Networking Model and Services

In Kubernetes

Arkan from Pahamify

Jakarta Kubernetes Meetup
11 Mei 2019

Intro

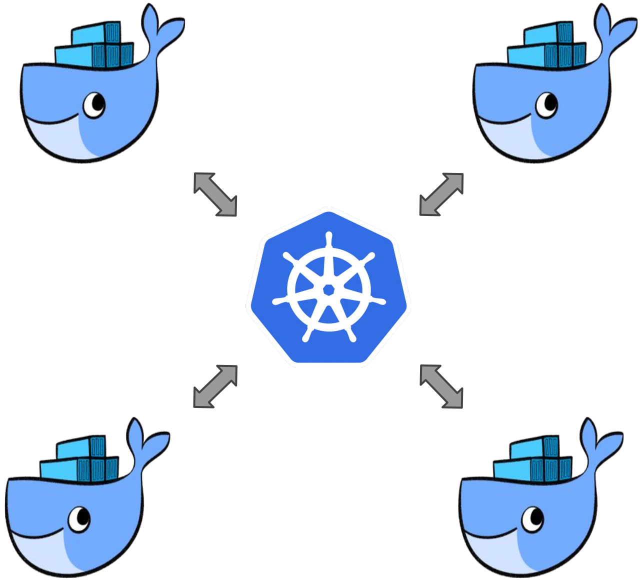
- ➔ Early Stage Startup(<= 1 year development). 4 month MVP. 2 engineer
- ➔ Startup School Awardee, YCombinator
- ➔ 99% Container Tech, Microservice, Orchestrated and Built on top of k8s

pahamify

TEMUKAN DI Google Play

Overview of K8S Model

Center of K8s: Networking



- ➔ using cluster concept
- ➔ manage communication and regulation
  - ◆ between apps/container inside cluster
  - ◆ between cluster and external traffic

Objective Menu

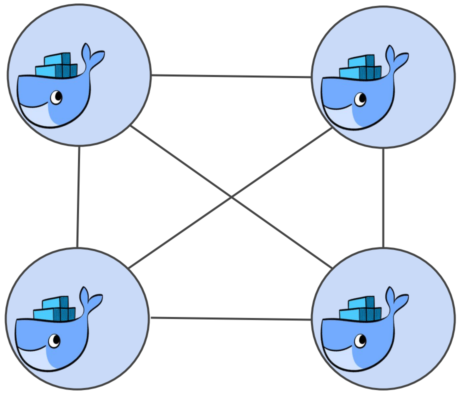
- Pod Ip Model
- Services
- Cluster to External Communication
- Network Policy

Pod Ip Model

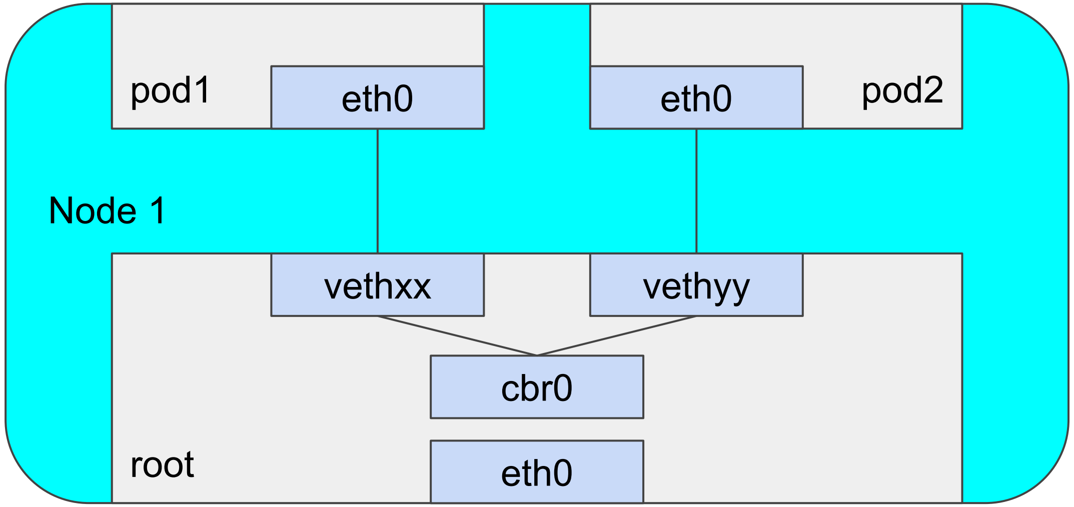
Every Pod Has Real IP Address

Basic Rule

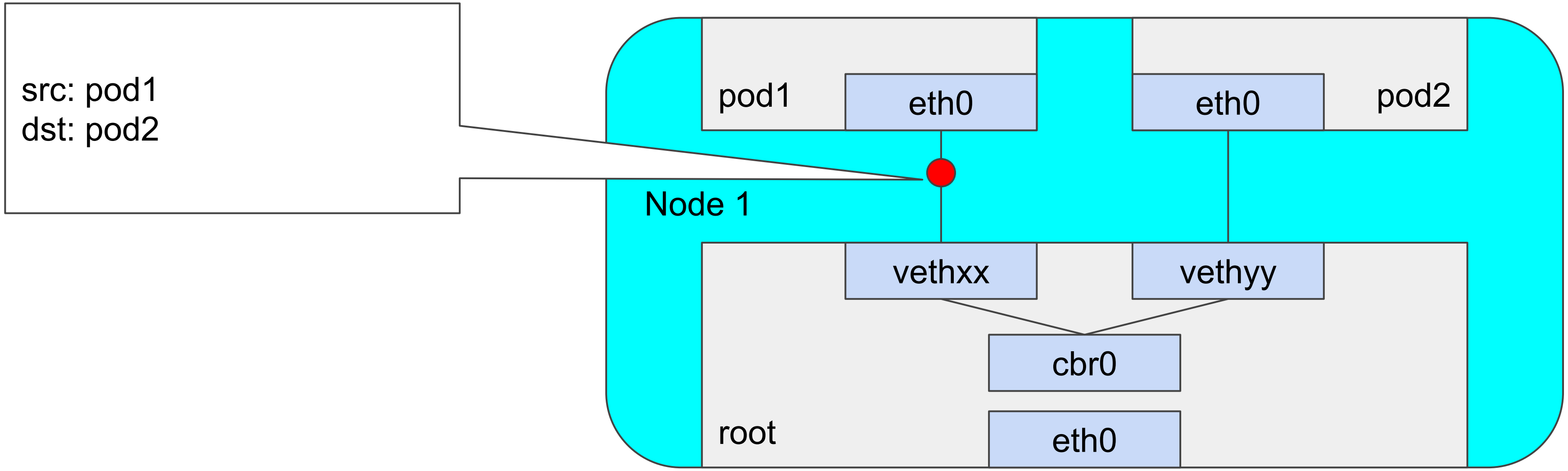
- Pod can be accessed from other pod, no matter where they are (can be distributed across the node, **flat**)
- Every node has CIDR(IP Block)
- K8s **DOESNT CARE** how (L2, L3, overlay, carrier pigeons,dll)
- Pods vs Docker
    - no port mapping
    - no shared machine private IP



