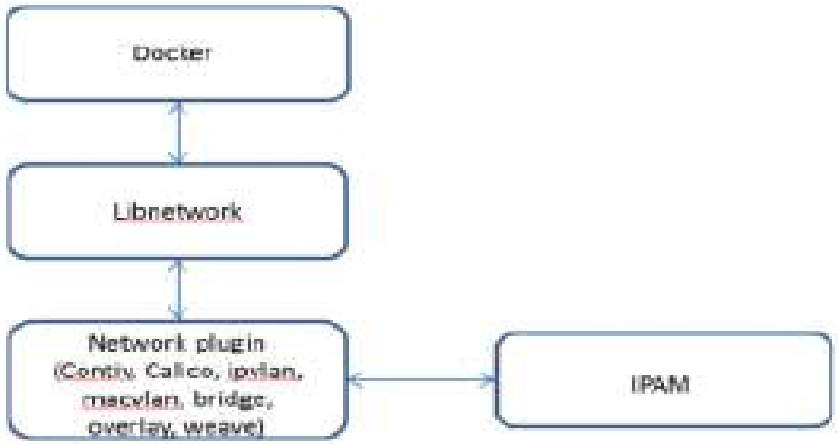
- INPUT
- PREROUTING
- FORWARD
- POSTROUTING
- OUTPUT


-A DOCKER -i docker0 -j RETURN
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 8080 -j DNAT --to-destination 172.17.0.2:8080
-A DOCKER -d 172.17.0.2/32 ! -i docker0 -o docker0 -p tcp -m tcp --dport 8080 -j ACCEPT
LOCAL PROCESS
LOCAL PROCESS
YES
tunnel?
raw P
OUTPUT
conntrack
filter P
INPUT
mangle P
OUTPUT
mangle P
INPUT
filter
DOCKER-ISOLATION
nat
DOCKER
nat P
OUTPUT
filter
DOCKER
filter P
OUTPUT
raw P
PREROUTING
conntrack
mangle P
PREROUTING
nat P
PREROUTING
routing decision
mangle P
FORWARD
filter P
FORWARD
mangle P
POSTROUTING
nat P
POSTROUTING
YES
tunnel?
-A FORWARD -j DOCKER-ISOLATION
-A FORWARD -o docker0 -j DOCKER
-A FORWARD -o docker0 -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -i docker0 ! -o docker0 -j ACCEPT
-A FORWARD -i docker0 -o docker0 -j ACCEPT

Container Network Model

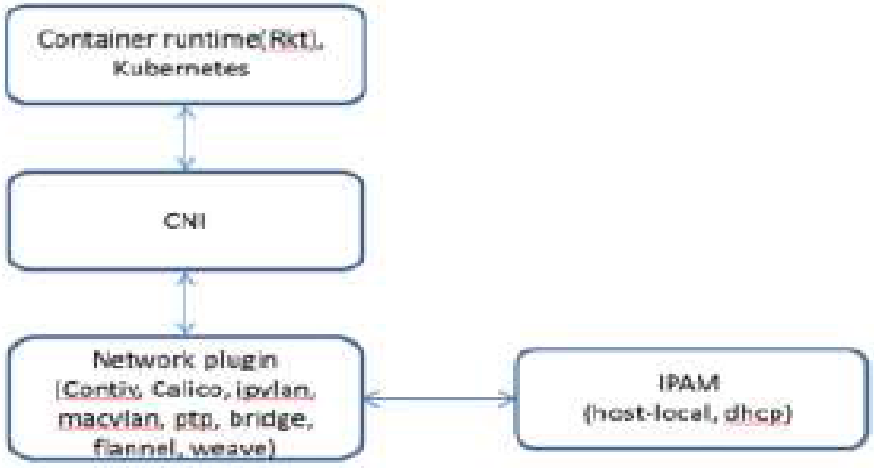
CNM VS CNI

CNM



- Project started by Docker.
- Keep networking as a library separate from the Container runtime.
- Networking implementation will be done as a plugin implemented by drivers.
- IP address assignment for the Containers is done using local IPAM drivers and plugins.
- Supported local drivers are bridge, overlay, macvlan, ipvlan. Supported remote drivers are Weave, Calico, Contiv etc.

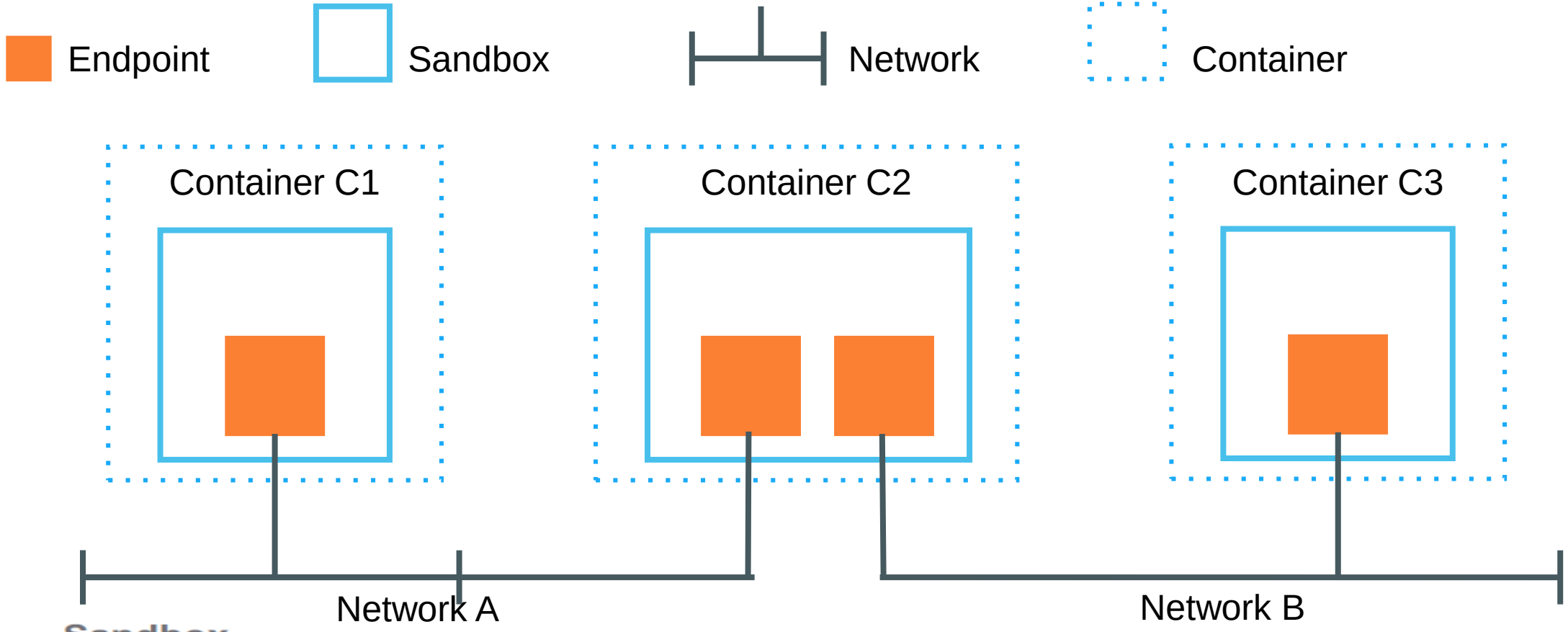
CNI



- Project started by CoreOS. Used by Cloudfoundry, Mesos and Kubernetes.
- The CNI interface calls the API of the CNI plugin to set up Container networking.
- The CNI plugin calls the IPAM plugin to set up the IP address for the container.
- Available CNI plugins are Bridge, macvlan, ipvlan, and ptp. Available IPAM plugins are host-local and DHCP.
- External CNI plugins examples – Flannel, Weave, Contiv etc

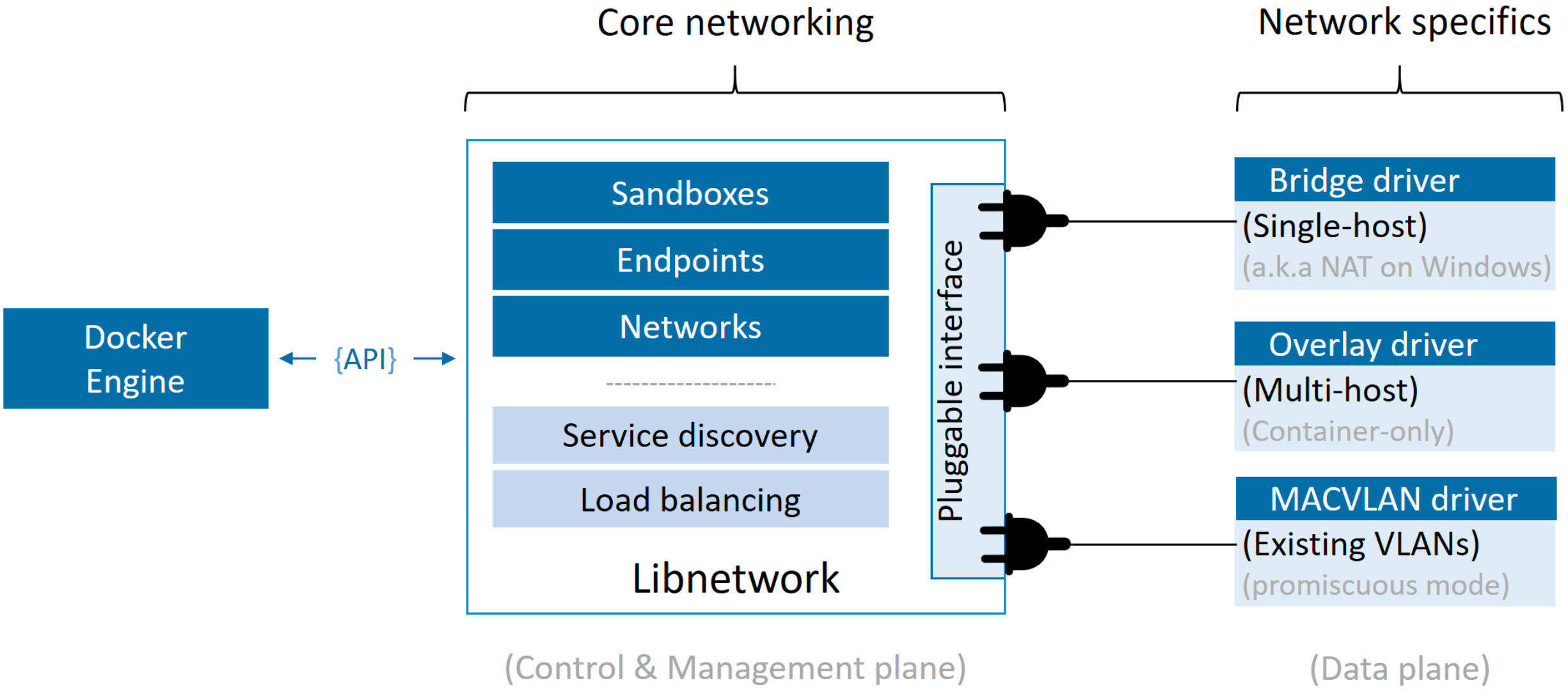
Note: https://kubernetes.io/blog/2016/01/why-kubernetes-doesnt-use-libnetwork/

Containers and the CNM



- **Sandbox**
  - A Sandbox contains the configuration of a container's network stack. This includes management of the container's interfaces, routing table and DNS settings. An implementation of a Sandbox could be a Linux Network Namespace, a FreeBSD Jail or other similar concept.
- **Endpoint**
  - An Endpoint joins a Sandbox to a Network. An implementation of an Endpoint could be a veth pair, an Open vSwitch internal port or similar
- **Network**
  - A Network is a group of Endpoints that are able to communicate with each-other directly. An implementation of a Network could be a VXLAN Segment, a Linux bridge, a VLAN, etc.

CNM Driver Interfaces

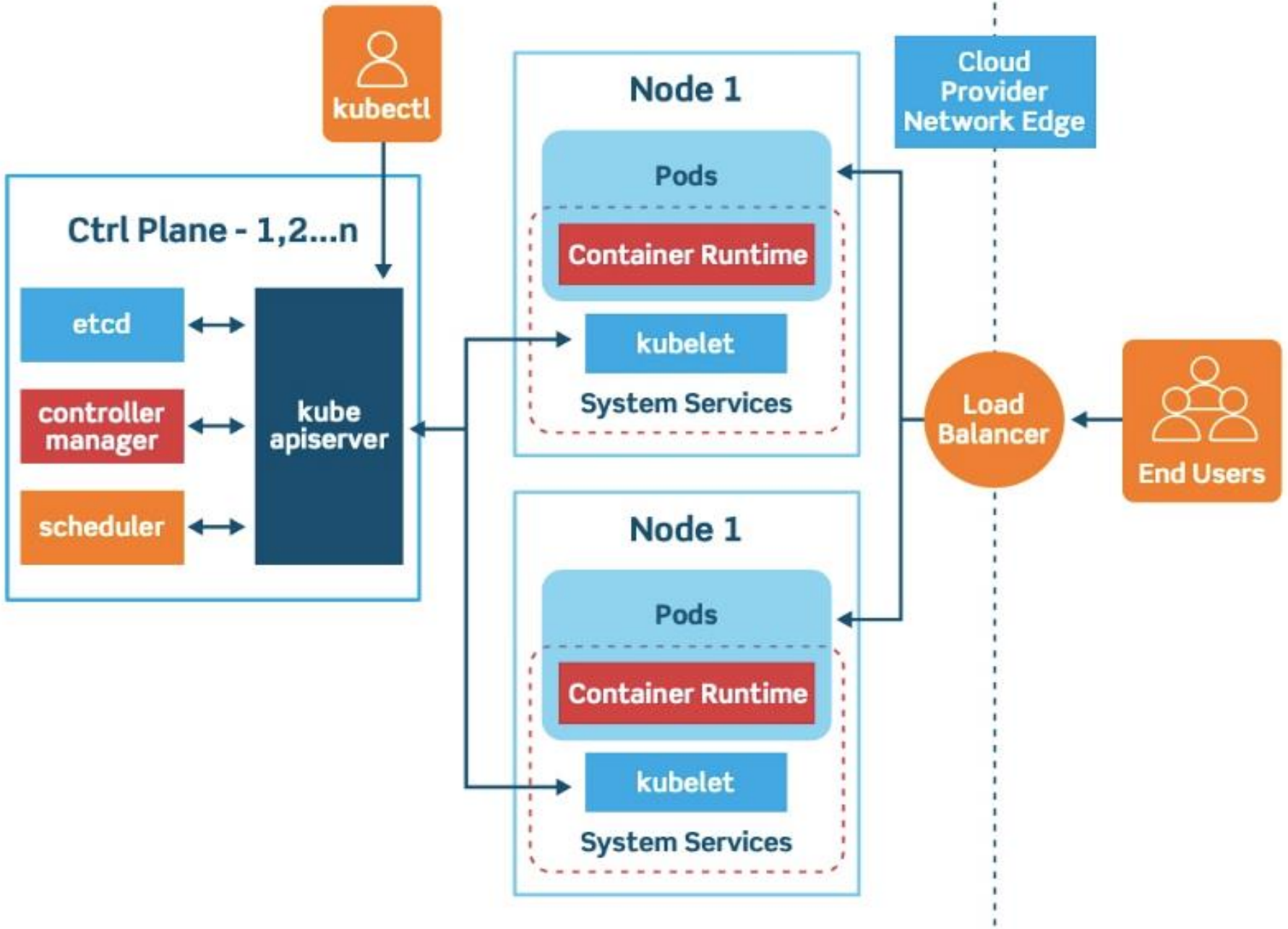


Docker Default Network Drivers

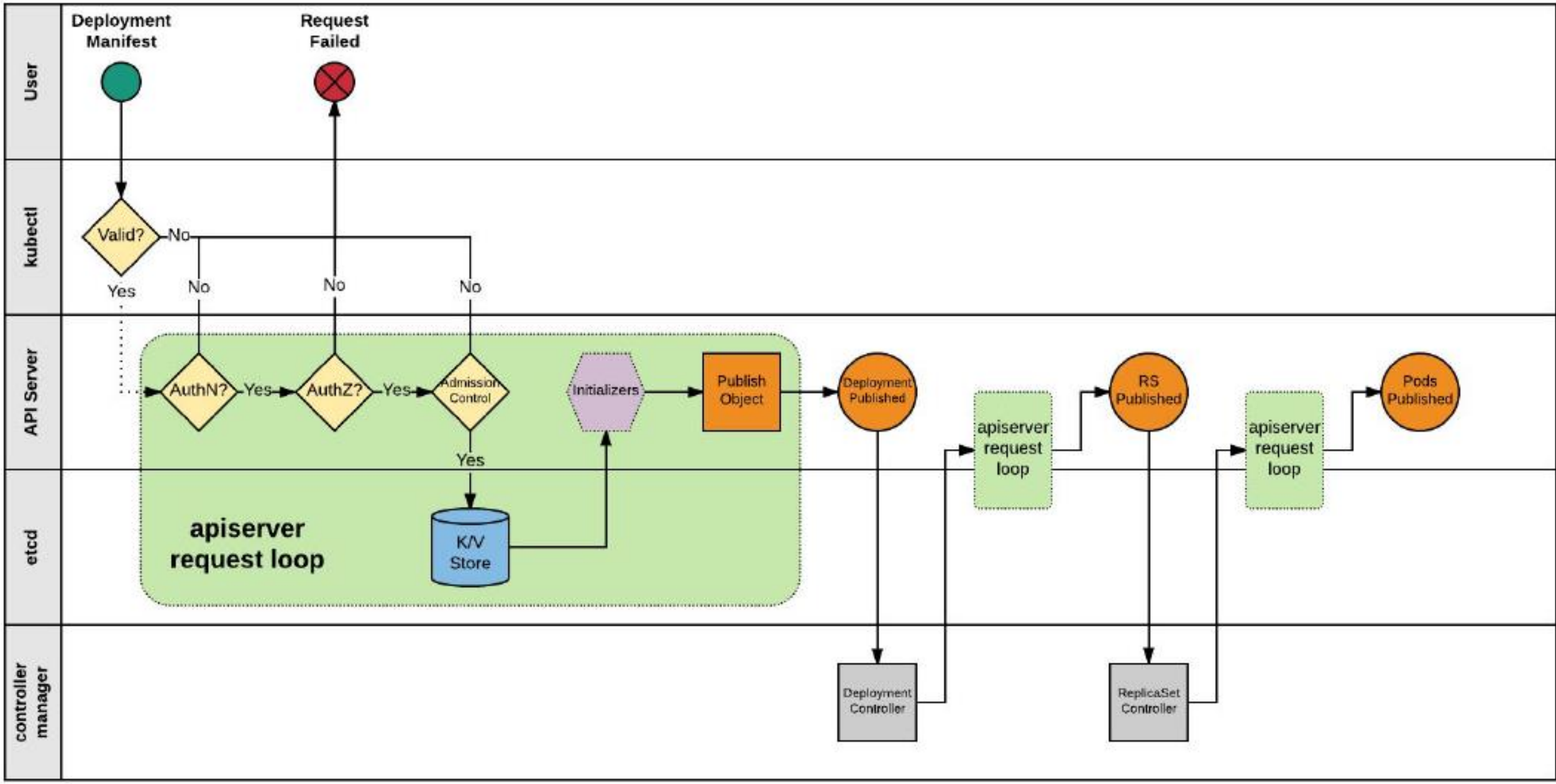
Driver	Description
Host	With the `host` driver, a container uses the networking stack of the host. There is no namespace separation, and all interfaces on the host can be used directly by the container
Bridge	The `bridge` driver creates a Linux bridge on the host that is managed by Docker. By default containers on a bridge can communicate with each other. External access to containers can also be configured through the `bridge` driver
Overlay	The `overlay` driver creates an overlay network that supports multi-host networks out of the box. It uses a combination of local Linux bridges and VXLAN to overlay container-to-container communications over physical network infrastructure
MACVLAN	The `macvlan` driver uses the MACVLAN bridge mode to establish a connection between container interfaces and a parent host interface (or sub-interfaces). It can be used to provide IP addresses to containers that are routable on the physical network. Additionally VLANs can be trunked to the `macvlan` driver to enforce Layer 2 container segmentation
None	The `none` driver gives a container its own networking stack and network namespace but does not configure interfaces inside the container. Without additional configuration, the container is completely isolated from the host networking stack

