Kubernetes 101 for Network Professionals

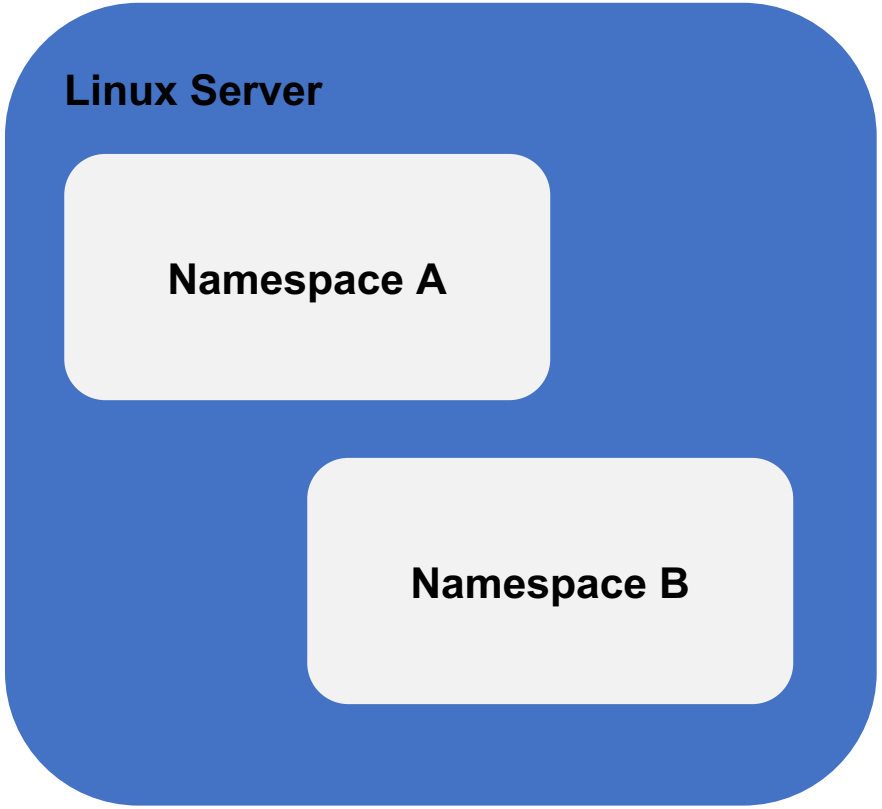
FEB-2023

Mau Rojas

bio.site/pinrojas

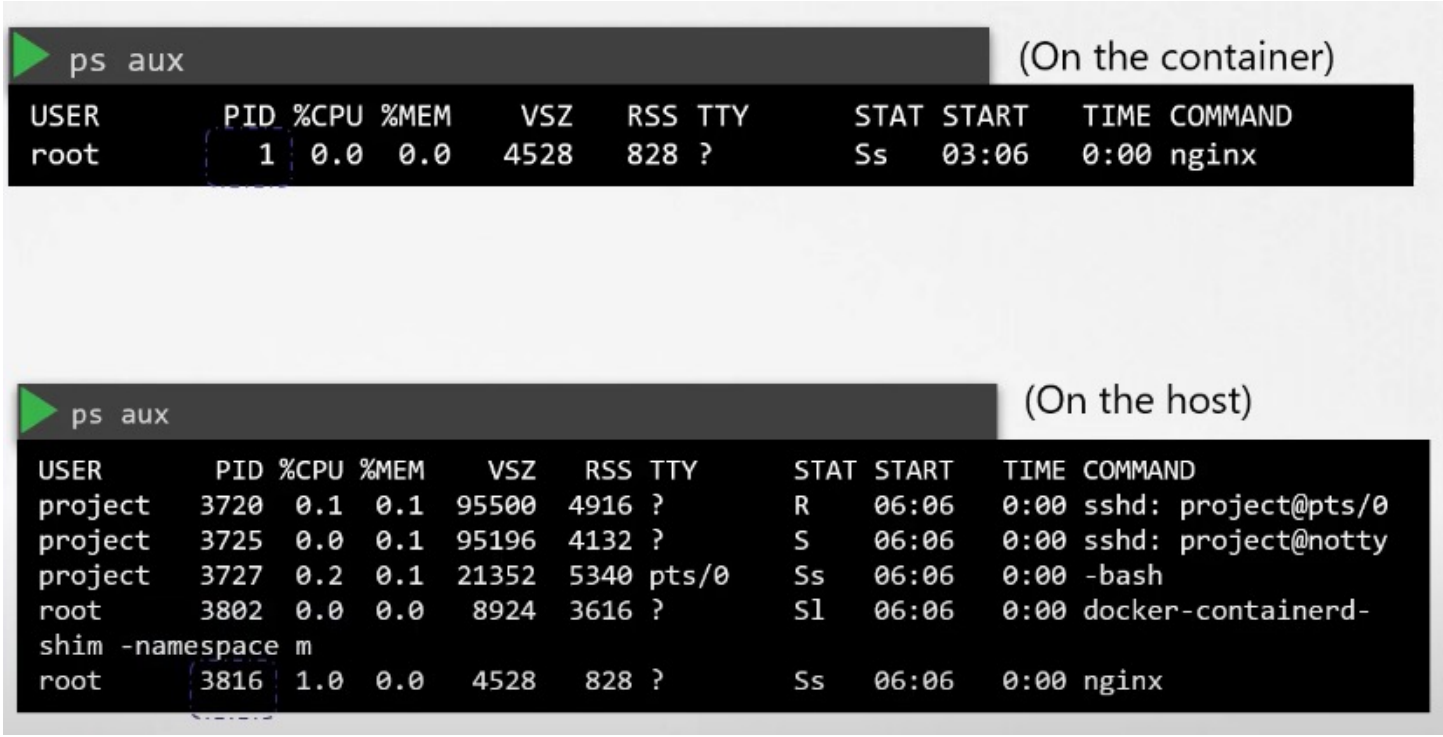
Network Namespaces

- What is a Namespace?
- Use cases
- How do they work and how to use it?



Network Namespaces

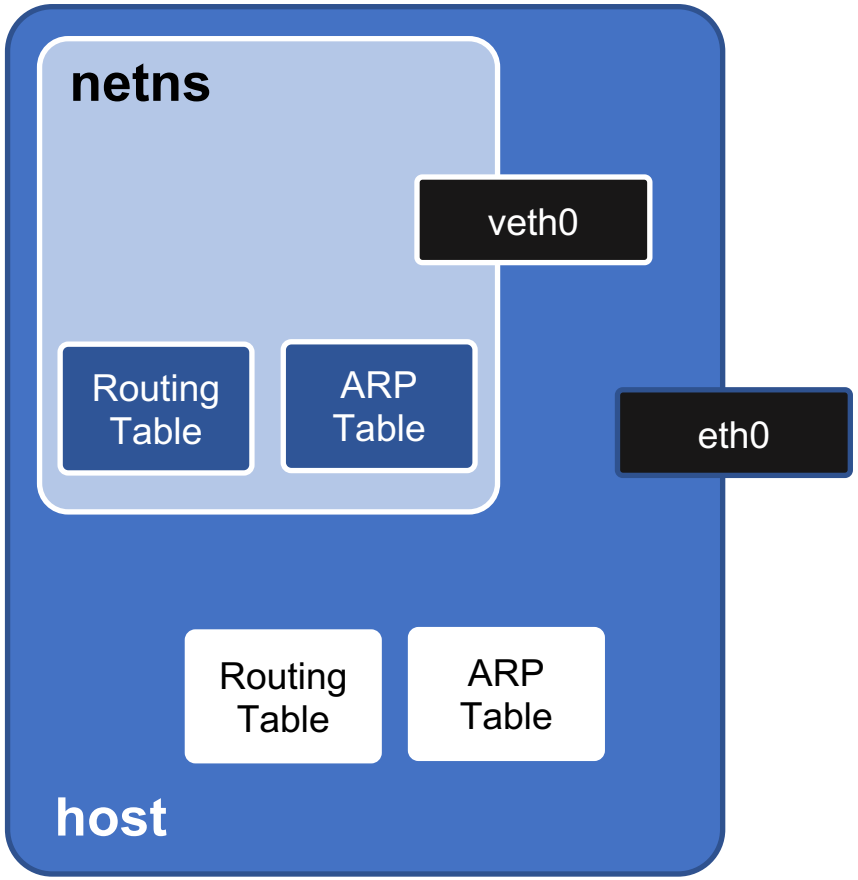
What is a namespace?



- Partition of kernel resources
- Resources may exist in multiple spaces:
  - Process IDs
  - Hostnames
  - File names
  - User IDs
  - **Network**
- Fundamental aspect of containers

Network Namespaces

Example

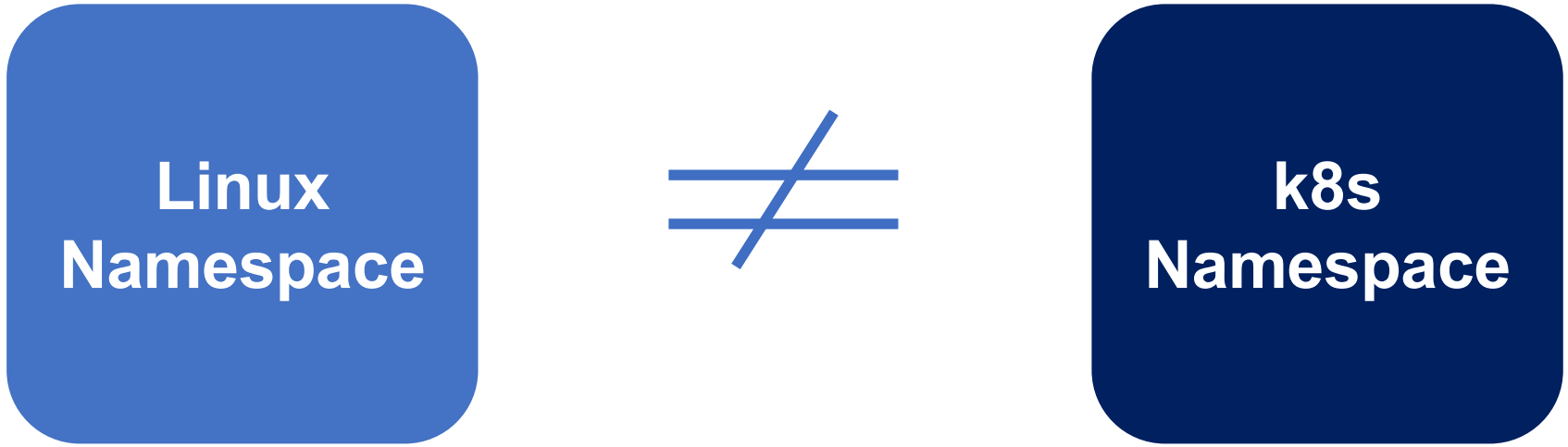



```
ip netns add texas
ip netns add california

ip netns exec texas ip link
ip netns exec california ip link
```

Network Namespaces

Linux namespaces vs K8s namespaces



Kernel resource partition

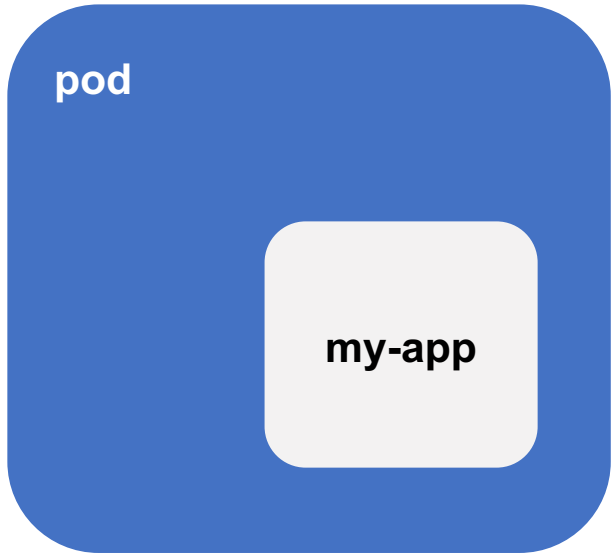
Pointer in etcd

(project to organize and secure pods)

Pods and Containers

Container Communication

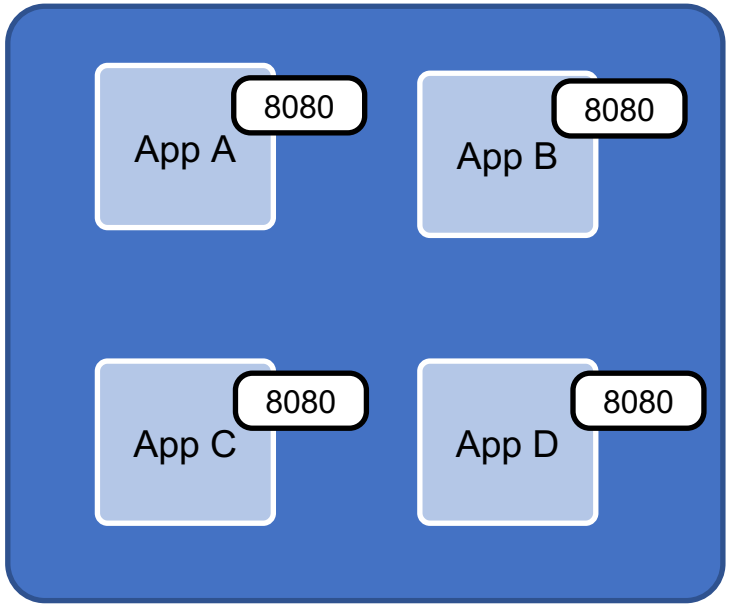
- Why is a **Pod abstraction useful**?
  - Pod fundamentals
- Container vs Pods
- When are **multiple containers** necessary in one pod?
- How do **containers communicate in a Pod**?



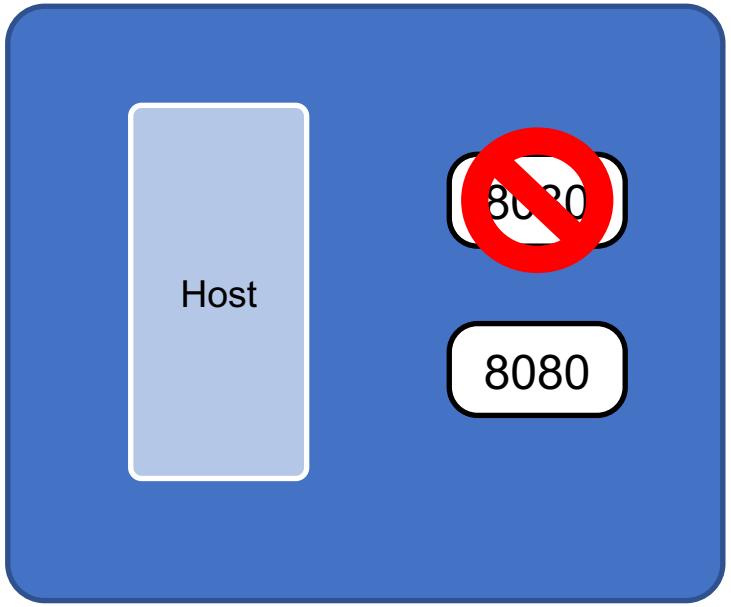
Pods and Containers

Container Port Mapping before Pods

```
docker run -o 5432:5432 -e POSTGRESS_PASSWORD=pwd postgres:9.6.17
```



How to allocate ports without conflicts?

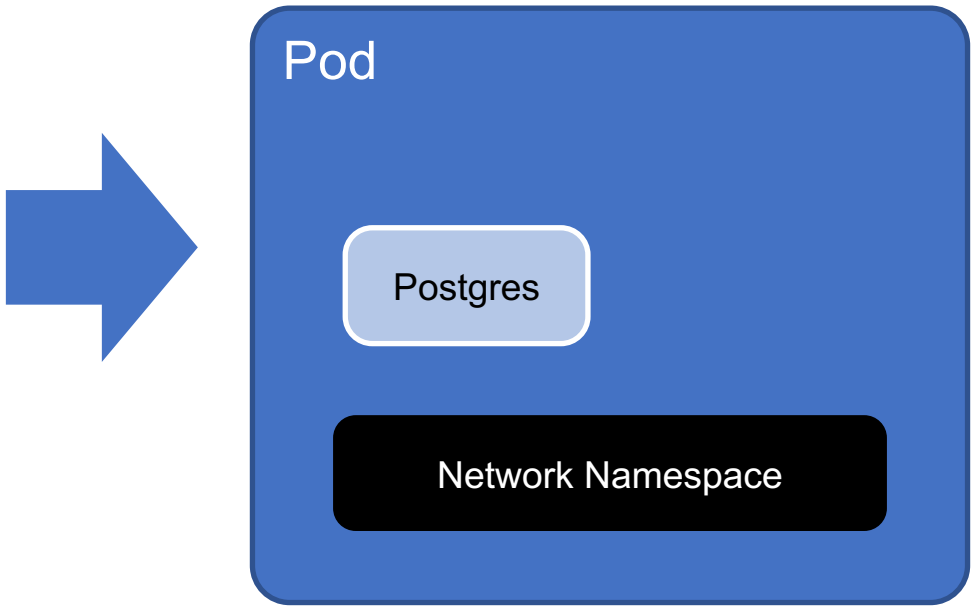


Pods and Containers

Pod Abstraction

Hundreds of containers per server. **Hard to keep track** of all ports to bind.