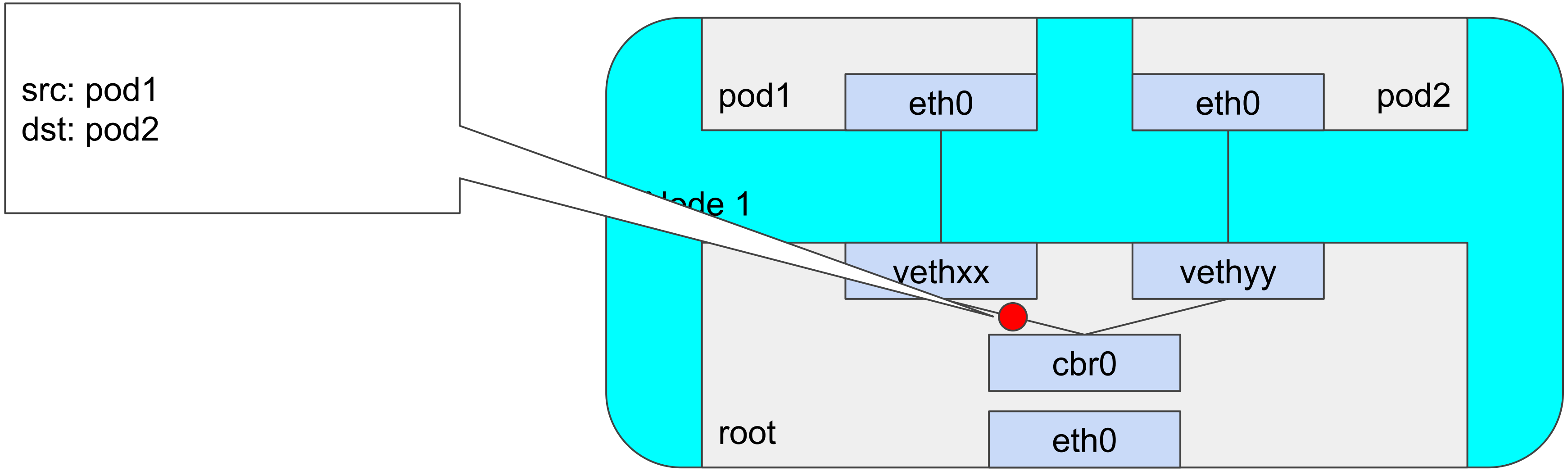
Network Namespace



Pod to Pod - Intra Node

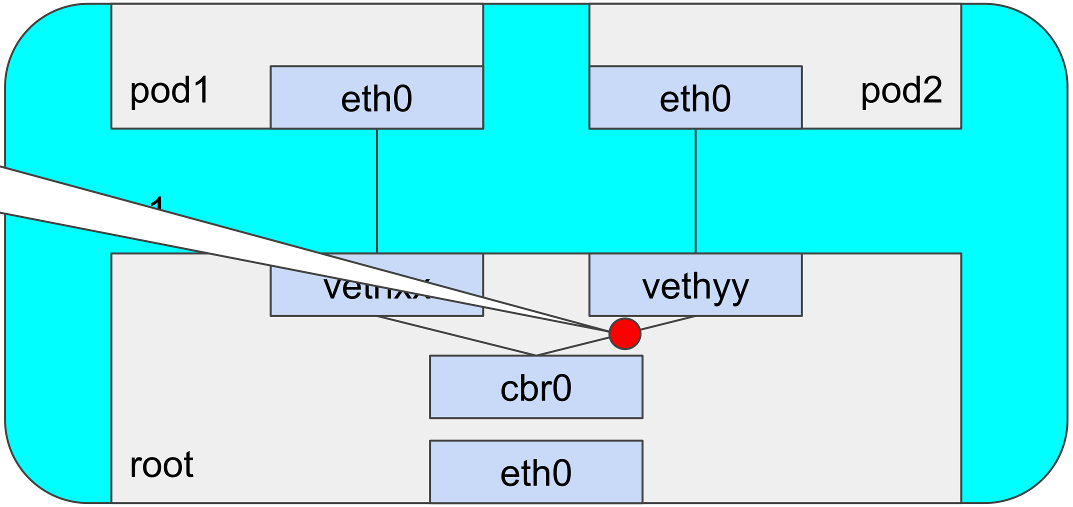


Pod to Pod - Intra Node

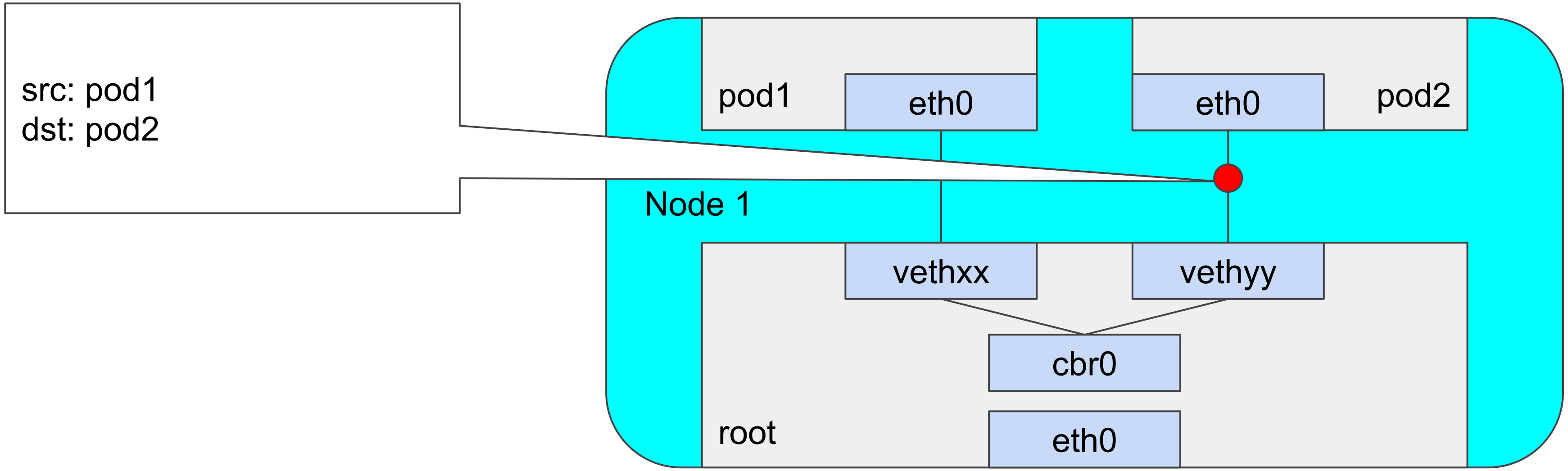


Pod to Pod - Intra Node

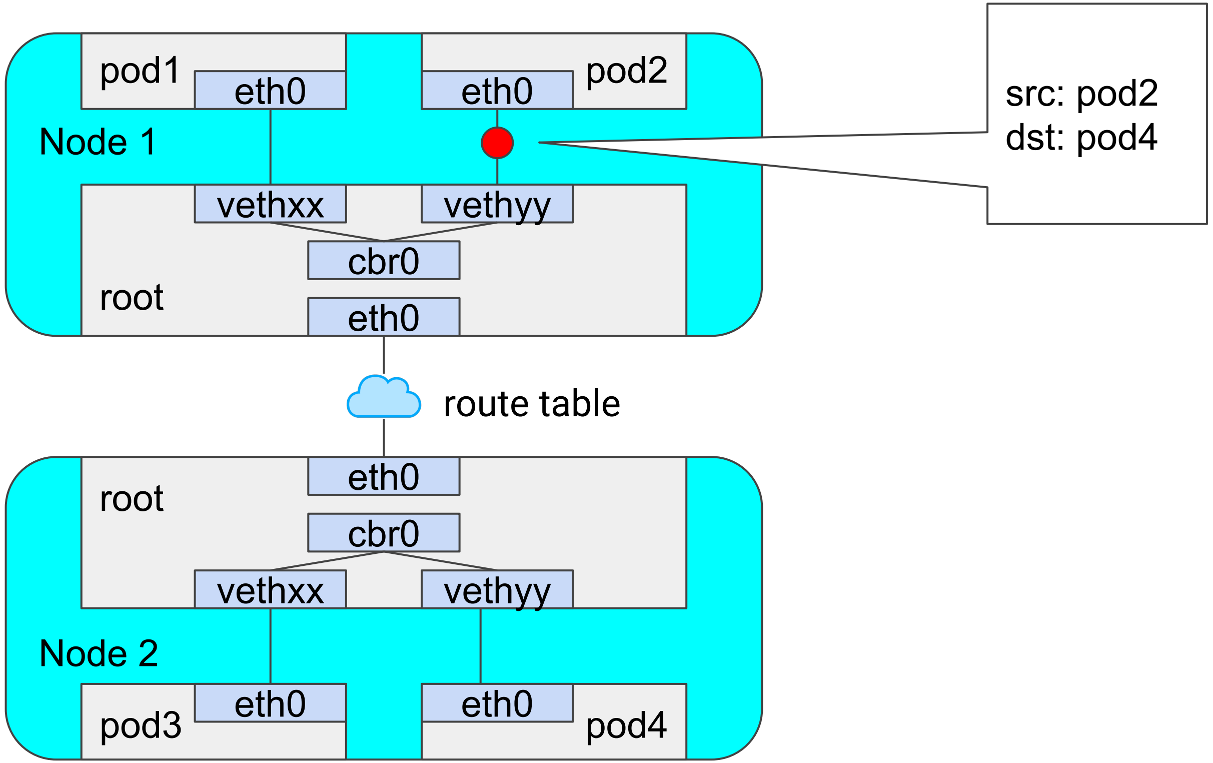
src: pod1
dst: pod2



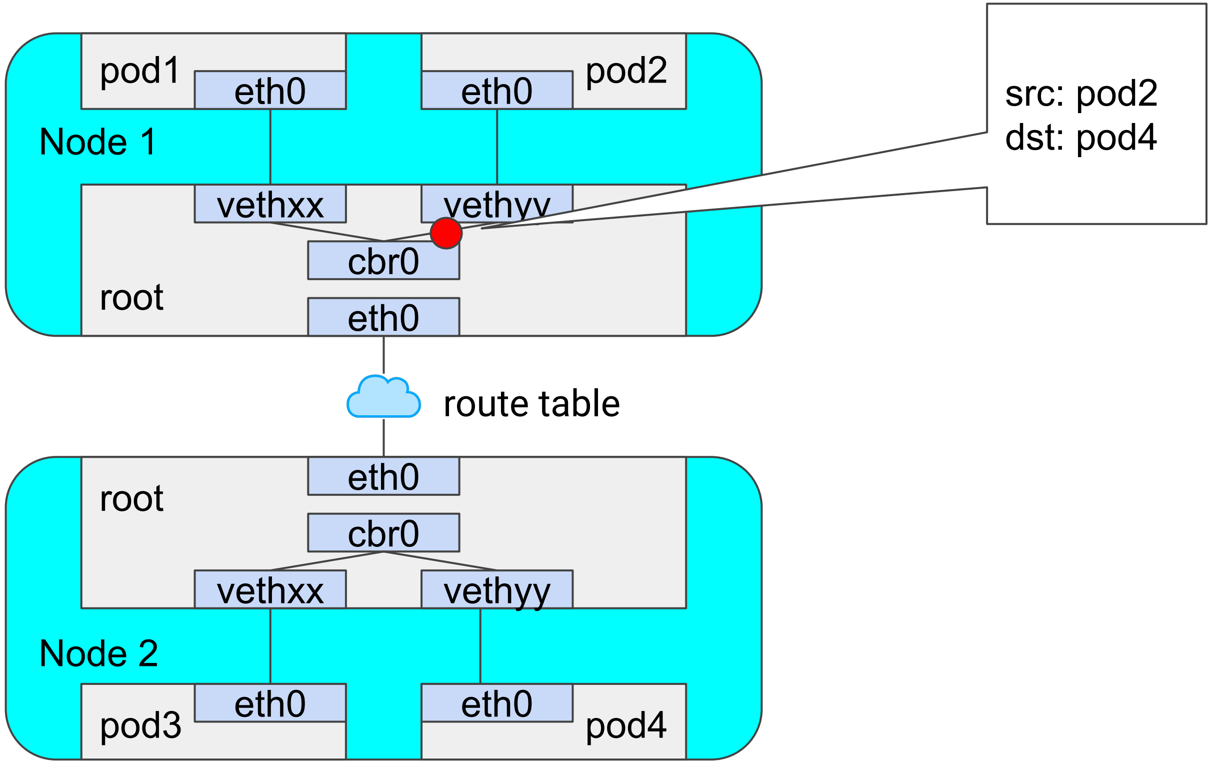
Pod to Pod - Intra Node



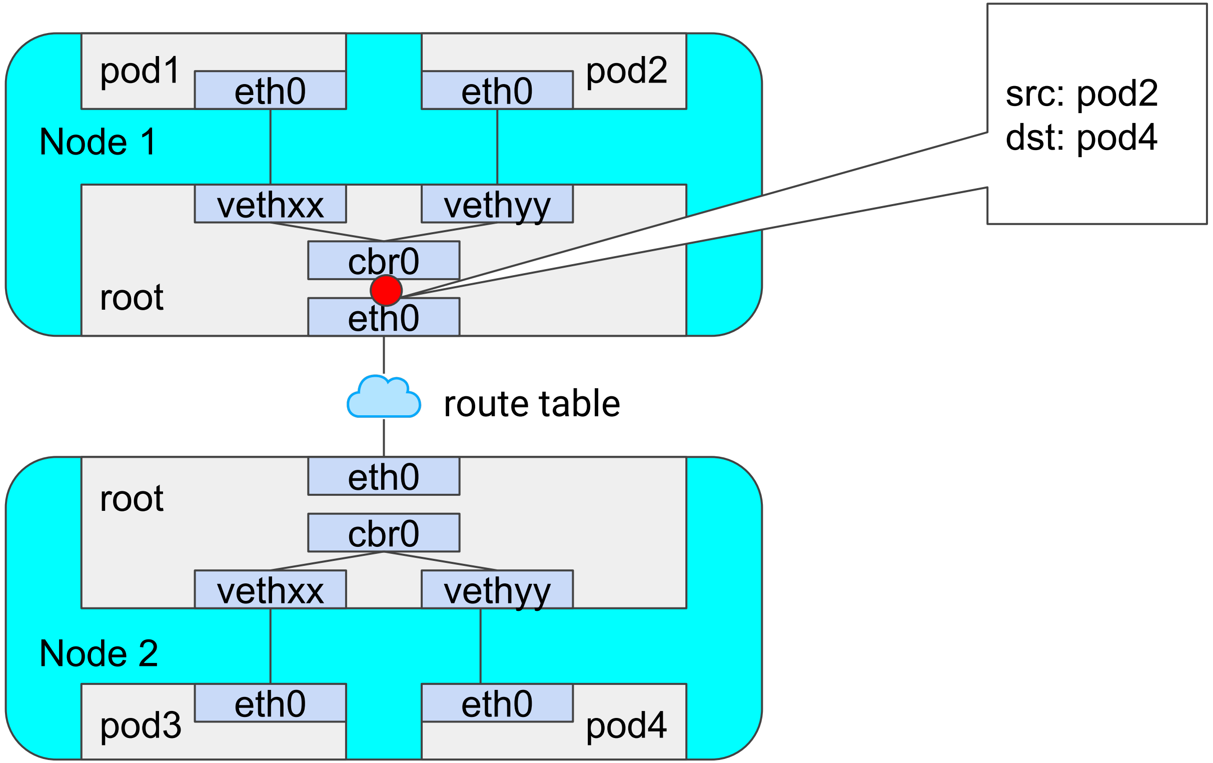
Pod to Pod - Inter Node



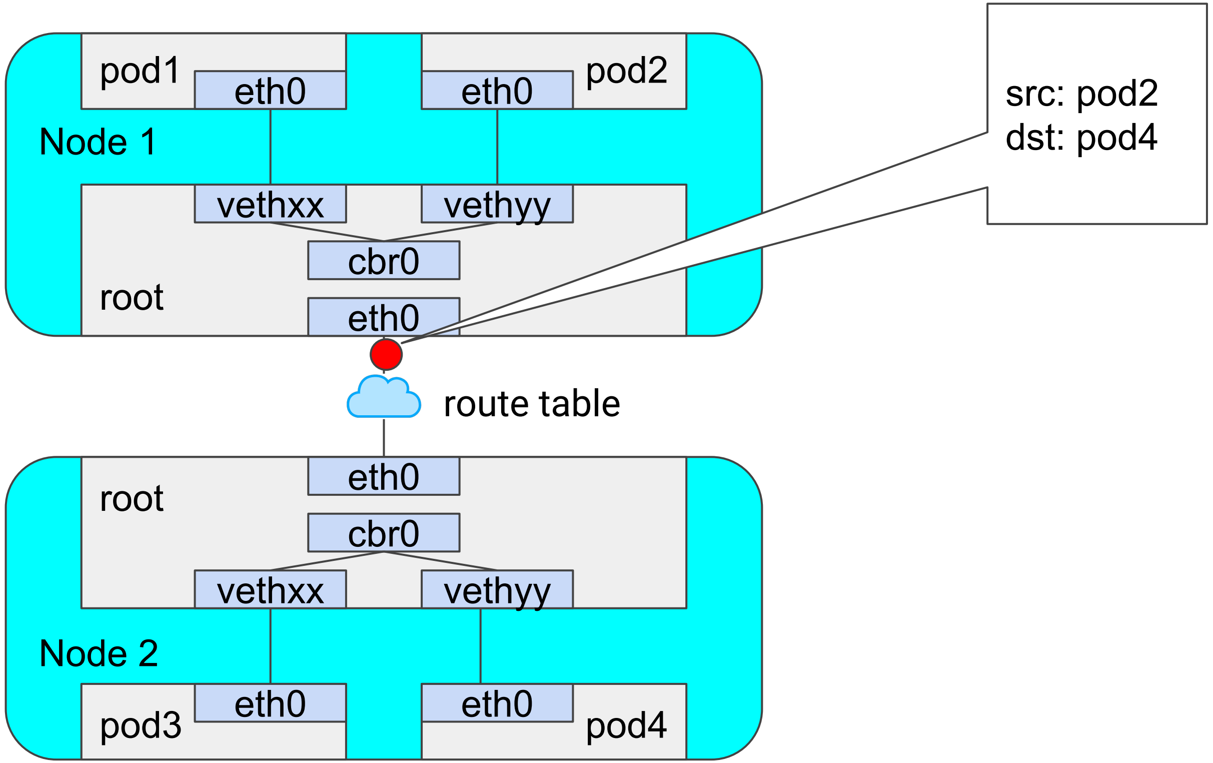
Pod to Pod - Inter Node



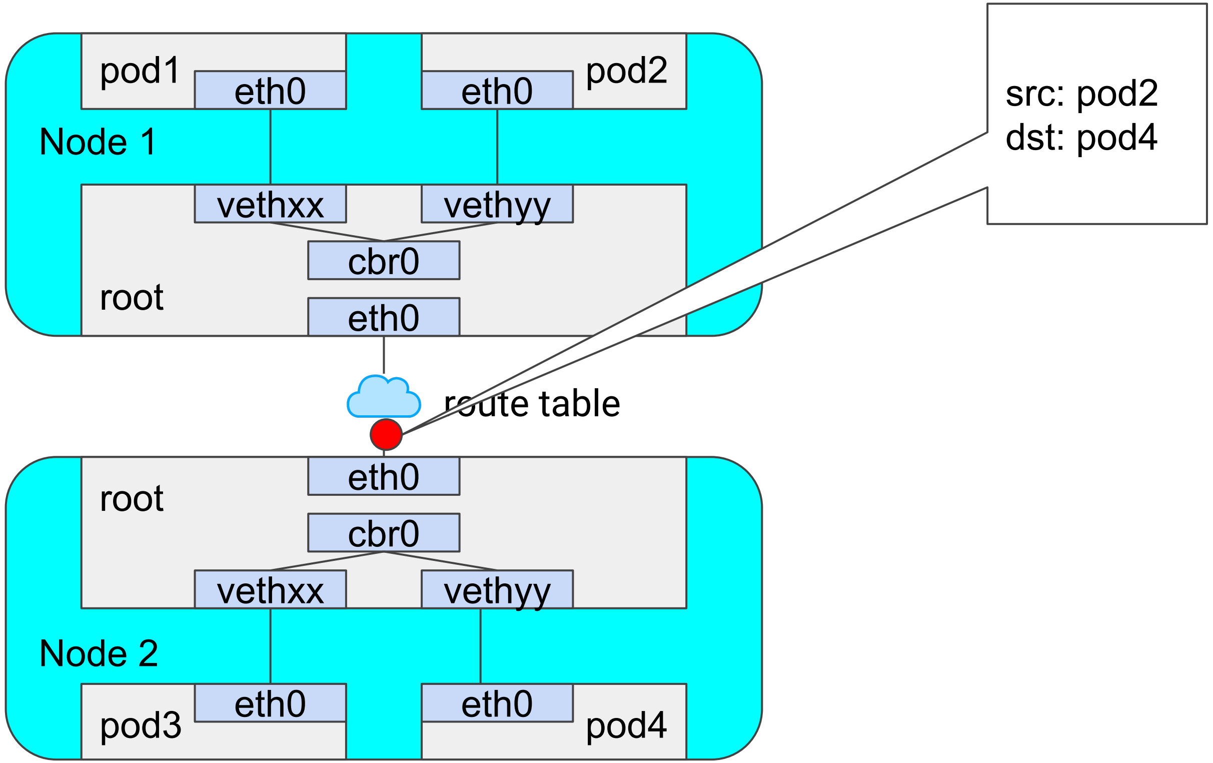
Pod to Pod - Inter Node



Pod to Pod - Inter Node

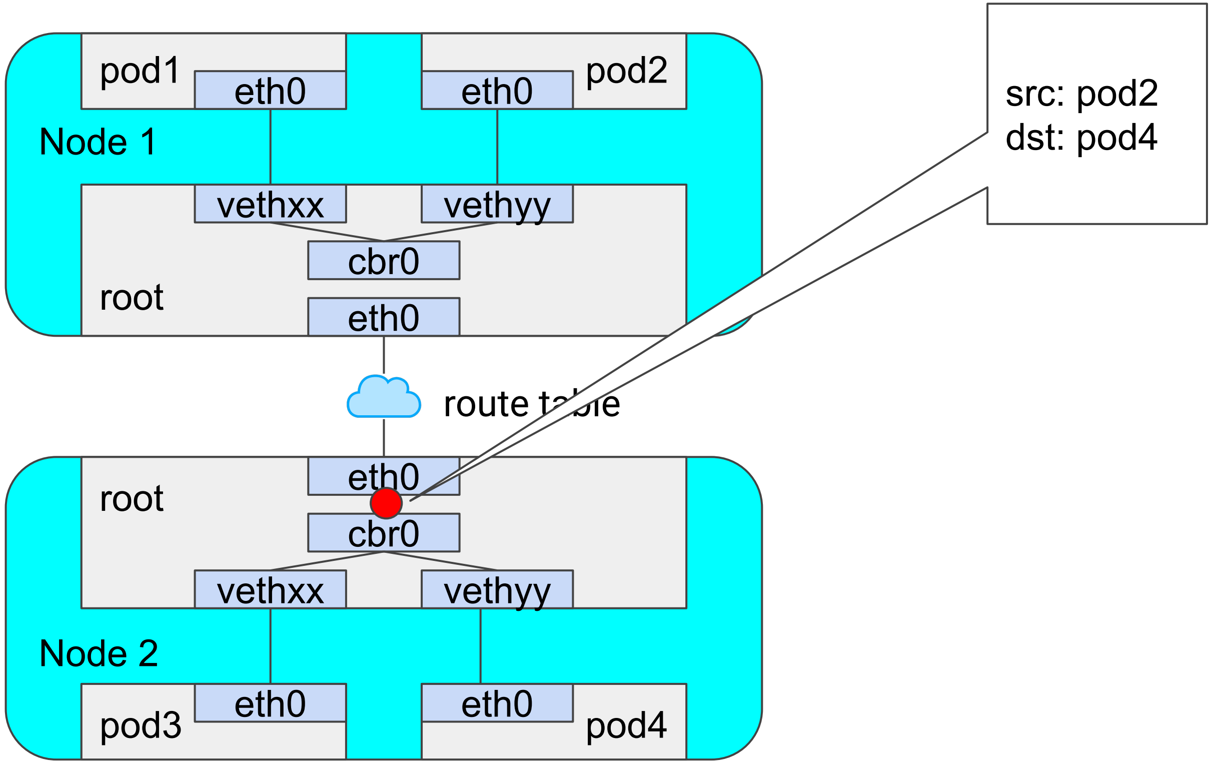


Pod to Pod - Inter Node

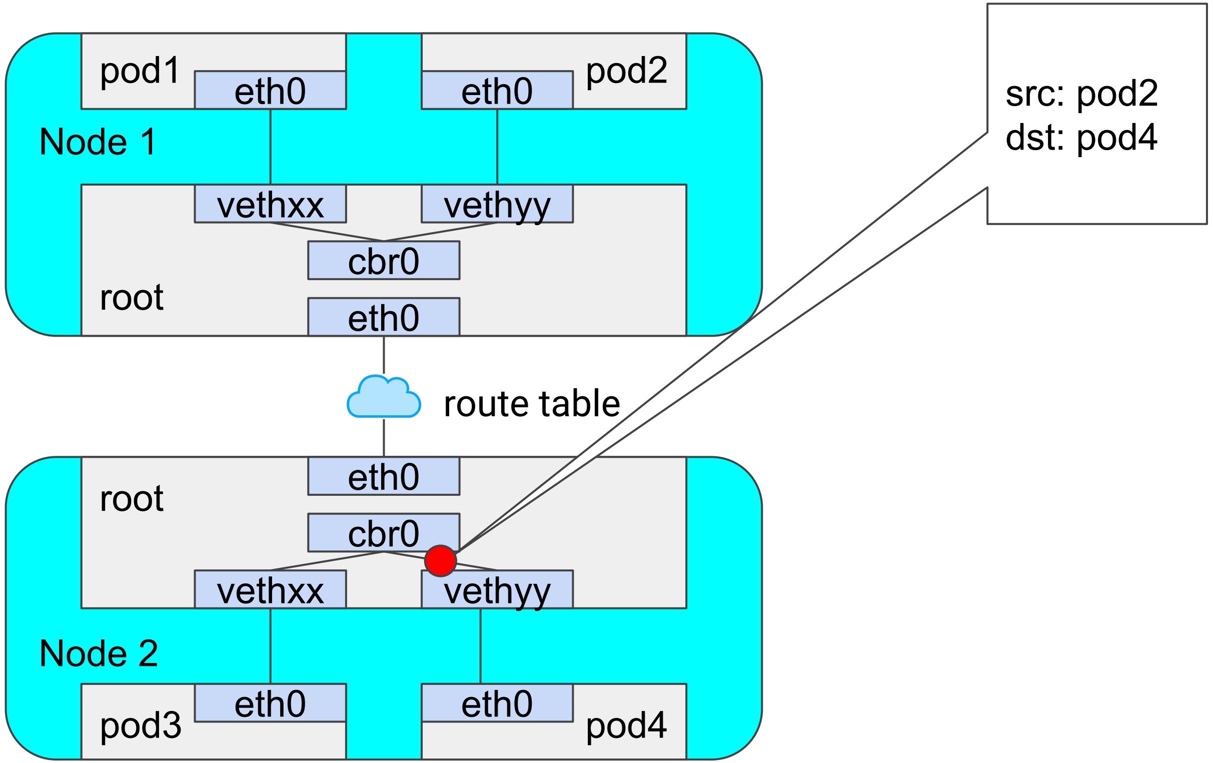


src: pod2
dst: pod4

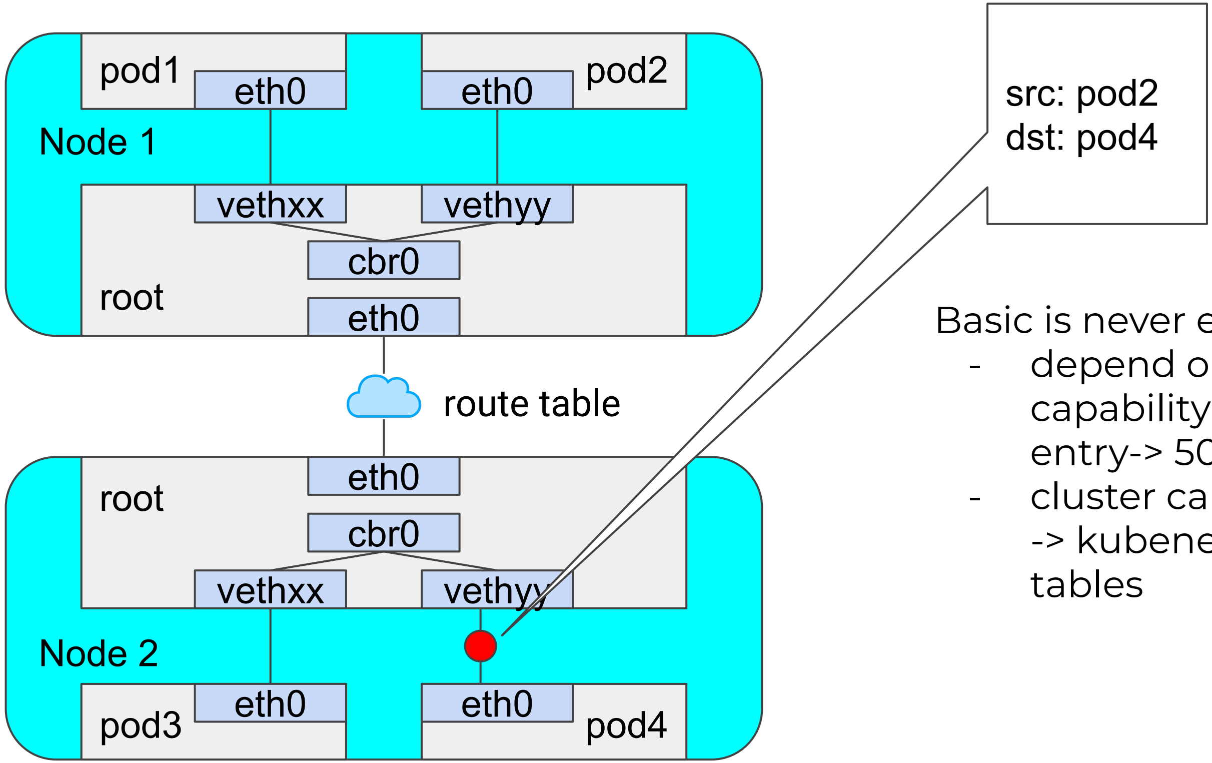
Pod to Pod - Inter Node



Pod to Pod - Inter Node



Pod to Pod - Inter Node



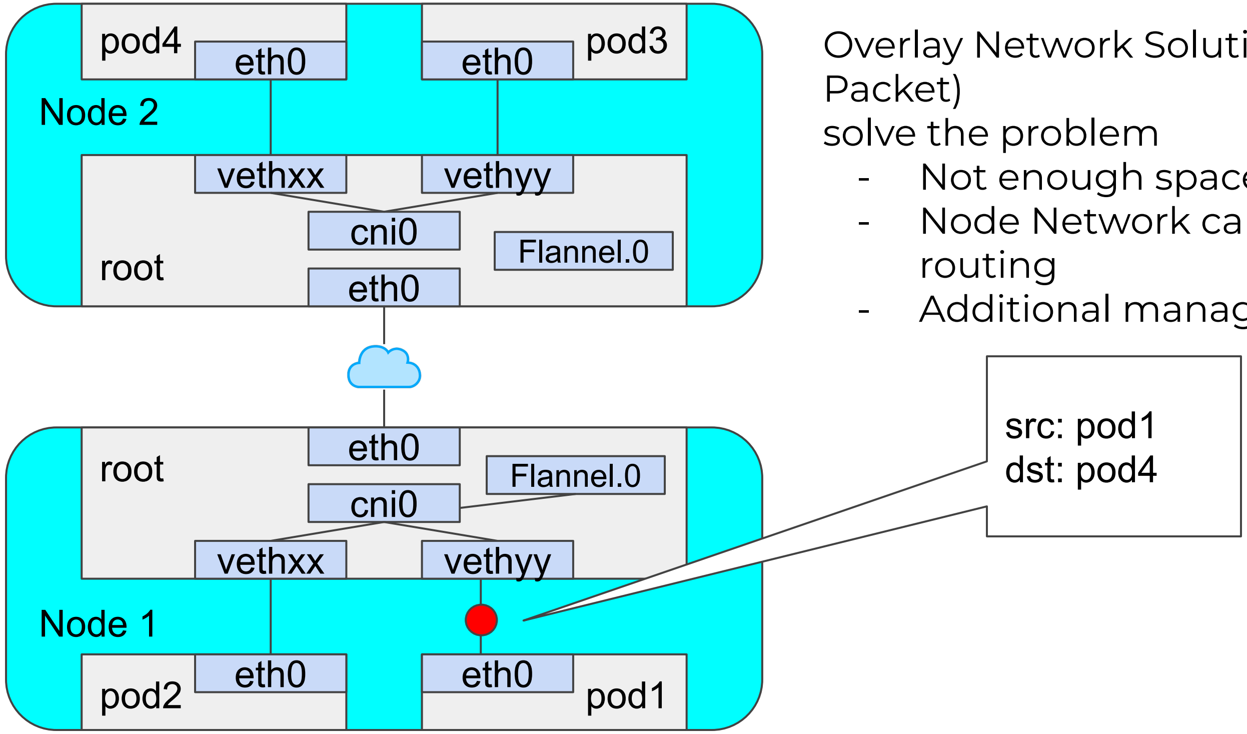
Basic is never enough sometimes

- depend on cloud provider route table capability (AWS VPC limit to 50 entry-> 50 node max per cluster)
- cluster cannot be setup in private vpc -> kubenet only support single route tables