Container Network Interface

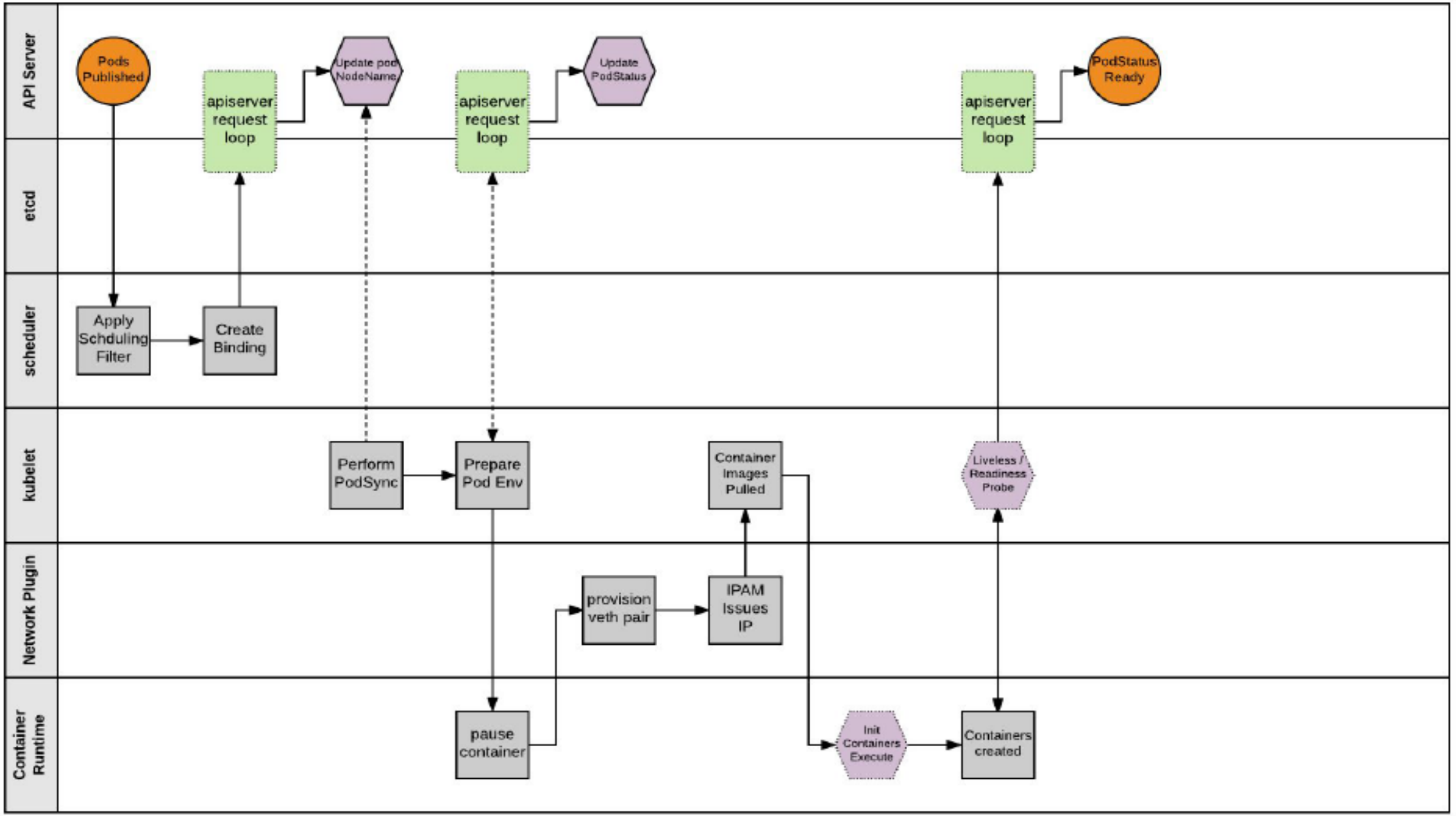
Kubernetes At a High Level



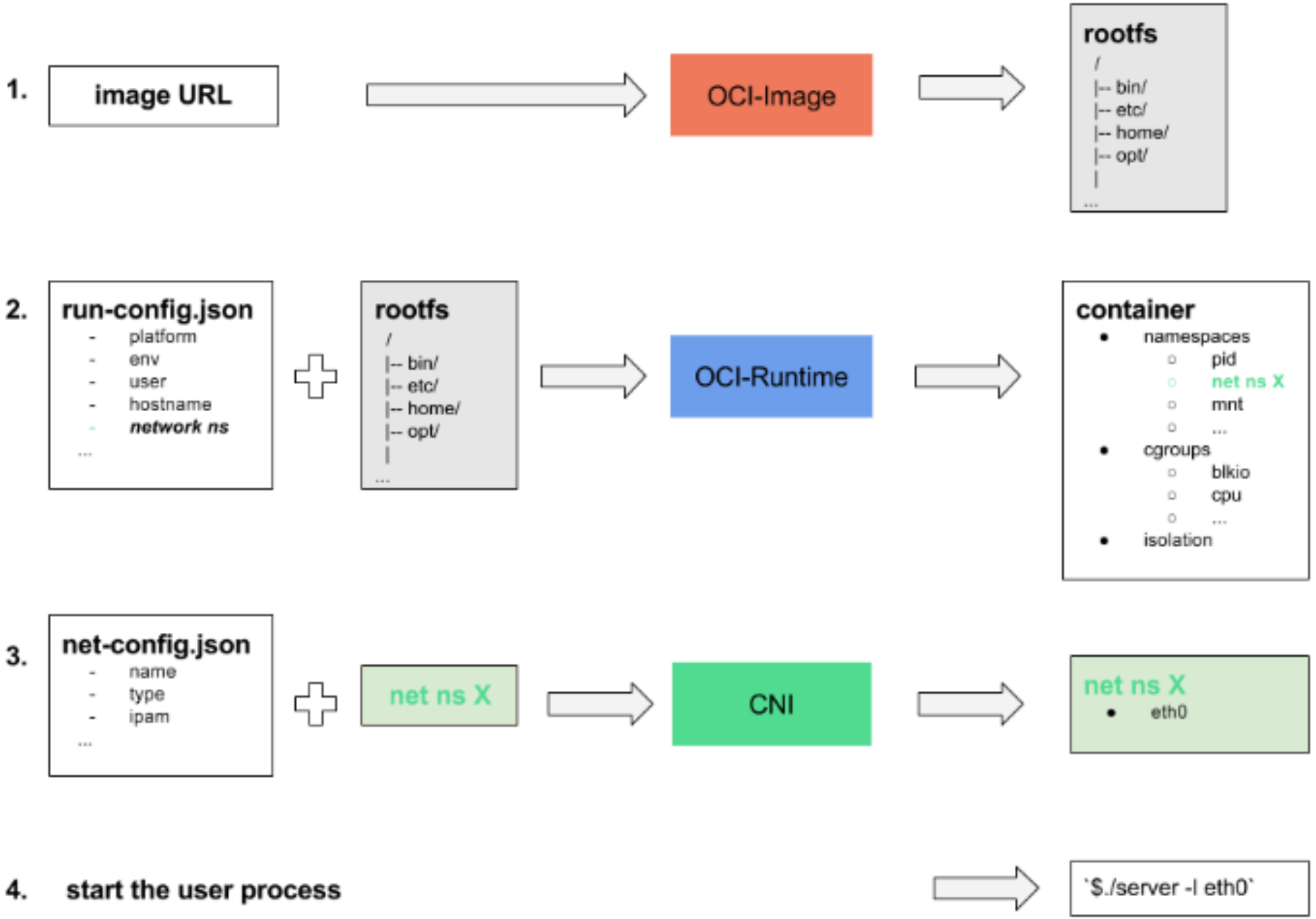
Kubernetes Fundamentals



Kubernetes Fundamentals



Kubernetes Flow Fundamentals



What is a POD? (In terms of networking)

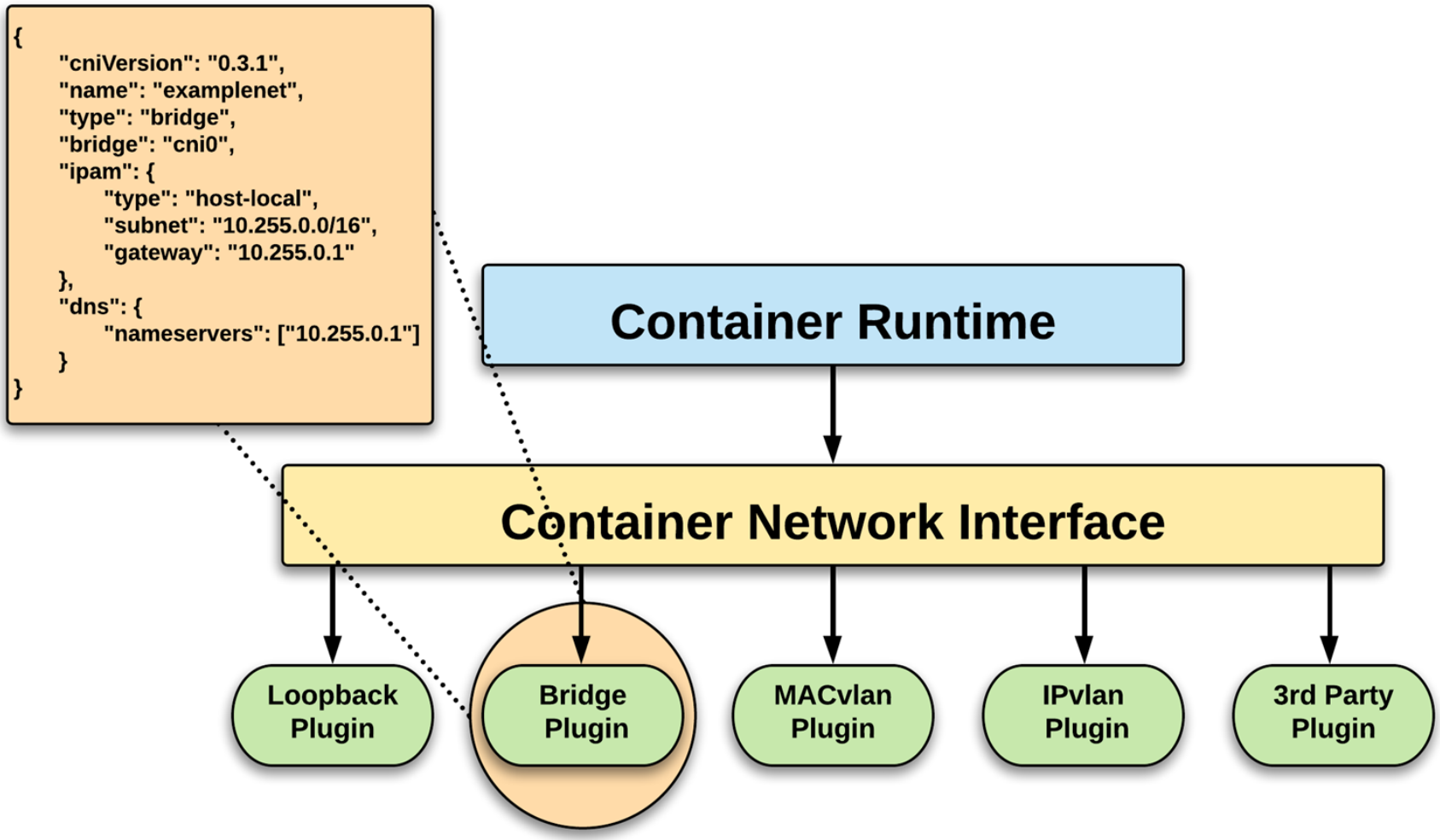
- A POD is two(at the very least) or more containers running together.
  - pause container + actual container(s)
- Pause container has the actual network namespace.
- Actual container uses the network namespace of the pause container.

Docker equivalent:

```
docker run --name pause -itd ubuntu sleep infinity

docker run --name actual-app --net=container:pause -itd appimage
```

CNI Overview



https://github.com/kelseyhightower/kubernetes-the-hard-way/blob/master/docs/09-bootstrapping-kubernetes-workers.md#configure-cni-networking

How does kubelet know which plugins to use?

- The plugin (or list of plugins) is set in the CNI configuration
- The CNI configuration is a *single file* in `/etc/cni/net.d`
- If there are multiple files in that directory, the first one is used

  (in lexicographic order)
- That path can be changed with the `--cni-conf-dir` flag of kubelet

CNI

- The spec defines a container as being a Linux network namespace. We should be comfortable with that definition as container runtimes like Docker create a new network namespace for each container.
- Network definitions for CNI are stored as JSON files.
- The network definitions are streamed to the plugin through STDIN. That is – there are no configuration files sitting on the host for the network configuration.
- Other arguments are passed to the plugin via environmental variables
- A CNI plugin is implemented as an executable.
- The CNI plugin is responsible wiring up the container. That is – it needs to do all the work to get the container on the network. In Docker, this would include connecting the container network namespace back to the host somehow.
- The CNI plugin is responsible for IPAM which includes IP address assignment and installing any required routes.

CNI plugins for kubernetes typically have two components:

1. A CNI binary that configures the pod's interface.
2. A daemon that manages routing.

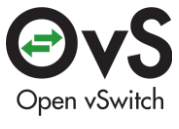
Points to consider on implementing CNI

- The container runtime must create a new network namespace for the container before invoking any plugins.
- The runtime must then determine which networks this container should belong to, and for each network, which plugins must be executed.
- The network configuration is in JSON format and can easily be stored in a file. The network configuration includes mandatory fields such as "name" and "type" as well as plugin (type) specific ones. The network configuration allows for fields to change values between invocations. For this purpose there is an optional field "args" which must contain the varying information.
- The container runtime must add the container to each network by executing the corresponding plugins for each network sequentially.
- Upon completion of the container lifecycle, the runtime must execute the plugins in reverse order (relative to the order in which they were executed to add the container) to disconnect the container from the networks.
- The container runtime must not invoke parallel operations for the same container, but is allowed to invoke parallel operations for different containers.
- The container runtime must order ADD and DEL operations for a container, such that ADD is always eventually followed by a corresponding DEL. DEL may be followed by additional DELs but plugins should handle multiple DELs permissively (i.e. plugin DEL should be idempotent).
- A container must be uniquely identified by a ContainerID. Plugins that store state should do so using a primary key of (network name, CNI_CONTAINERID, CNI_IFNAME).
- A runtime must not call ADD twice (without a corresponding DEL) for the same (network name, container id, name of the interface inside the container). This implies that a given container ID may be added to a specific network more than once only if each addition is done with a different interface name.