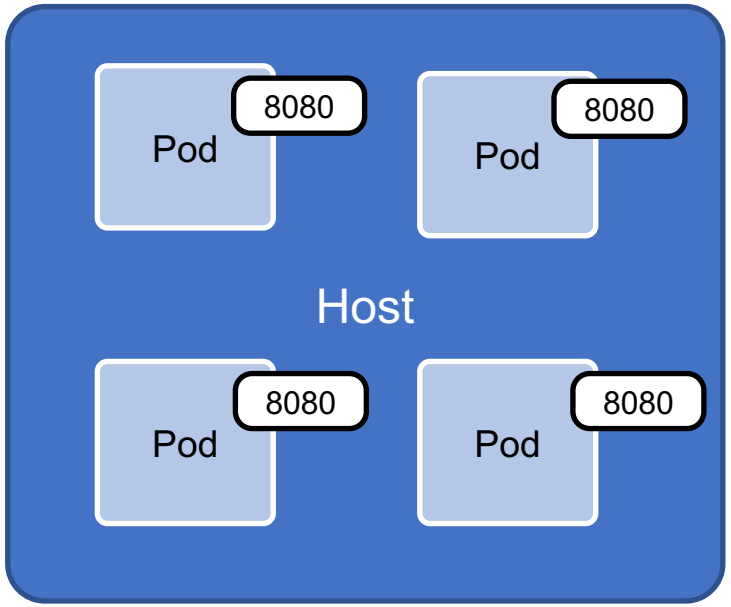
What ports are still free?



- **Network Namespace**
- **Virtual Ethernet Connection**
- **Pod is a Host**
  - IP addresses
  - Range of ports to allocate

Pods and Containers

- Every Pod has a unique IP address
- IP address reachable from all other Pods in the K8s cluster



No conflicts

```
apiVersion: v1
kind: Pod
metadata:
  name: postgress
  labels:
    app: postgress
spec:
  containers:
    - name: postgress
      image: postgress:9.6.17
      ports:
       - containerPort: 80
       env:
       - name: POSTGRESS_PASSWORD
          value: “pwd”
```

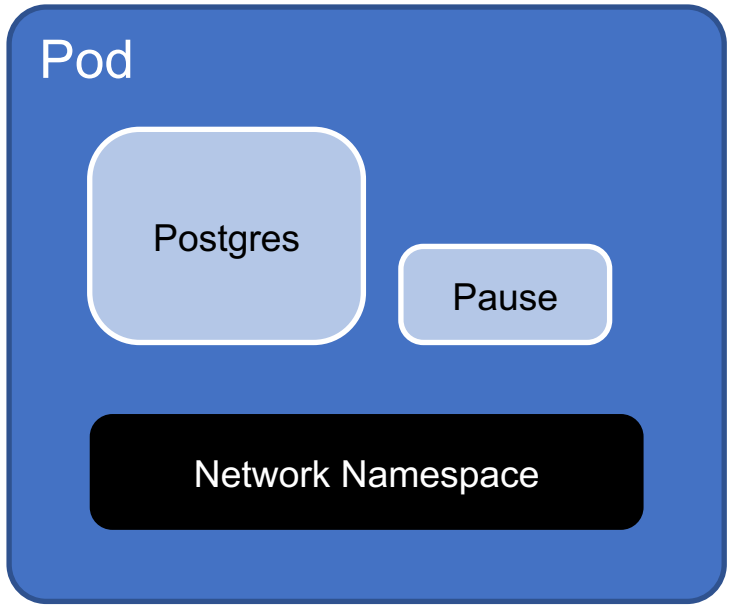
Example: netns/docker

https://github.com/cloud-native-everything/k8s-net-101


```
docker run -dt --rm --name netns-test1 --network none pinrojas/net-test:v0.3
docker run -dt --rm --name netns-test2 --network none pinrojas/net-test:v0.3
rm -rf /var/run/netns/netns-test1
rm -rf /var/run/netns/netns-test2
pid1=$(docker inspect -f '{{.State.Pid}}' netns-test1)
pid2=$(docker inspect -f '{{.State.Pid}}' netns-test2)
sudo ln -sf /proc/$pid1/ns/net /var/run/netns/netns-test1
sudo ln -sf /proc/$pid2/ns/net /var/run/netns/netns-test2
sudo ip link add veth1 type veth peer name veth2
sudo ip link set veth1 netns netns-test1
sudo ip link set veth2 netns netns-test2
sudo ip netns exec netns-test1 ip link set veth1 name eth0
sudo ip netns exec netns-test2 ip link set veth2 name eth0
sudo ip netns exec netns-test1 ip link set eth0 up
sudo ip netns exec netns-test2 ip link set eth0 up
sudo ip netns exec netns-test1 ip addr add 1.1.1.1/30 dev eth0
sudo ip netns exec netns-test2 ip addr add 1.1.1.2/30 dev eth0
```

Pods and Containers

Pause Container

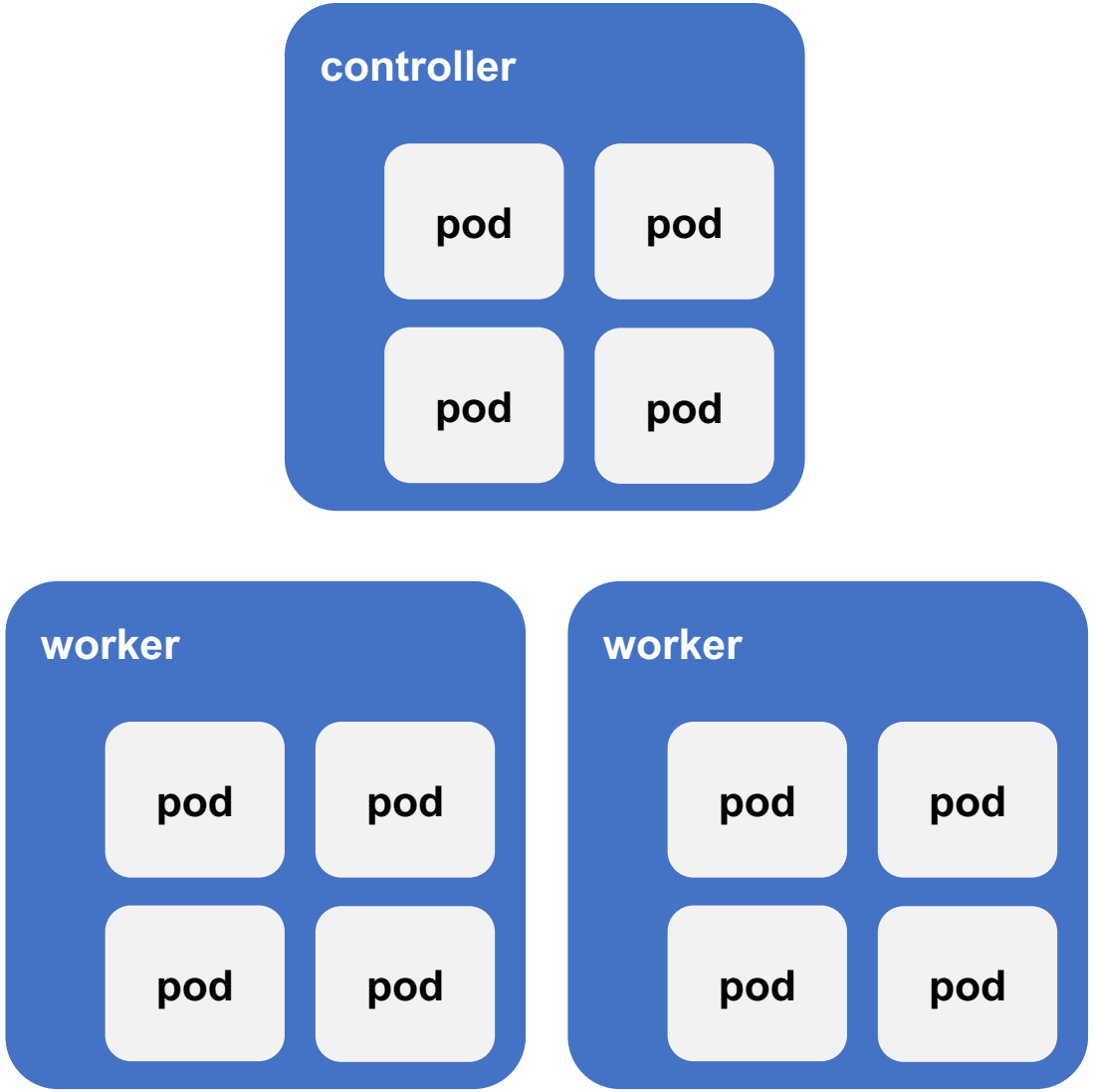


“pause” container in each pod

- **Reserves and hold netns**
  - **Main container crashes, pause hold IP settings**
  - **Pod crashes, IP settings change**
- **Enable communication between containers**

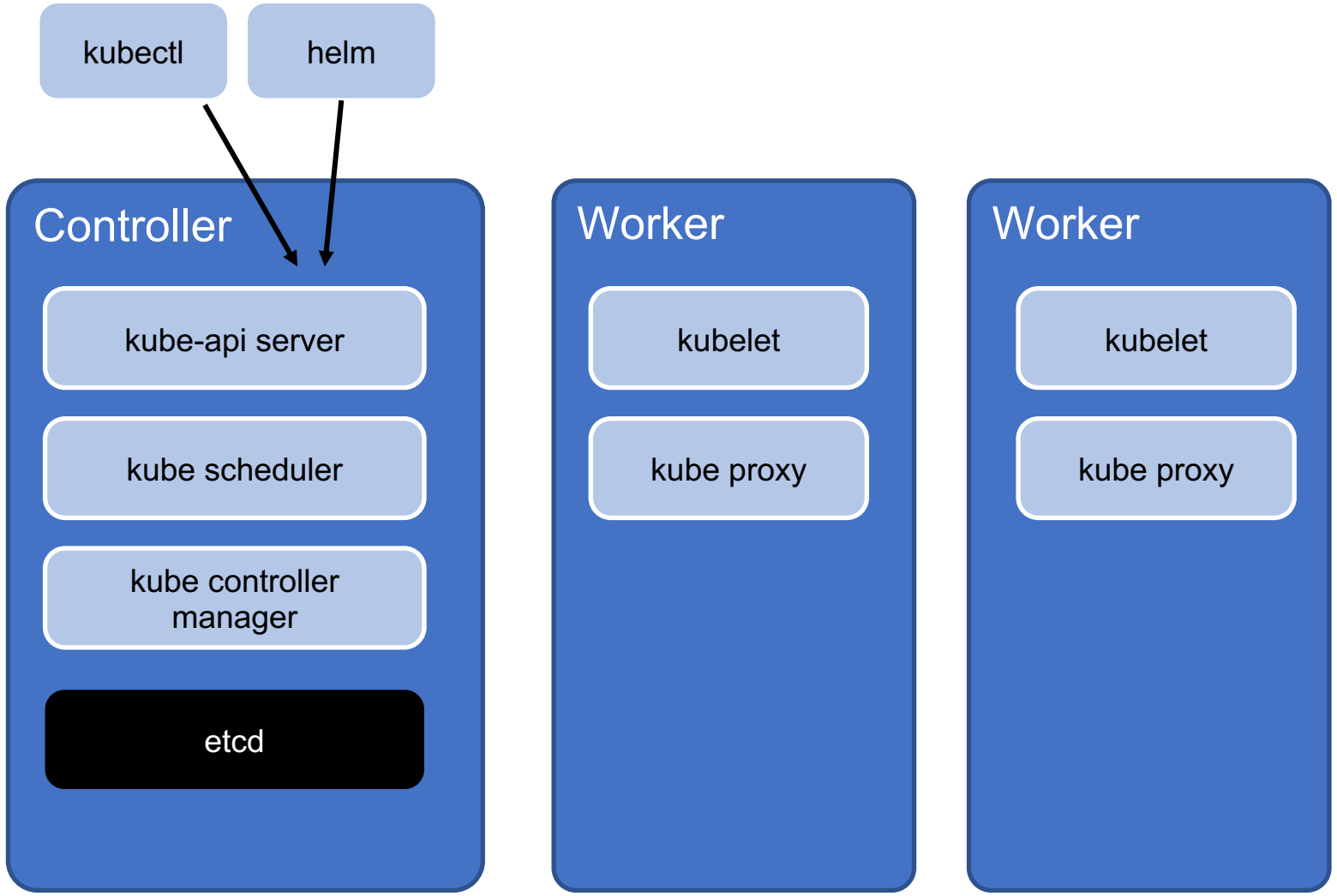
Kubernetes Cluster Fundamentals

- K8s components overview
- Communication between pods in the same node
- Communication between pods in different node



Kubernetes Cluster Fundamentals

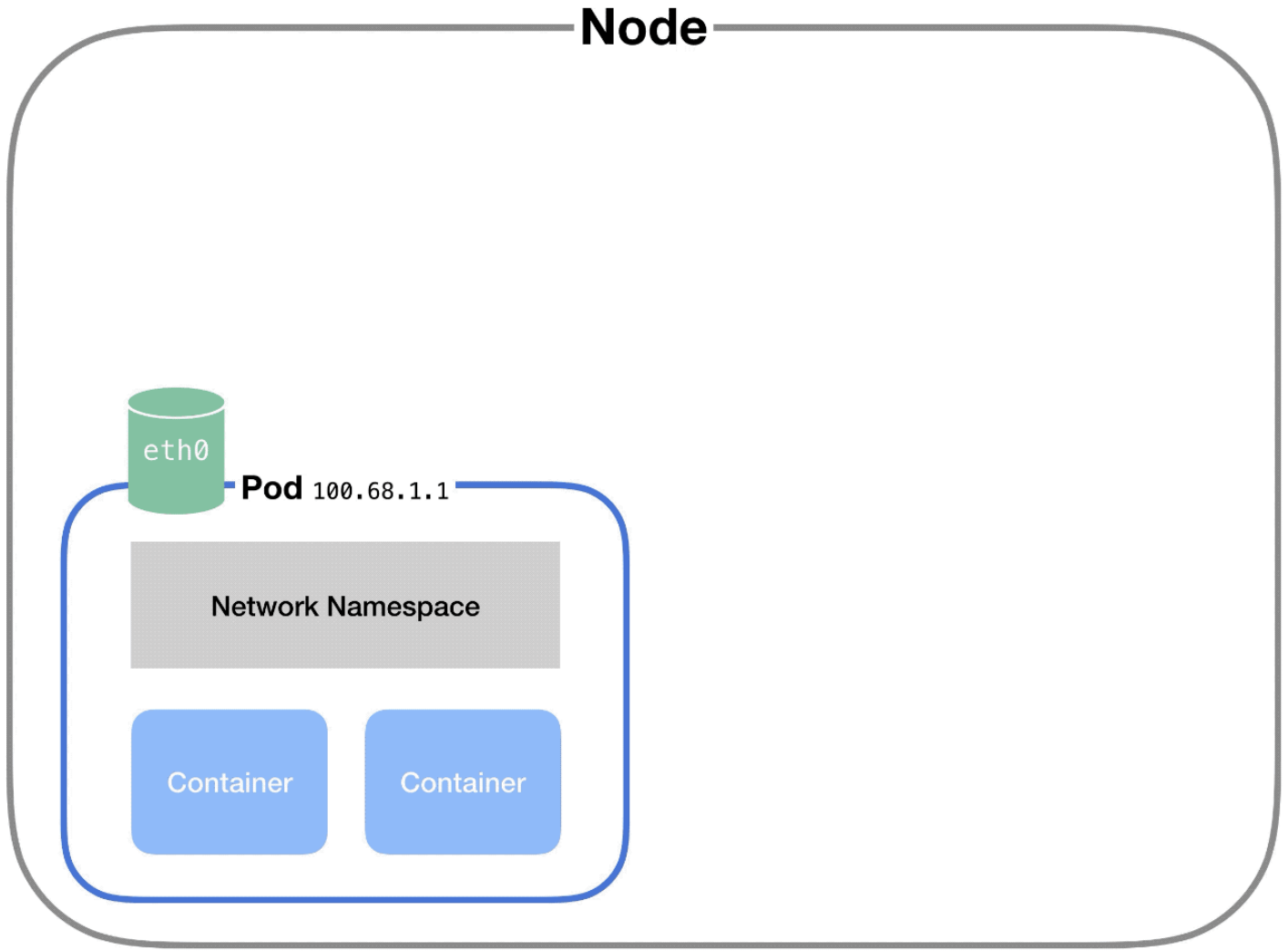
- **kube-api-server**: exposes the Kubernetes API
- **etcd**: highly-available key value store
- **kuber-scheduler**: watches for newly created Pods
- **kube-controller-manager**: service accounts, tokens, etc.
- **kubelet:** makes sure that containers are running in a Pod.
- **kubeproxy**: services, maintains network rules on nodes



K8s Cluster : Communication between pods in the same node

Linux Bridge

- Bridge is called cbr0
- Every pod on a node is part of the bridge
- Bridge connects all pods on the same node together.