Container Network Interface(CNI) Plugin

- Flannel
- Calico
- Romana
- Weavenet
- Canal
- Cilum

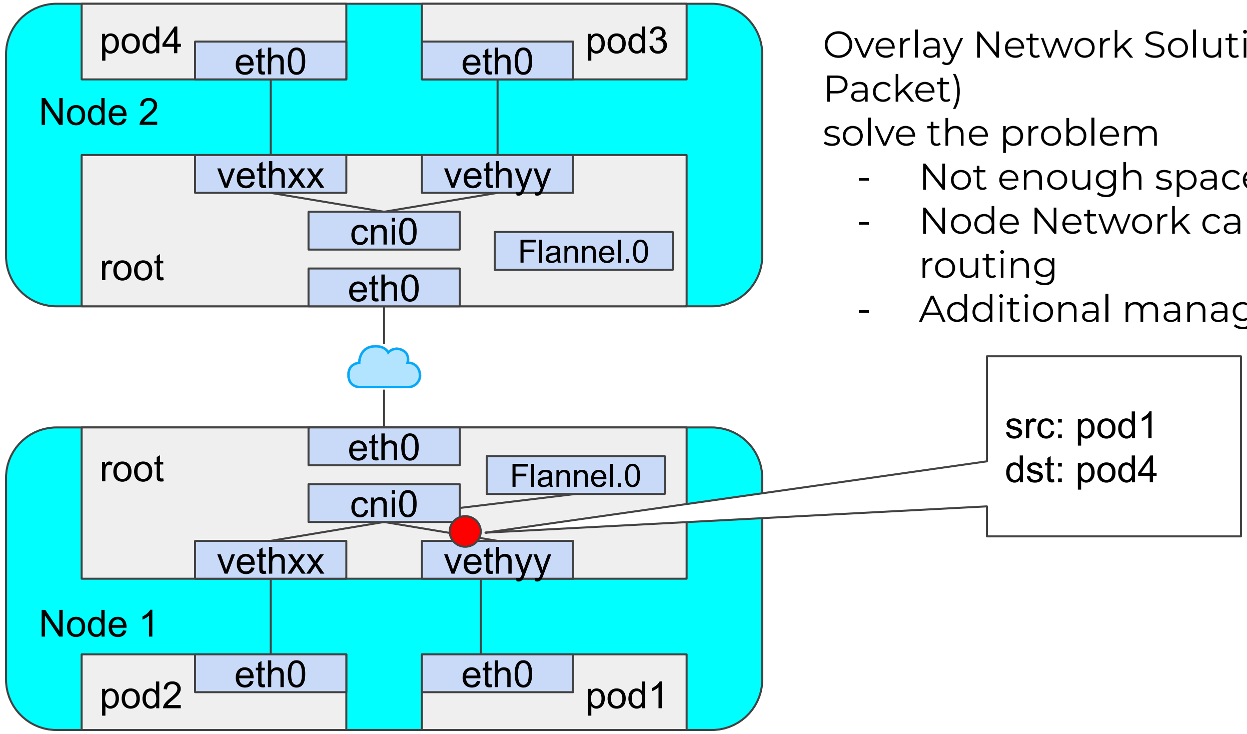
Pod to Pod - Inter Node-Flannel(concept)



Overlay Network Solution(Encapsulate Packet to Packet)
solve the problem

- Not enough space IP
- Node Network cannot handle too much routing
- Additional management feature

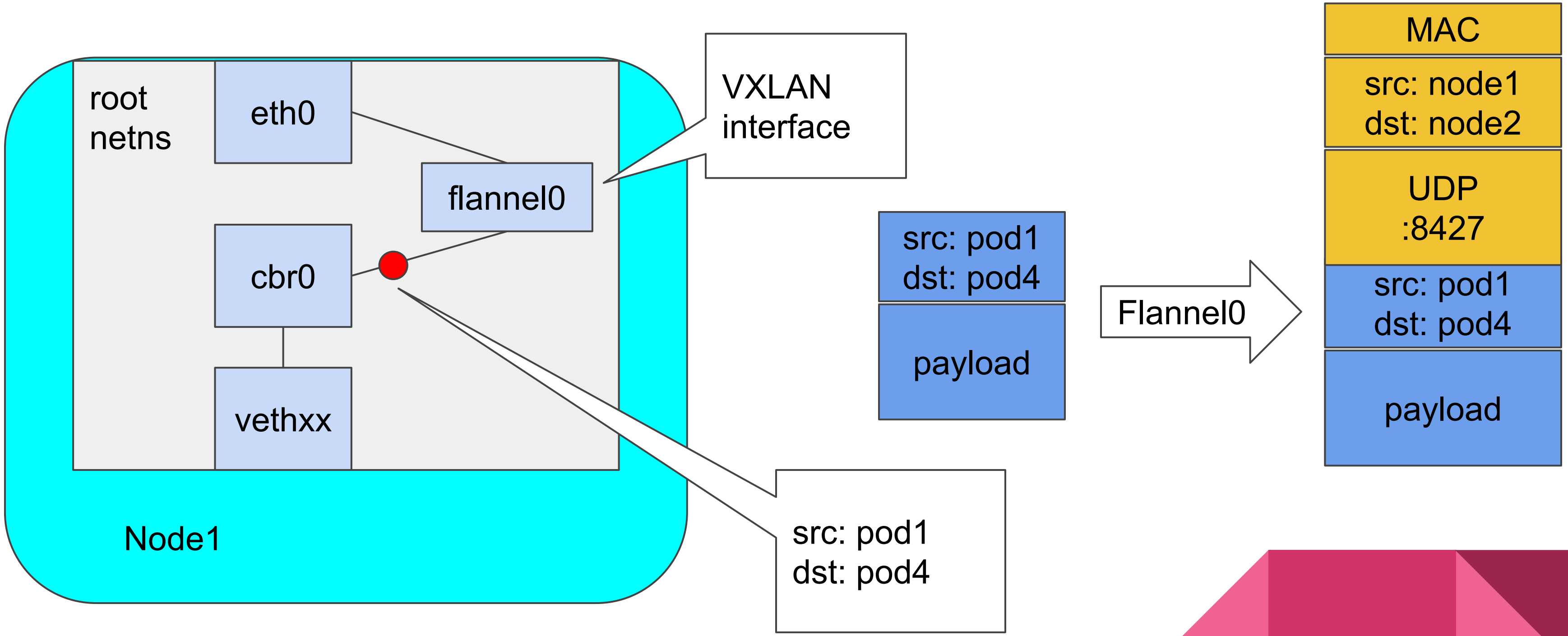
Pod to Pod - Inter Node-Flannel(concept)



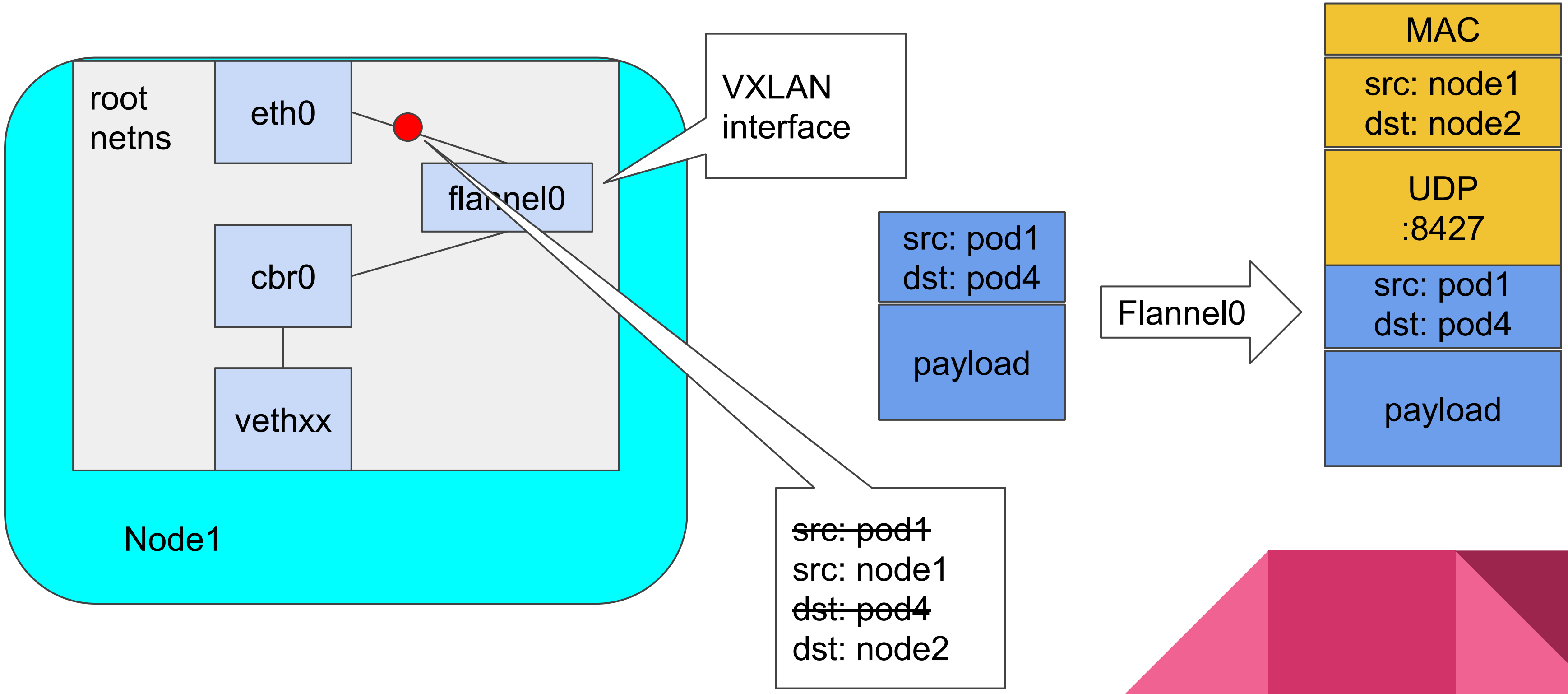
Overlay Network Solution(Encapsulate Packet to Packet)
solve the problem

- Not enough space IP
- Node Network cannot handle too much routing
- Additional management feature

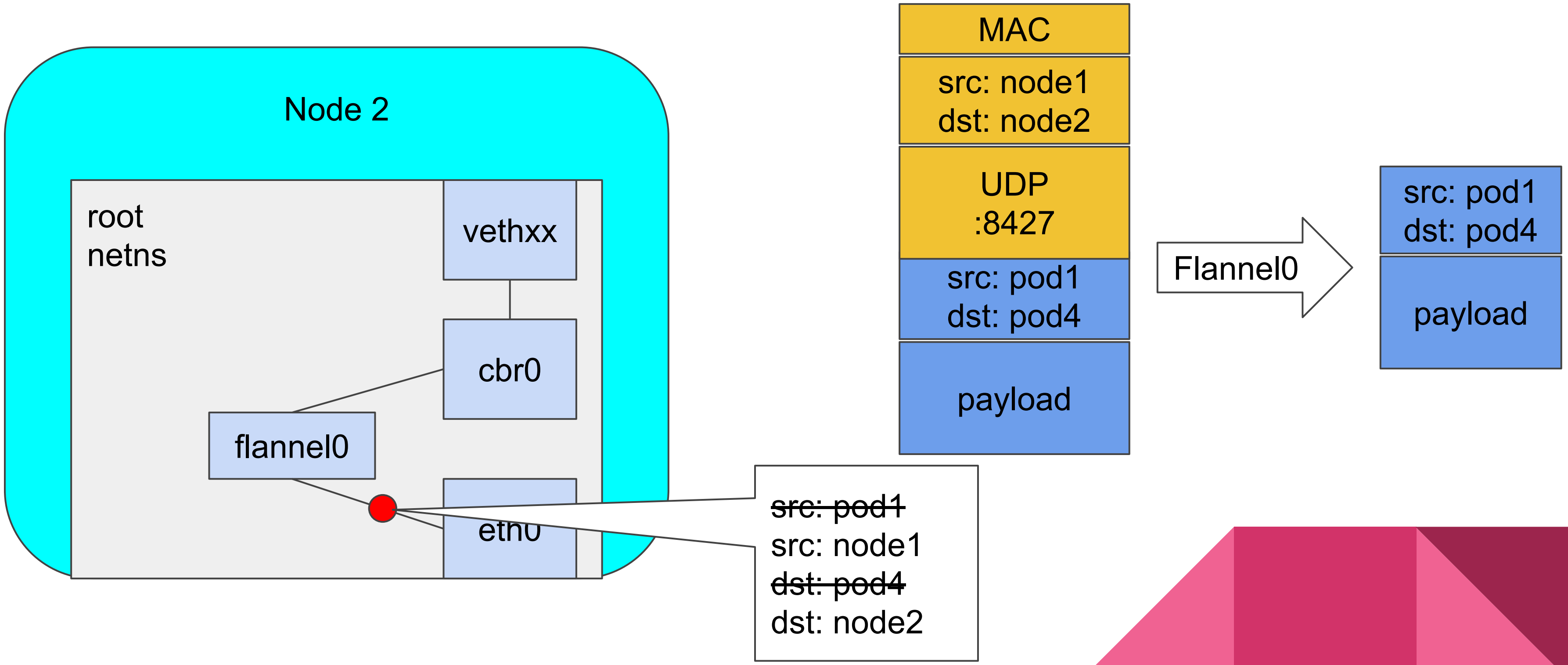
Pod to Pod - Inter Node-Flannel(concept)



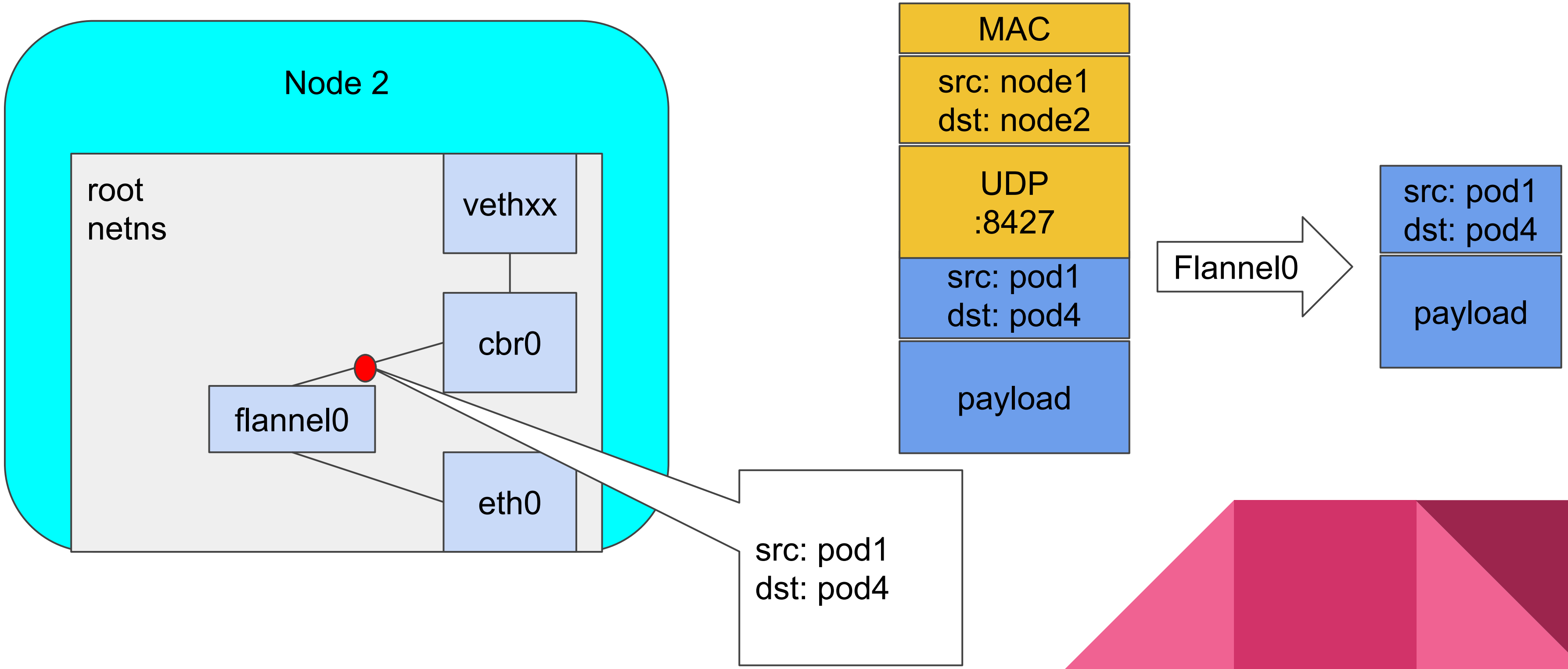
Pod to Pod - Inter Node-Flannel(concept)