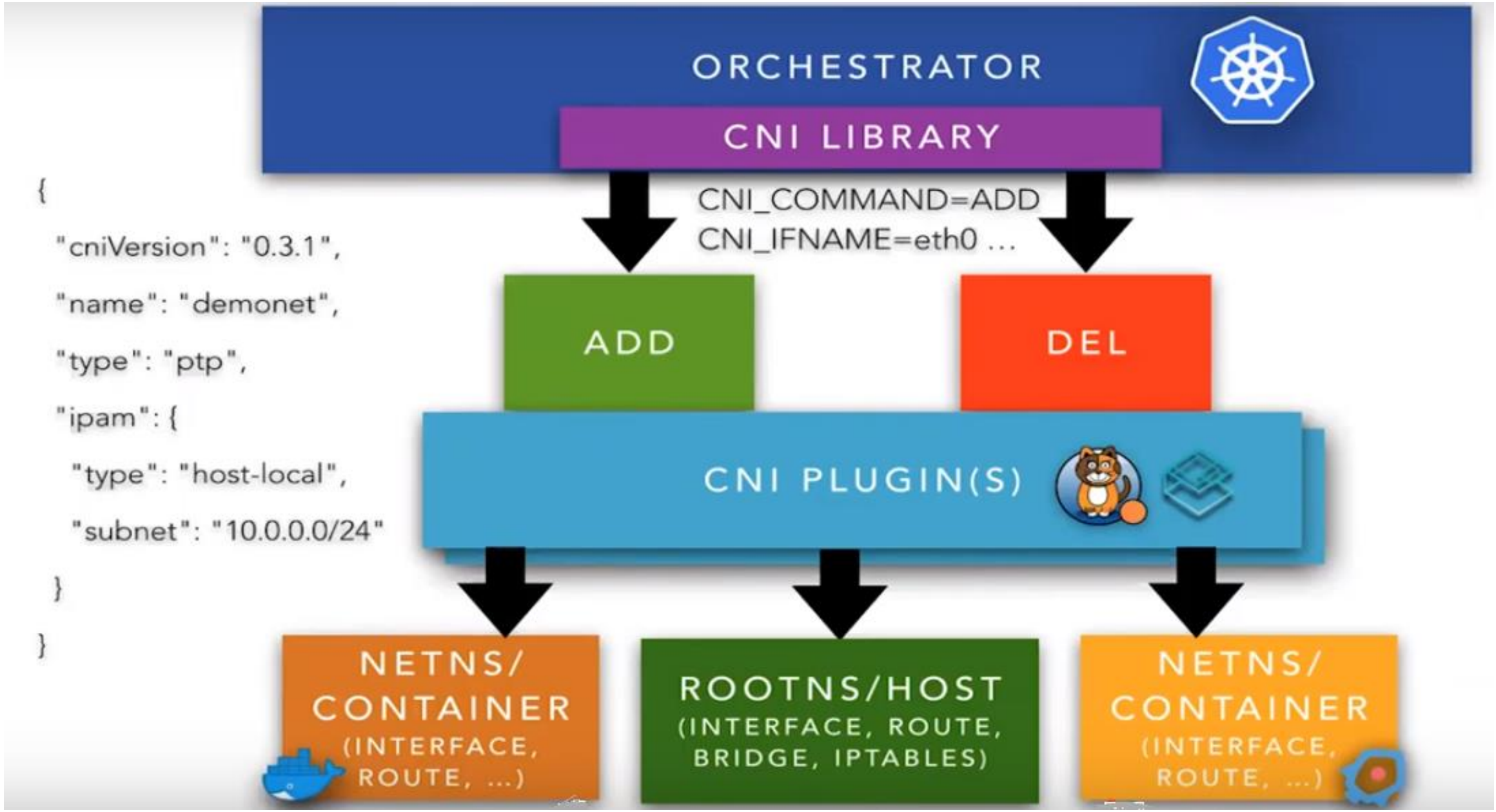
CNI Plugins

- Amazon ECS
- Calico
- Cillium
- Contiv
- Contrail
- Flannel

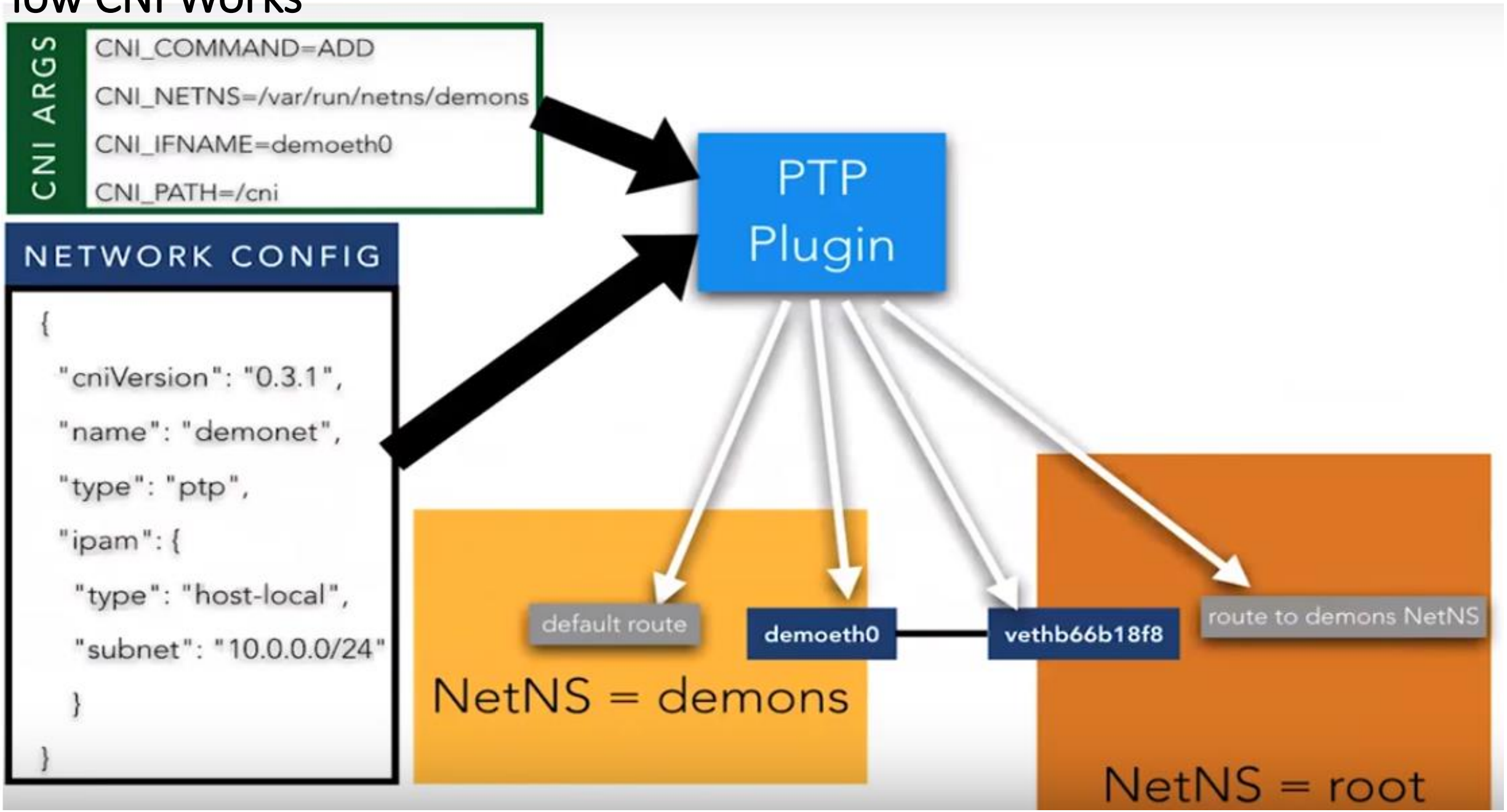
- GCE
- kube-router
- Multus
- OpenVSwitch
- Romana
- Weave

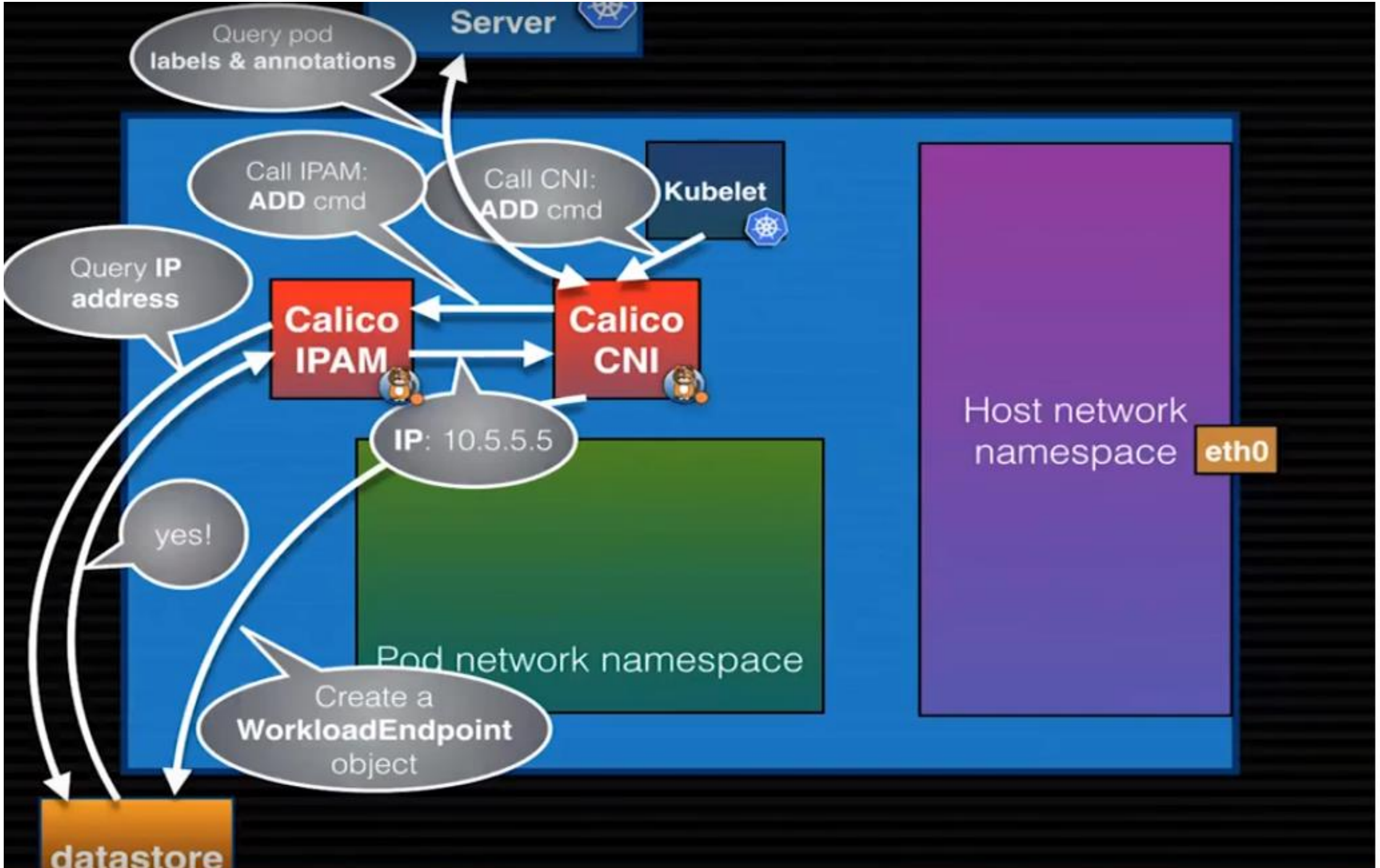


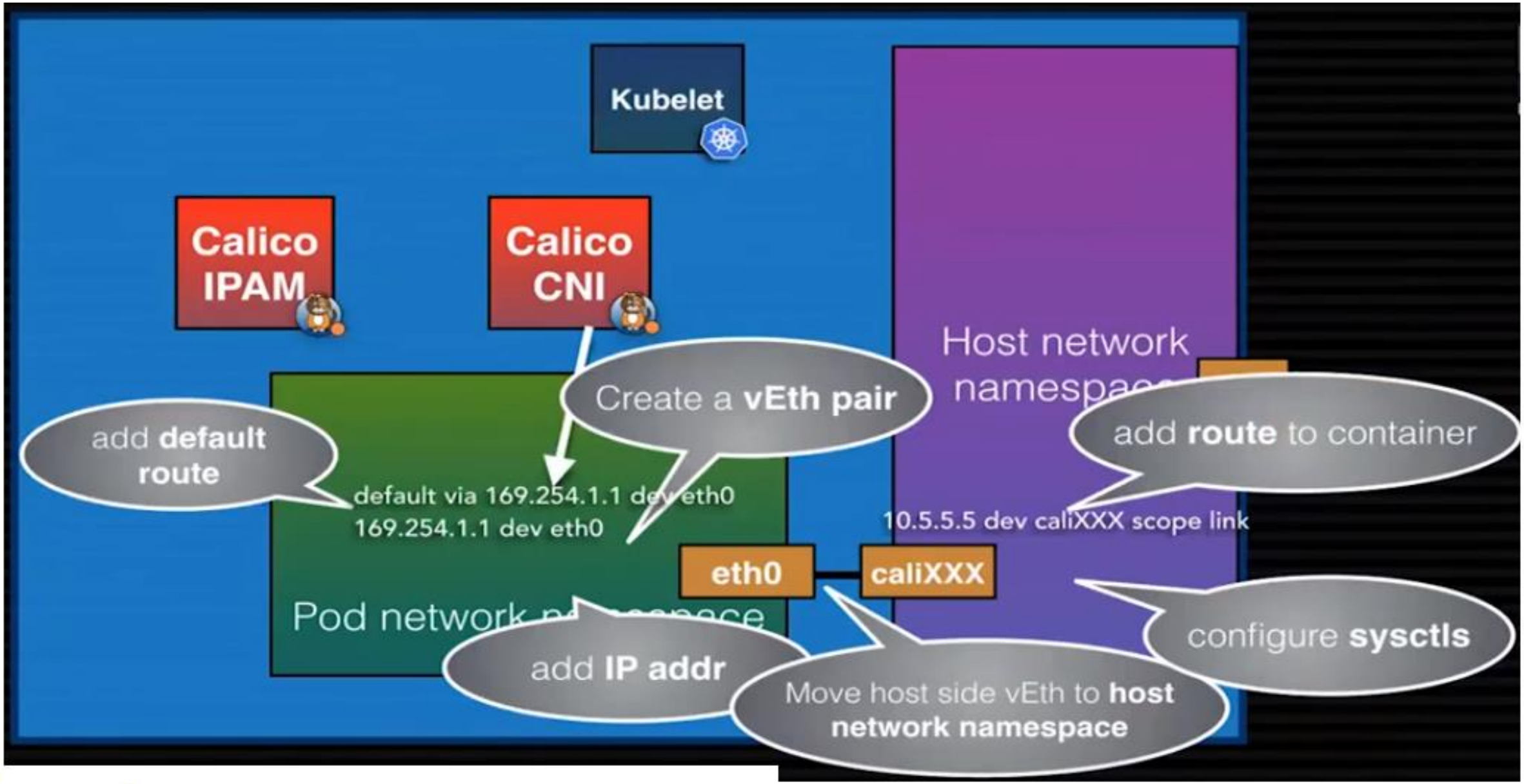
How CNI Works



How CNI Works




Server
Query pod
labels & annotations
Call IPAM:
ADD cmd
Call CNI:
ADD cmd
Kubelet
Query IP
address
Calico
IPAM
Calico
CNI
IP: 10.5.5.5
yes!
Host network
namespace
eth0
Pod network namespace
Create a
WorkloadEndpoint
object
datastore


Kubelet
Calico IPAM
Calico CNI
Host network namespace
Create a vEth pair
add default route
add route to container
default via 169.254.1.1 dev eth0
169.254.1.1 dev eth0
10.5.5.5 dev caliXXX scope link
eth0
caliXXX
Pod network namespace
add IP addr
configure sysctls
Move host side vEth to host network namespace

Kubernetes Networking Model

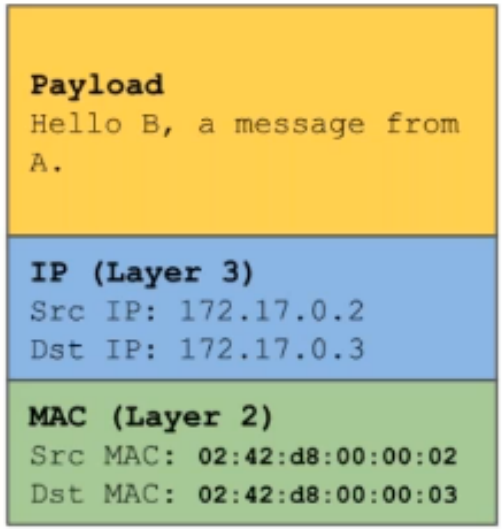
- all Pods can communicate with all other Pods without using *network address translation* (NAT).
- all Nodes can communicate with all Pods without NAT.
- the IP that a Pod sees itself as is the same IP that others see it as.

Given the above constraints , below problems to be solved in Kubernetes Networking

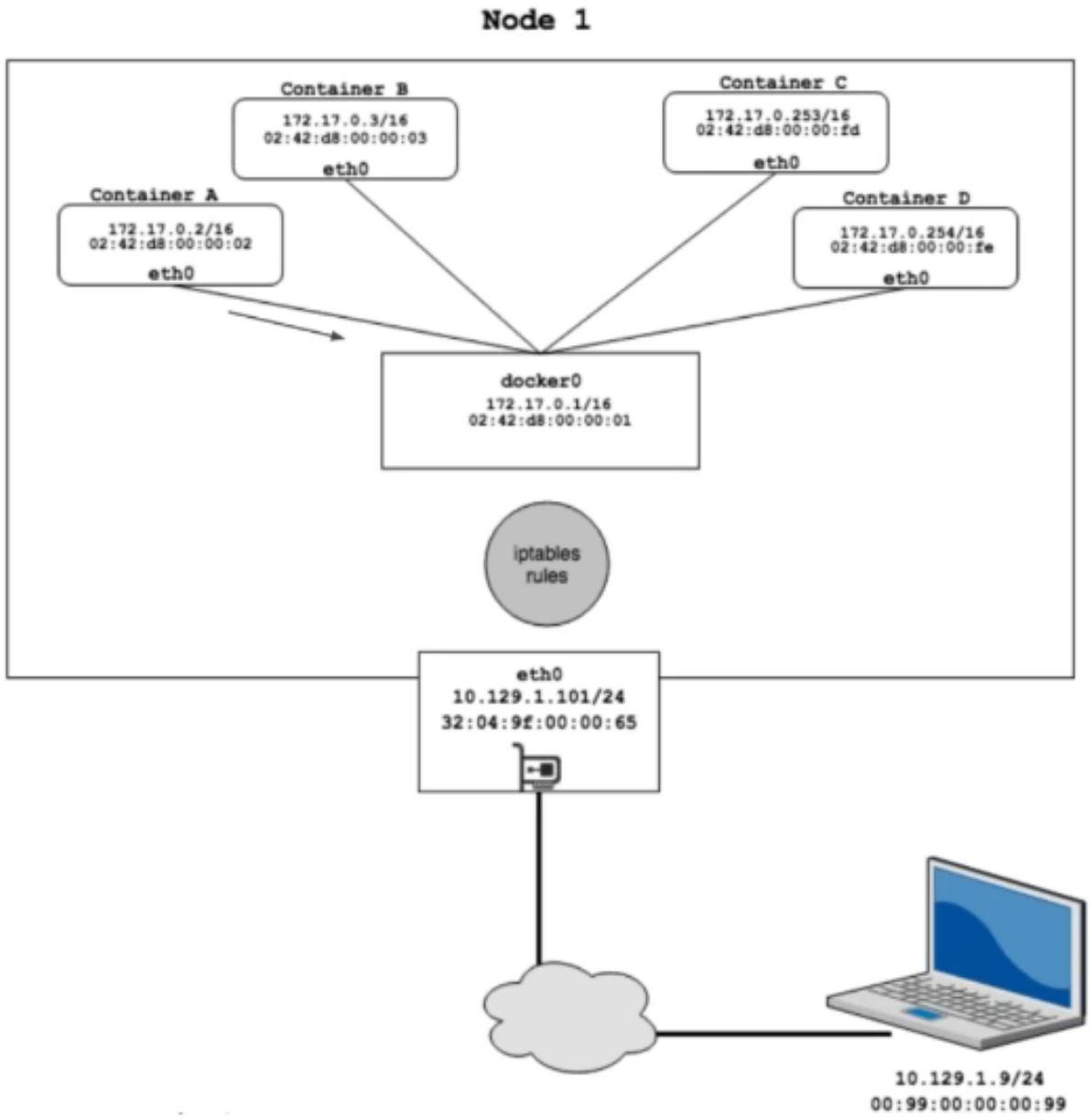
1. Container-to-Container networking
2. Pod-to-Pod networking
3. Pod-to-Service networking
4. Internet-to-Service networking