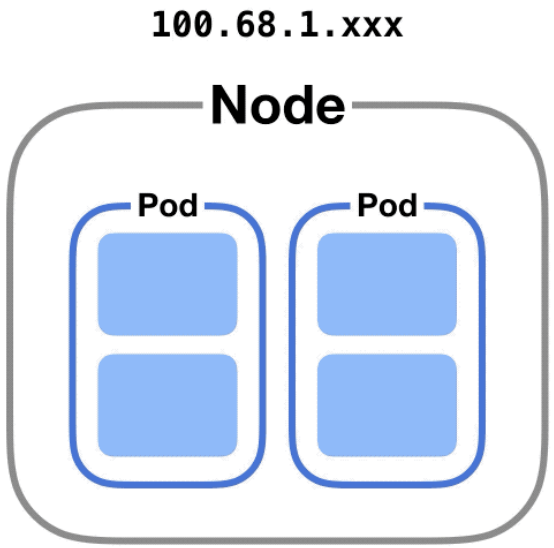


Source: https://matthewpalmer.net/

K8s Cluster: Communication between pods in different nodes

Routing table

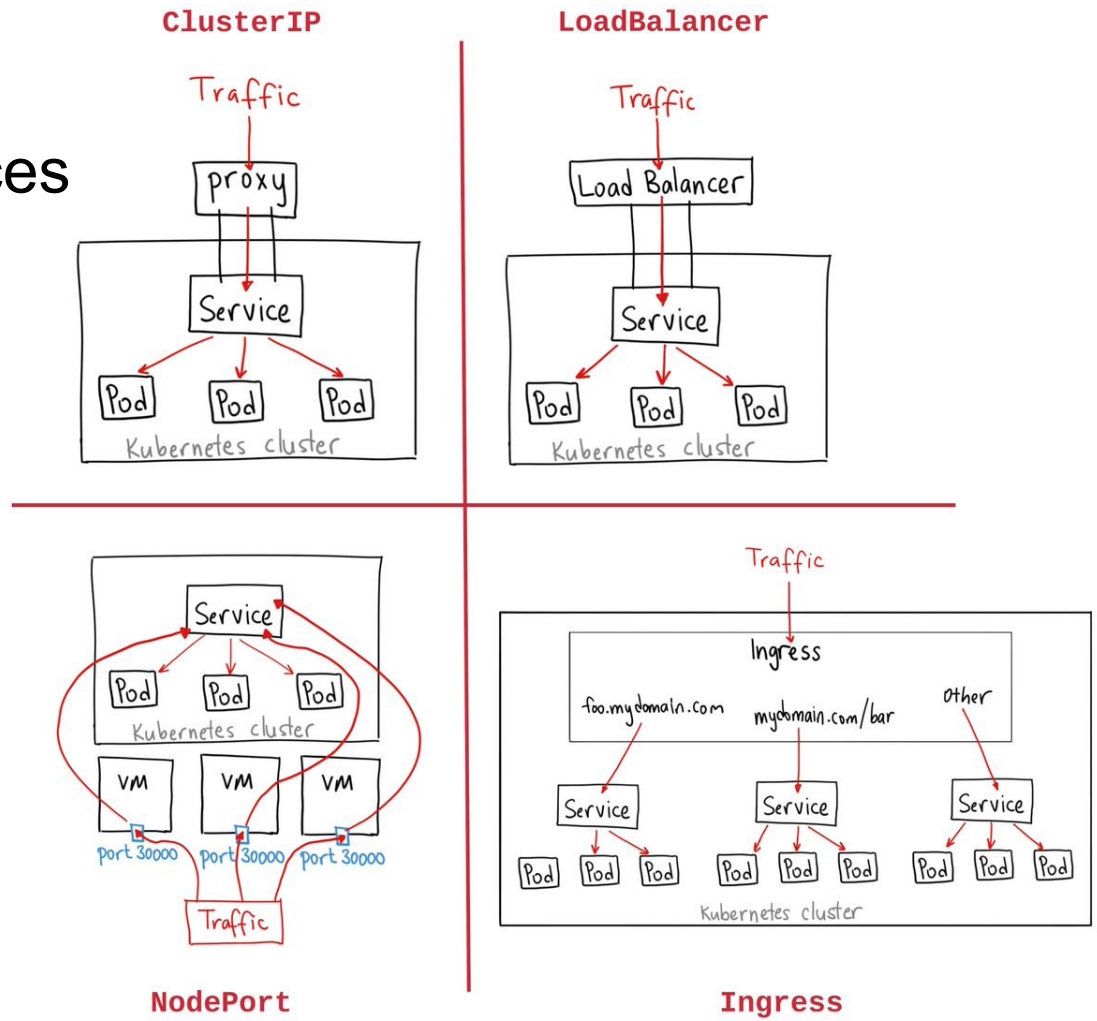
- Can vary based on the **cloud provider** and **networking plugins**
- Goes up to the cluster level and looks for the IP address



Source: https://matthewpalmer.net/

Kubernetes Services

- What are the most common services provided in Kubernetes
  - ClusterIP
  - NodePort
  - LoadBalancer
  - Ingress



K8s Services : What is and why they are needed?

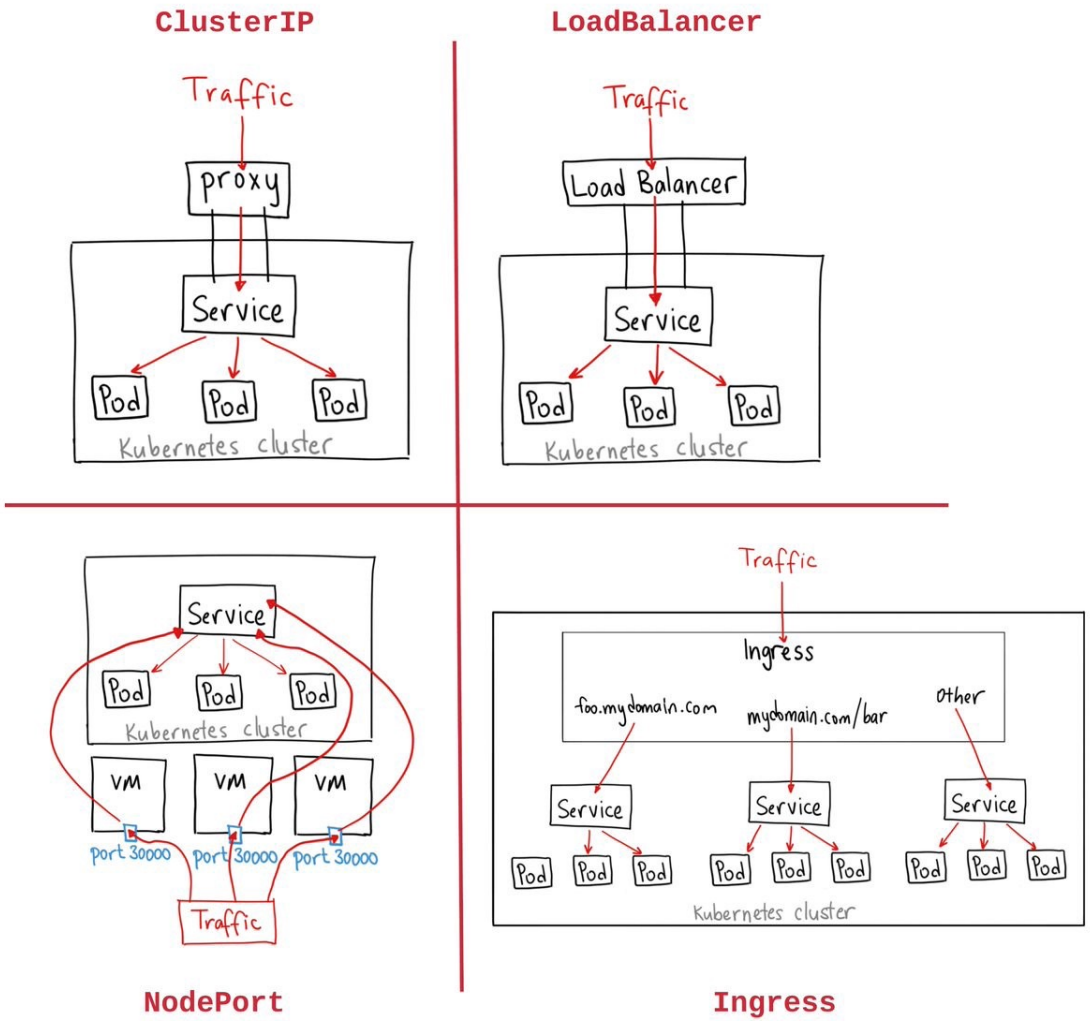
Overview

Each pod has its own IP address

- Pods are ephemeral – are destroy frequently

Services

- Stable IP address
- Load-balancing
- Within and outside cluster

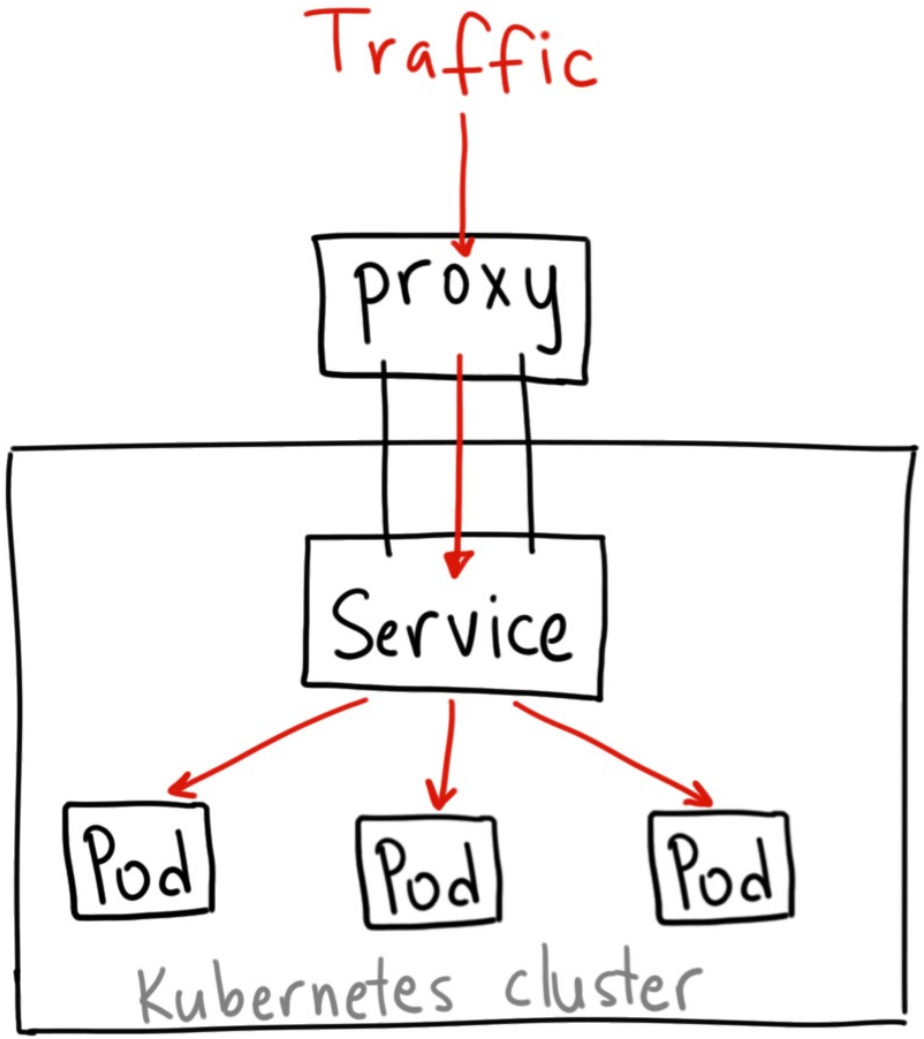


Source: https://medium.com/@kubernetes-advocate

K8s Services : ClusteIP

- Most common and default type
- Internal to the Cluster **only**
- Between backend/frontend pods

```
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: ClusterIP
  selector:
    app: "nginx"
```

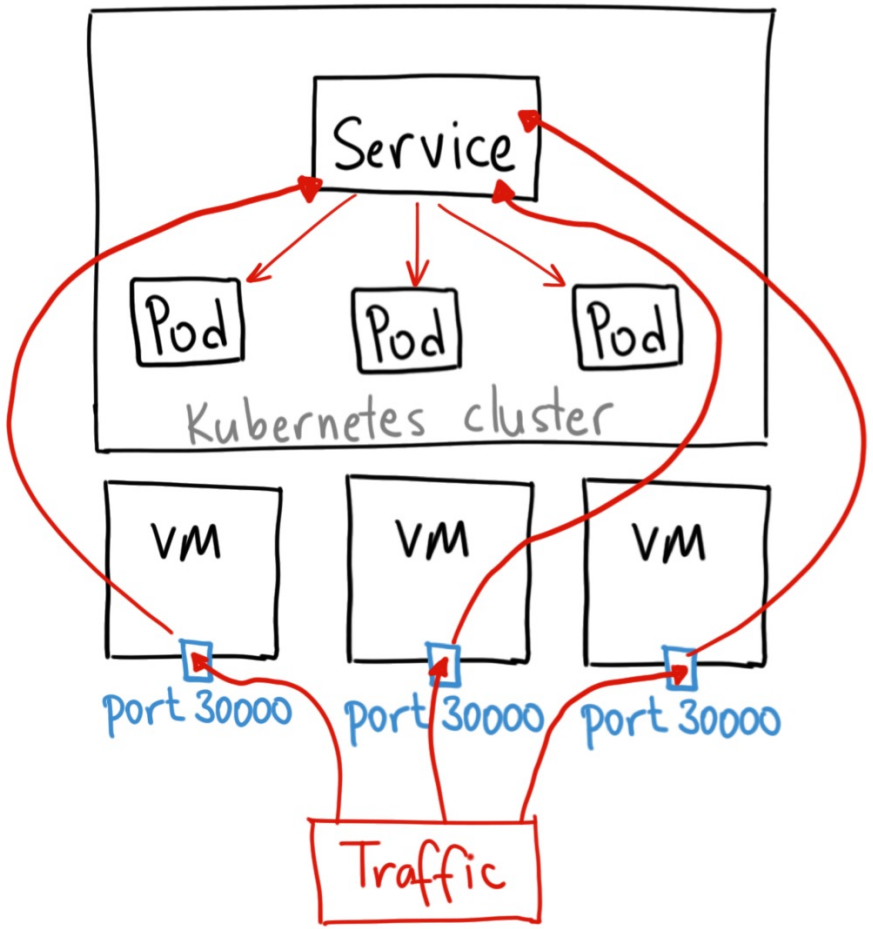


Source: https://medium.com/@kubernetes-advocate

K8s Services : NodePort

- 30000-32767 ports range.

```
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
      nodePort: 30001
  type: NodePort
  selector:
    app: "nginx"
```