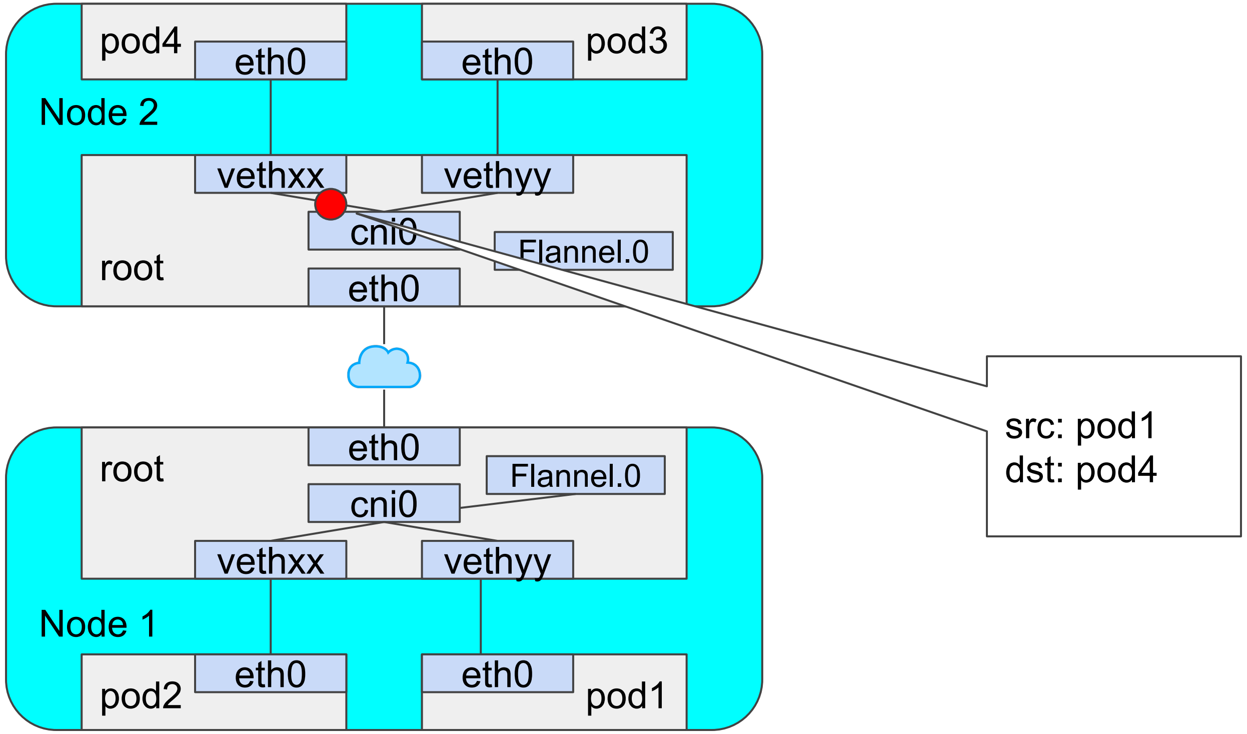
Pod to Pod - Inter Node-Flannel(concept)



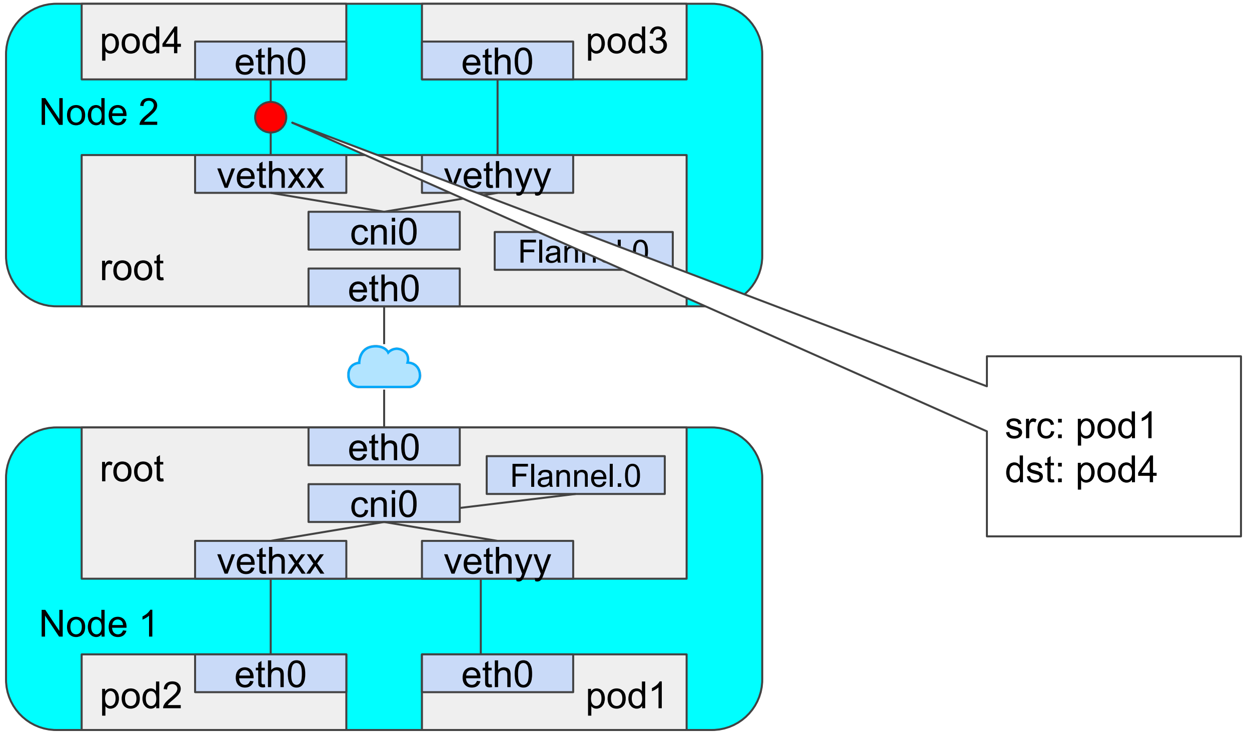
Pod to Pod - Inter Node-Flannel(concept)



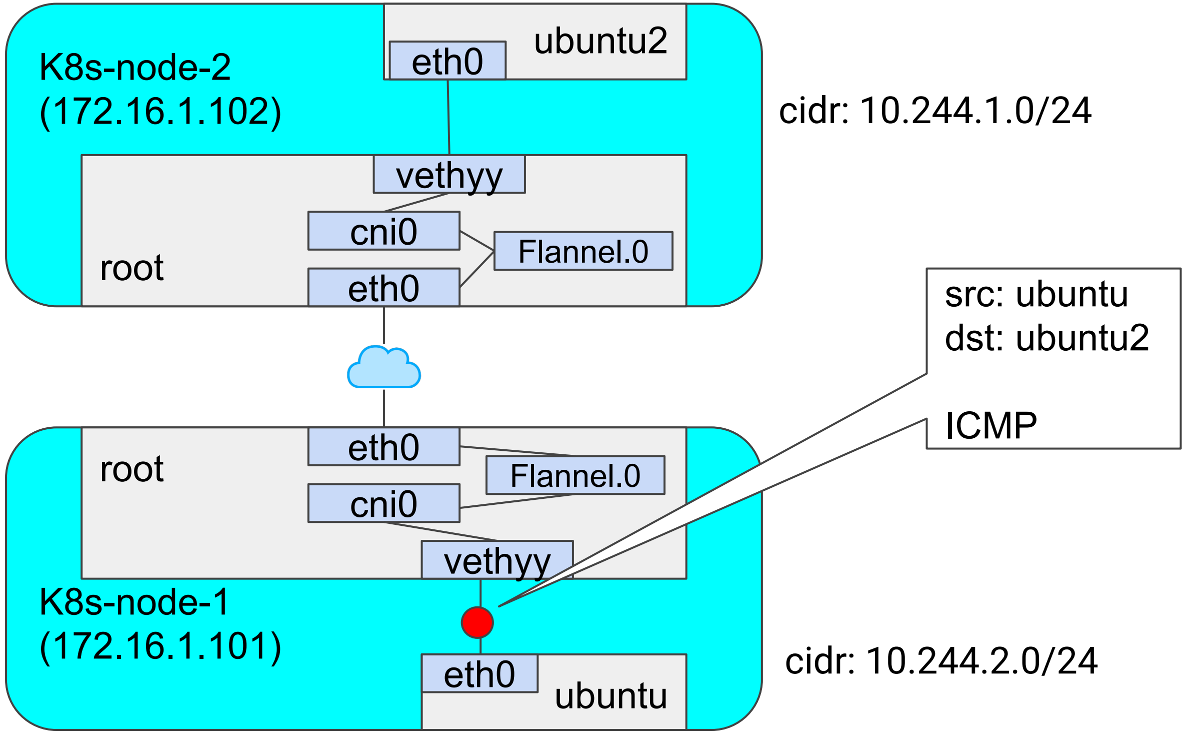
Pod to Pod - Inter Node-Flannel(concept)



Pod to Pod - Inter Node-Flannel(concept)



Pod to Pod - Inter Node-Flannel(demo)

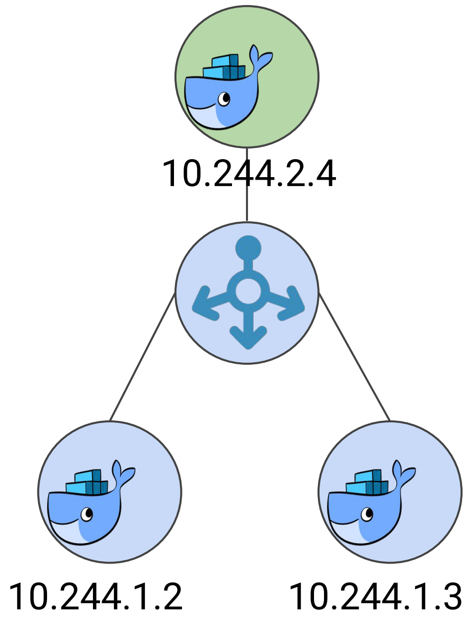


Life is never flat...

rolling update
scale up/down
pods crash/hang
nodes reboot

pods is mortal. come and go, never get ressurected

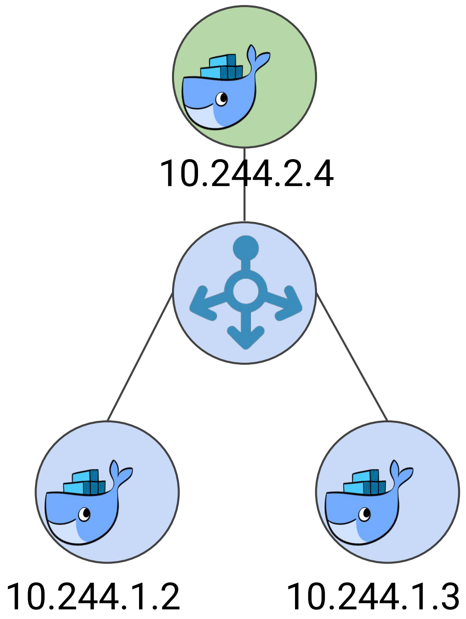
we need some interface abstraction to handle dynamics in pod life rotation



Services

Abstraction

- *load balancer pods”*
- Stable Virtual IP
- Group of Endpoint
- Support Port Forwarding



Link Service to Pods(demo)

```
apiVersion:
extensions/v1beta1
kind: Deployment
metadata:
 name: deploy-nginx
spec:
 replicas: 1
 template:
   metadata:
     labels:
       app: proxy
   ----
```

```
apiVersion: v1
kind: Service
metadata:
 name: srv-nginx
spec:
 ports:
 - port: 80
   protocol: TCP
   targetPort: 80
 selector:
   app: proxy

   ----
```

```
apiVersion: v1
kind: Service
metadata:
  --
  name: srv-nginx
  namespace: default
  --
spec:
  clusterIP: 100.68.232.5
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: proxy
  sessionAffinity: None
  type: ClusterIP
```

Services Type

- **ClusterIp**:

  assigned IP inside cluster and only accessible from inside only

- **NodePort**

  service mapped into particular node port and node IP

- **LoadBalancer**

  exposed to external traffic.

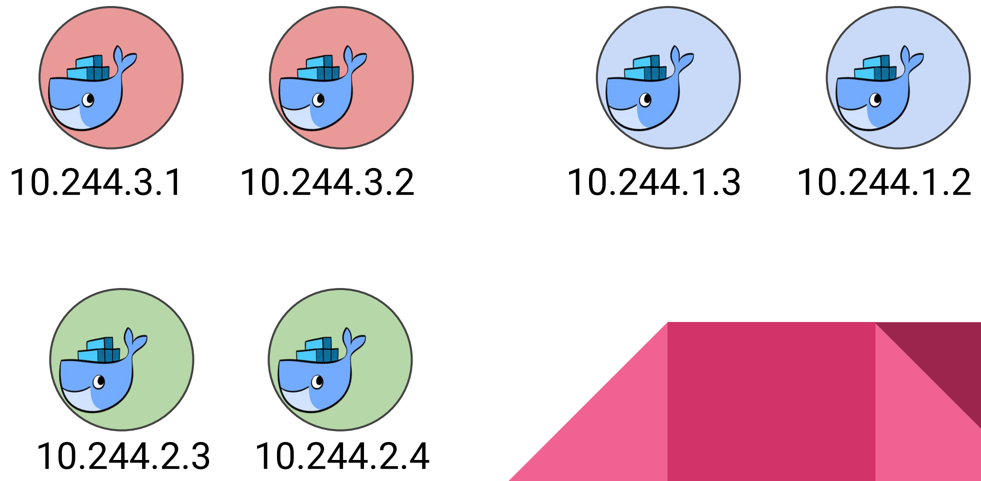
- **ExternalName**

  without pods selector, access external resource or other namespace

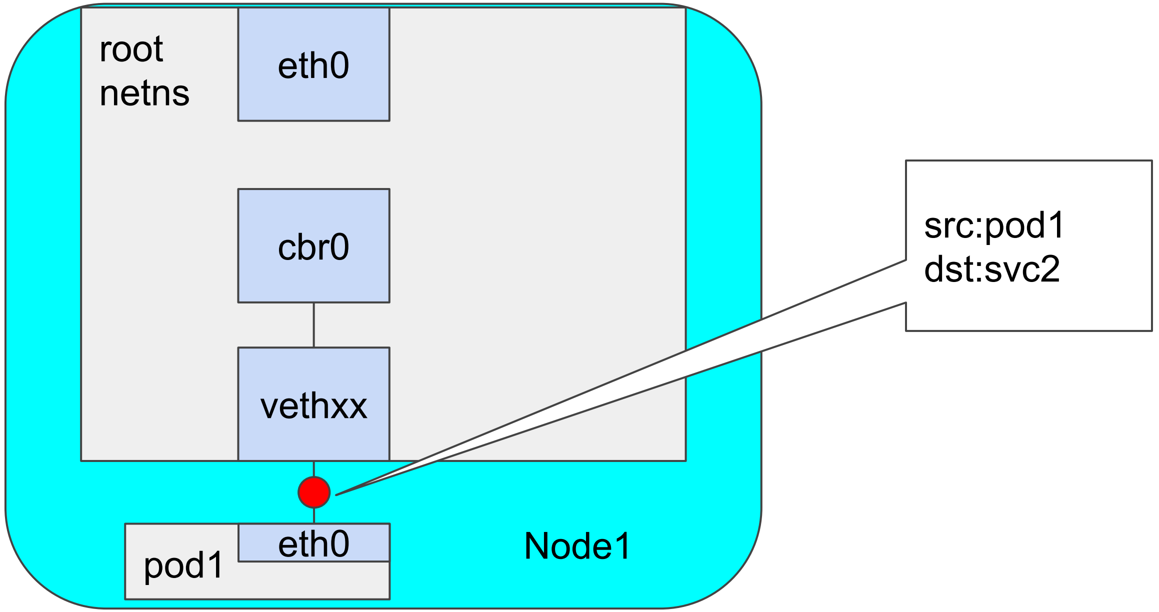
Endpoints

- API Object that automatically created when deploy service
- list all pods below particular service

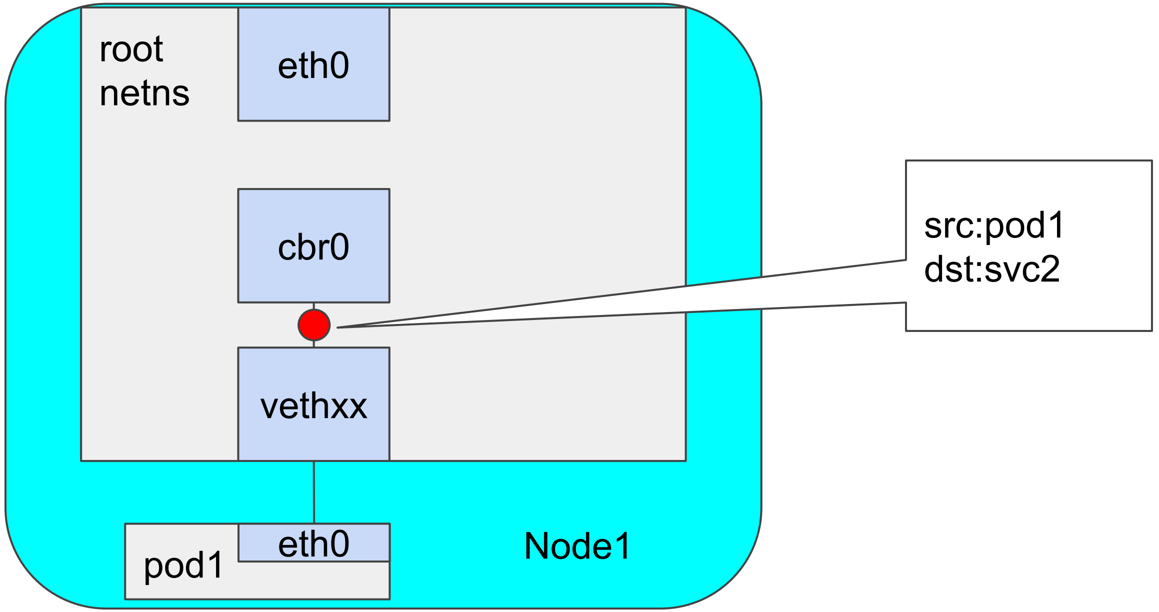
```
apiVersion: v1
kind: Endpoints
---
subsets:
- addresses:
  - ip: 10.244.3.1
    nodeName:k8s-node-1
    targetRef:
      -----
  - ip: 10.244.3.2
    nodeName: k8s-node-2
    targetRef:
      -----
```