Container to container networking

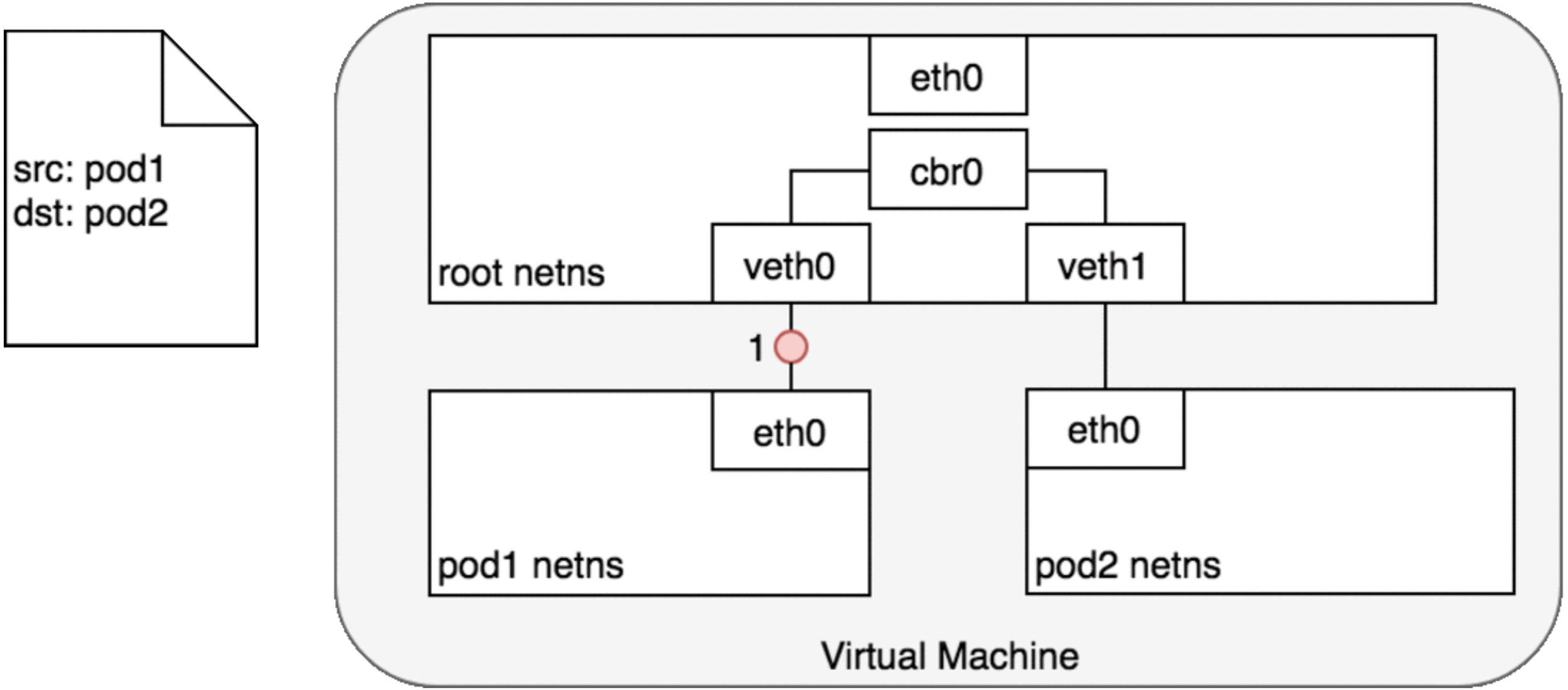
Container to Container Communication



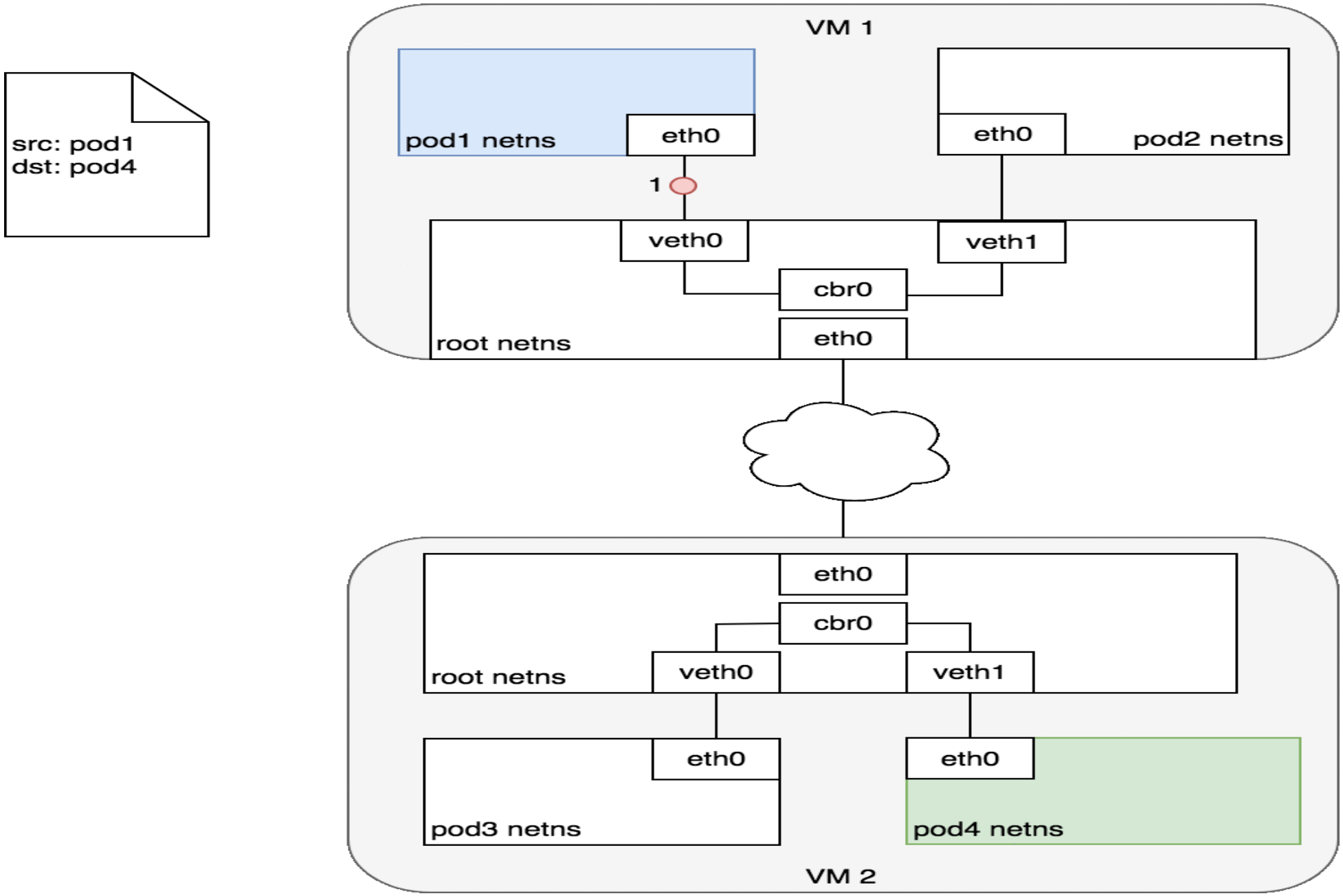
Container A ----------> Container B



Pod to Pod Networking same node



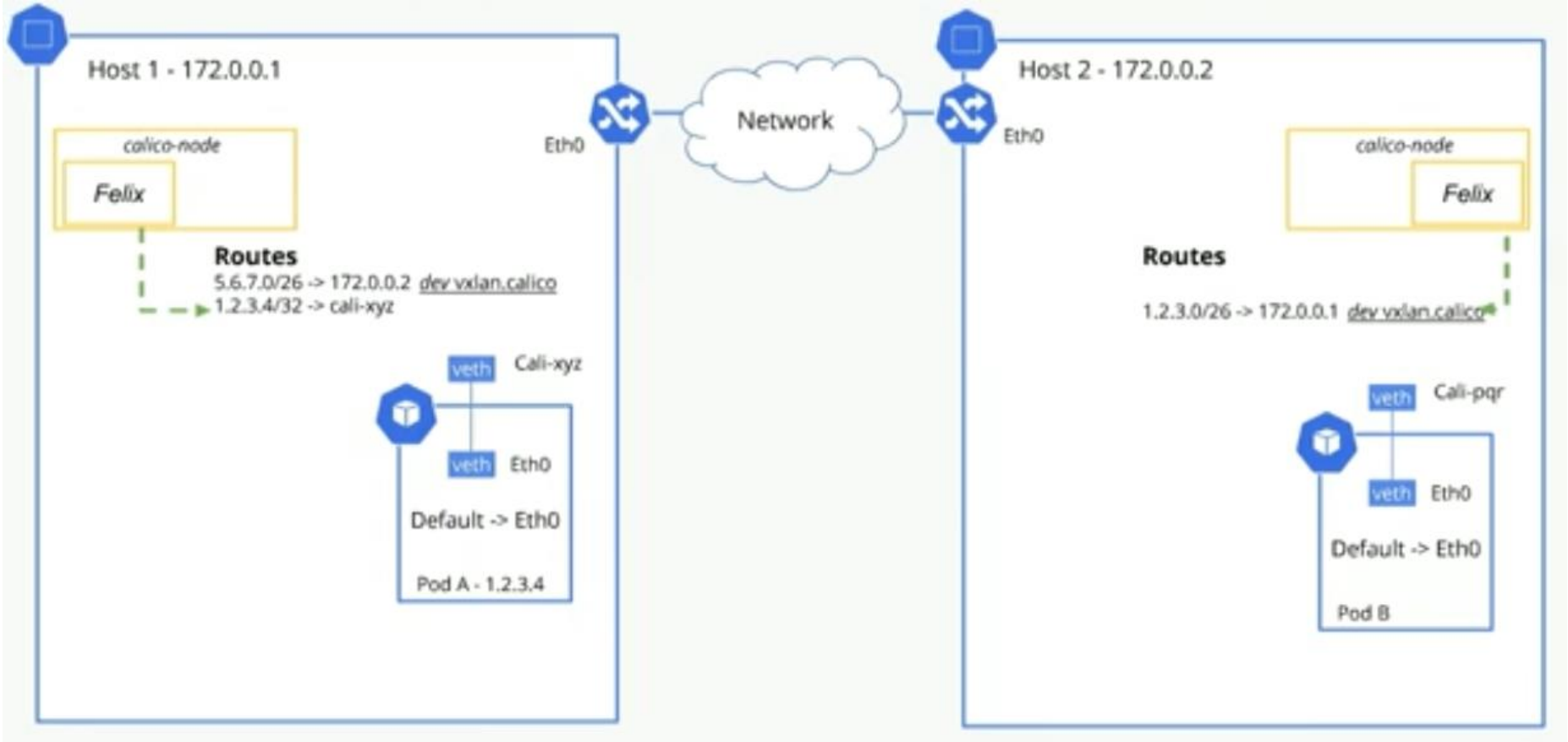
Pod to Pod Networking different node



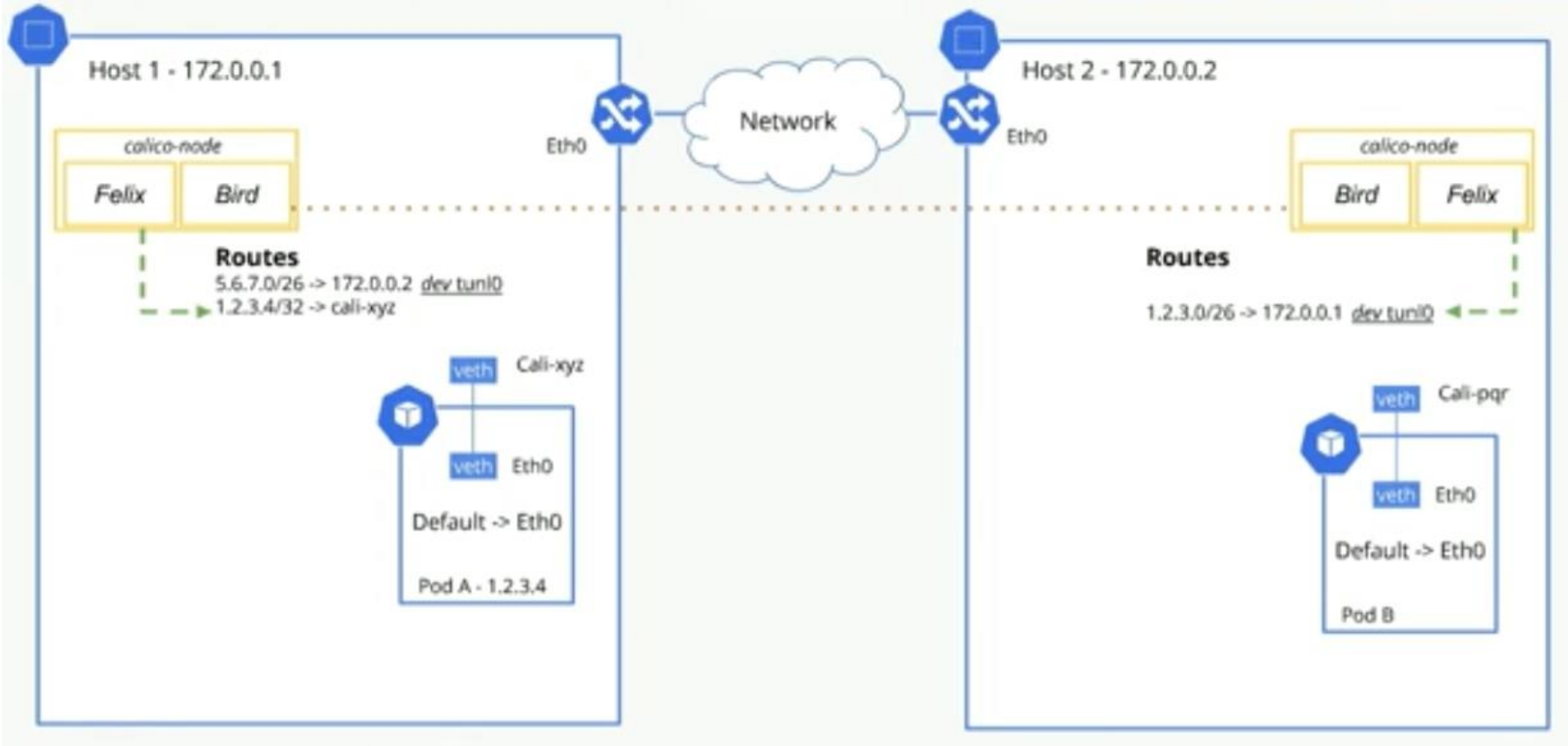
Pod Networking

Type/Features	L2	L3	Overlay	Cloud
Summary	Pods communicate using L2	Pod traffic is routed in underlay network	Pod traffic is encapsulated and uses underlay for reachability	Pod traffic is routed in cloud virtual network
Underlying technology	L2 arp, broadcast	Routing protocol like BGP	VXLAN	Pre-programmed fabric using controller(eg: GKE)
Encapsulation	No double encapsulation	No double encapsulation	Double encapsulation	No double encapsulation
Examples		Calico	Flannel, Weave	GKE, ACS, EKS