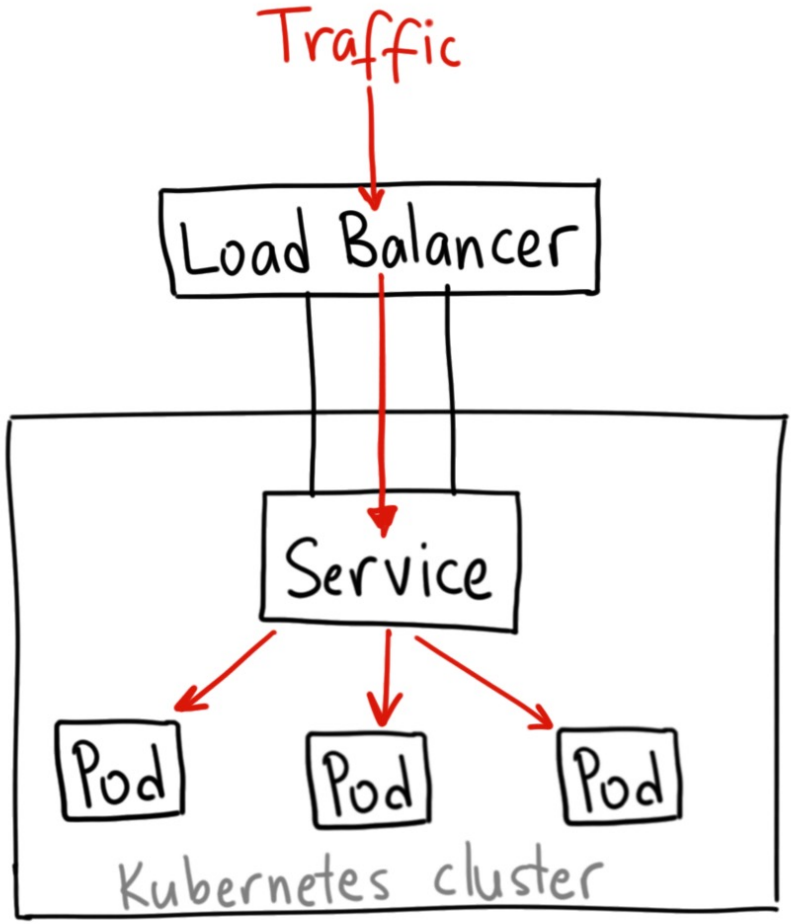


Source: https://medium.com/@kubernetes-advocate

K8s Services : LoadBalancer

- The most used Service type.

```
apiVersion: v1
kind: Service
metadata:
  name: "nginx-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: LoadBalancer
  selector:
    app: "nginx"
```

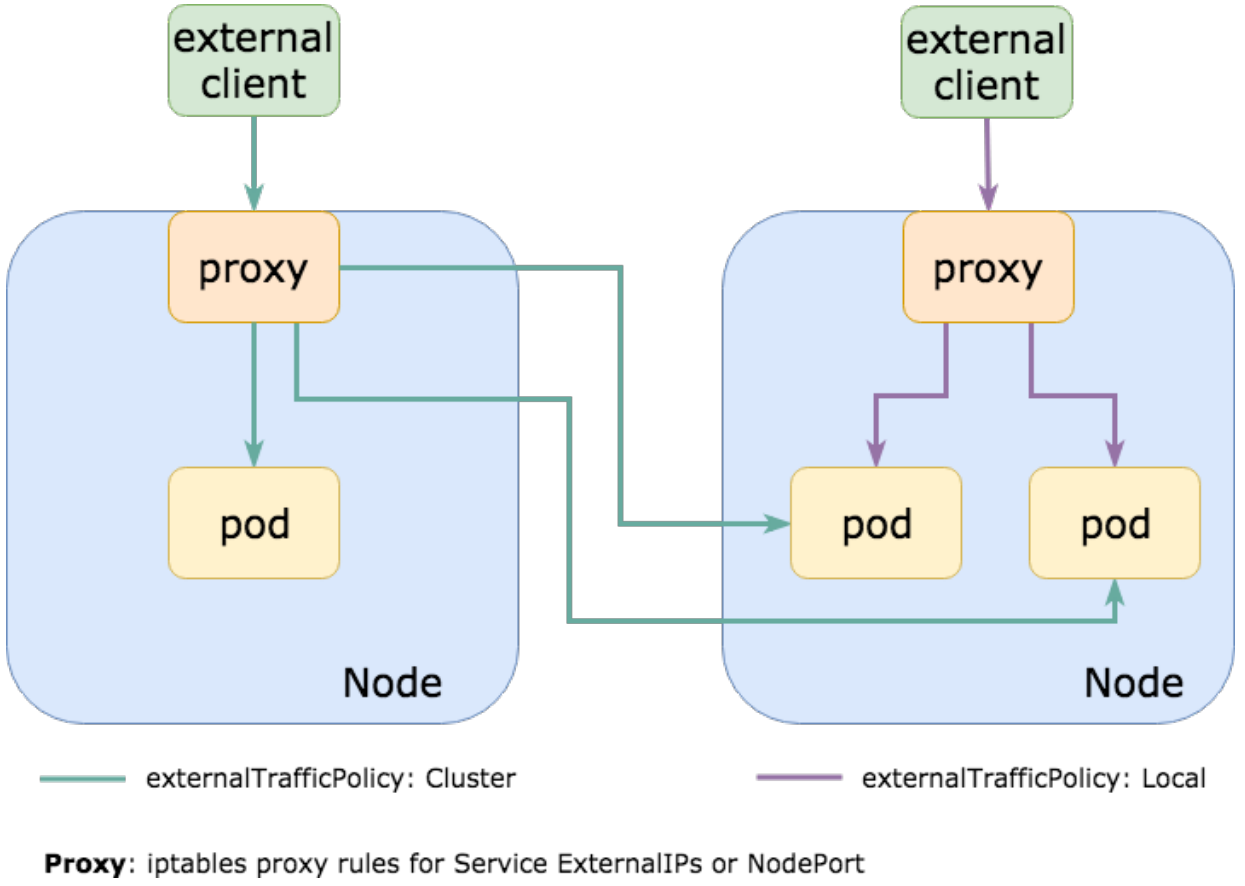


Source: https://medium.com/@kubernetes-advocate

externalTrafficPolicy: Cluster or Local

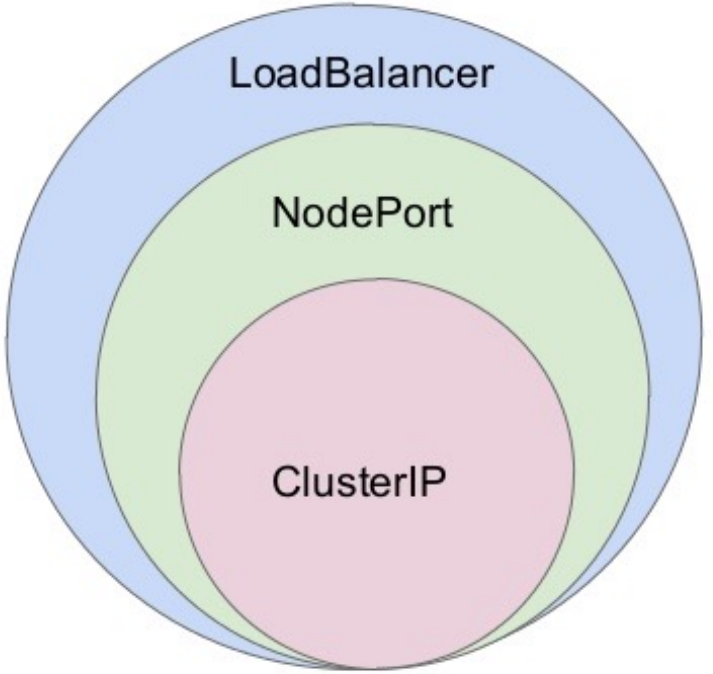
externalTrafficPolicy route external traffic to node-local or cluster-wide endpoints.

- "**Local**" preserves the client source IP and avoids a second hop for LoadBalancer and NodePort type services.
- "**Cluster**" obscures the client source IP and may cause a second hop to another node.



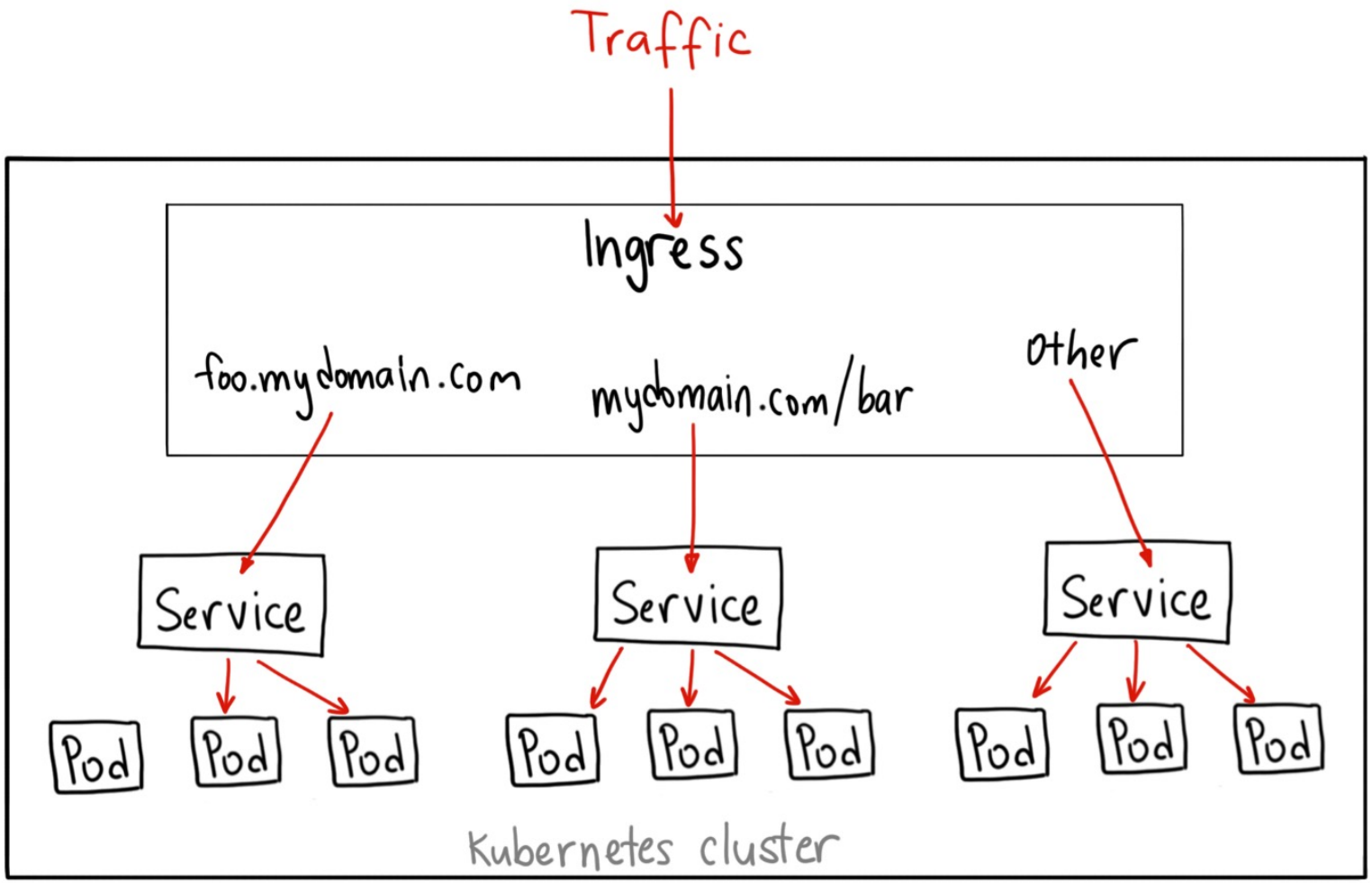
Kubernetes Services: LoadBalancer

```
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
hello-lb     LoadBalancer   10.102.210.54   10.254.254.240   8080:32142/TCP   20d
[root@ctl-a1 ~]# kubectl describe svc hello-lb
Name:                     hello-lb
Namespace:                default
Labels:                   <none>
Annotations:              externalTrafficPolicy: local
Selector:                 app=hello-node
Type:                     LoadBalancer
IP Families:              <none>
IP:                       10.102.210.54
IPs:                      10.102.210.54
LoadBalancer Ingress:     10.254.254.240
Port:                     <unset>  8080/TCP
TargetPort:               8080/TCP
NodePort:                 <unset>  32142/TCP
Endpoints:
172.17.135.129:8080,172.17.135.130:8080,172.17.208.148:8080 + 3 more...
Session Affinity:         None
External Traffic Policy:  Cluster
Events:                   <none>
```



NANOG™

K8s Services : Ingress



Source: https://medium.com/@kubernetes-advocate

K8s Services : Ingress Example

```
apiVersion: v1
kind: Service
metadata:
  name: "nginx-1-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: NodePort
  selector:
    app: "nginx-1"
---
apiVersion: v1
kind: Service
metadata:
  name: "nginx-2-service"
  namespace: "default"
spec:
  ports:
    - port: 80
  type: NodePort
  selector:
    app: "nginx-2"
```

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: "nginx-ingress"
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
  labels:
    app: "nginx"
spec:
  rules:
  - http:
      paths:
      - path: /svc1.html
        backend:
          serviceName: "nginx-1-service"
          servicePort: 80
      - path: /svc2.html
        backend:
          serviceName: "nginx-2-service"
          servicePort: 80
```

Container Network Interface (CNI)

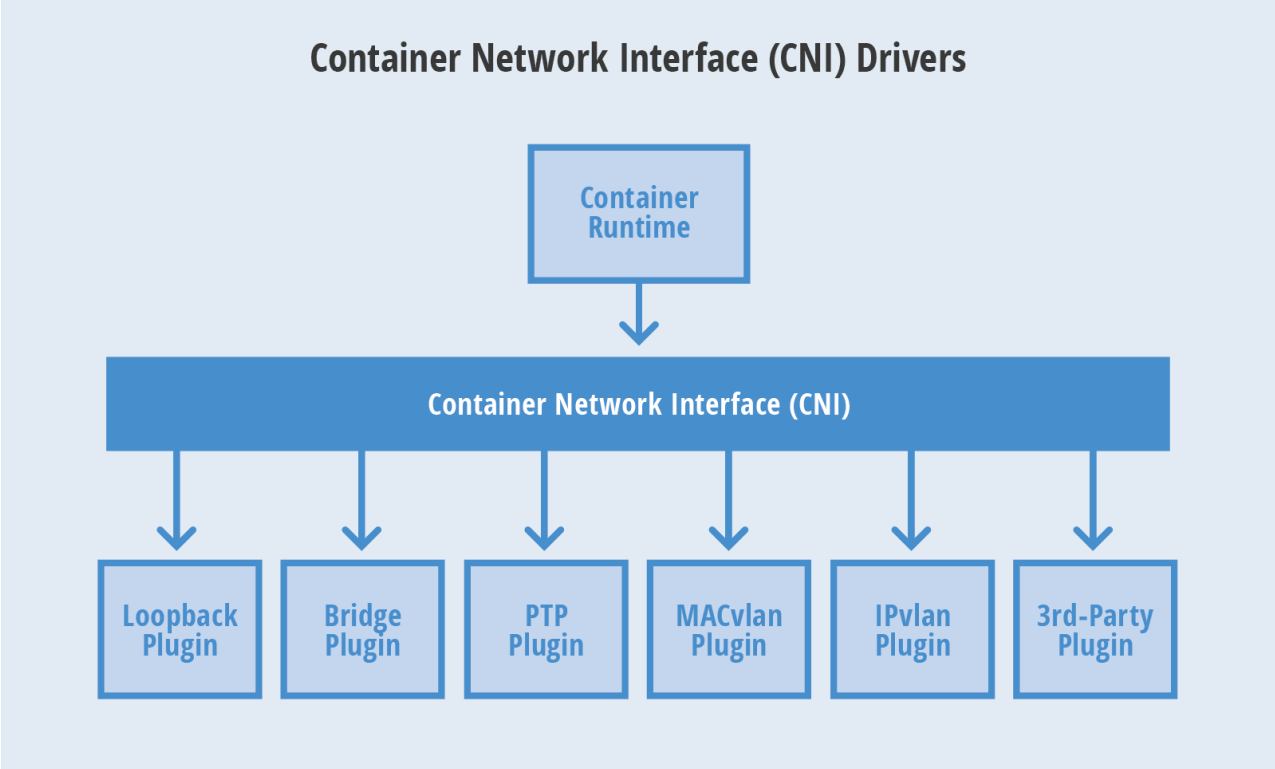
Why do we need this standard?