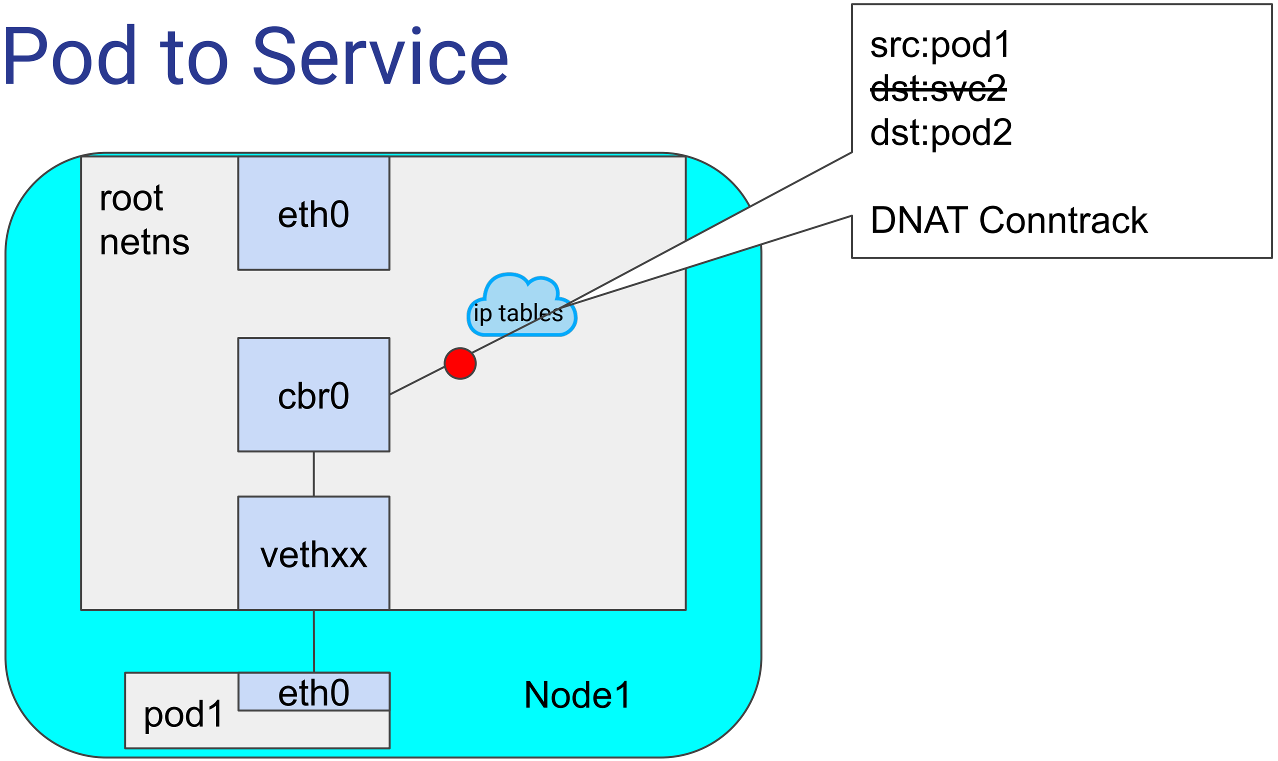
Pod to Service



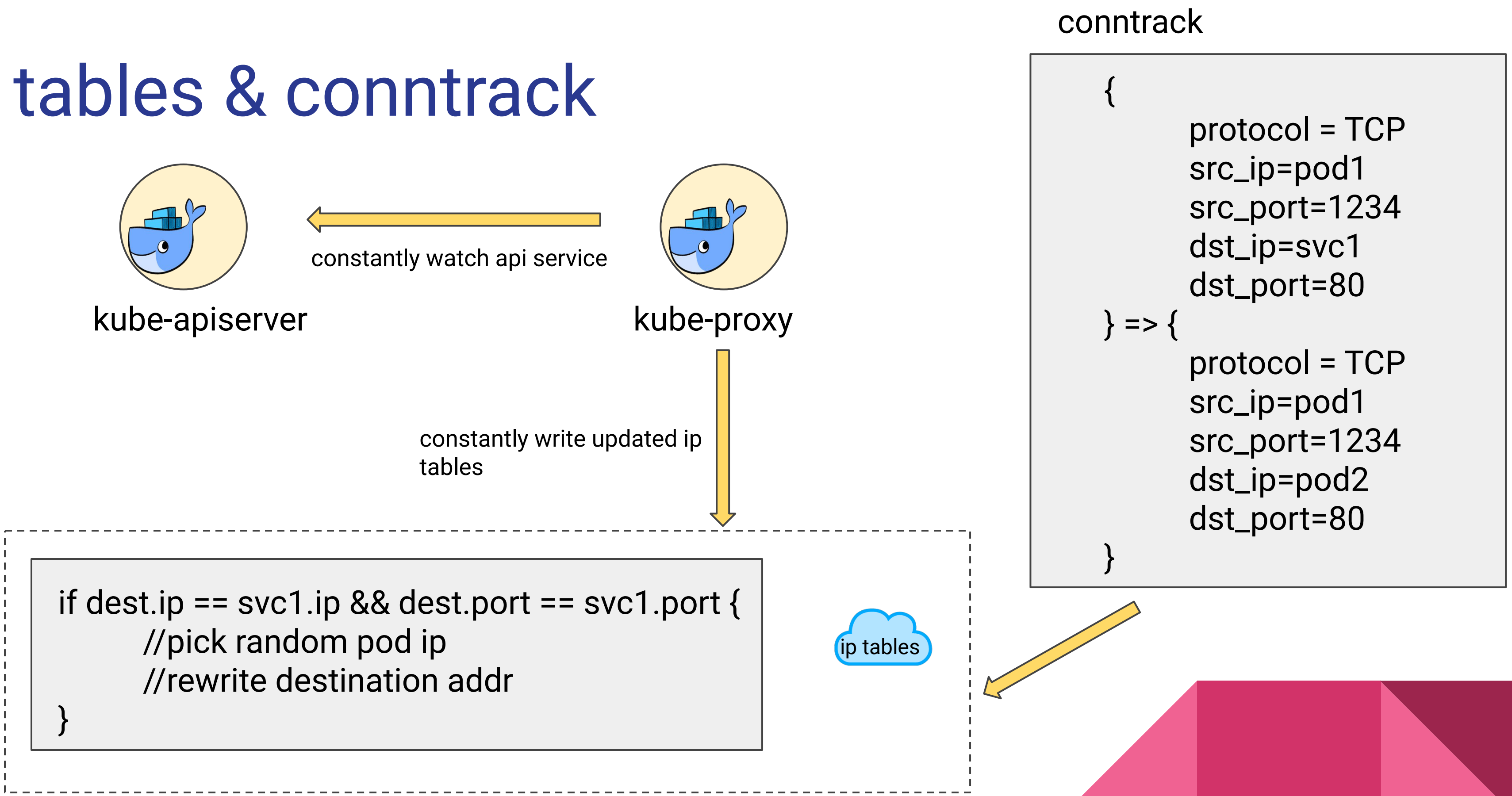
Pod to Service



Pod to Service



ip tables & conntrack

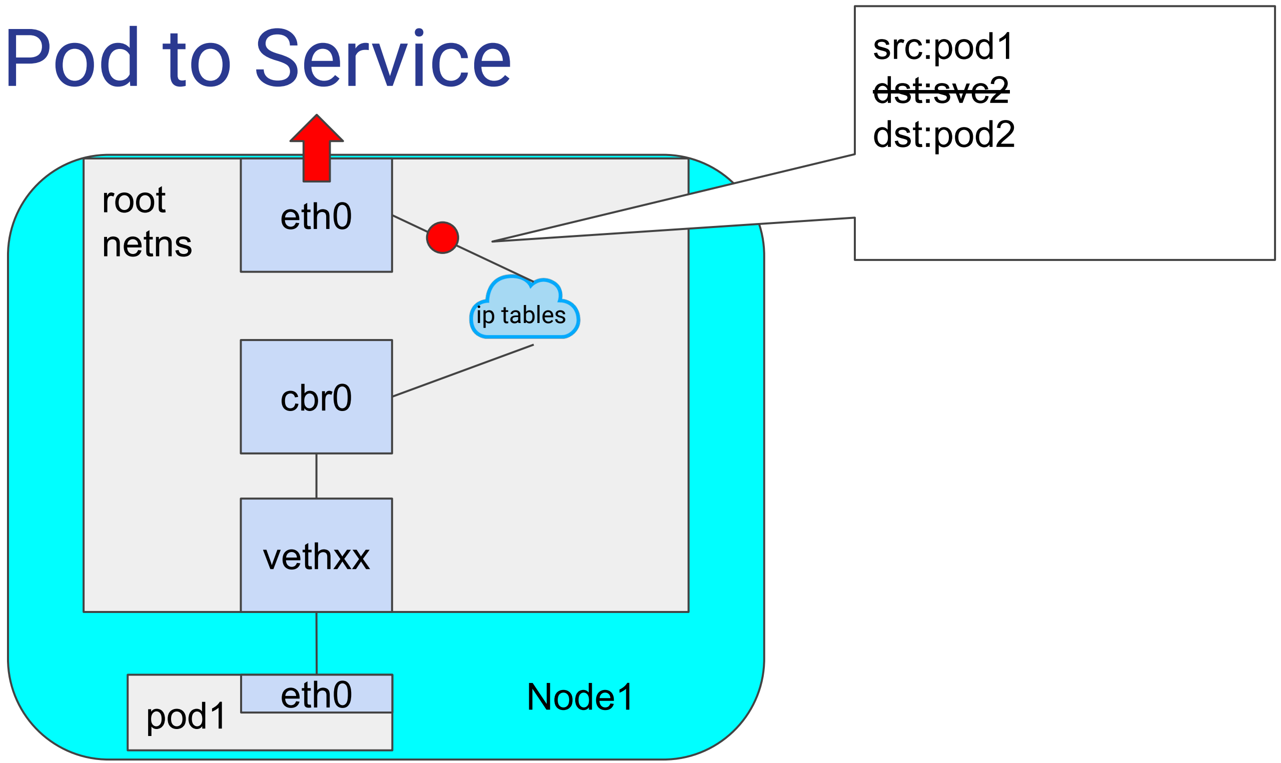

kube-apiserver
constantly watch api service
kube-proxy
constantly write updated ip tables
ip tables
conntrack

```
if dest.ip == svc1.ip && dest.port == svc1.port {
        //pick random pod ip
        //rewrite destination addr
}
```

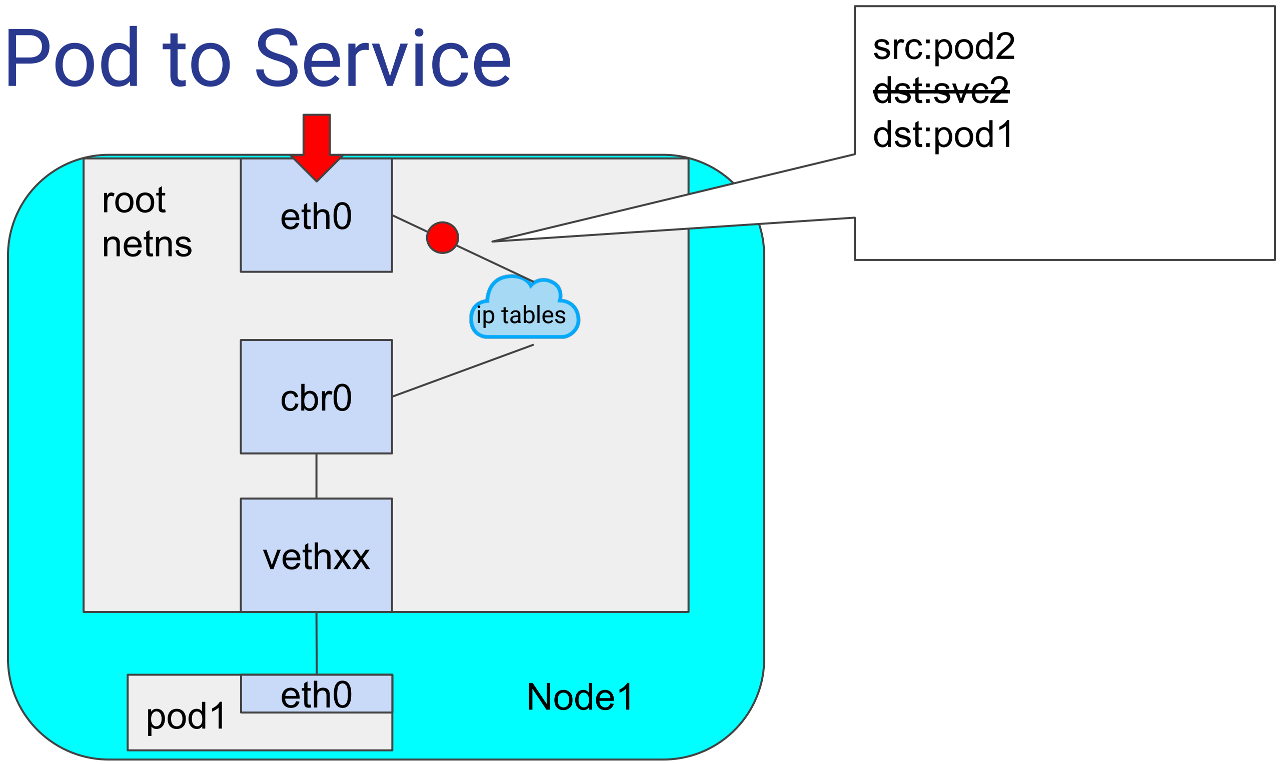
conntrack

```
{
        protocol = TCP
        src_ip=pod1
        src_port=1234
        dst_ip=svc1
        dst_port=80
} => {
        protocol = TCP
        src_ip=pod1
        src_port=1234
        dst_ip=pod2
        dst_port=80
}
```