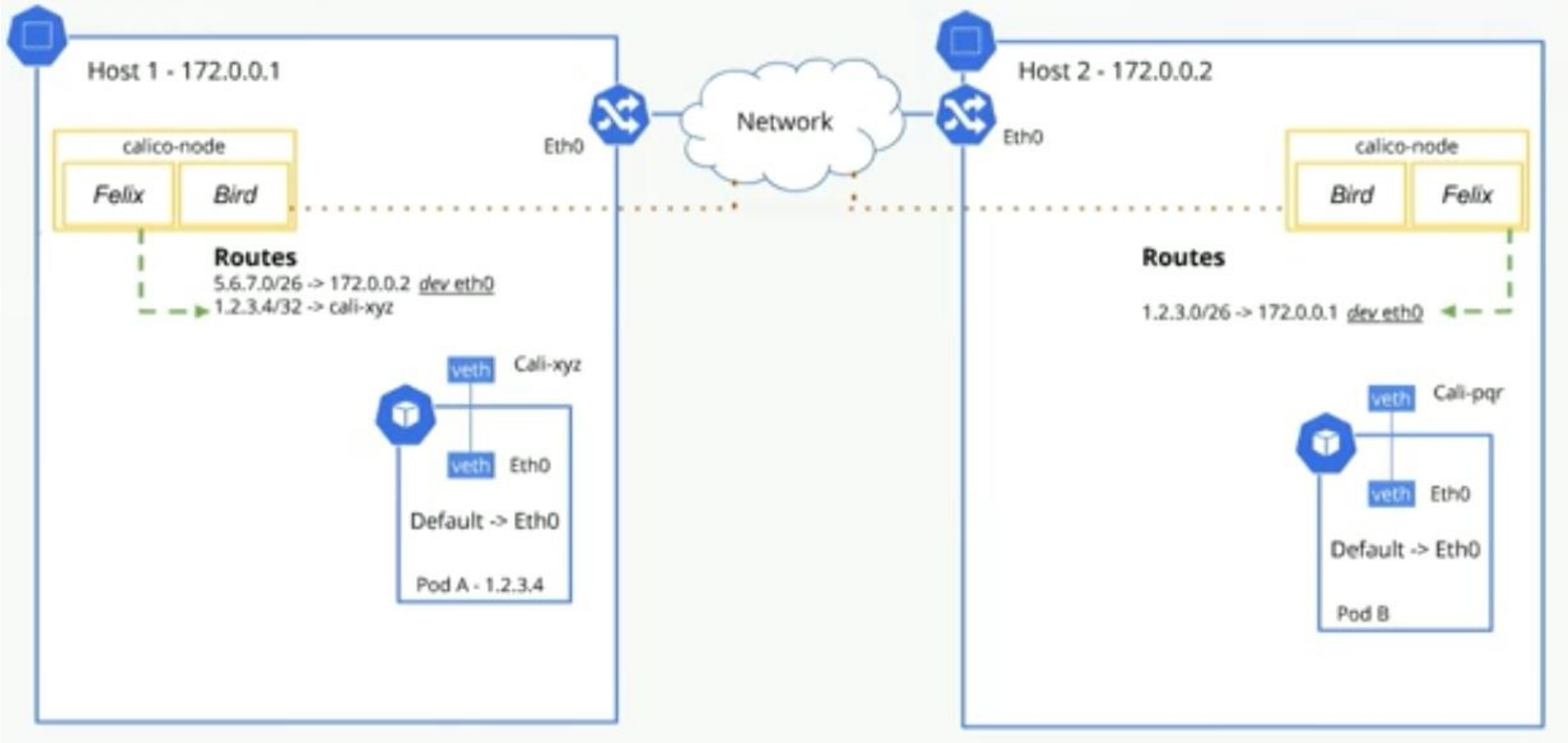
Pod networking –Vxlan Mode



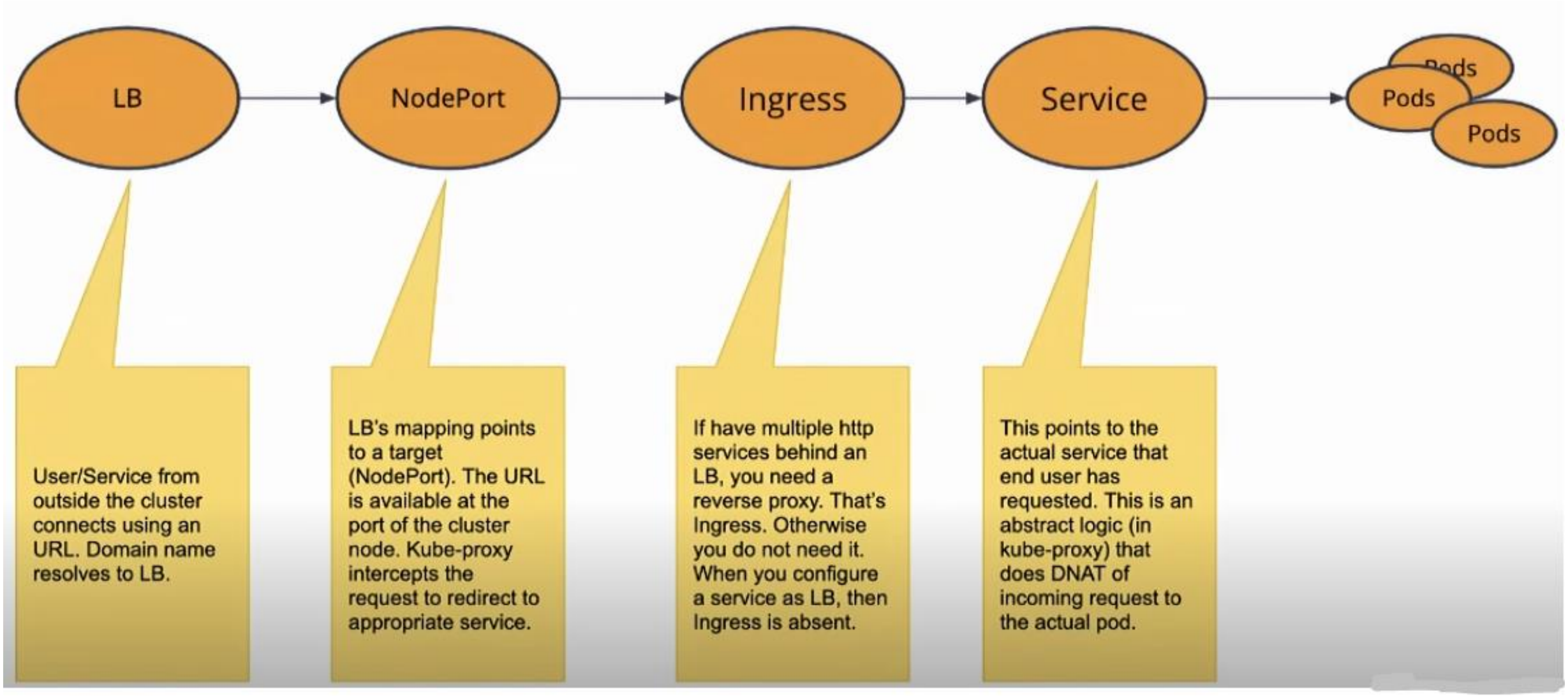
Pod networking –IP-IP/ L3 Mode



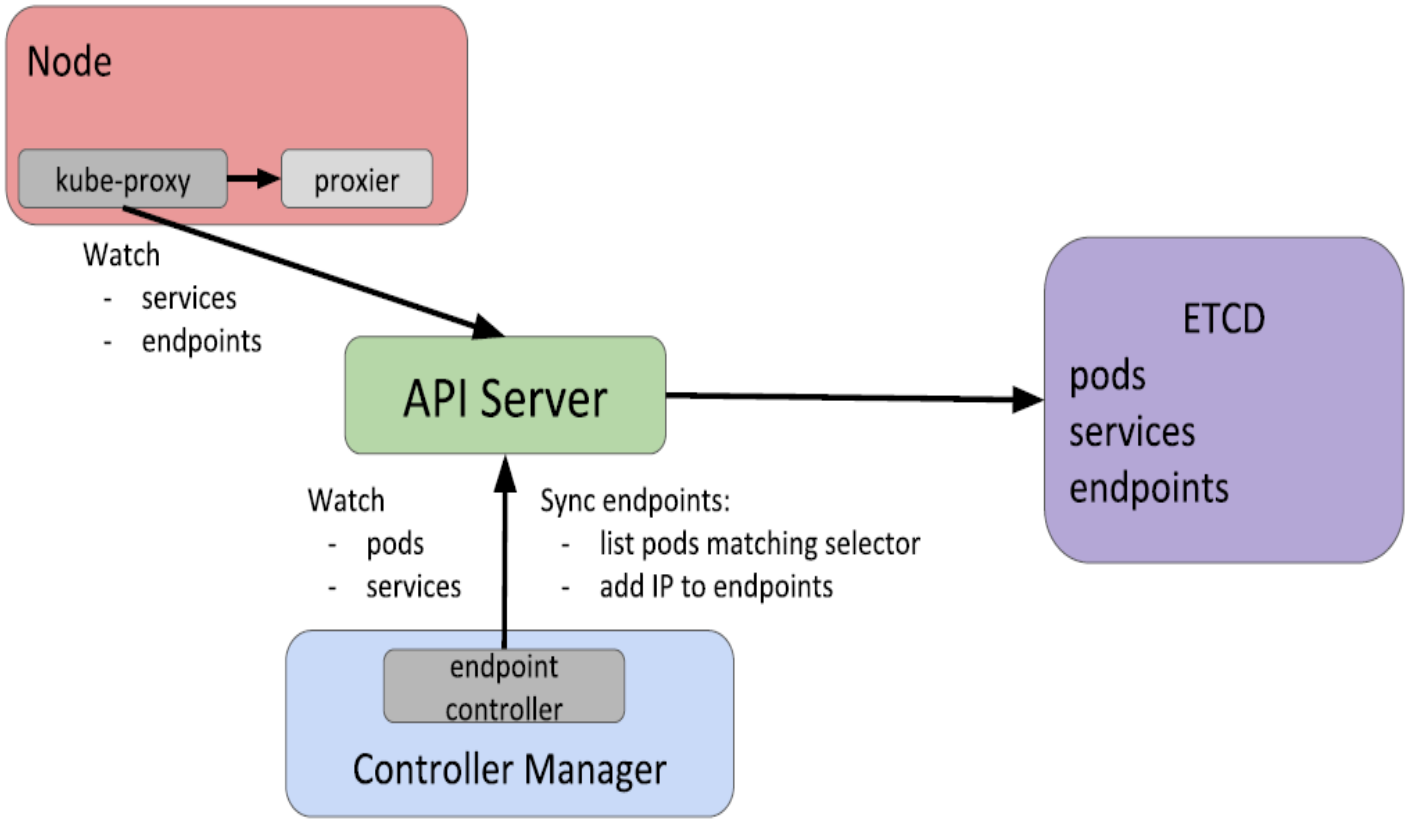
Pod networking –L2 Non encapsulated Mode



Traffic to a pod



Kube-Proxy



- **Userspace**
  - Original implementation
  - Userland TCP/UDP proxy
- **IPtables**
  - Default since Kubernetes 1.2
  - Use iptables to load-balance traffic
  - Faster than userspace
- **IPVS**
  - GA since Kubernetes 1.11
  - Use Kernel load-balancing (LVS)
  - Still relies on iptables for some NAT rules
  - Faster than iptables, scales better with large number of services/endpoints

Alternatives to Kube-proxy

Kube-router
- Pod Networking with BGP + Network Policies
- **IPVS based service-proxy**

Cilium
- **Relies on eBPF to implement service proxying**
- Implement security policies with eBPF

Service

- Service IP is accessible only from within the cluster
- Type “ClusterIP” is used for internal communication within the cluster
- Traffic sent to Service IP gets load balanced to pods that belong to service IP
- Source NAT is used for pods to communicate to external world
- IPtables are used extensively for load balancing and NAT
- Nodeport, Network load balancer and Ingress controller are service types for external world to reach services inside the cluster.

Service/Cluster IP Range

- The subnet reserved to allocate Cluster IPs in a cluster for various Services.
- **Non overlapping with the host network CIDR.**
- The size of the subnet depends on how many services are planned to be hosted in the cluster.
- Can be much smaller than Pod subnet.

```
apiVersion: v1
kind: Service
metadata:
  name: productpage
spec:
  selector:
    app: productpage
  type: ClusterIP
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
```

Kube-dns

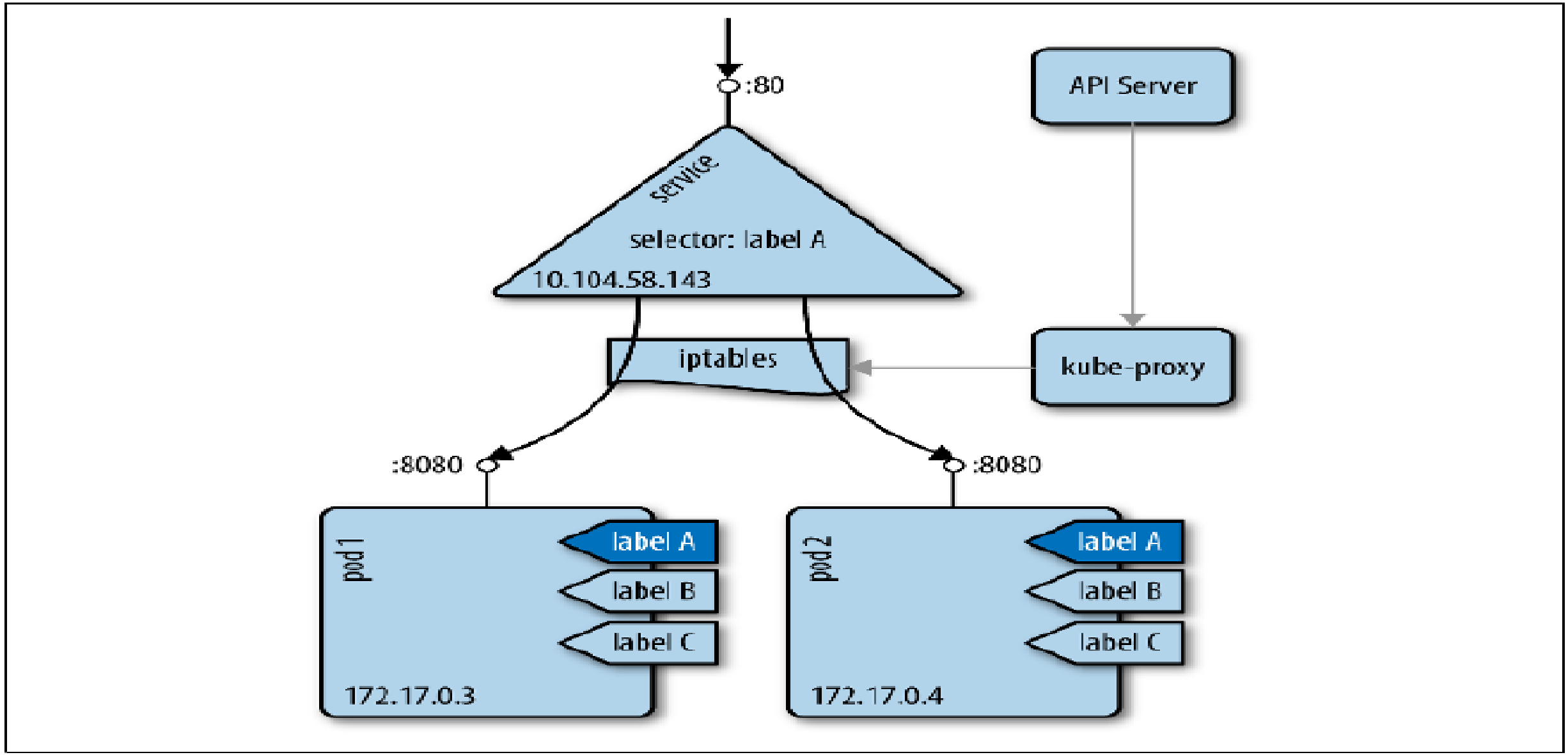
kube-dns

- The component responsible for providing Service Discovery functionality in a kubernetes cluster.
- The DNS pods are exposed to other pods in the cluster using DNS Service's a Cluster IP.
- The scale of the Deployment depends on the actual cluster size/load.

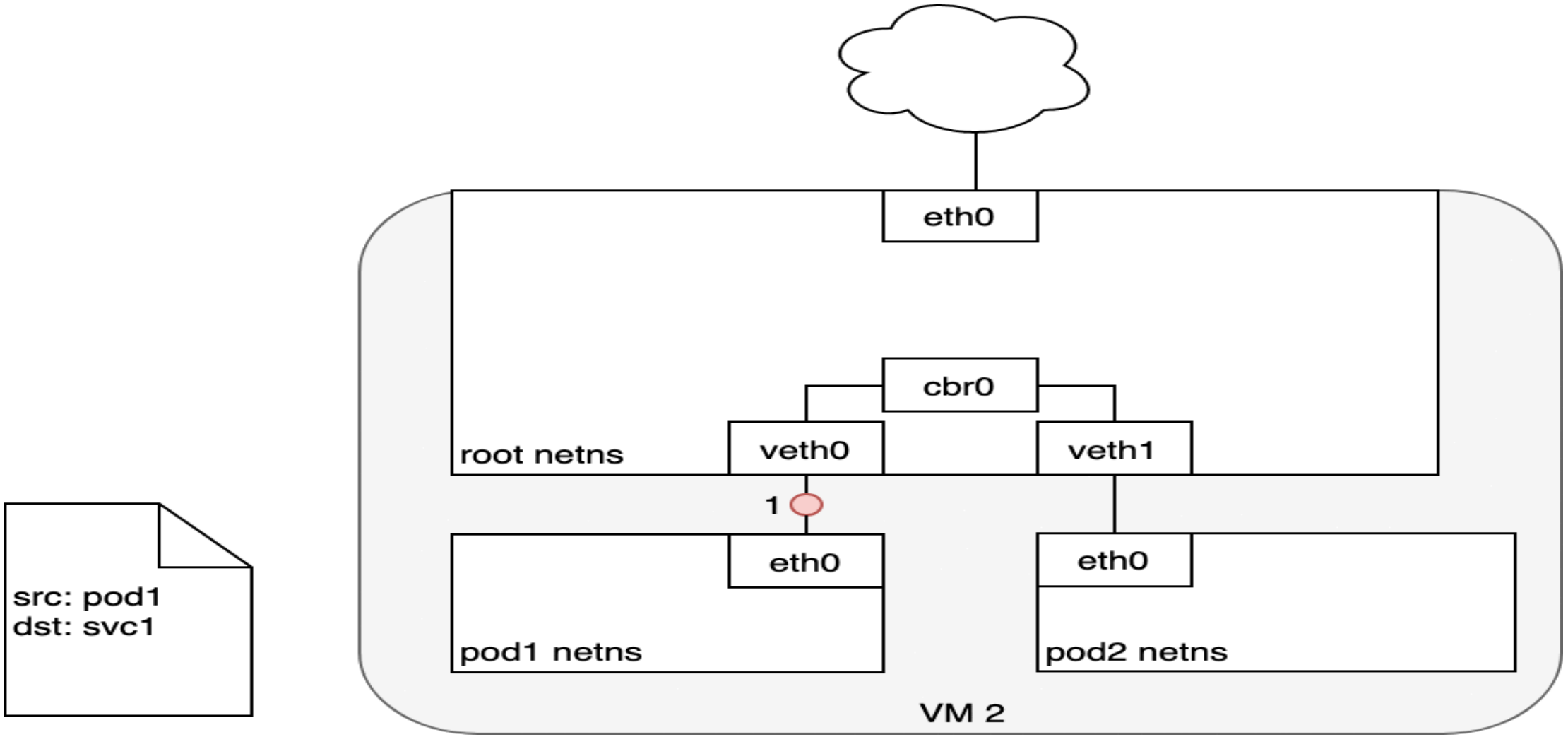
Service discovery

- Within the same namespace, services are discoverable using their names.
- Across different namespaces, services are discoverable using the format:
  - service-name.namespace-name
- Default domain: cluster.local
- DNS A records: service-name.namespace-name.svc.cluster.local