```
>> bridge add 2a4e9ab45 /var/run//netns/2a4e9ab45
```

Container Network Interface

Why do we need this standard?

- **Container Runtime**
  - Must create **netns**
  - **Identify network** container must attached to
  - Invoke Network Plugin when container is **ADD**ed.
  - Invoke Network Pluggin when container is **DEL**eted
  - **JSON** format in the network configuration

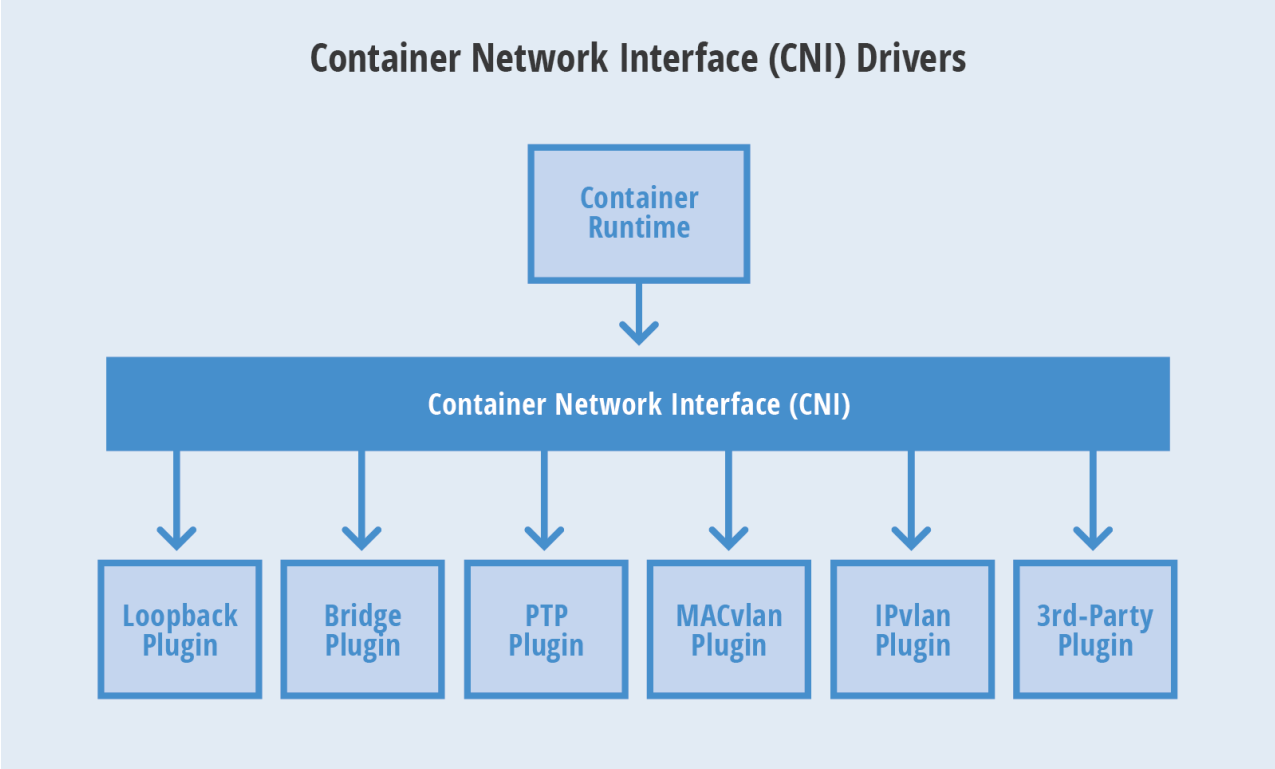


Source: https://saitejabellam.medium.com/

Container Network Interface

Why do we need this standard?

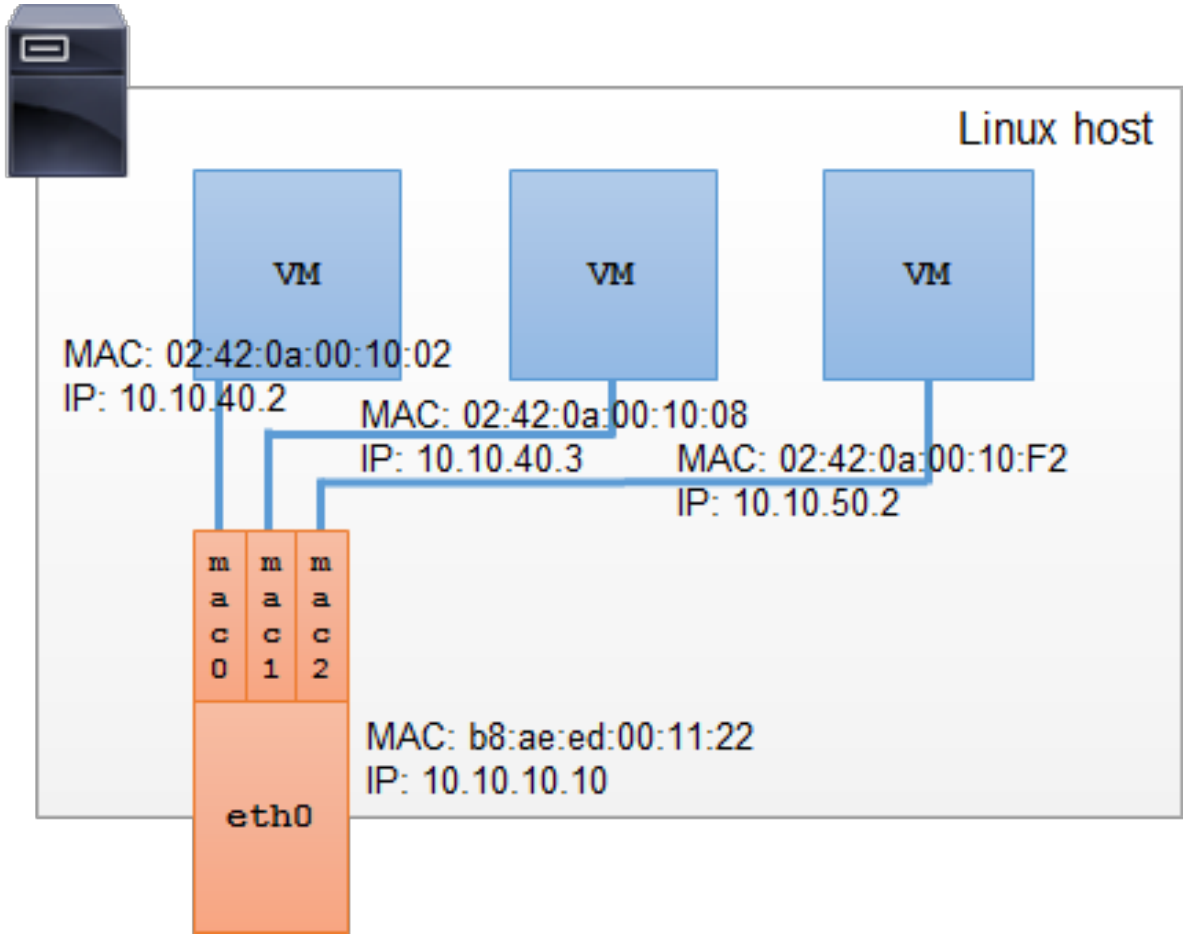
- **Network Plugin**
  - Must support ADD/DEL/CHECK
  - Must support container-id and netns-id
  - Must manage IP assignments to Pods
  - Must return results in a specific format



Source: https://saitejabellam.medium.com/

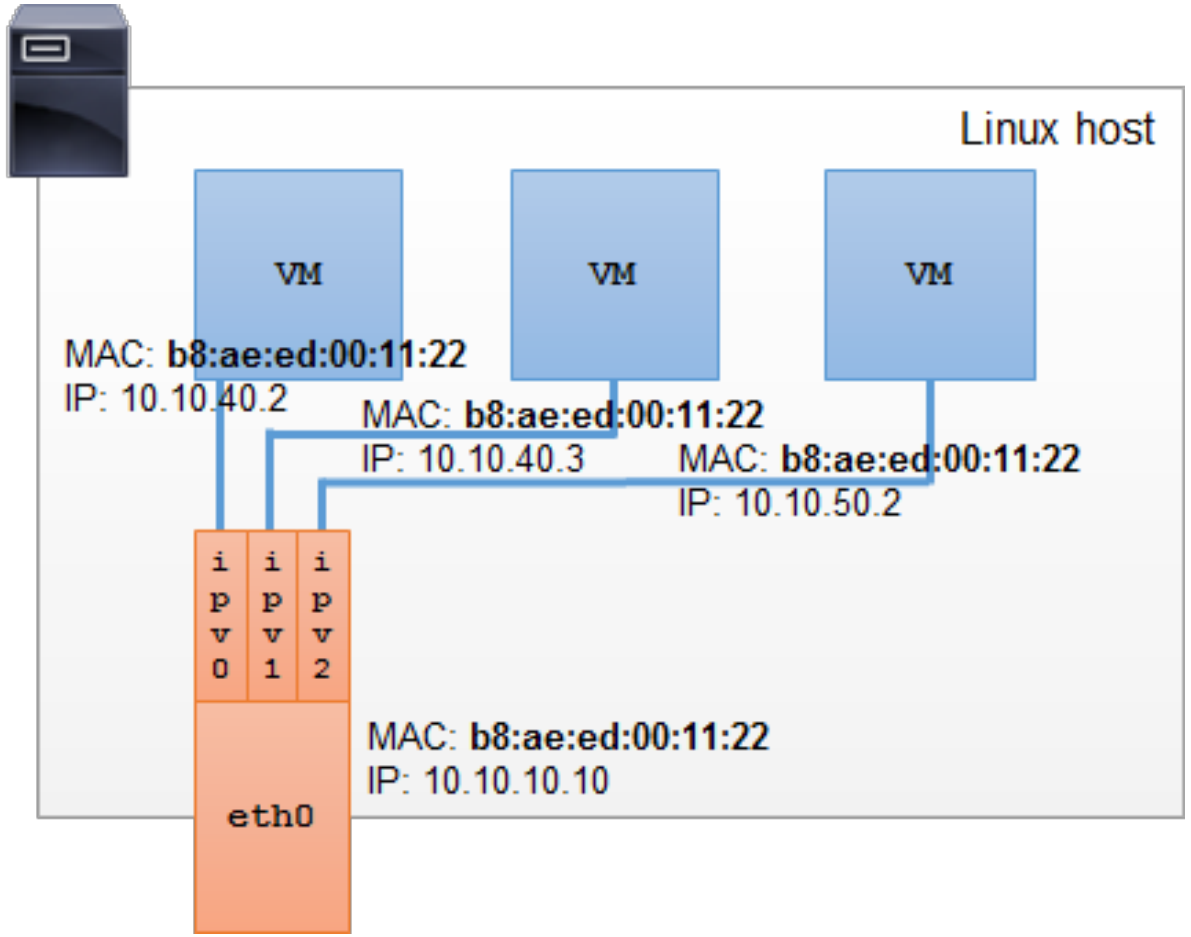
Container Network Interface
MACVLAN

- **Preferred**
- Configure sub-interfaces of a parent, physical Ethernet interface.
- Each with its own unique MAC and IP address.
- It could bind to a specific VLAN in the Underlay Network



Container Network Interface
IPVLAN

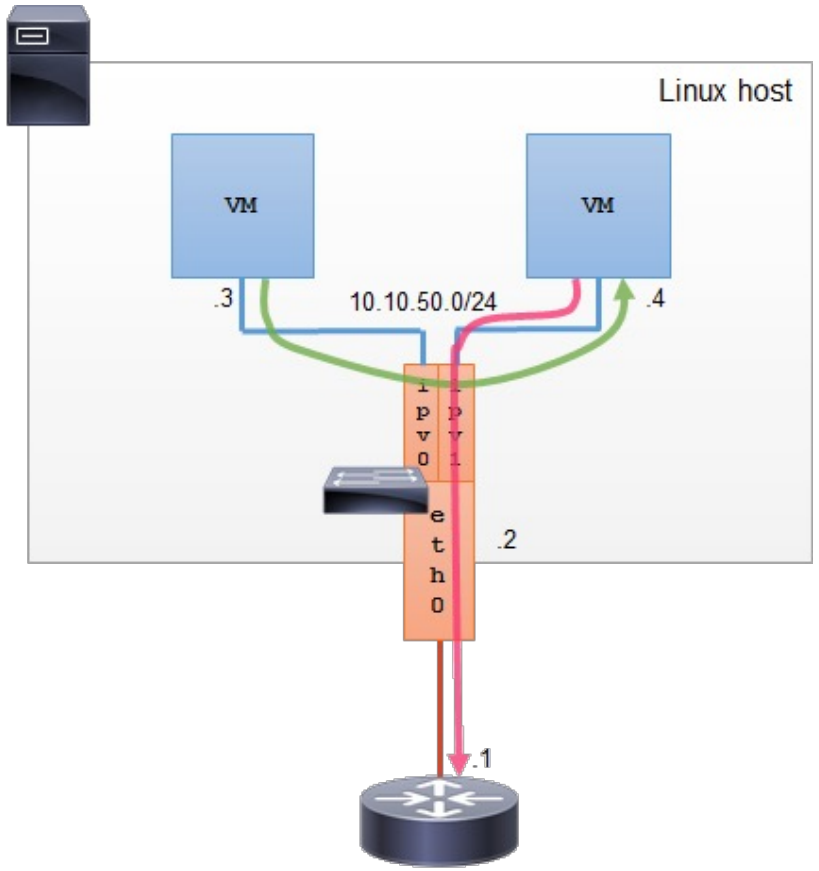
- **Use case: Limited number of MACs.**
- Does not assign unique MAC addresses to created sub-interfaces
- All sub-interfaces share parent's interface MAC address.
- Mode Layer2 or Layer3



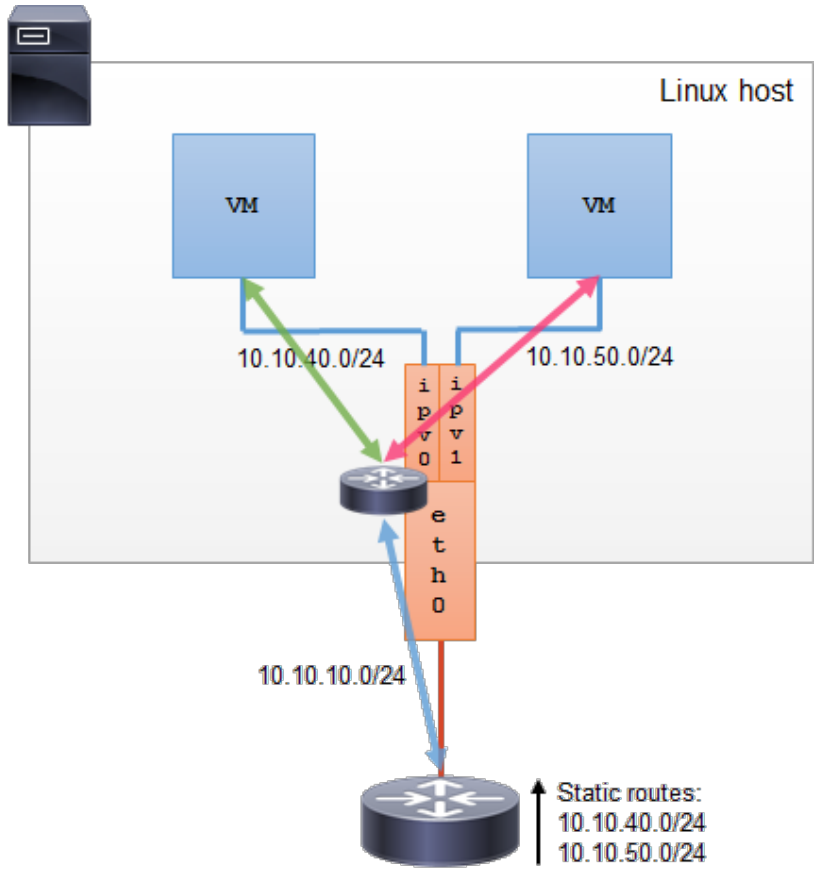
Container Network Interface

IPVLAN

Layer2

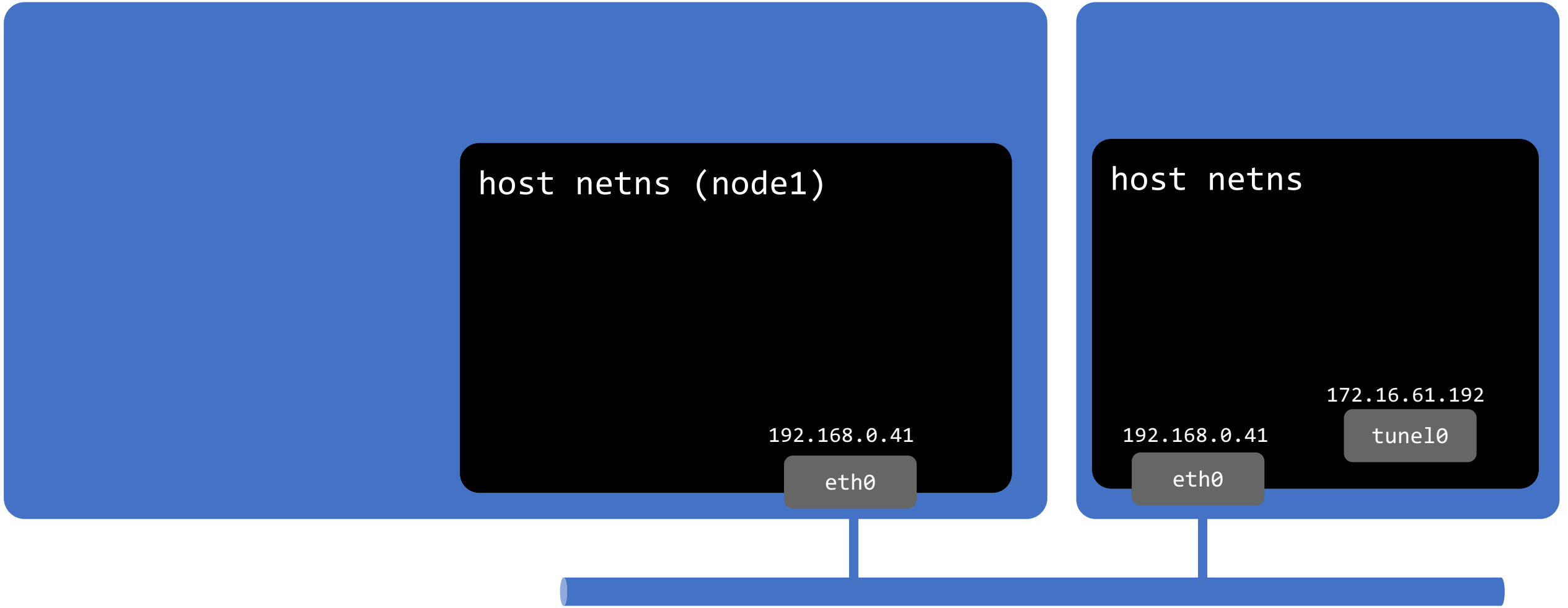


Layer3



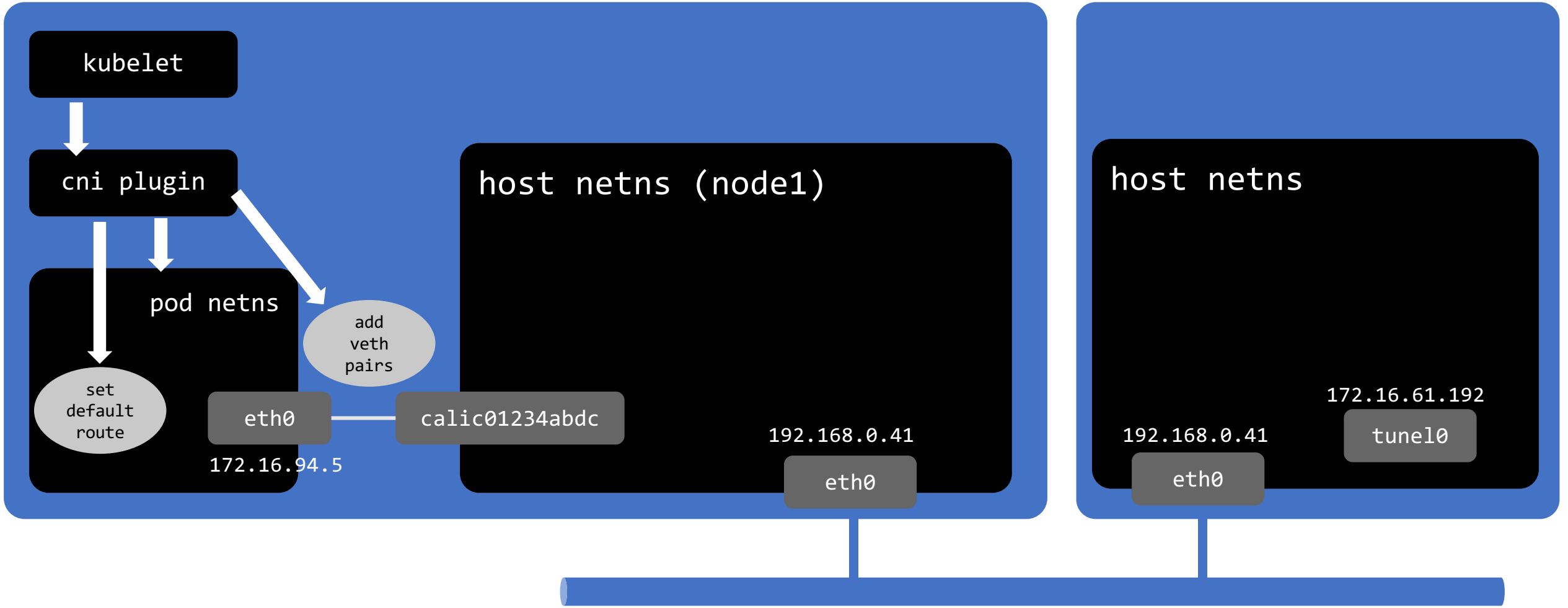
Container Network Interface

Third party: Calico



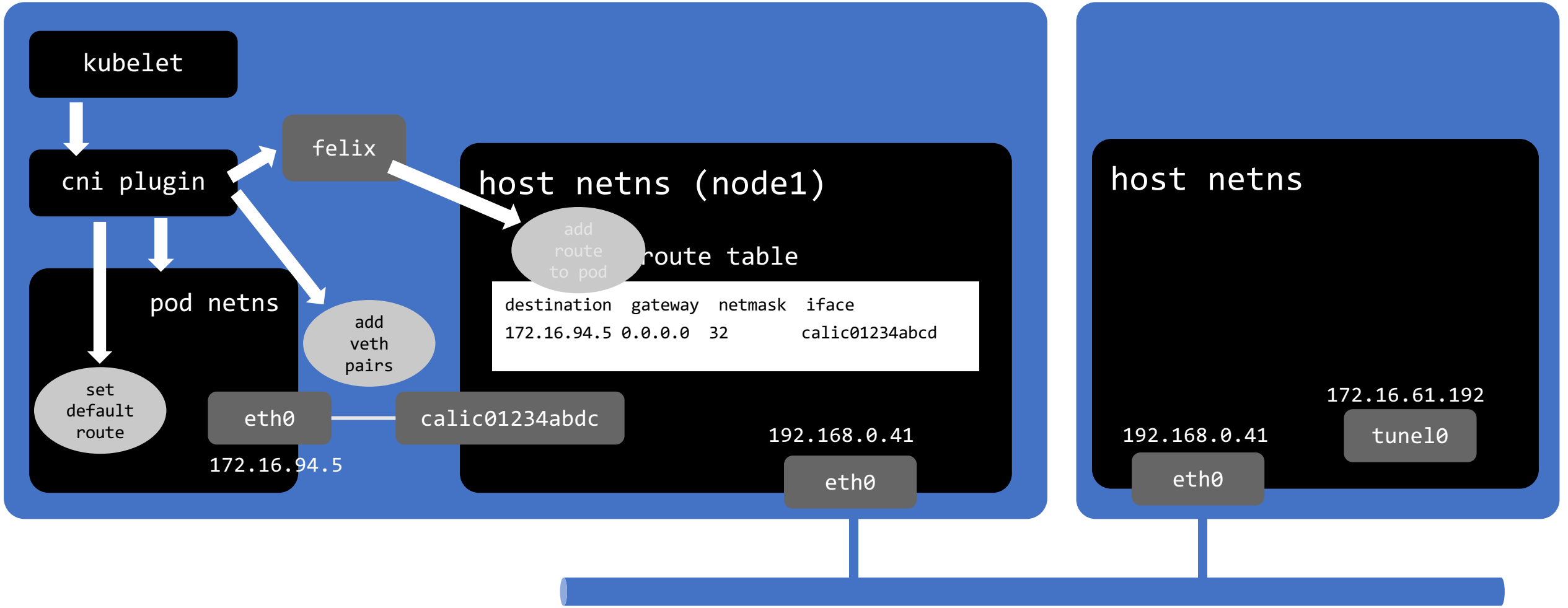
Container Network Interface

Third party: Calico



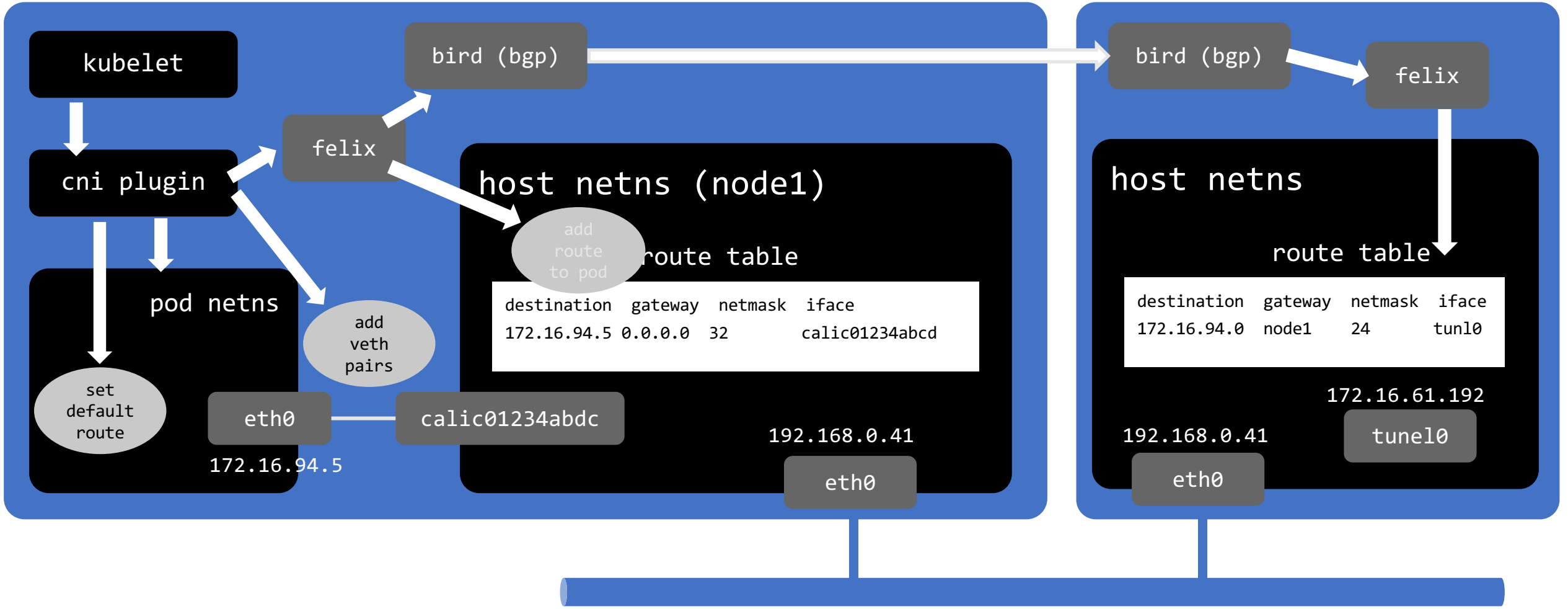
Container Network Interface

Third party: Calico



Container Network Interface

Third party: Calico



Container Network Interface

Third party: Calico

On-prem network options

- Underlay-Network: Better performance
  - BGP peer to TOR
  - Single L2 non-routable
    - IP Pods a no routable outside the cluster
    - BGP peering only between nodes
- Overlay-Networks
  - IP-in-IP
  - VxLAN