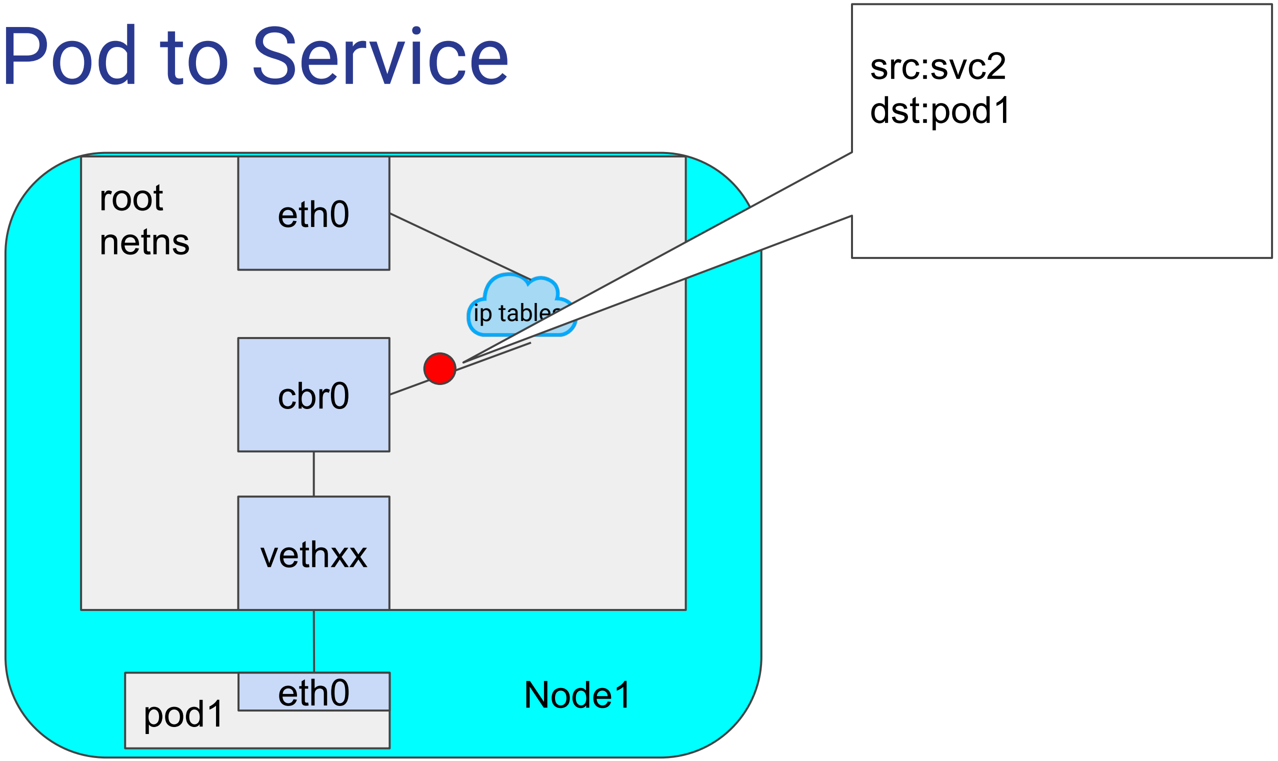
Pod to Service



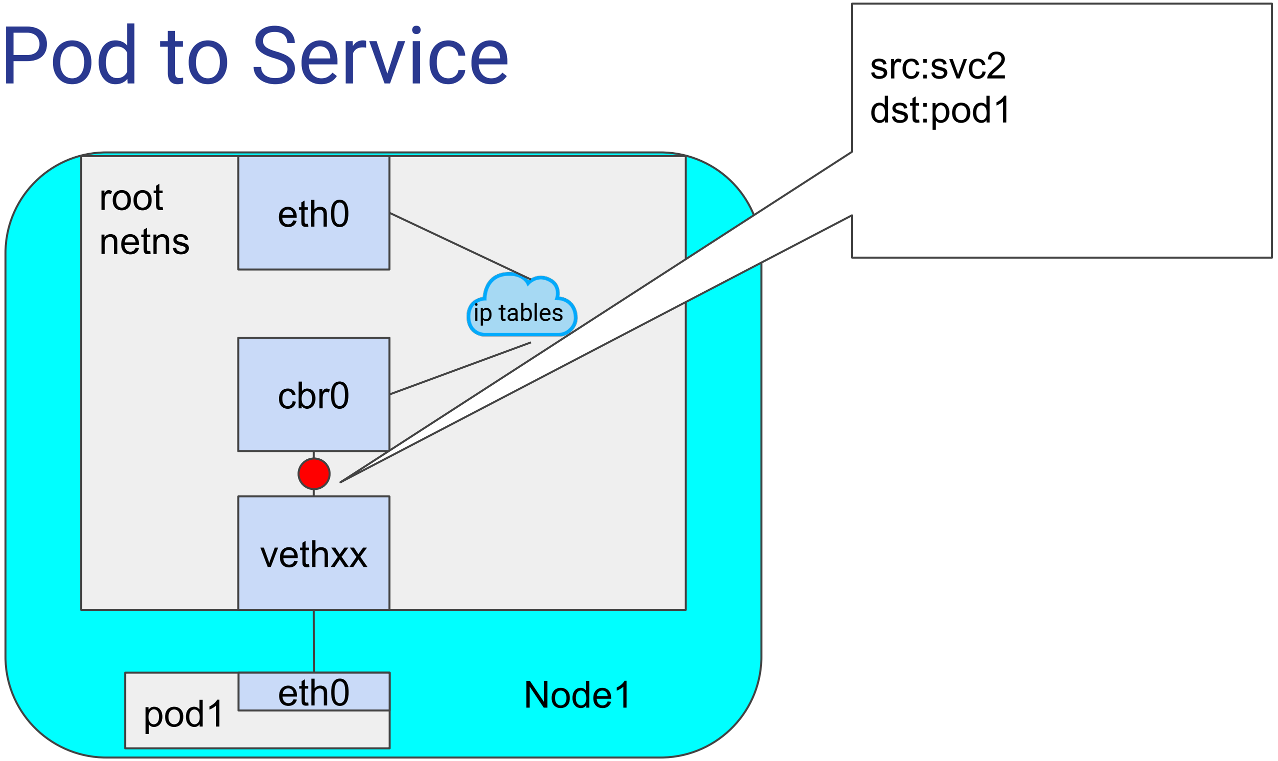
Pod to Service



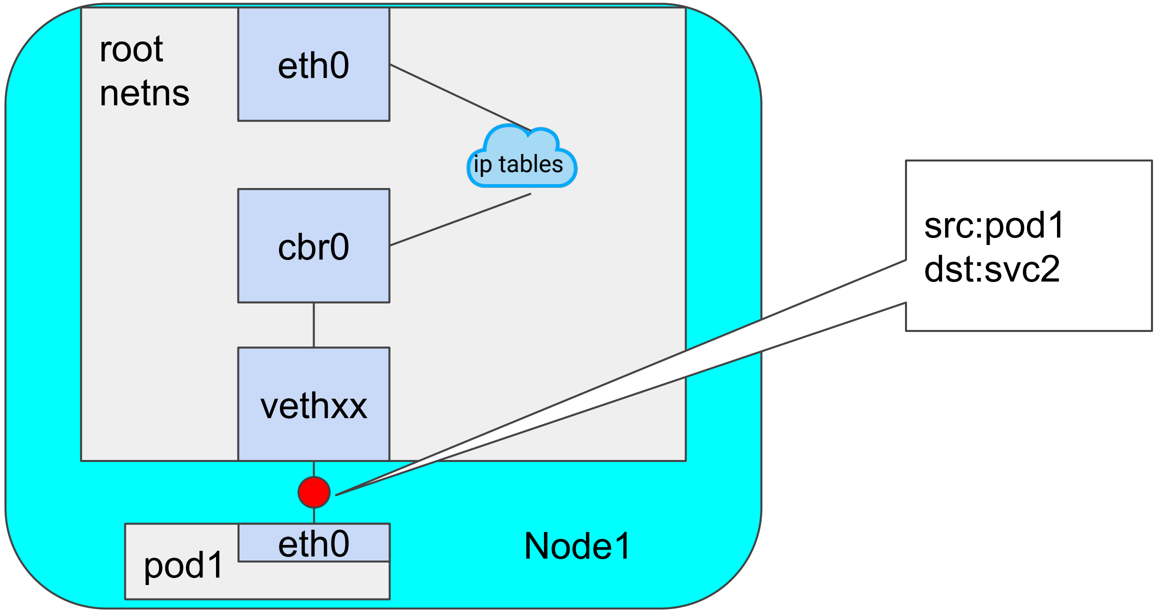
Pod to Service



Pod to Service

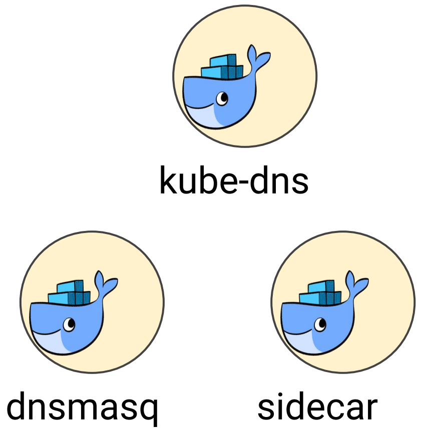


Pod to Service

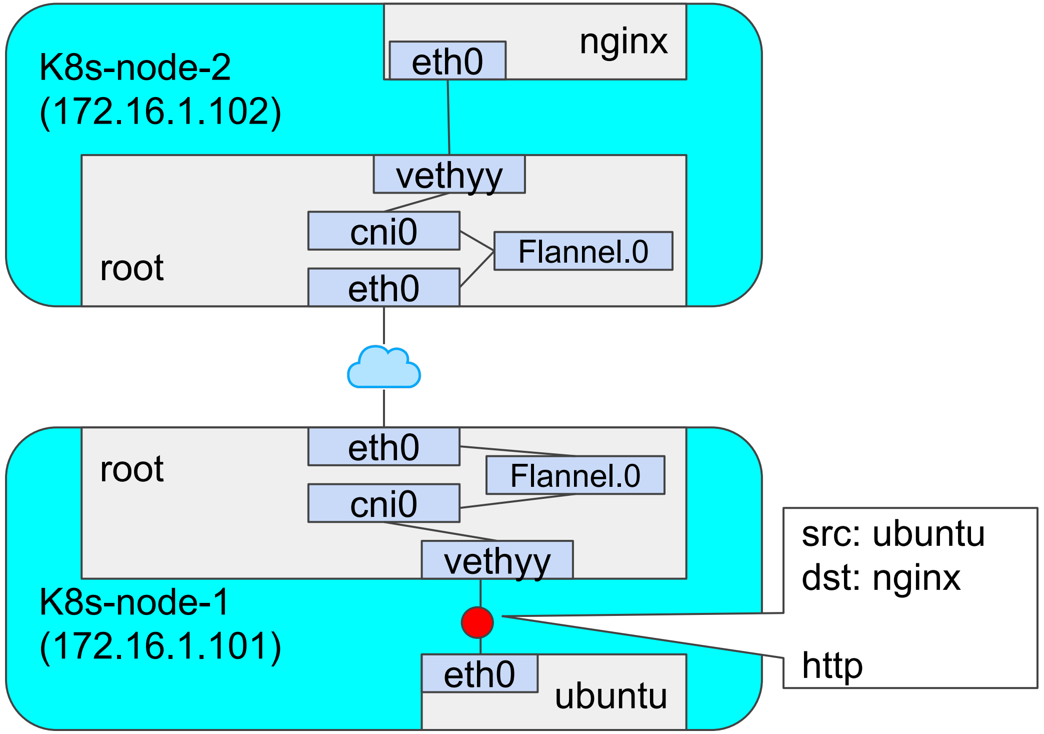


DNS

- say no to hardcode ip service. use friendly hostname, bonded with endpoint resource
- provide “A” & “SRV” records
- run in pods as kube-dns

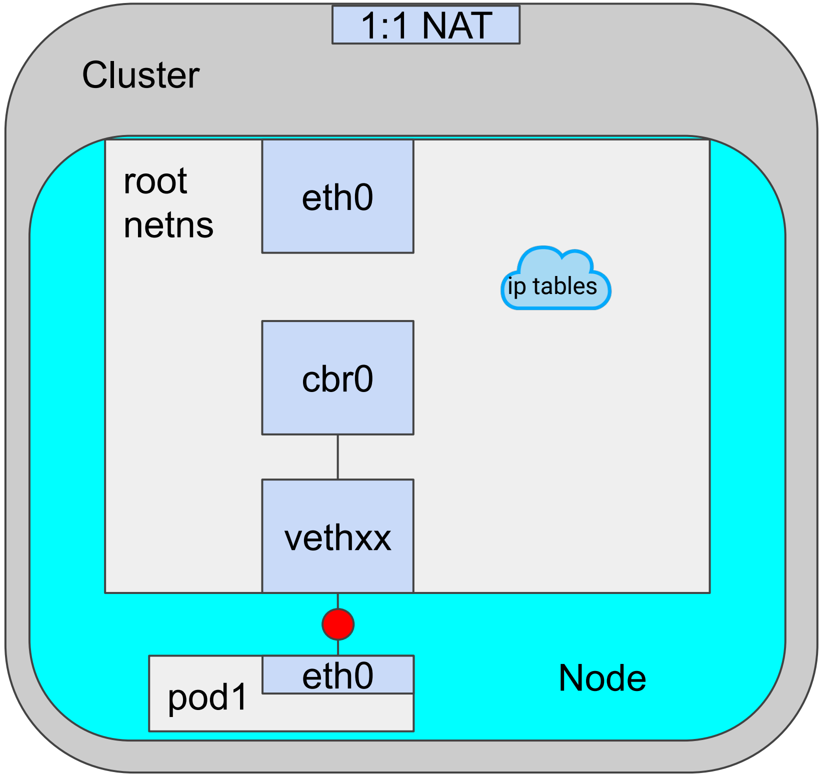


Pod to Service(demo)