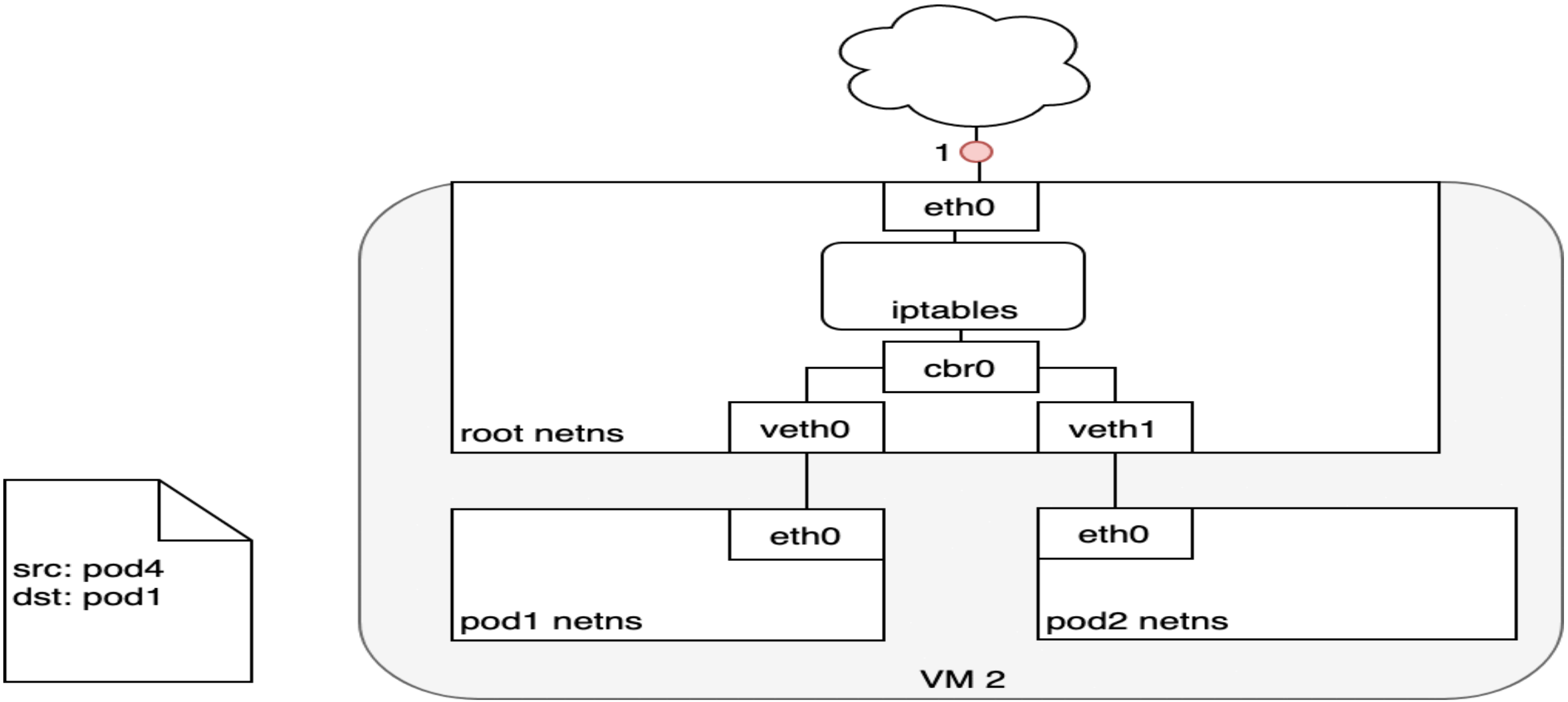
Kubernetes service concept



Pod to Service Networking



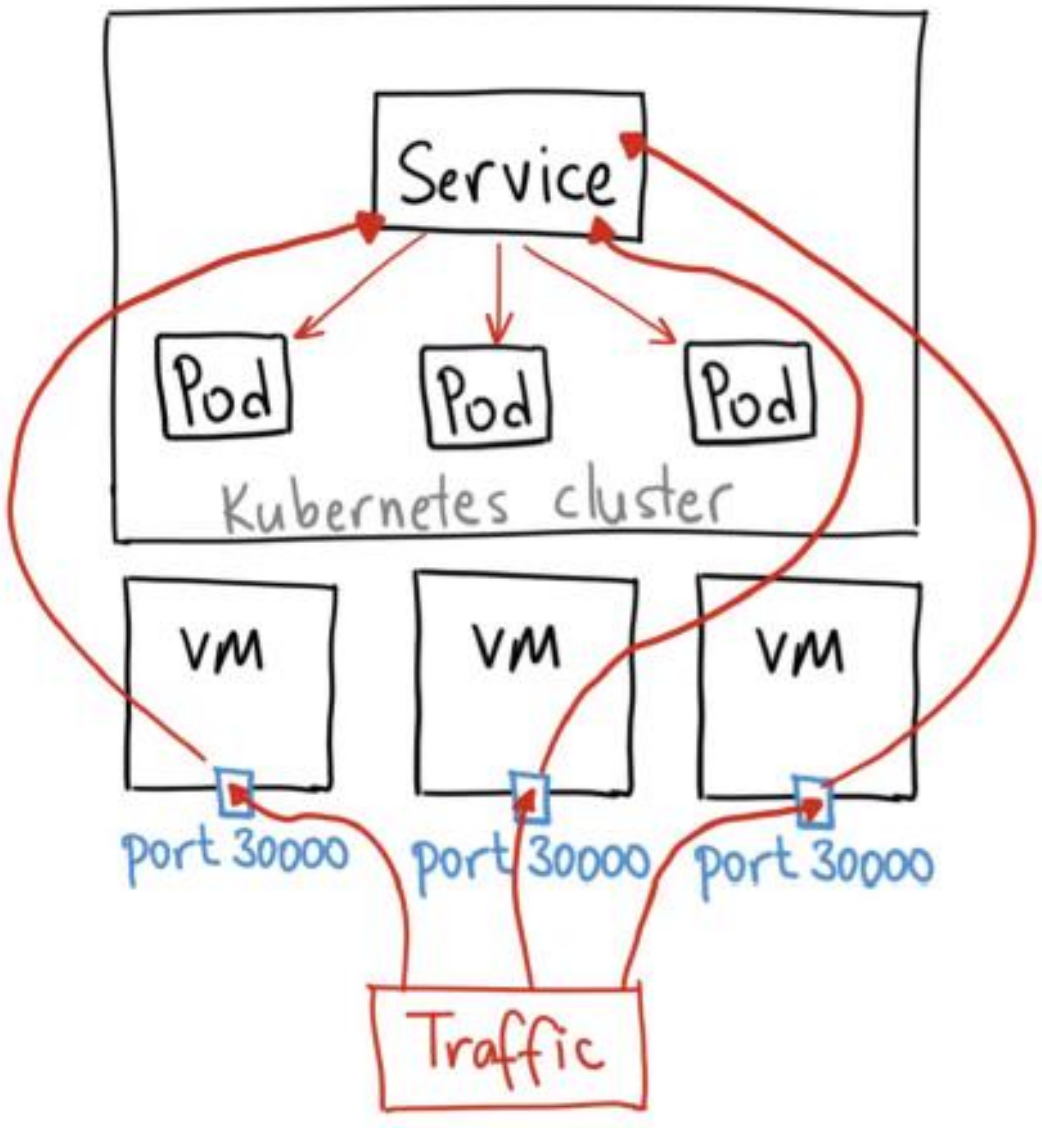
Service to Pod Networking



Service Networking Options

Feature	Nodeport	Load balancer	Ingress
Summary	Service is exposed using a reserved port in all nodes of cluster(Default: 32000-32767)	Typically implemented as network load balancer	Typically implemented as http load balancer
IP address	Node IP is used for external communication	Each service needs to have own external IP	Many services can share same external IP, uses path based demux
Use Case	Demo purposes	L3 services	L7 services
Examples		GKE Network load balancer	GKE http load balancer, nginx, Istio

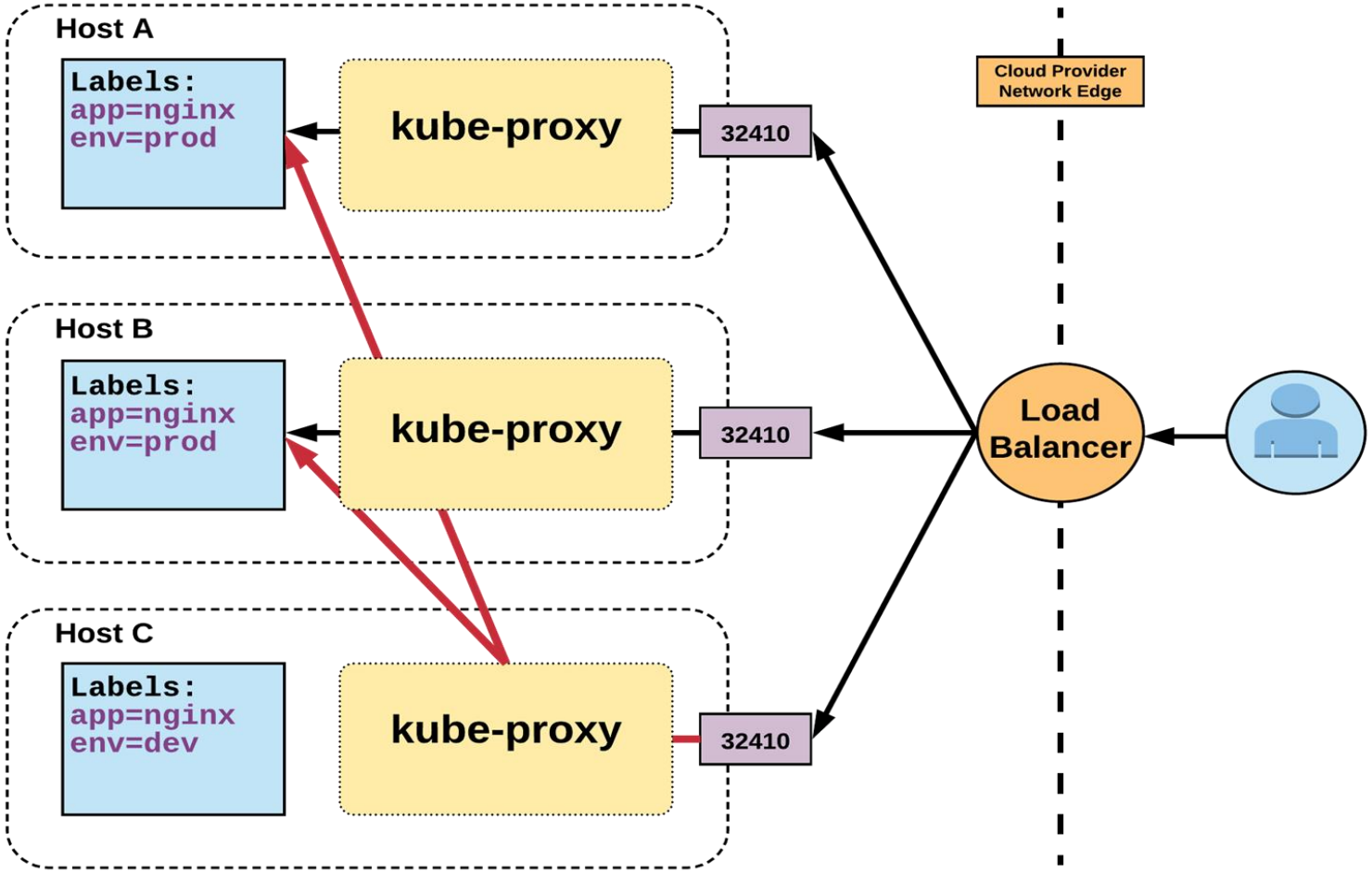
Nodeport




```
apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
spec:
  type: NodePort
  ports:
  - port: 30000
    targetPort: 9080
  selector:
    app: productpage
```

Load Balancer

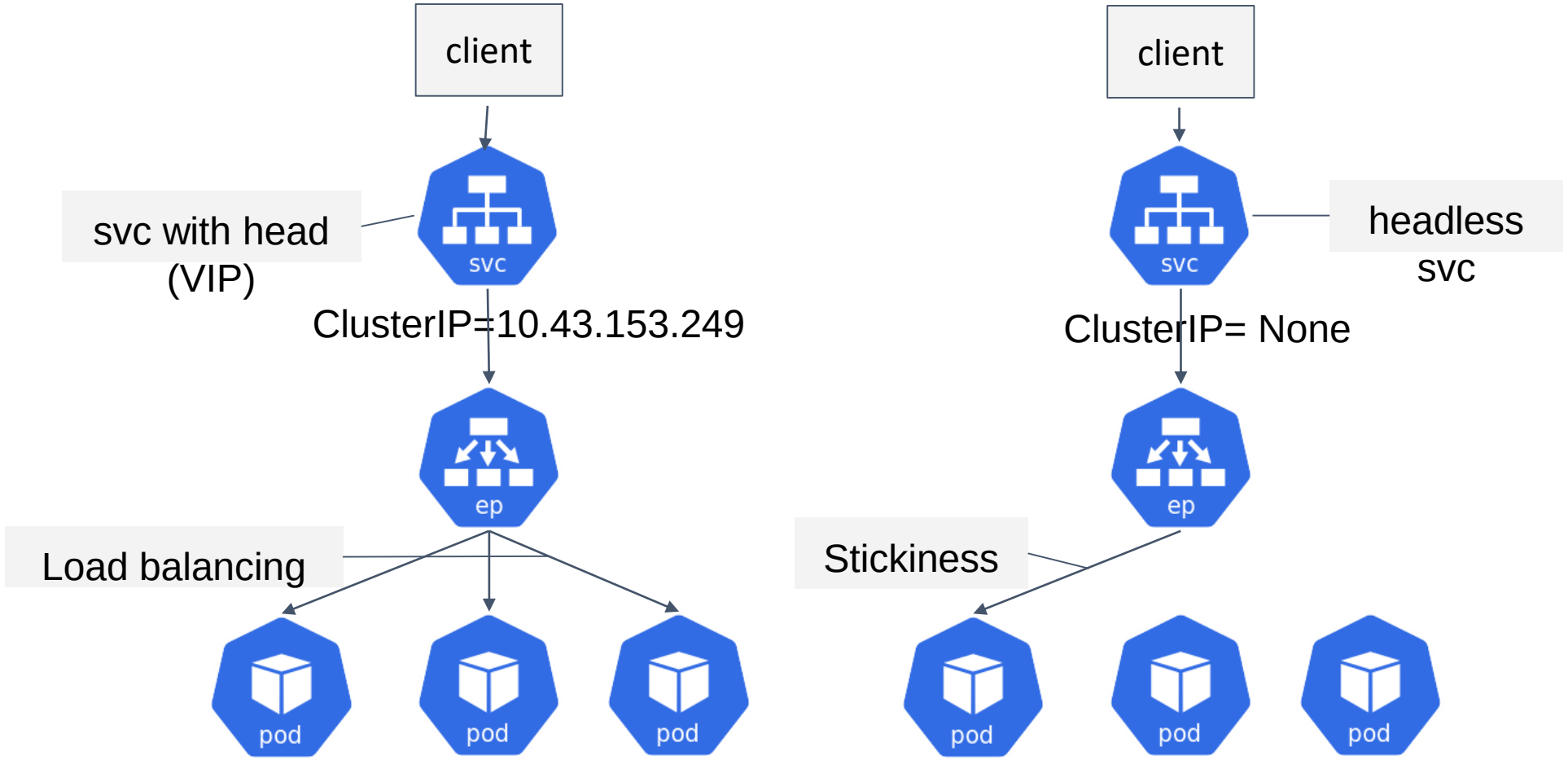
- **LoadBalancer** services extend **NodePort.**
- Works in conjunction with an external system to map a cluster external IP to the exposed service.




```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    env: prod
  ports:
    protocol: TCP
    port: 80
    targetPort: 80
```

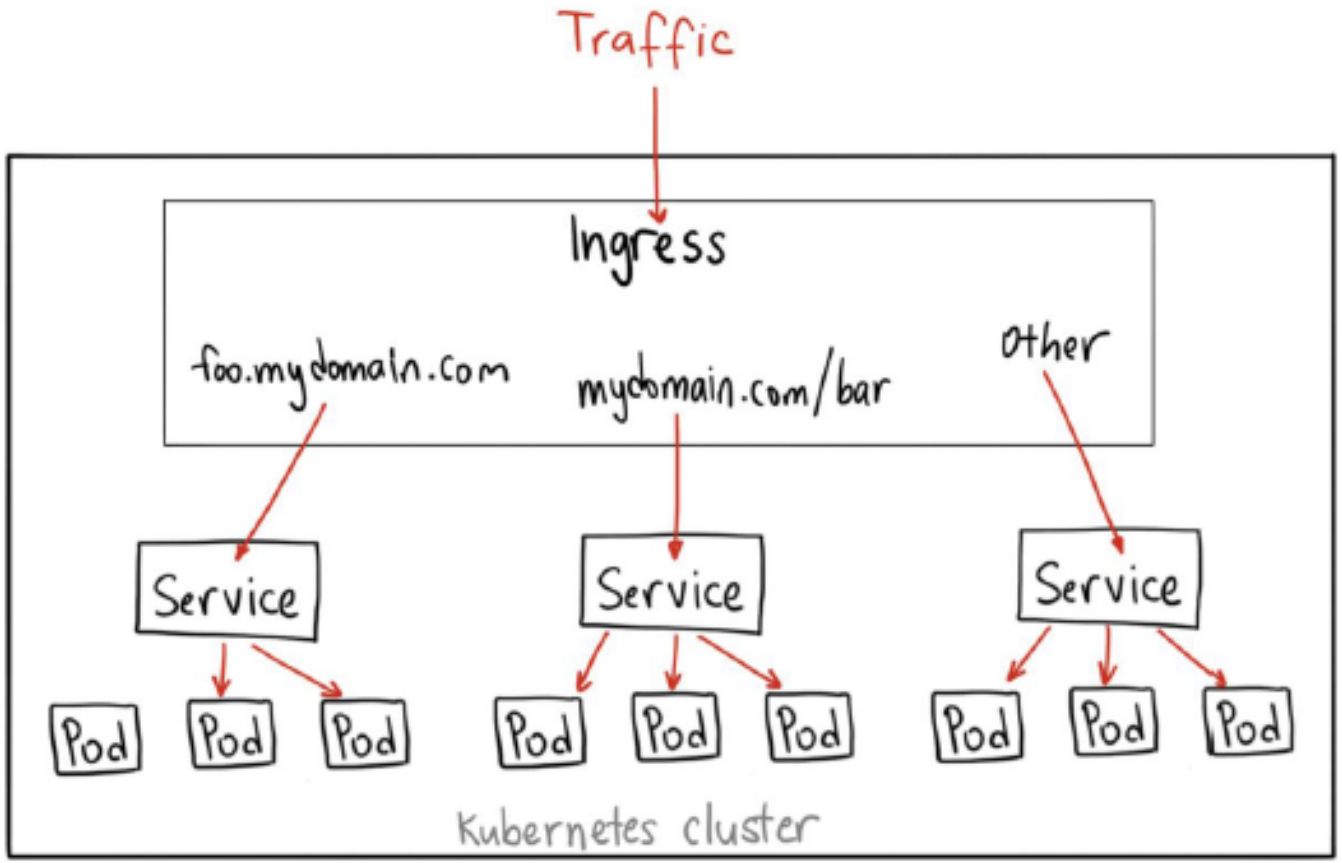
```
Name:          example-prod
Selector:      app=nginx,env=prod
Type:          LoadBalancer
IP:            10.96.28.176
LoadBalancer
Ingress:       172.17.18.43
Port:          <unset>  80/TCP
TargetPort:    80/TCP
NodePort:      <unset>  32410/TCP
Endpoints:     10.255.16.3:80,
               10.255.16.4:80
```

Kubernetes Headless vs. ClusterIP and traffic distribution



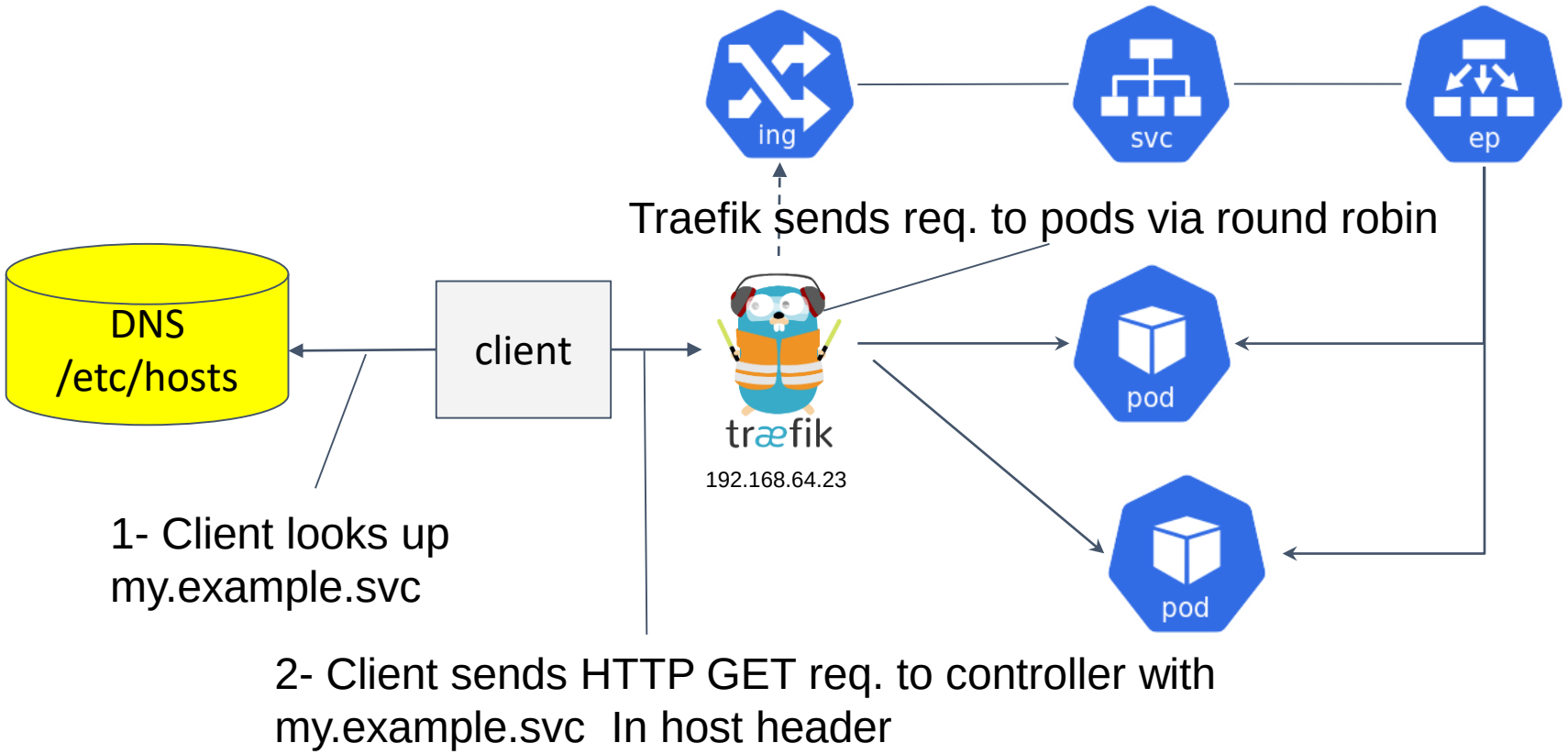
https://github.com/arashkaffamanesh/practical-kubernetes-problems#headless-services-for-stickiness