```
{
  "name": "k8s-pod-network",
  "cniVersion": "0.3.1",
  "plugins": [
    {
      "type": "calico",
      "log_level": "info",
      "log_file_path": "/var/log/calico/cni/cni.log",
      "datastore_type": "kubernetes",
      "nodename": "ctl-a1",
      "mtu": 0,
      "ipam": {
          "type": "calico-ipam"
      },
      "policy": {
          "type": "k8s"
      },
      "kubernetes": {
          "kubeconfig": "/etc/cni/net.d/calico-kubeconfig"
      }
  ]
}...
```

Benchmark results of Kubernetes network plugins (CNI) over 10Gbit/s network (Updated: August 2020)

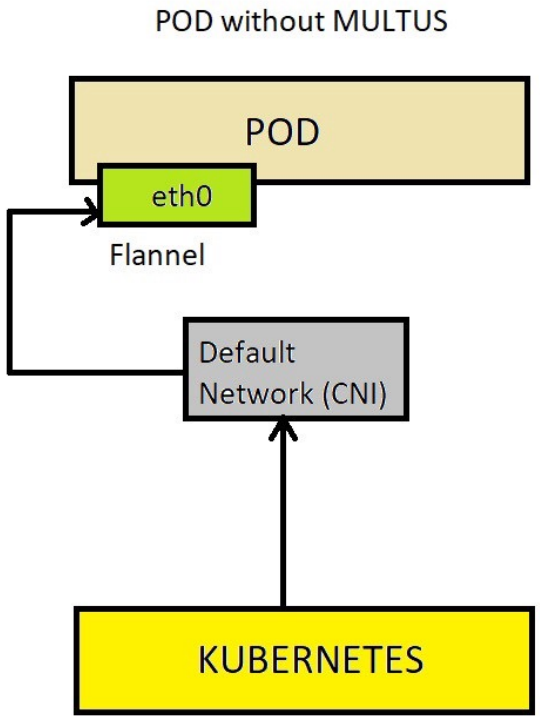
CNI Benchmark August 2020 infraBuilder	Config	Performances (bandwidth)				Resources consumption (cpu/ram)					Security features			
	MTU	Pod to Pod		Pod to Service		Idle	Pod to Pod		Pod to Service		Network Policies		Encryption	
	setting	TCP	UDP	TCP	UDP	none	TCP	UDP	TCP	UDP	in	out	activation	Performance
Antrea	auto	Very fast	Very fast	Very fast	Slow	Low	Low	Low	Low	Low	yes	yes	at deploy time	Slow
Calico	manual	Very fast	Very fast	Very fast	Fast	Low	Very low	Very low	Very low	Very low	yes	yes	anytime	Very fast
Canal	manual	Very fast	Very fast	Very fast	Very fast	Low	Very low	Very low	Very low	Very low	yes	yes	no	n/a
Cilium	auto	Fast	Very fast	Very fast	Very fast	High	High	High	High	High	yes	yes	at deploy time	Slow
Flannel	auto	Very fast	Very fast	Very fast	Very fast	Very low	Very low	Very low	Very low	Very low	no	no	no	n/a
Kube-OVN	auto	Fast	Very slow	Fast	Very slow	High	High	High	High	High	yes	yes	no	n/a
Kube-router	none	Slow	Very slow	Slow	Very slow	Low	Very low	Low	Very low	Low	yes	yes	no	n/a
Weave Net	manual	Very fast	Very fast	Very fast	Fast	Very low	Low	Low	Low	Low	yes	yes	at deploy time	Slow

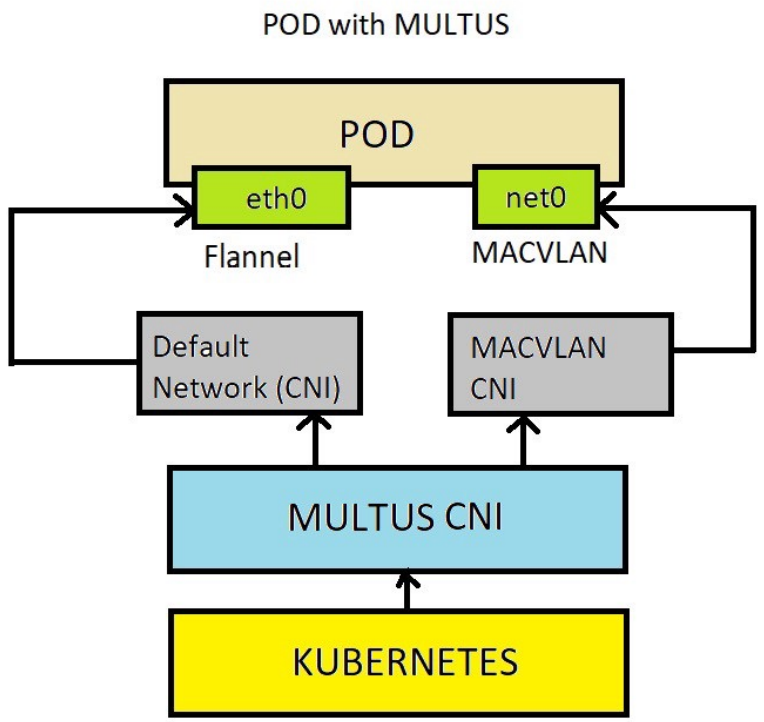
https://medium.com/@infrabuilder

Container Network Interface

Third party: Multus

- Attaching multiple network interfaces to pods
- Acting as a "meta-plugin”





Container Network Interface

Third party: Multus

- Attaching multiple network interfaces to pods
- Acting as a "meta-plugin"