Cluster to External Communication

Send to External Traffic(Egress) - GCP Case

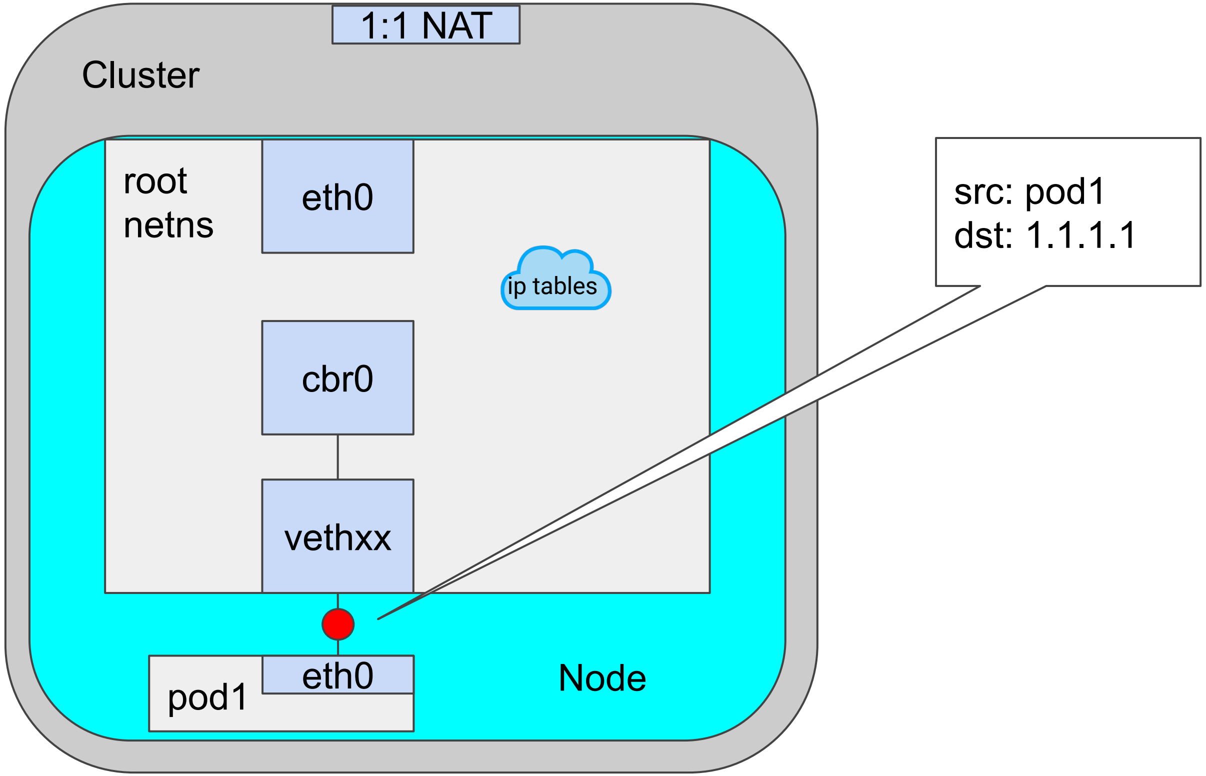


IP Private(inter-node) and maybe have public IP also

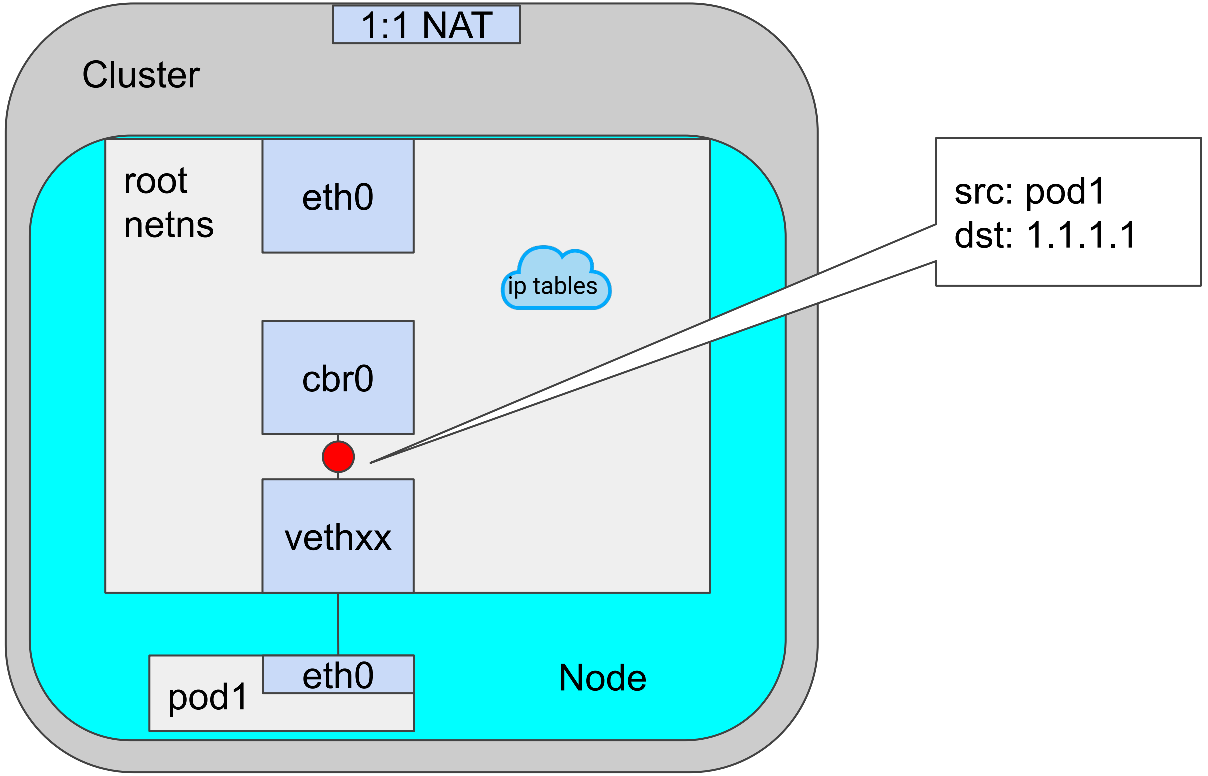
1:1 NAT

private	public
10.147.0.12	35.245.146.195

Life of packet: pod-to-internet

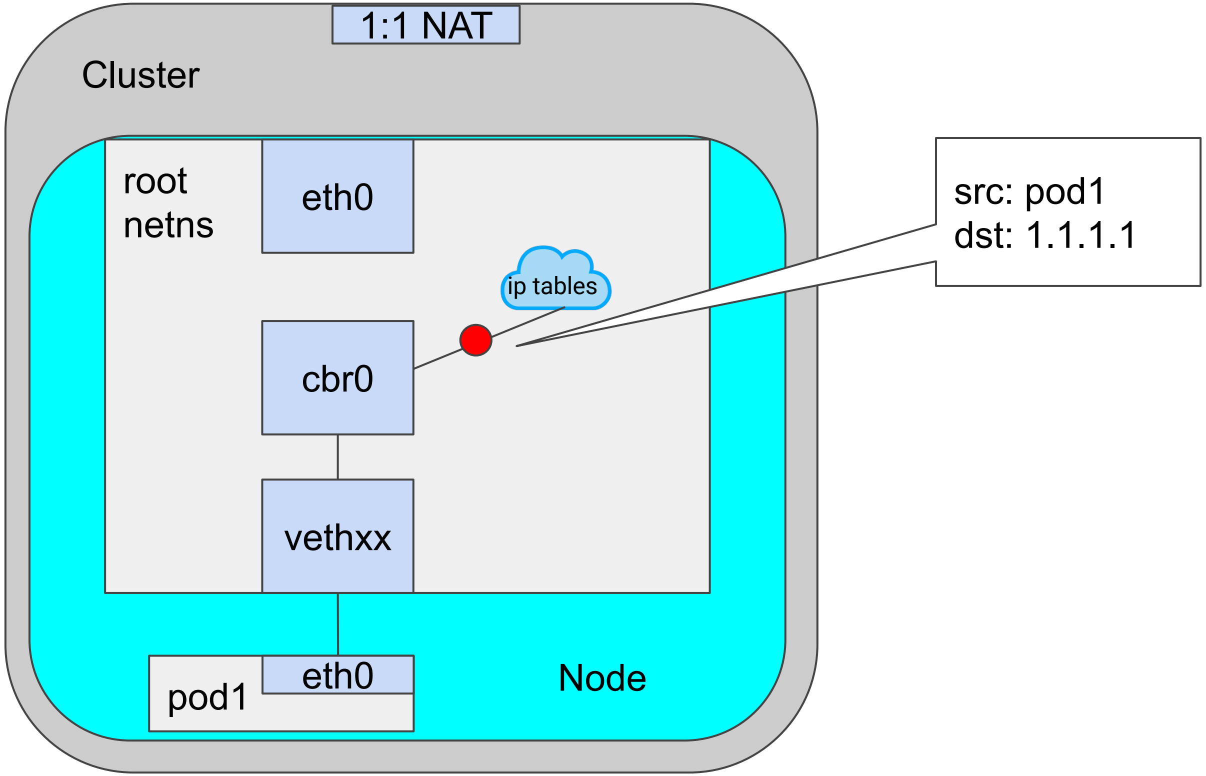


Life of packet: pod-to-internet

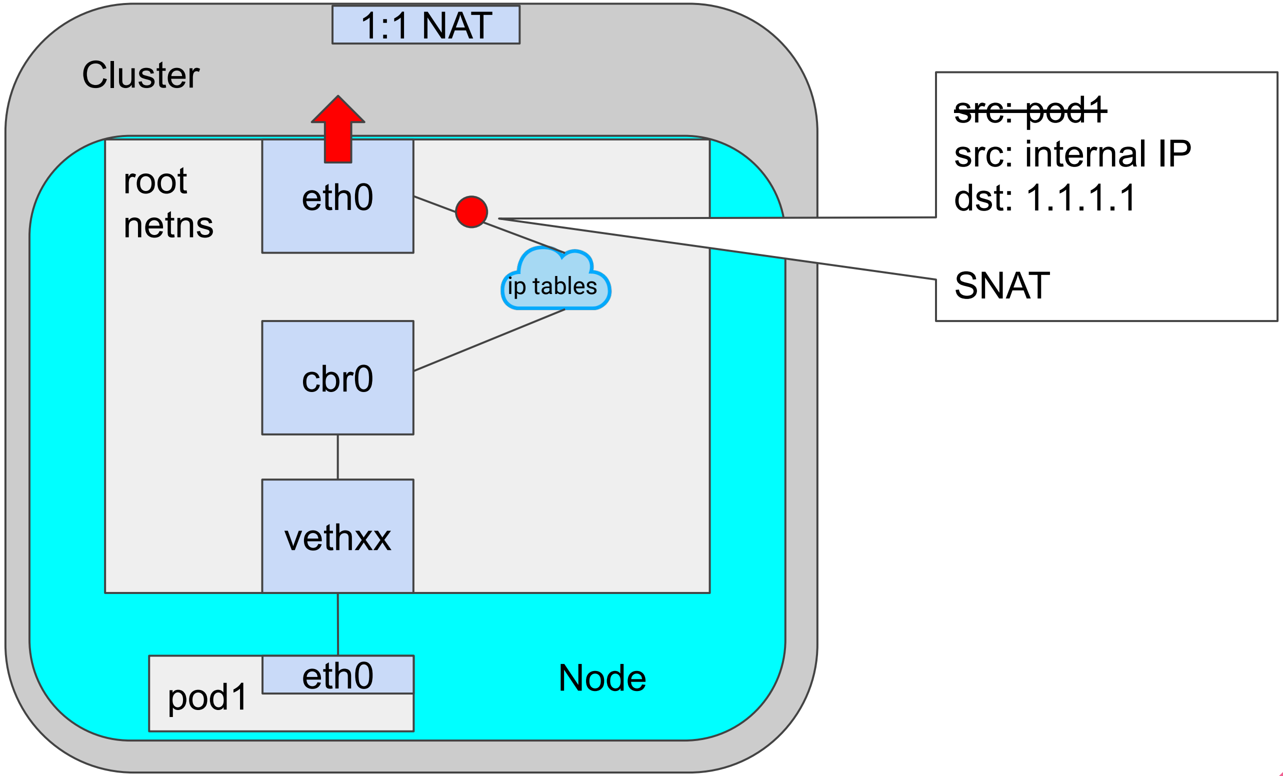


src: pod1
dst: 1.1.1.1

Life of packet: pod-to-internet



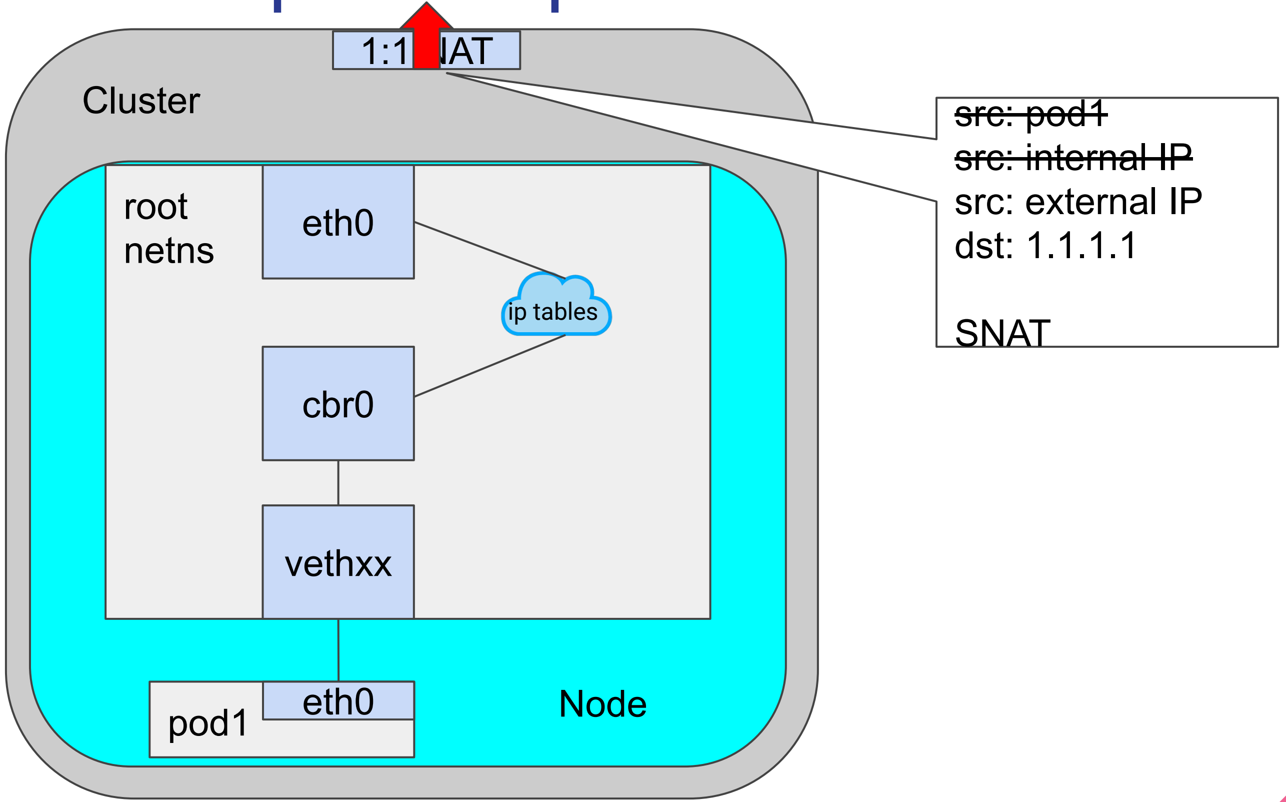
Life of packet: pod-to-internet



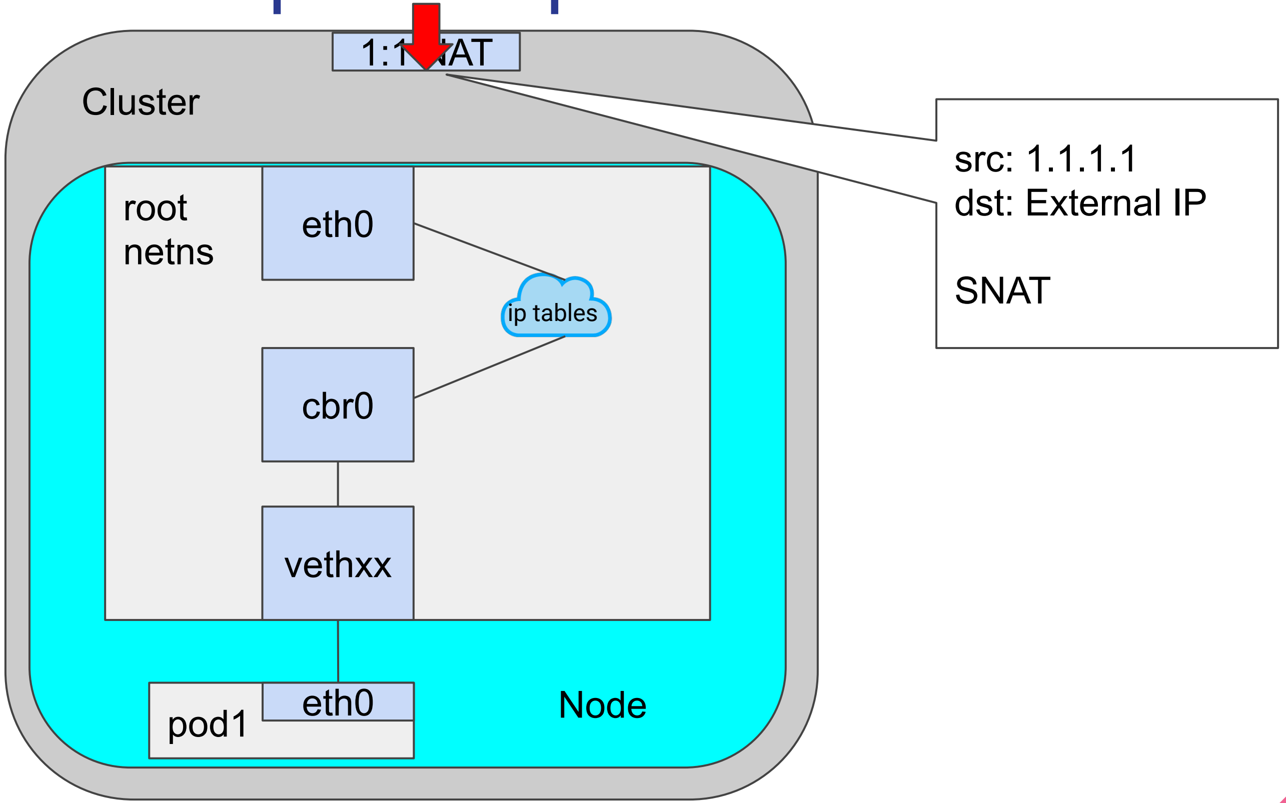
~~src: pod1~~
src: internal IP
dst: 1.1.1.1

SNAT

Life of packet: pod-to-internet



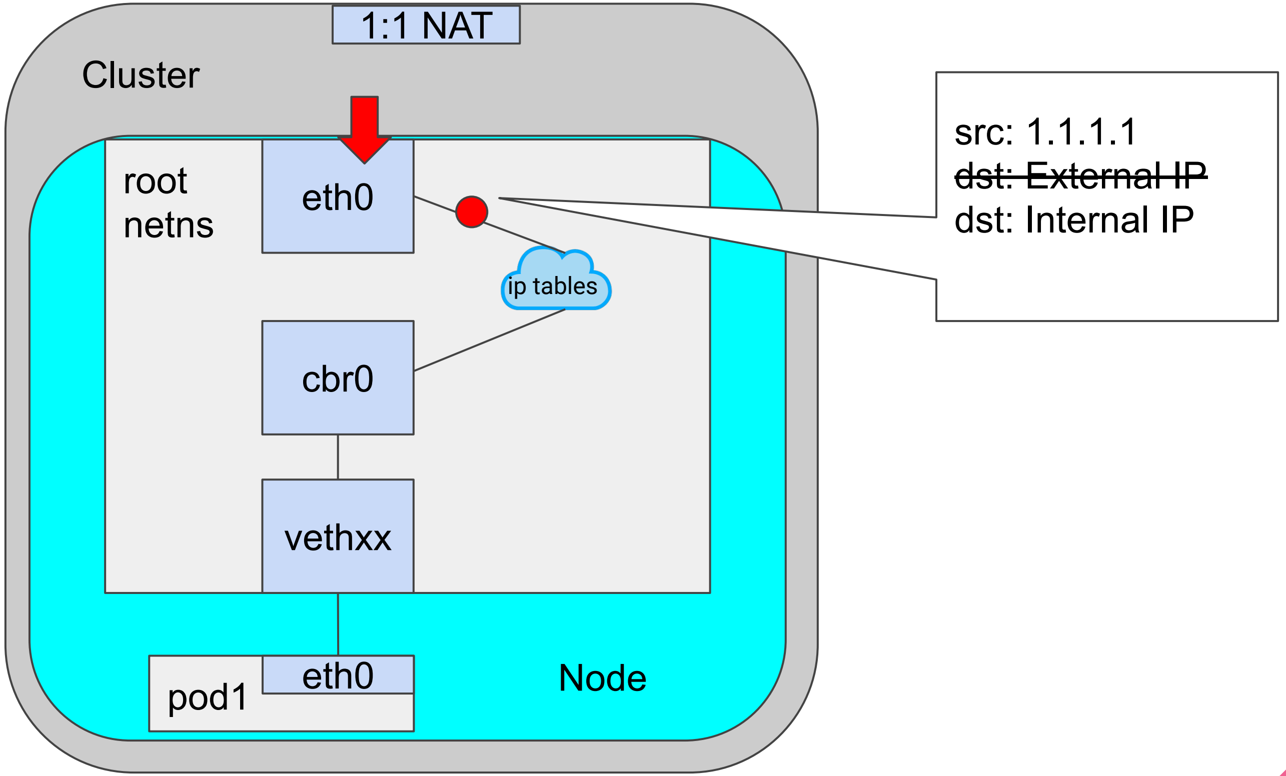
Life of packet: pod-to-internet



src: 1.1.1.1
dst: External IP

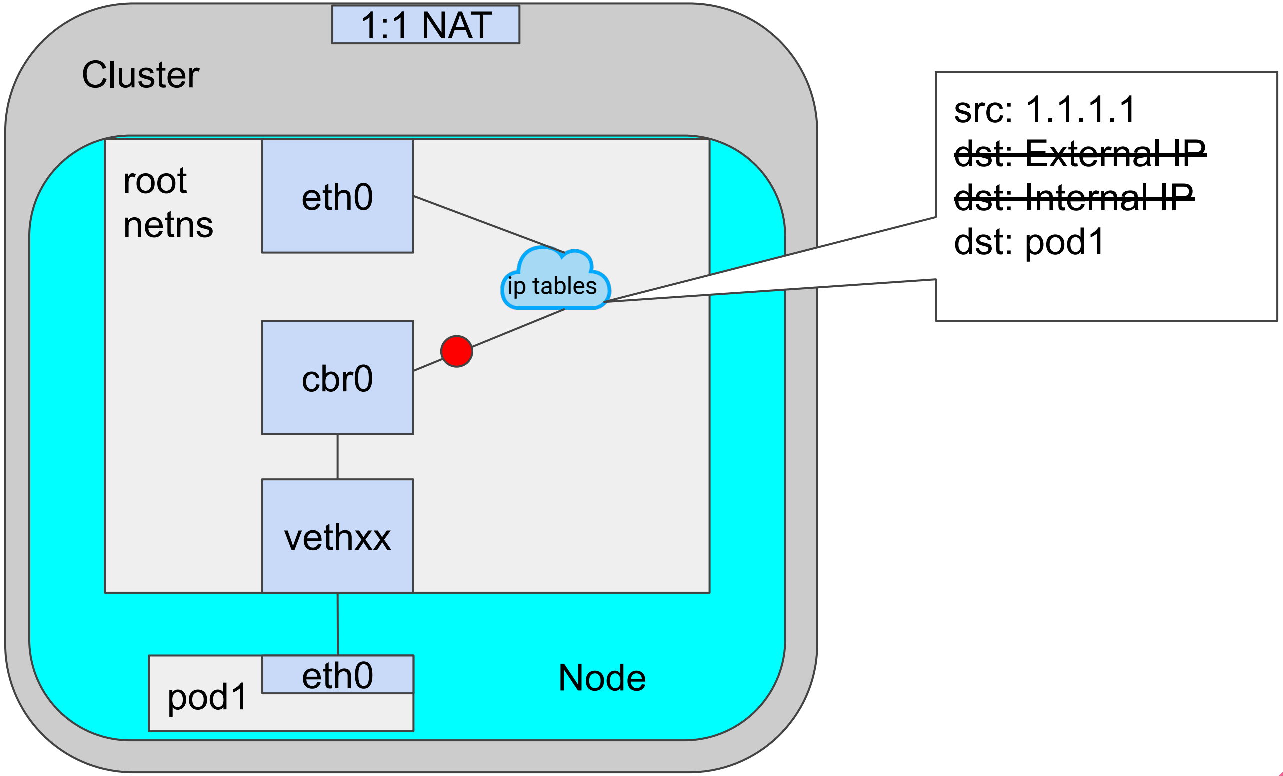
SNAT

Life of packet: pod-to-internet



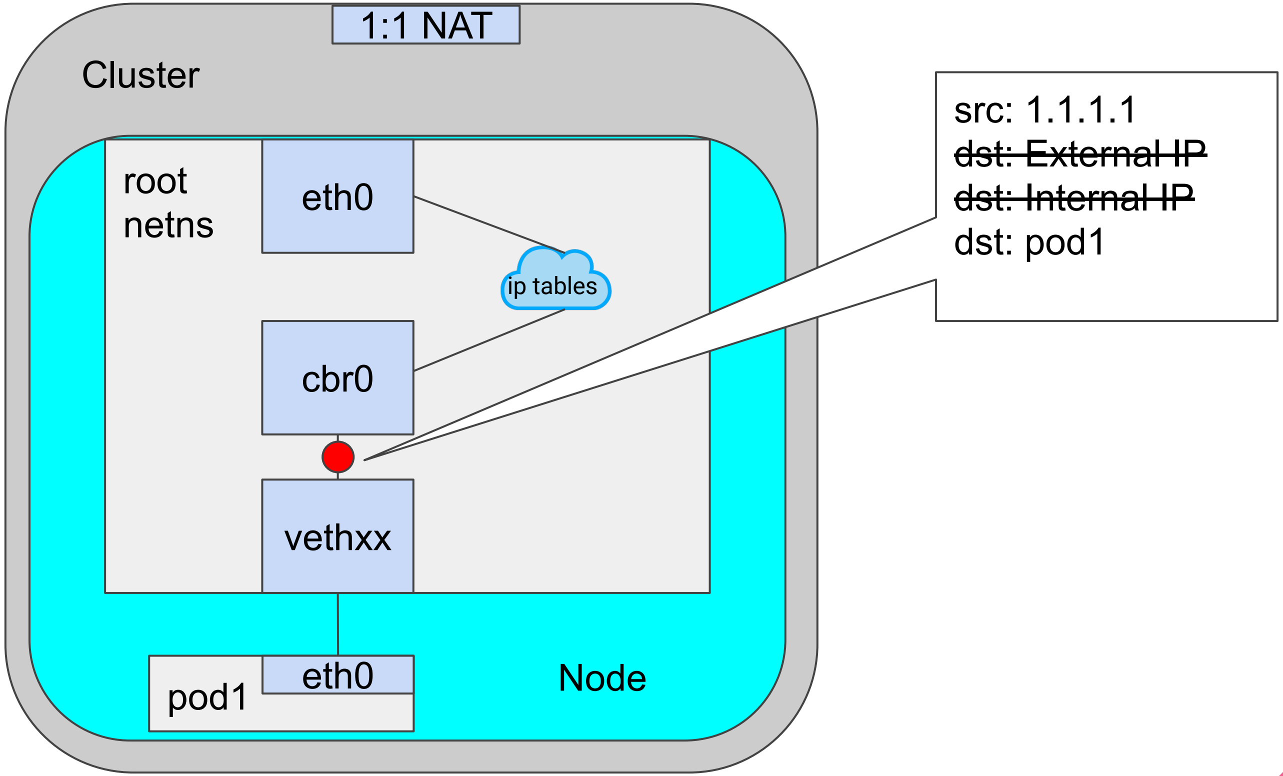
src: 1.1.1.1
~~dst: External IP~~
dst: Internal IP

Life of packet: pod-to-internet



src: 1.1.1.1
~~dst: External IP~~
~~dst: Internal IP~~
dst: pod1

Life of packet: pod-to-internet



src: 1.1.1.1
~~dst: External IP~~
~~dst: Internal IP~~
dst: pod1

Receive from External Traffic(Ingress)

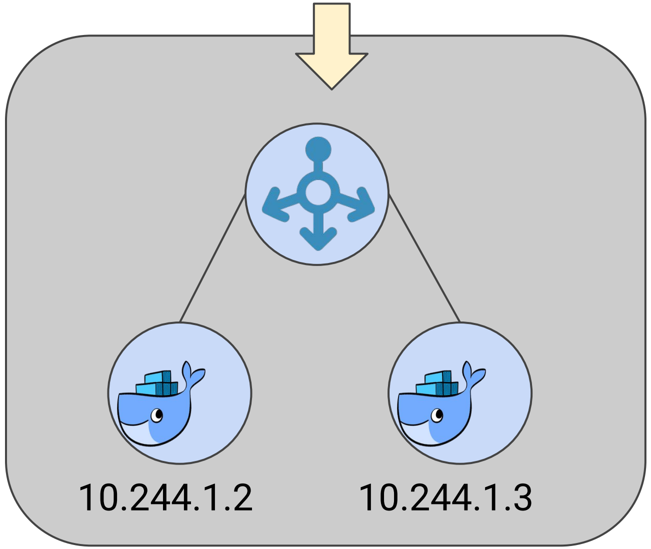
k8s support

L4: TCP

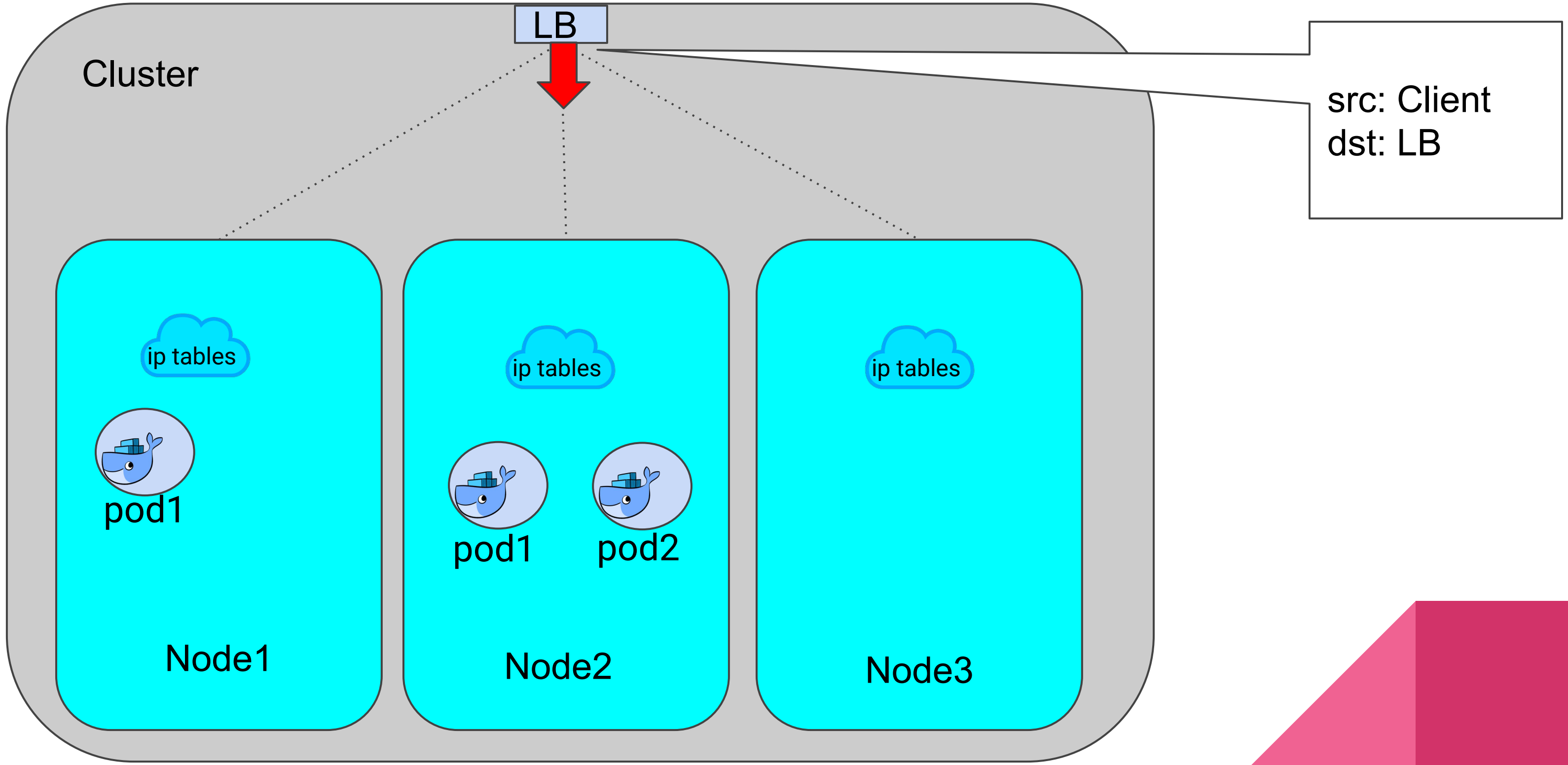
L7: HTTP/S (GCP)