Ingress

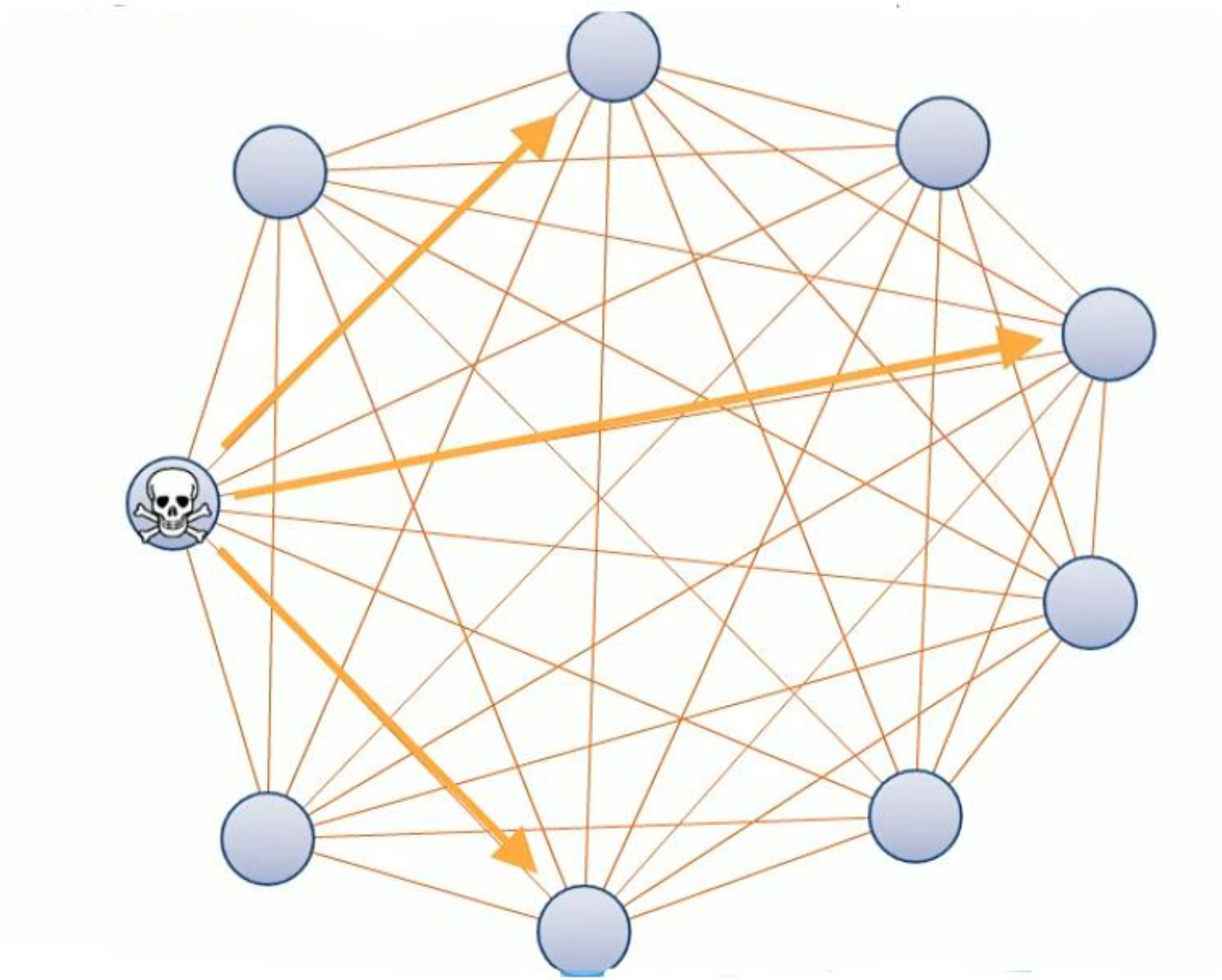



```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gateway
spec:
  backend:
    serviceName: productpage
    servicePort: 9080
  rules:
  - host: mydomain.com
    http:
      paths:
      - path: /productpage
        backend:
          serviceName: productpage
          servicePort: 9080
      - path: /login
        backend:
          serviceName: productpage
          servicePort: 9080
      - path: /*
        backend:
          serviceName: productpage
          servicePort: 9080
```

Ingress Layer7 Load balancing



Path way of attack



Network policy resources

Specifies how group of pods are allowed to communicate with each other and network endpoint using

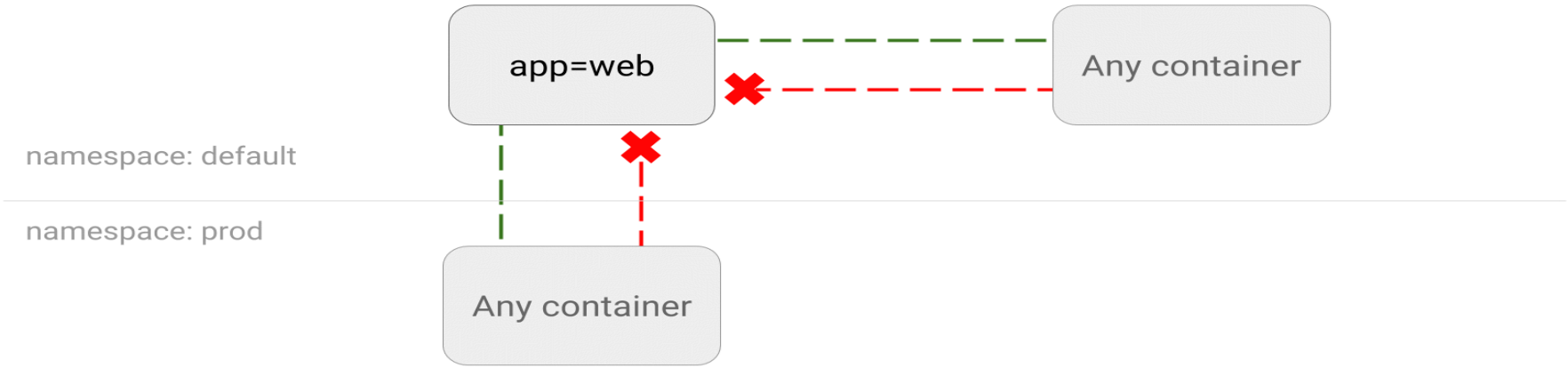
- pod label selector
- Namespace label selector
- protocols + ports
- service account based policy are possible with 3rdparty CNI llike calico

Network policy use cases

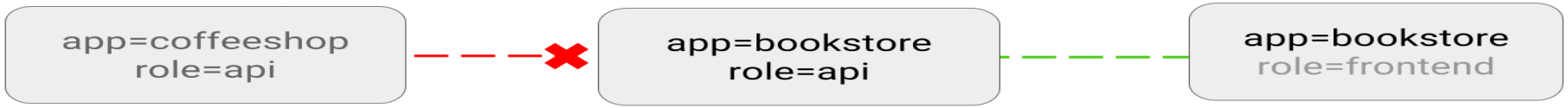
- Isolate dev/test/prod instances
- Translation traditional firewall rules eg 3tier architecture
- Tenant separation with namespaces along with network isolation
- Fine grained firewall rules to reduce surface attack within microservice based apps.
- Compliance eg: PCI