```
{
  "cniVersion": "0.3.1",
  "name": "node-cni-network",
  "type": "multus",
  "kubeconfig": "/etc/cni/net.d/multus.d/multus.kubeconfig”,
  "delegates": [
    {
      "name": "k8s-pod-network",
      "cniVersion": "0.3.1",
      "plugins": [
         { "type": "calico",
           "log_level": "info",
           "log_file_path": "/var/log/calico/cni/cni.log",
           "datastore_type": "kubernetes",
           "nodename": "k8s-node01",
           "mtu": 1440,
           "ipam": {
             "type": "calico-ipam”
...
```

Container Network Interface

Third party: Multus

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: macvlan-conf-172-254-1
spec:
  config: '{
      "cniVersion": "0.3.0",
      "type": "macvlan",
      "master": "eth1",
      "mode": "bridge",
      "ipam": {
        "type": "host-local",
        "subnet": "172.254.1.0/24",
        "rangeStart": "172.254.1.200",
        "rangeEnd": "172.254.1.216"
      }
    }'
```

```
## multi-int-test-pod.yaml file
apiVersion: v1
kind: Pod
metadata:
  name: multi-int-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: macvlan-conf-172-254-1
spec:
  containers:
  - name: multi-int-pod
    command: [
      "/bin/bash",
      "-c",
      "trap : TERM INT; sleep infinity & wait”
    ]
    image: alpine
```

Container Network Interface

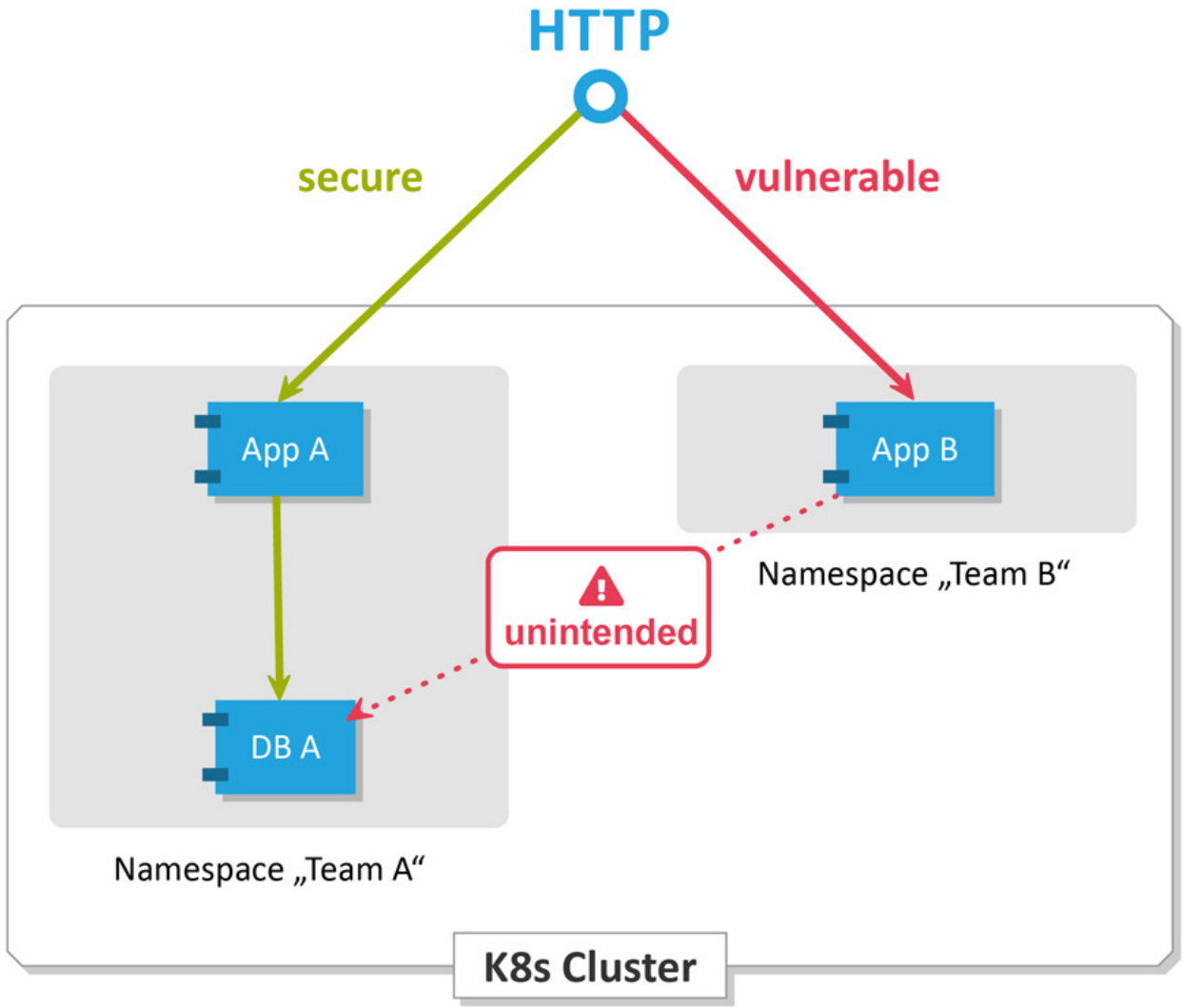
Third party: Multus

kubectl describe pods (network annotations)

```
Annotations:  cni.projectcalico.org/podIP: 10.140.58.205/32
              cni.projectcalico.org/podIPs: 10.140.58.205/32
              k8s.v1.cni.cncf.io/network-status:
                [{
                    "name": "",
                    "ips": [
                        "10.140.58.205"
                    ],
                    "default": true,
                    "dns": {}
                },{
                    "name": "default/macvlan-conf-172-254-1",
                    "interface": "net1",
                    "ips": [
                        "172.254.1.200"
                    ],
                    "mac": "22:74:5e:0d:35:8b",
                    "dns": {}
                }]
```

Container Network Interface
Network Policies

- By default, pods are non-isolated; they accept traffic from any source
- Pods become isolated by having a NetworkPolicy that **selects** them
- Network policies are implemented by the network plugin
- combination of the following 3 identifiers
  - Other **pods** that are allowed
  - **Namespaces** that are allowed
  - **IP blocks** (traffic to and from the node where a Pod is running is always allowed)



Container Network Interface

Network Policies

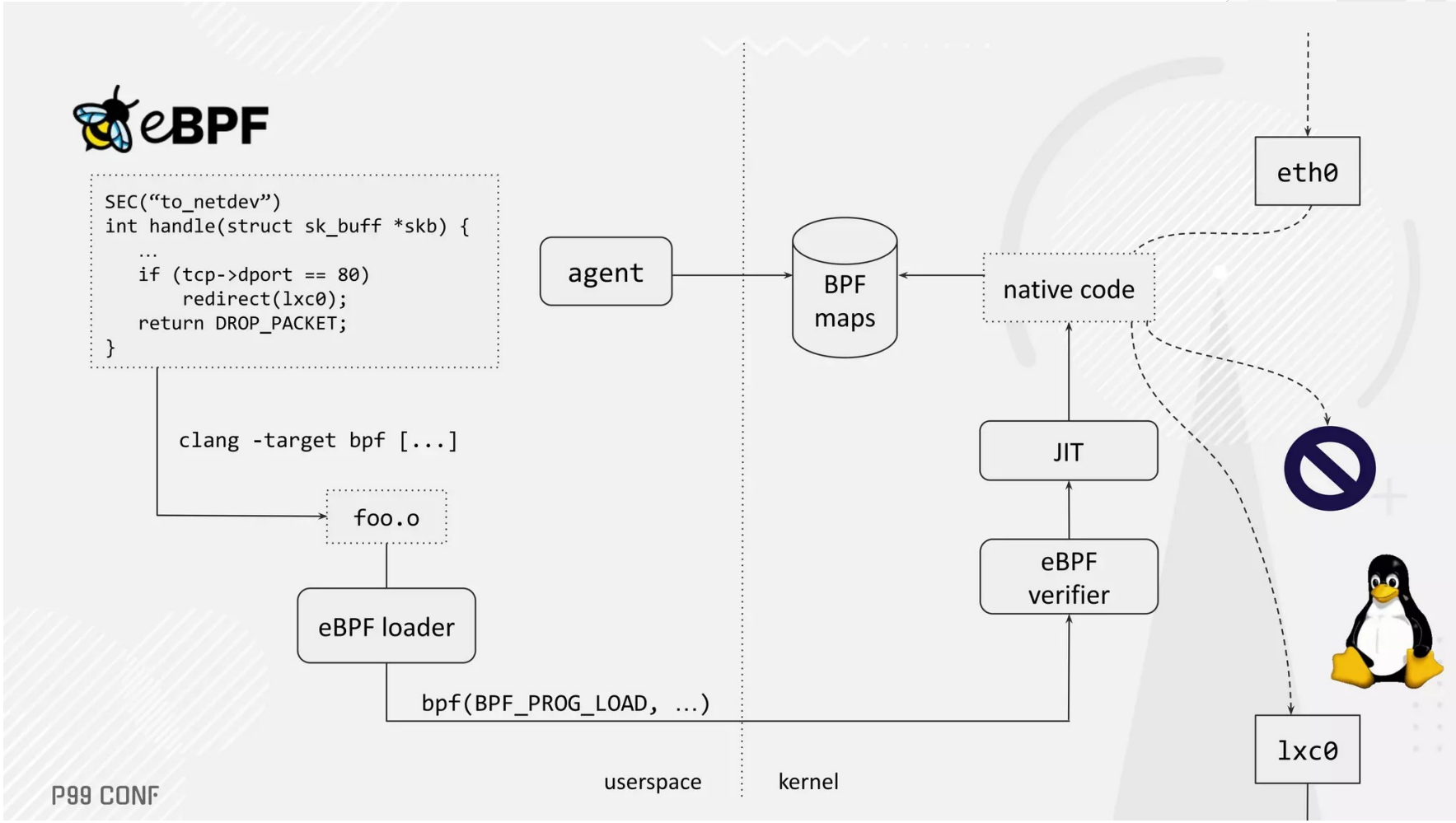
```
apiVersion:
networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  - Egress
```

```
ingress:
  - from:
    - ipBlock:
        cidr: 172.17.0.0/16
        except:
        - 172.17.1.0/24
    - namespaceSelector:
        matchLabels:
          project: myproject
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

```
egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 30000
      endPort: 32768
```

What's happening now with Kubernetes?

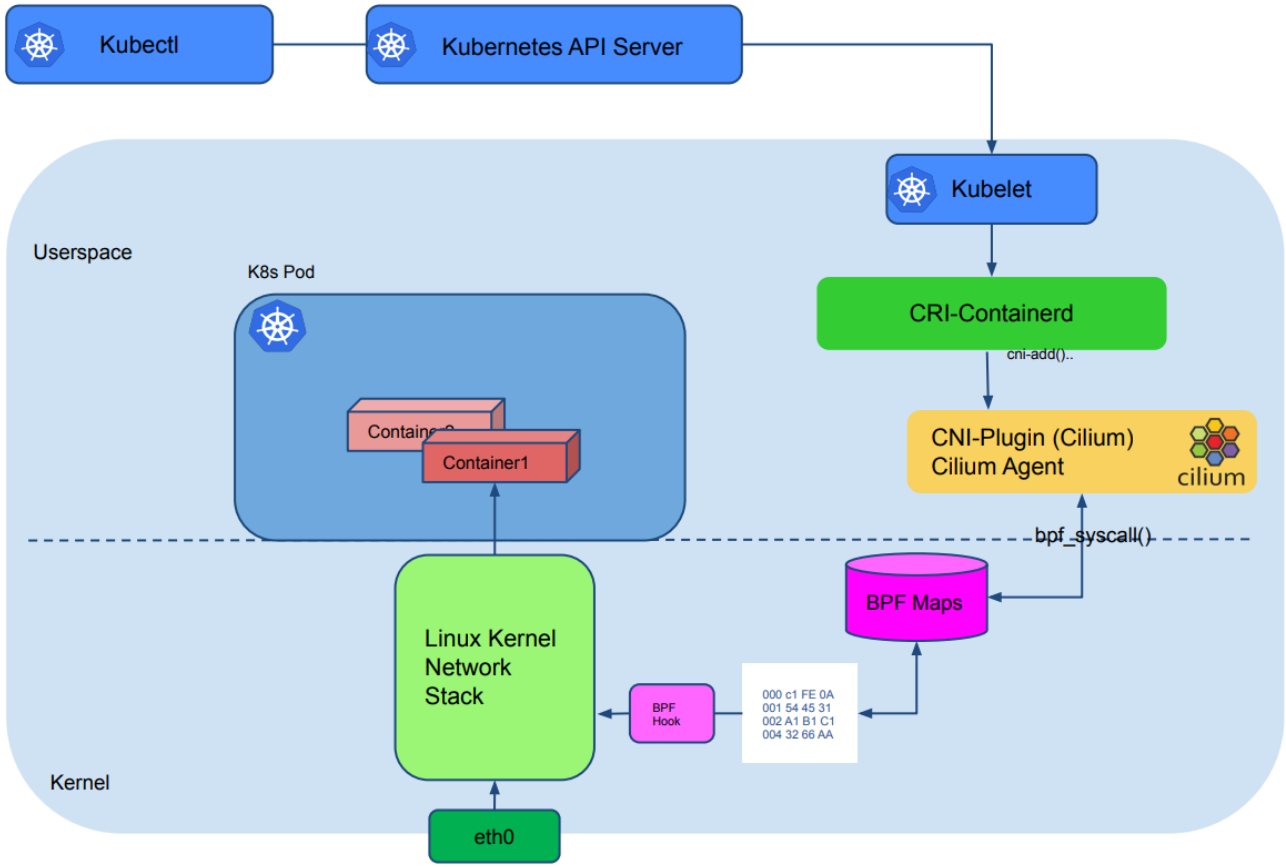
eBPF



source: @lizrice

extended Berkeley Packet Filter (eBPF)

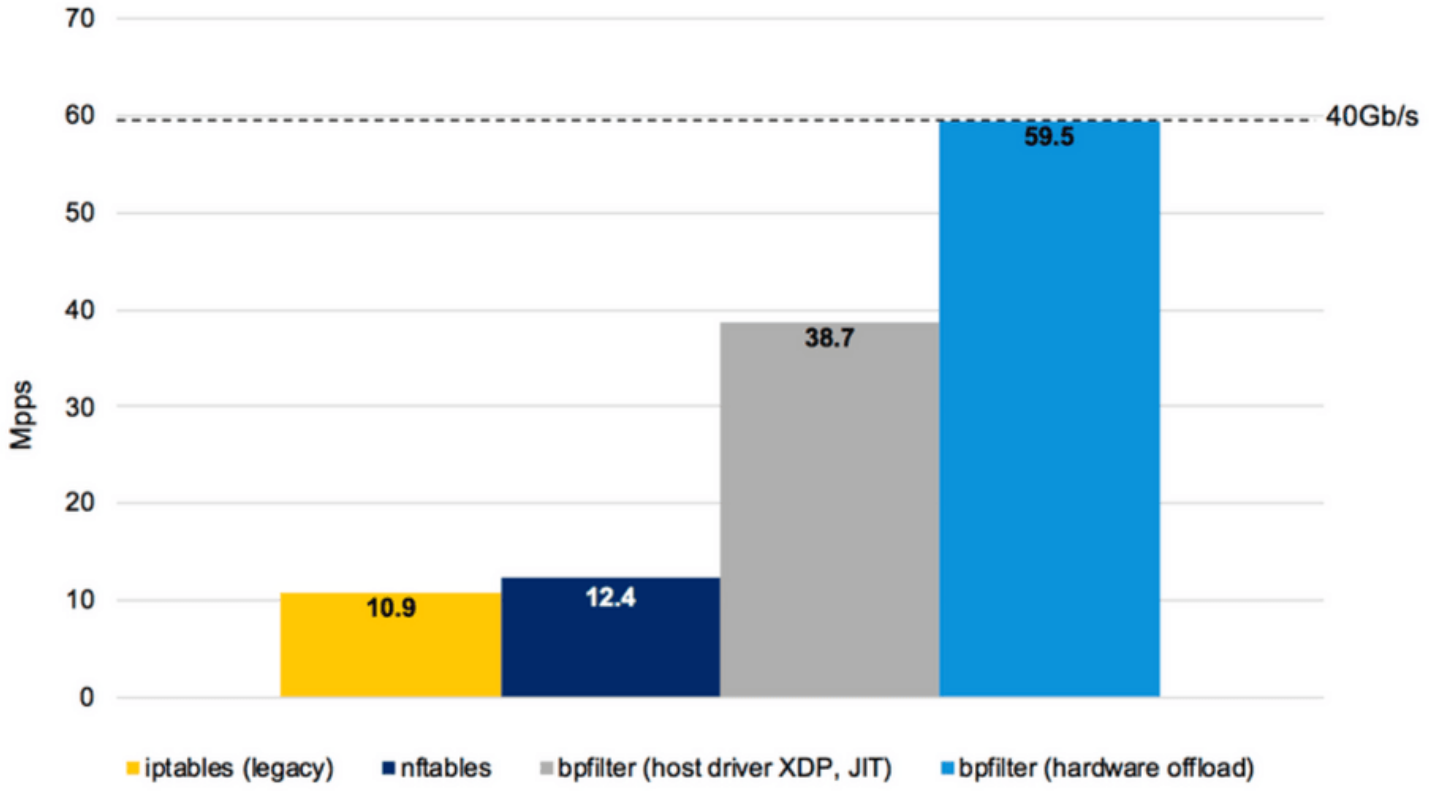
- Cilium uses eBPF heavily in its data path
- It can completely replace kube-proxy



source: https://cilium.io/blog

bpfilter vs iptables

If you need to alter the packet flow in the kernel, bpfilter would be the better choice



source: https://cilium.io/blog

Kubernetes CRDs

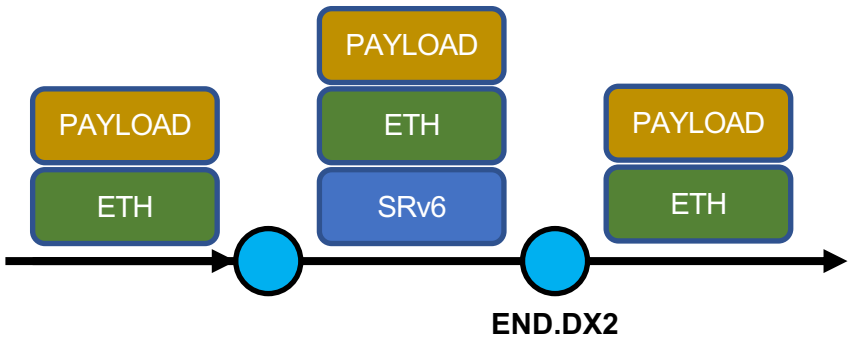
- Custom resources are extensions of the Kubernetes API
- Making Kubernetes more modular
- Use for Network Automation/Orchestration purposes

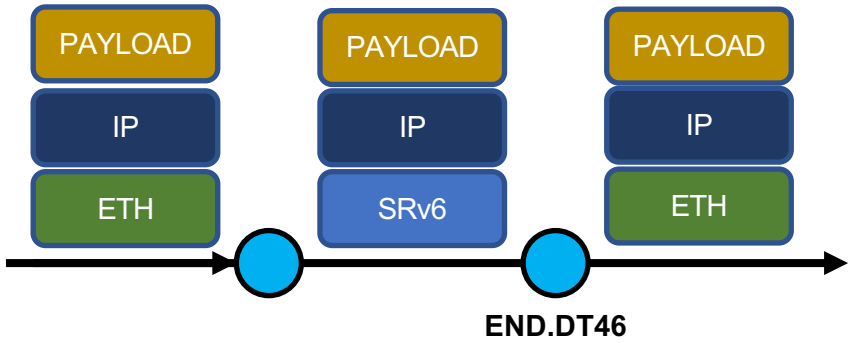
```
apiVersion: nwi.enc.nokia.com/v1alpha2
kind: SROSConfig
metadata:
  name: vpls-01
  annotations:
    nwi.enc.nokia.com/depends_on: |
      [{
        "apiVersion": "nwi.enc.nokia.com/v1alpha2",
        "kind": "SROSConfig",
        "name": "customer-1234",
        "namespace": "enc-nwi-system"
      }]
spec:
  switchID: 172.20.20.200
  path: /service/vpls[service-name=vpls-01]
  properties:
    admin-state: enable
    customer: "1234"
    service-id: 4321
    service-name: vpls-01
```

Kubernetes and SRv6

SRV6 Policies

- Layer2 Cross-Connect (END.DX2) / Layer3 VPNs (END.DT46), Service Chains
- Ingress Traffic is steered into policies based in various criteria.



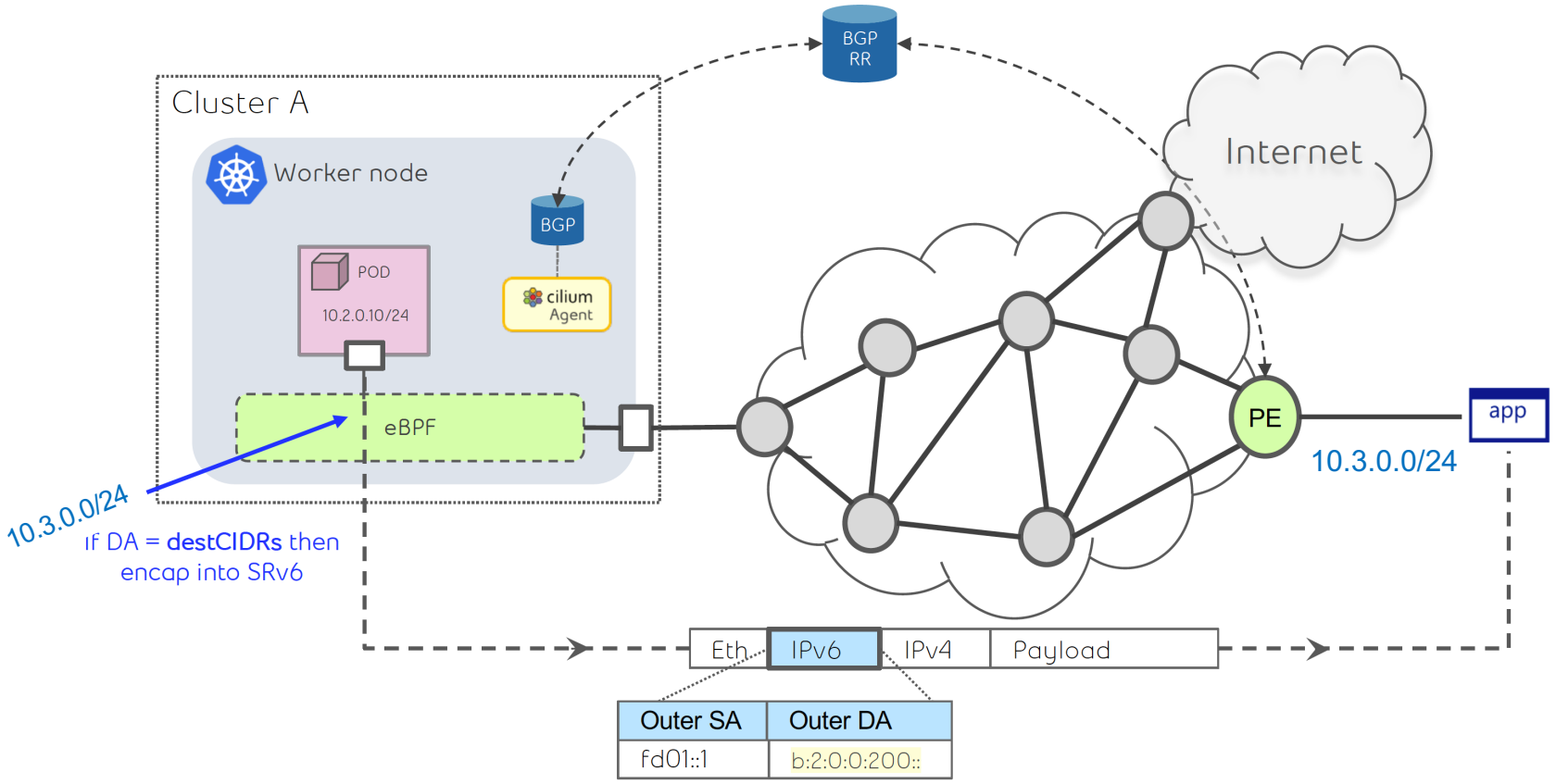


Source: @danbernier01

Kubernetes and SRv6

Cilium SRv6 – K8s-cluster to PE for L3VPN

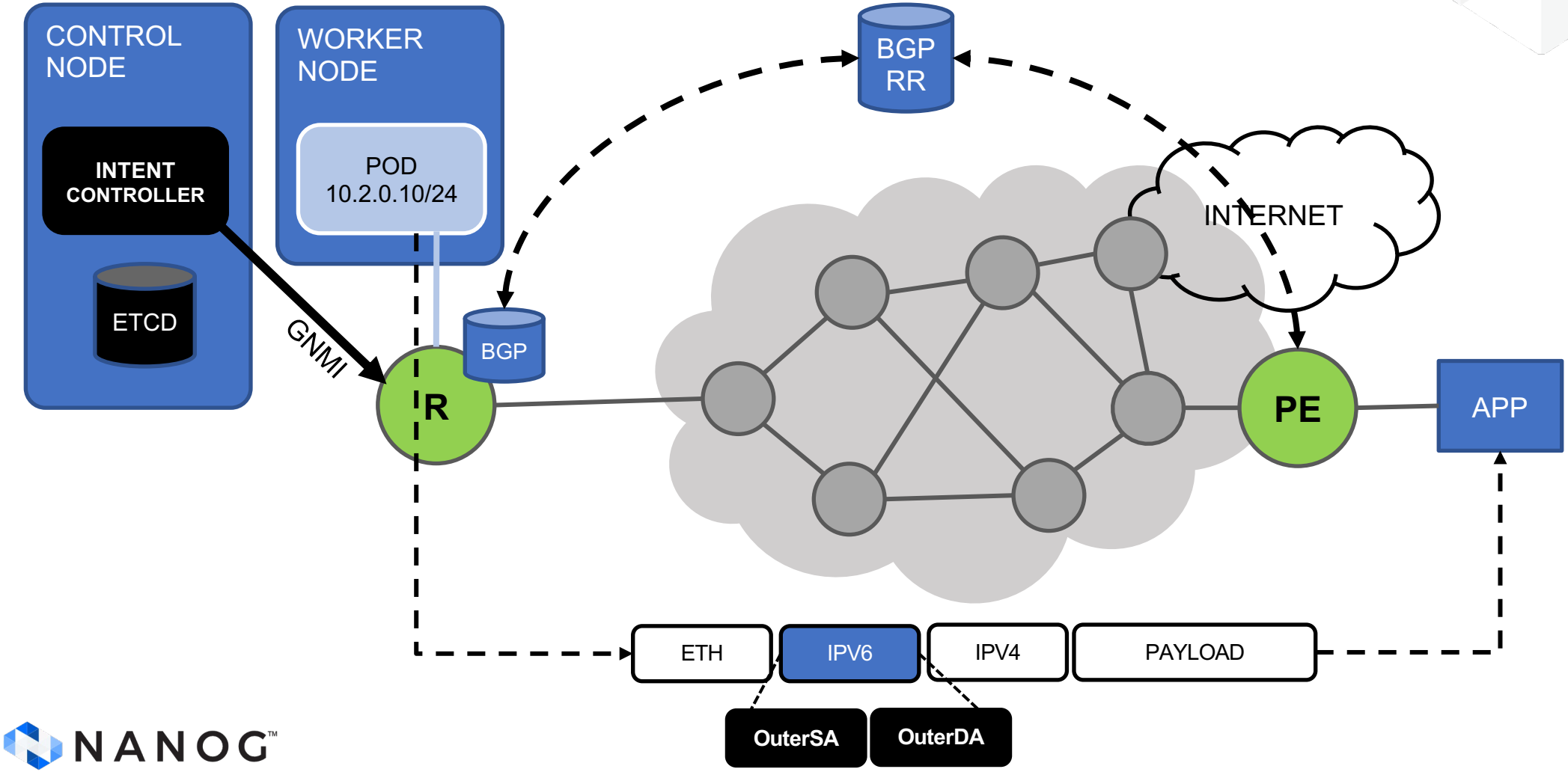
- POD has a single interface with only a default route
- POD is associated with one or more VRFs if required by owner
- CRD defines VRF attributes (RT, name, etc.)
- Once BGP received within VRF, Cilium builds associated dynamic ciliumSrv6EgressPolicy



Source: @danbernier01

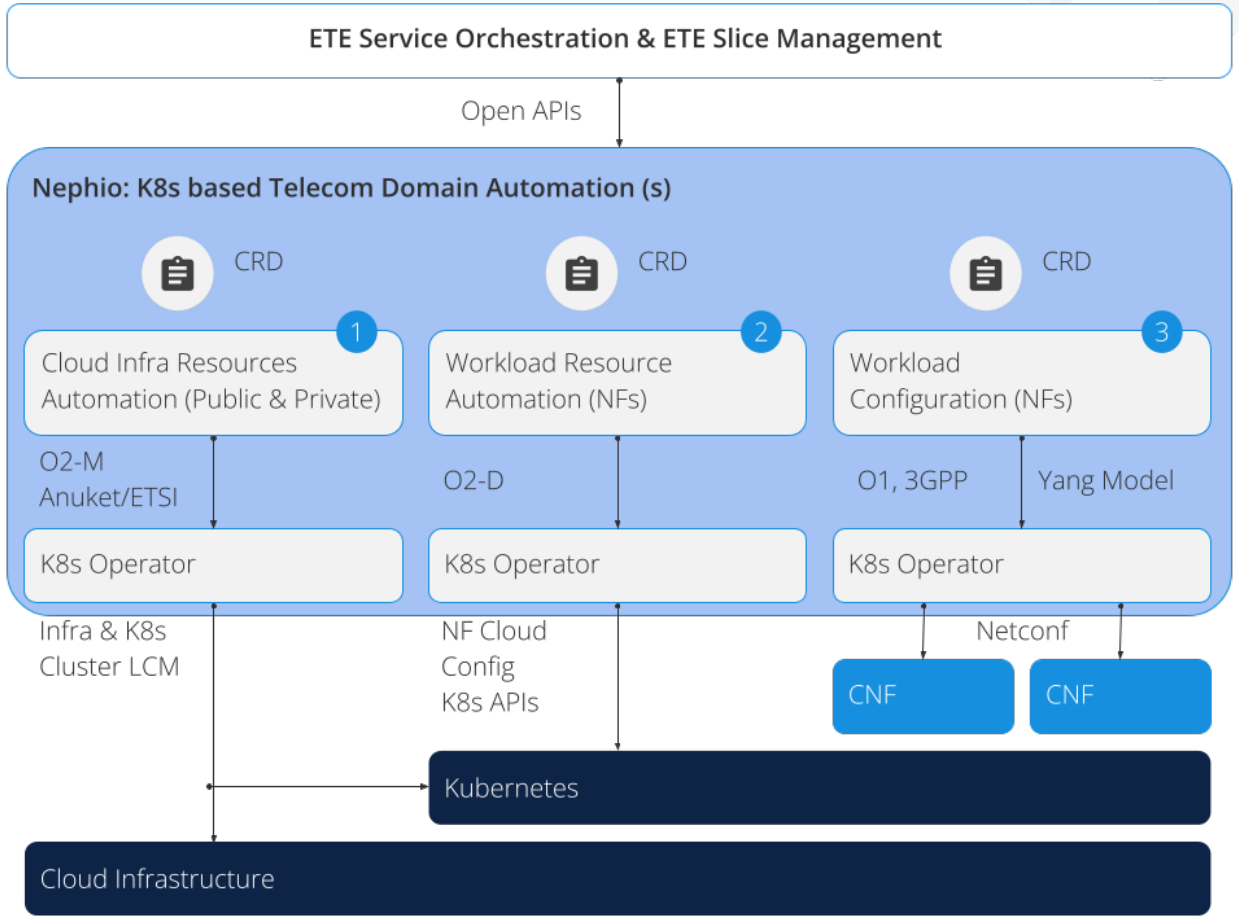
Other Kubernetes CRDs approach

Edge Controller and SRv6 – K8s-cluster to PE for L3VPN



More from Kubernetes CRDs

- Linux Foundation Project
- Kubernetes-based intent-driven automation



Thanks!

bio.site/pinrojas

NANOG™