Tool: inspector gadget as Network Policy Advisor

DENY all traffic to an application

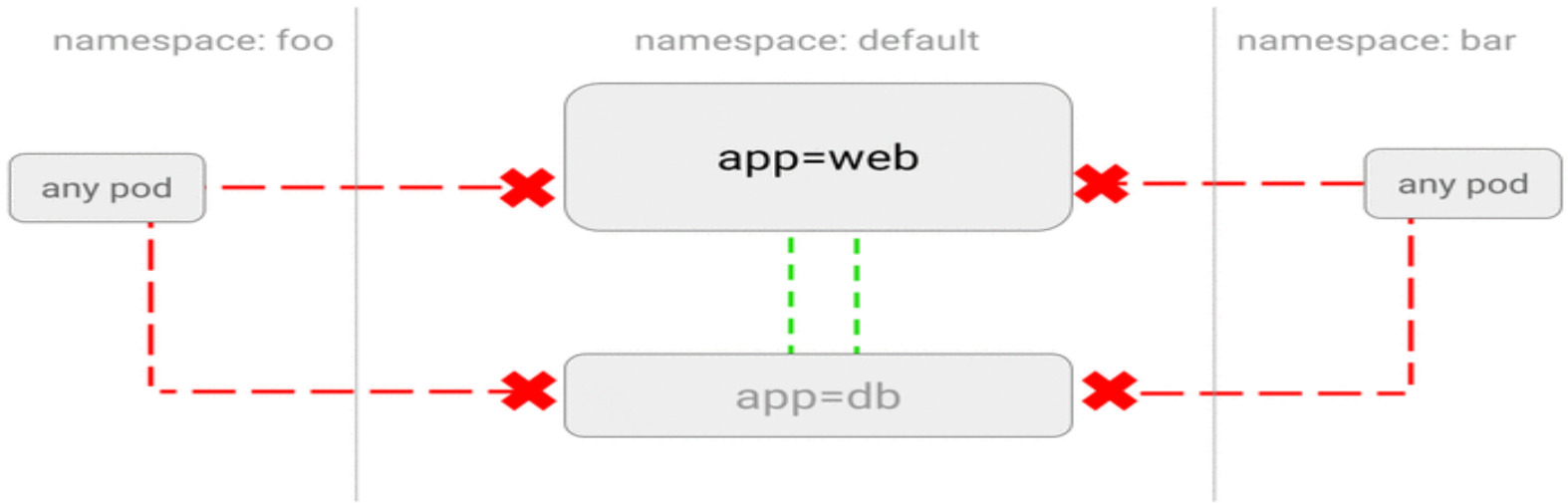


LIMIT traffic to an application

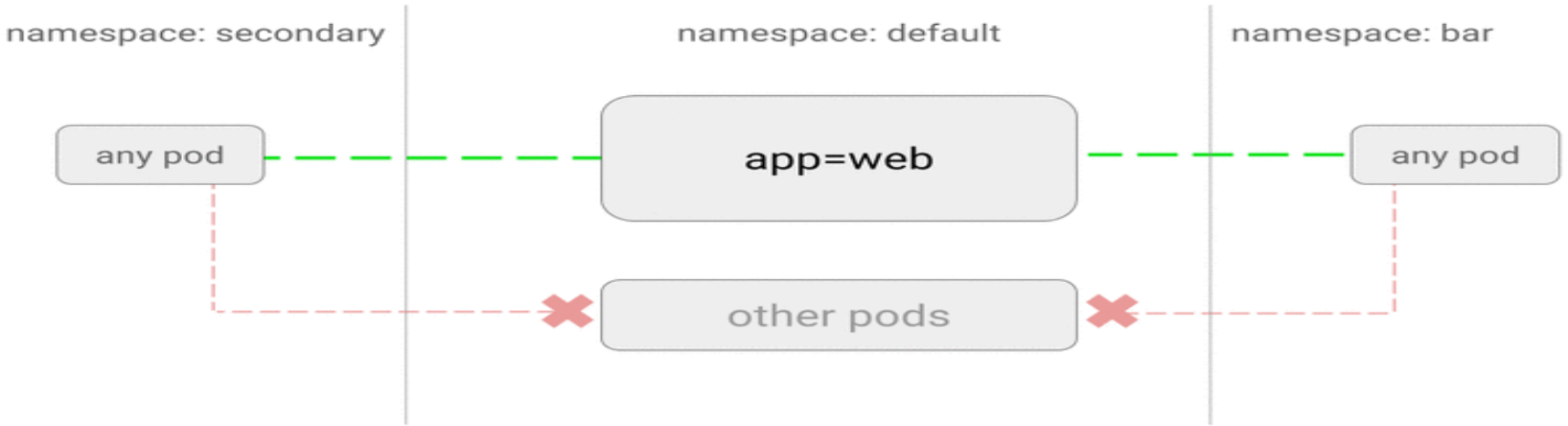


DENY all non-whitelisted traffic in a namespace

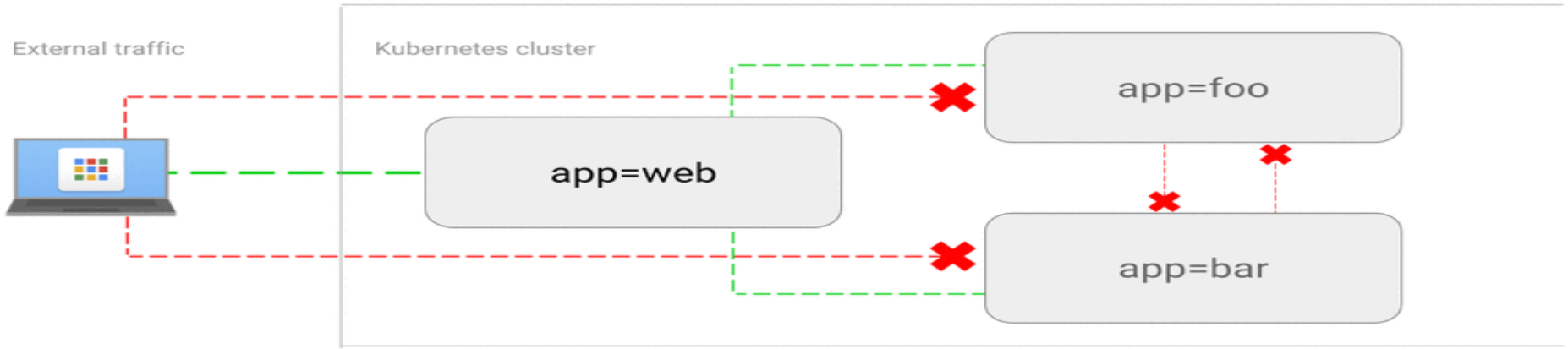
DENY all traffic from other namespaces



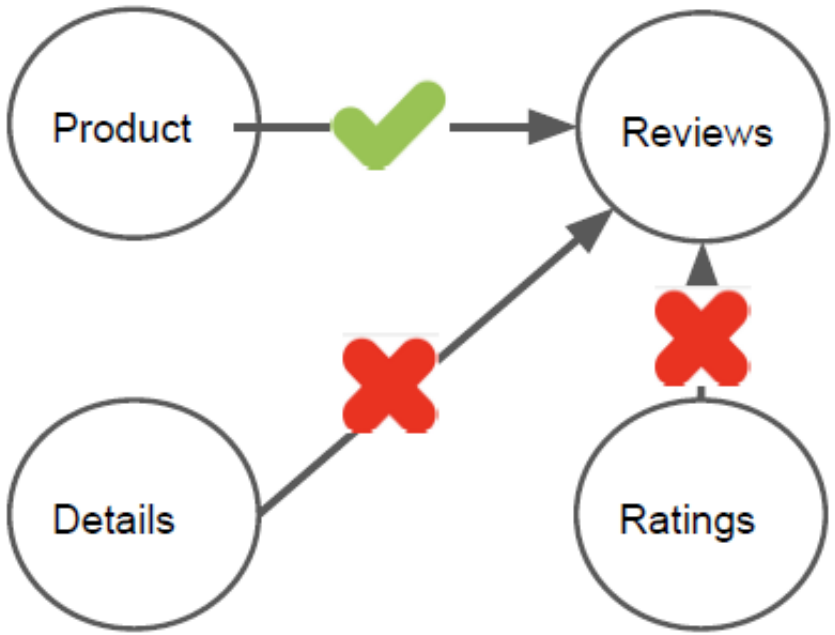
ALLOW traffic from other namespaces



ALLOW traffic from external clients

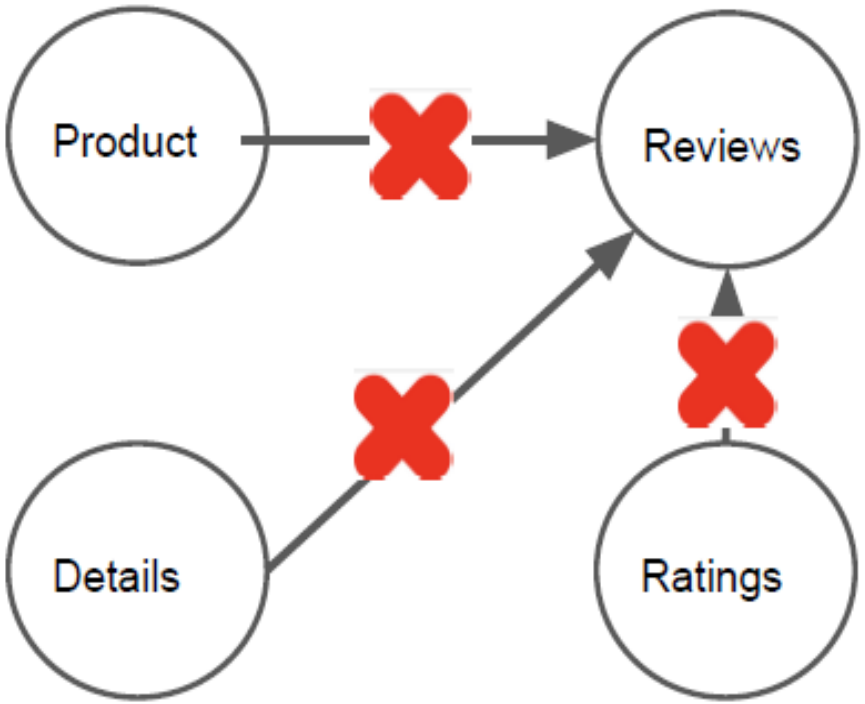


Network control policy example1



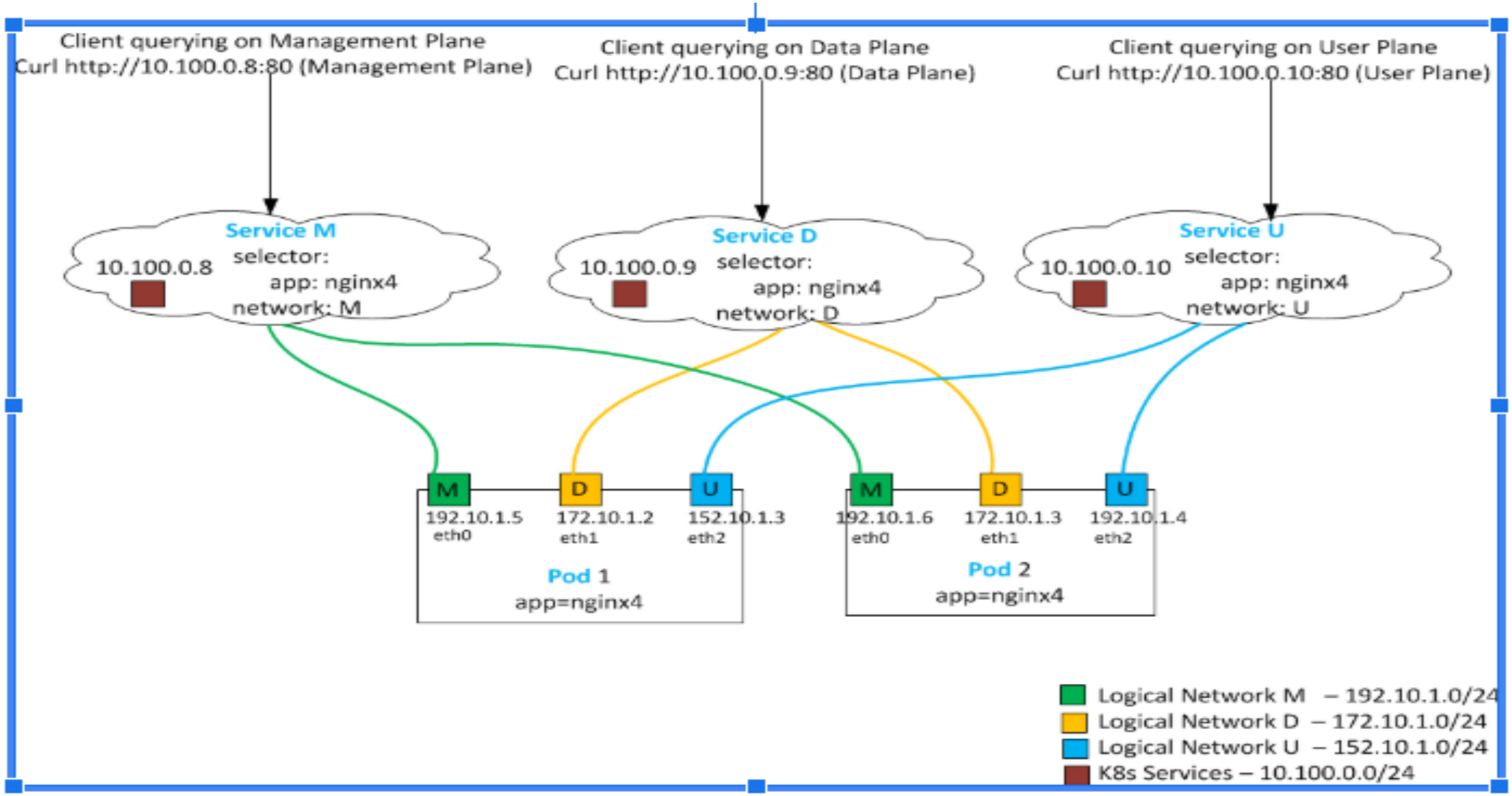

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: hello-allow-from-product
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: reviews
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: productpage
```

Network control policy example2

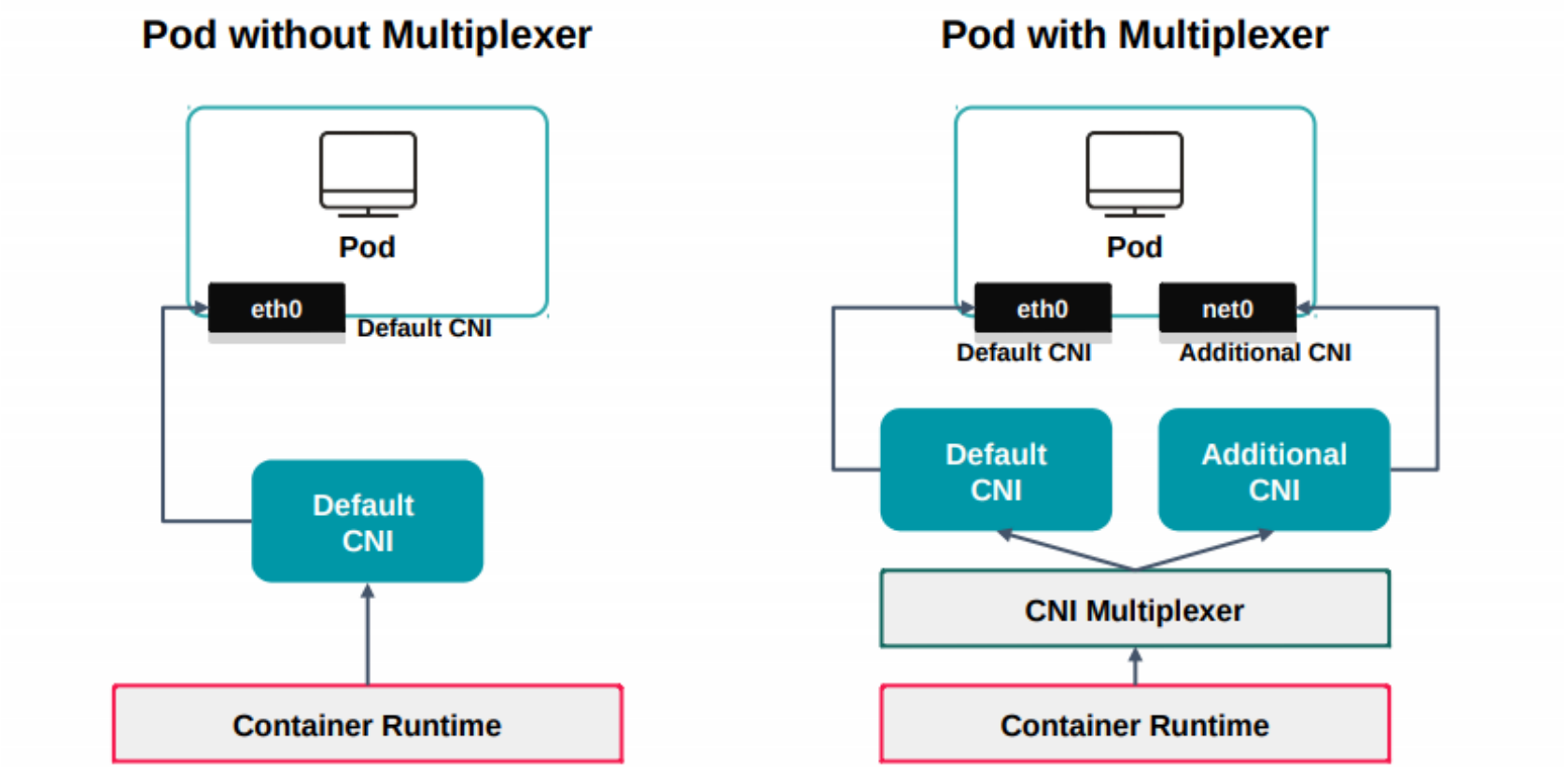



```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: hello-allow-from-product
spec:
  policyTypes:
  - Ingress
  podSelector:
    matchLabels:
      app: reviews
  ingress: []
```

Multi networking pods



Multi networking pods



- https://github.com/intel/multus-cni/
- https://github.com/cni-genie/CNI-Genie
- https://github.com/nokia/danm
- https://github.com/ZTE/Knitter
- https://tungsten.io/

Kubernetes networking –Planning

- Existing Network
  - Does the cluster span L3 segments?
  - BGP
  - HA between TOR and host
  - MTU
- CIDR
  - Ex. 40 servers/rack, 60 pods/server, and 120-node cluster = 7200 pods = /19 mask.
  - You need ip addresses for pods and services.
  - Consider autoscaling.
- Single vs. Multi-cluster

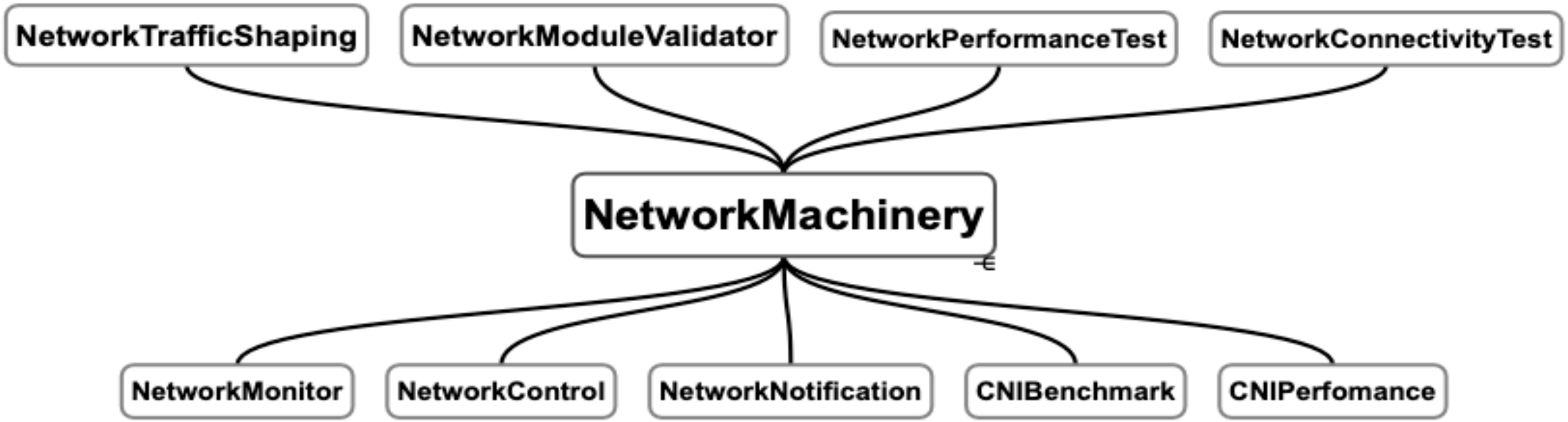
Pod Network troubleshooting

- pod might show up and it might not get an IP address. by looking at the Kubelet logs we will come to know which binary is called
- Check if the pod sending traffic to the right IP address & check network policies, labels and selectors
- Pod to service connection times out.Tcpdump could show that lots of repeated SYN packets are sent, but no ACK is received. Check sysctl net.ipv4.ip_forward
- If service and pod are on different network and getting timeout . Check sysctl net.bridge.bridge-nf-call-iptables
- If you're trying to reach another pod of IP over the network, is the pod sending traffic over the virtual Ethernet and is it getting to the host? It's getting to the host. Does the host have the right routes to be able to route traffic to the remote host to get to the other pod? In other words, is the routing table correct?. Firewall rules block overlay network traffic. Update it accordingly
- does the destination node have the right set of routes and is it sending it to the destination pod?
- For the return traffic back from the destination port back to the source, you would essentially retrace the steps in the opposite direction.
- Pod CIDR Conflicts. Double-check what RFC1918 private network subnets are in use in your network, VLAN or VPC and make certain that there is no overlap.

Tools

- https://github.com/nicolaka/netshoot
- https://github.com/eldadru/ksniff
- https://github.com/gravitational/satellite
- https://github.com/draios/sysdig-inspect

Networking CRD



Ref: https://github.com/networkmachinery/networkmachinery-operators

https://operatorhub.io/operator/skydive-operator

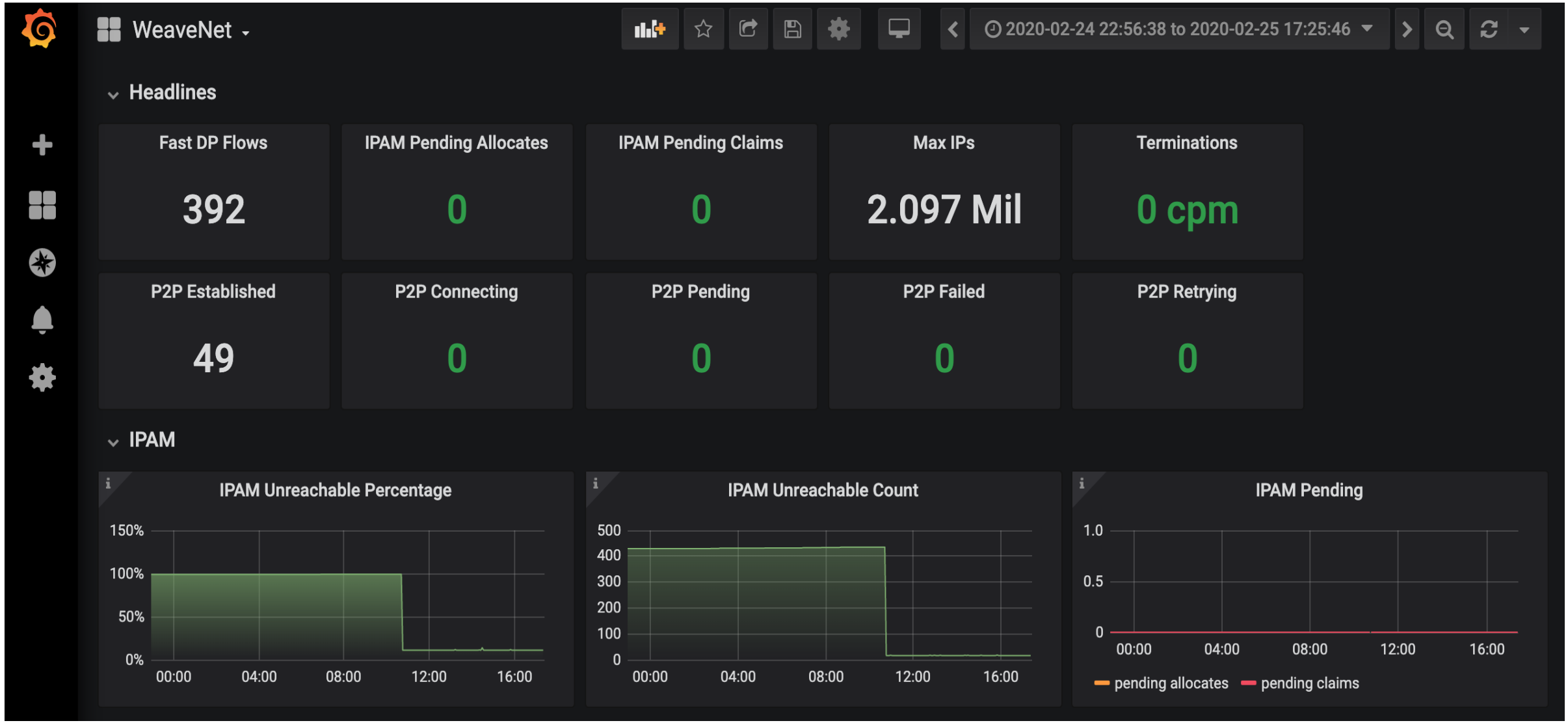
Kubernetes Networking Best Practices

- Keep control plane and data plane on separate subnet while initializing cluster
- IP address management on workloads.(Cluster,service,node)
- Encapsulation technique(vxlan,ip-ip)
- Implement labeling on pods,deployments,services
- Make sure IPVS mode is enabled in kubeproxy
- Avoid unnecessary external load balancer IP allocation to service.
- Apply network policy to control communication inside the cluster.
- Revisit pod scheduling characteristics across nodes
- Research your overlay network’s back-end configuration options (including potential integrations with network hardware SDN management systems) to ensure that you’re picking the best back-end for your network design requirements.
- HA between top of the rack switches

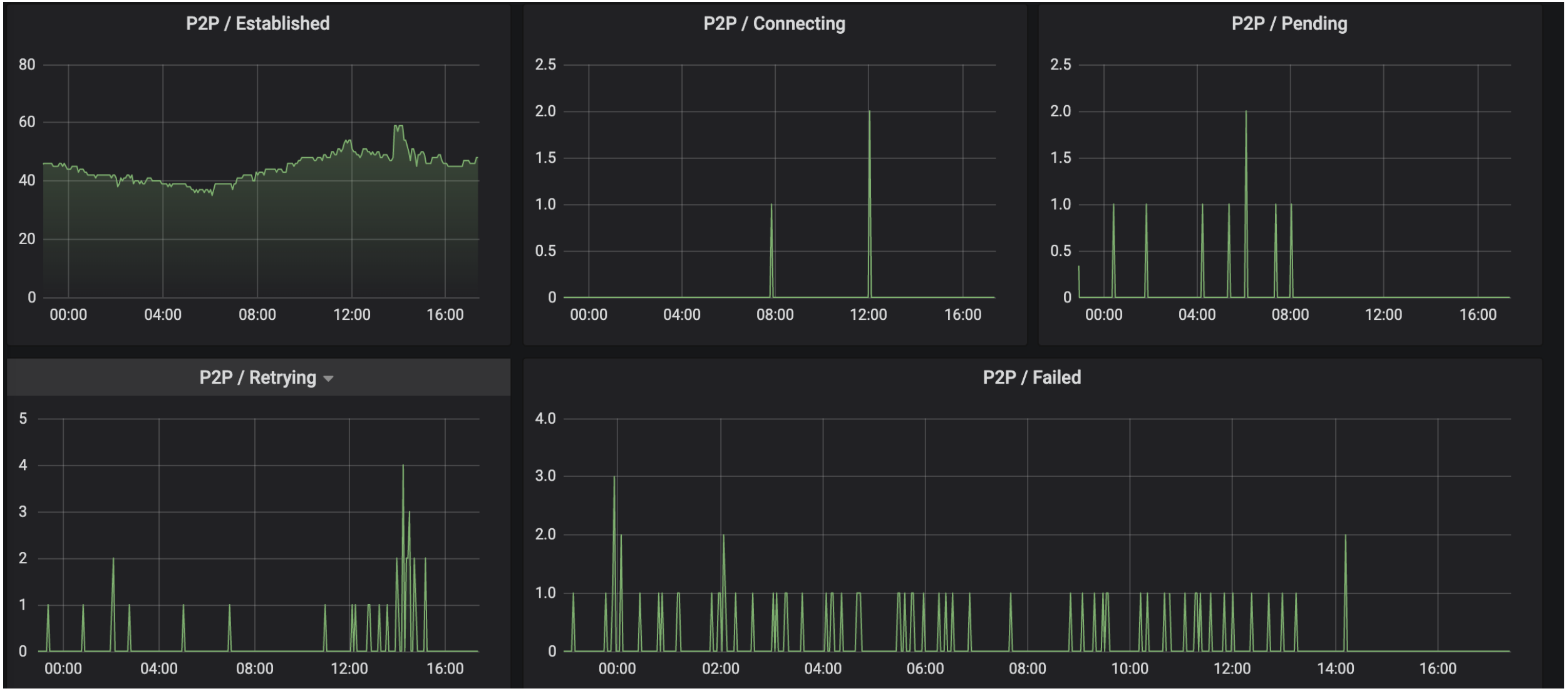
Kubernetes Networking Metrics

- Some of the metrics about CNI plugin on Prometheus
  - Number of peer-to-peer connections
  - Number of peer-to-peer connections **terminated**
  - Number of IP addresses
  - Size of IP address space used by allocator
  - Number of DNS entries
  - Number of unreachable peers that own IPAM addresses
  - Percentage of all IP addresses owned by unreachable peers
  - Number of pending allocates

Kubernetes Networking Metrics



Kubernetes Networking Metrics



Reference

- https://github.com/collabnix/dockerlabs
- https://docs.docker.com
- http://www.collabnix.com
- https://kubernetes.io/docs/concepts/cluster-administration/networking/
- https://sookocheff.com/post/kubernetes/understanding-kubernetes-networking-model/
- https://www.digitalocean.com/community/tutorials/how-to-inspect-kubernetes-networking
- https://success.docker.com/article/docker-ee-best-practices#astandarddeploymentarchitecture
- https://success.docker.com/article/networking
- https://kubernetes.io/blog/2016/12/container-runtime-interface-cri-in-kubernetes/
- https://medium.com/@reuvenharrison/an-introduction-to-kubernetes-network-policies-for-security-people-ba92dd4c809d
- https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0
- https://sreeninet.wordpress.com/2016/05/29/docker-macvlan-and-ipvlan-network-plugins/

Thank You