mapped to

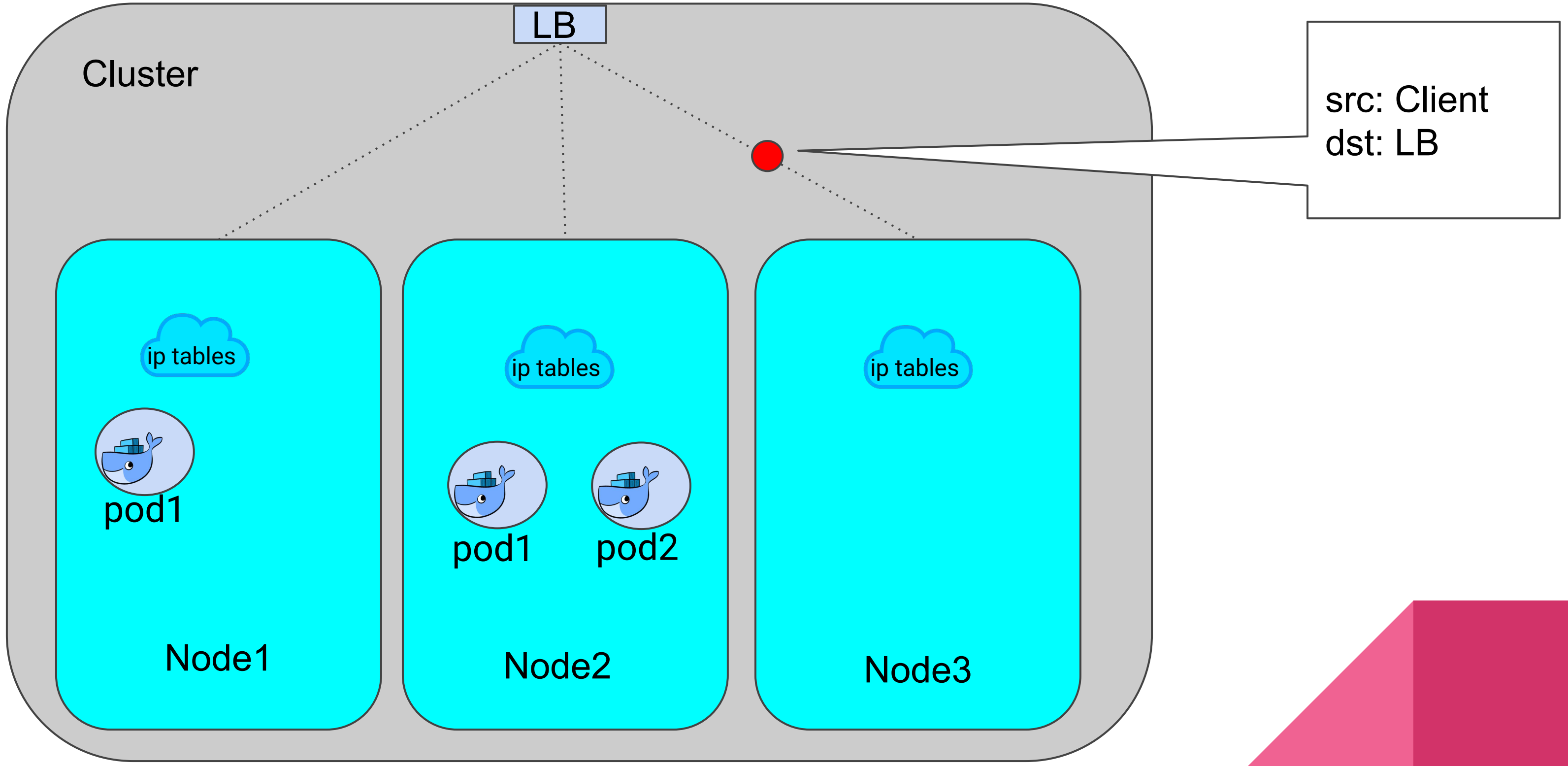
- Service type=LoadBalancer
- Ingress



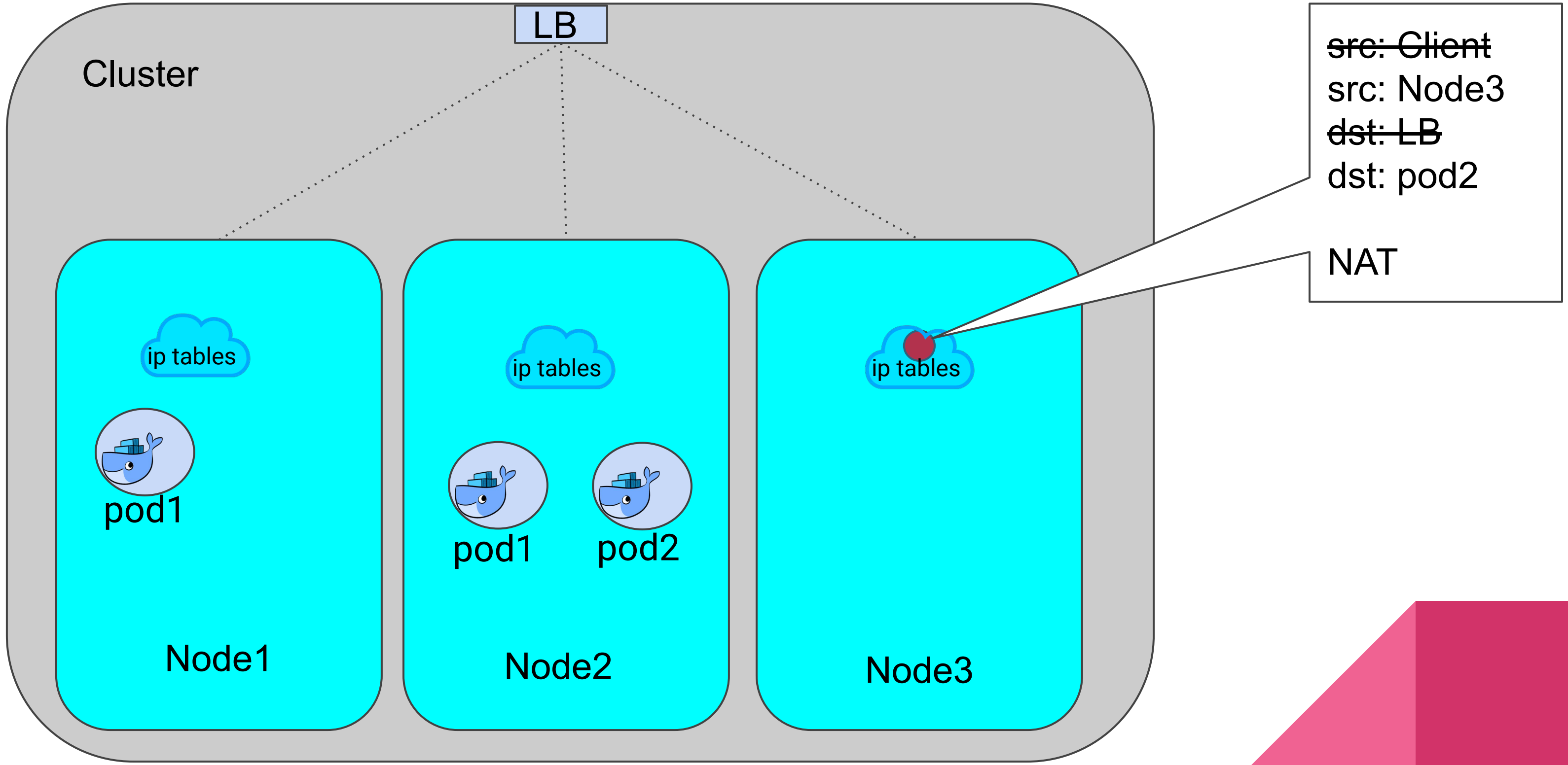
Life of packet: external to service(L4 case)



Life of packet: external to service(L4 case)



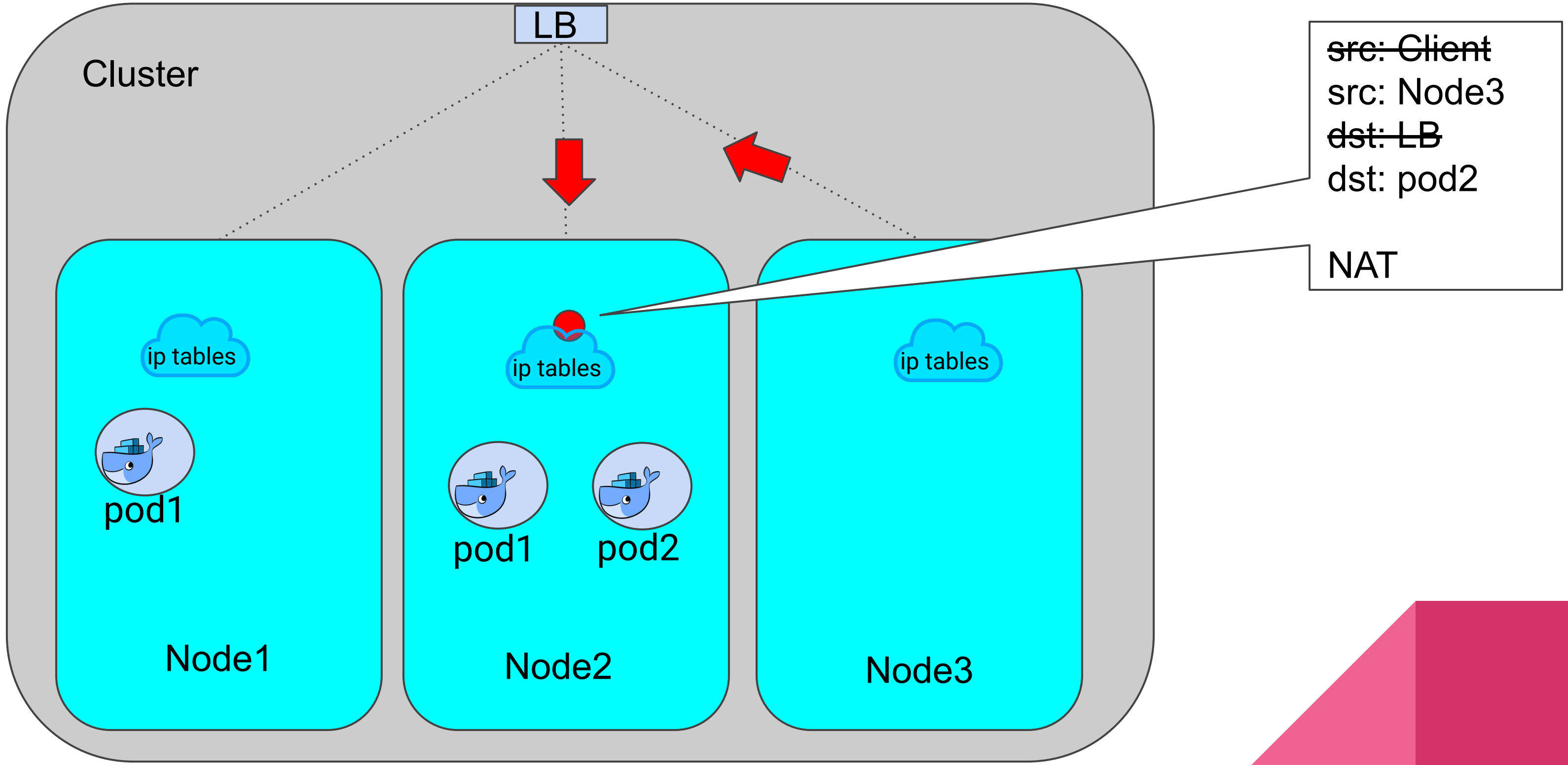
Life of packet: external to service(L4 case)



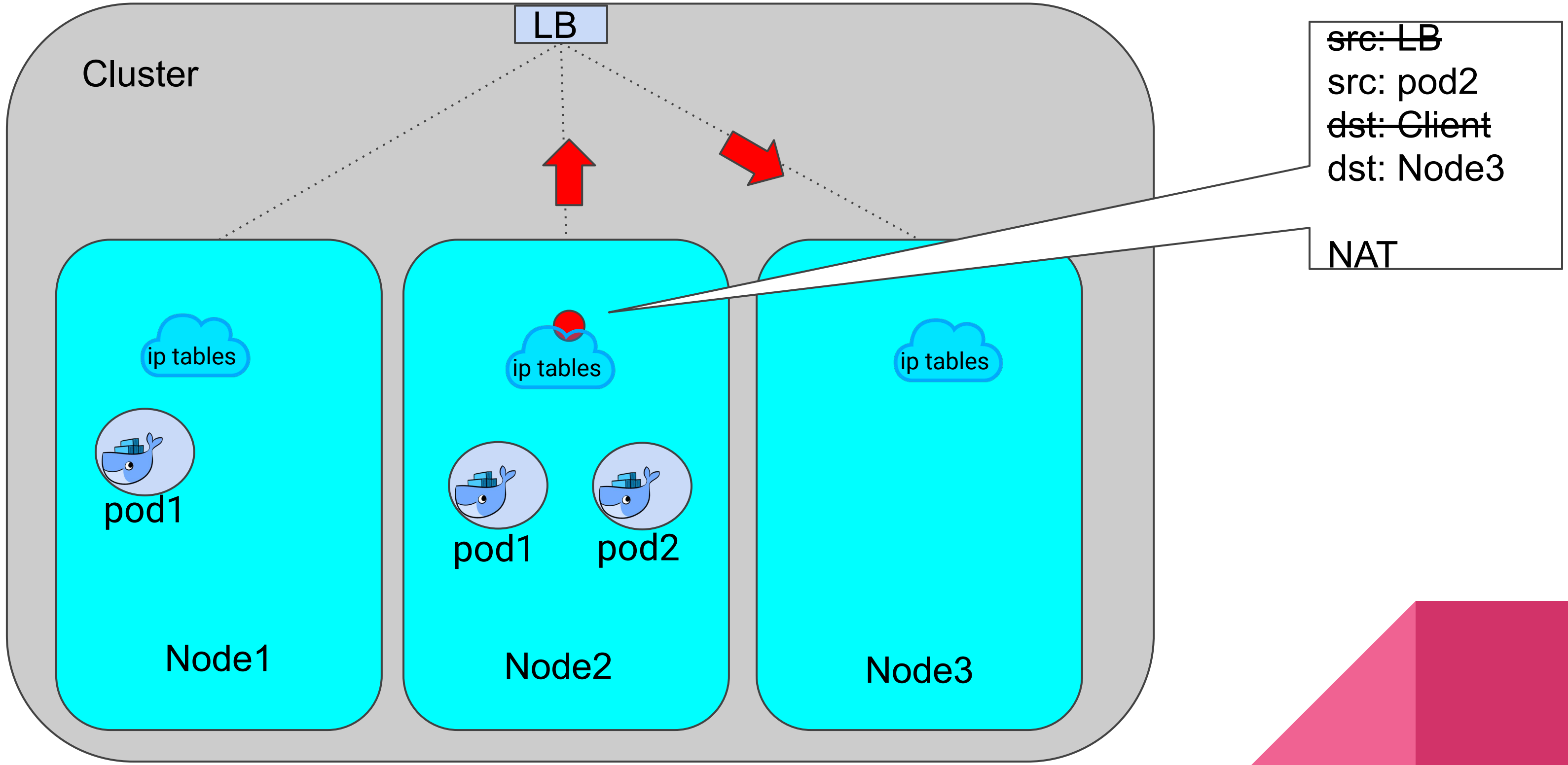
~~src: Client~~
src: Node3
~~dst: LB~~
dst: pod2

NAT

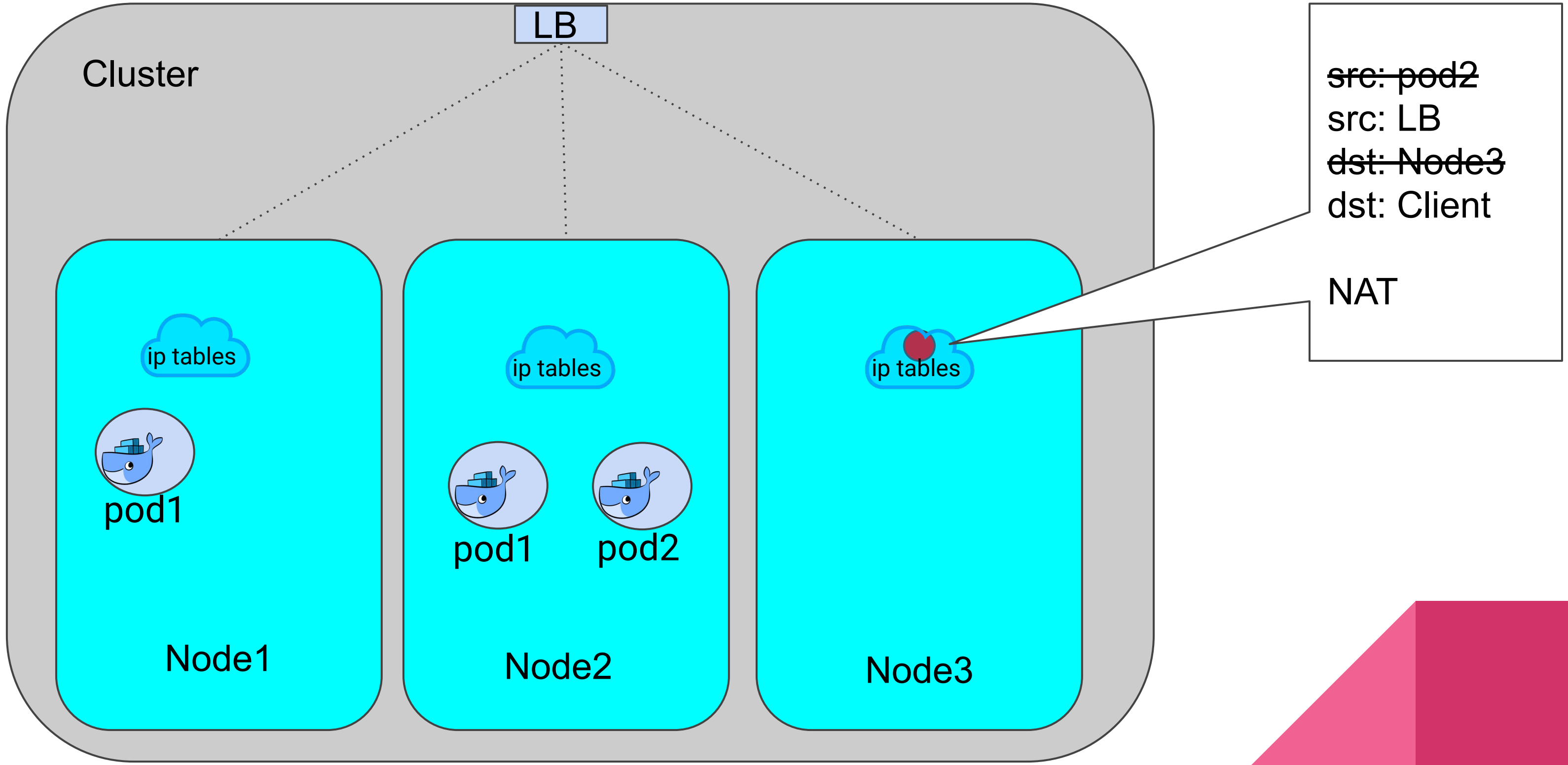
Life of packet: external to service(L4 case)



Life of packet: external to service(L4 case)



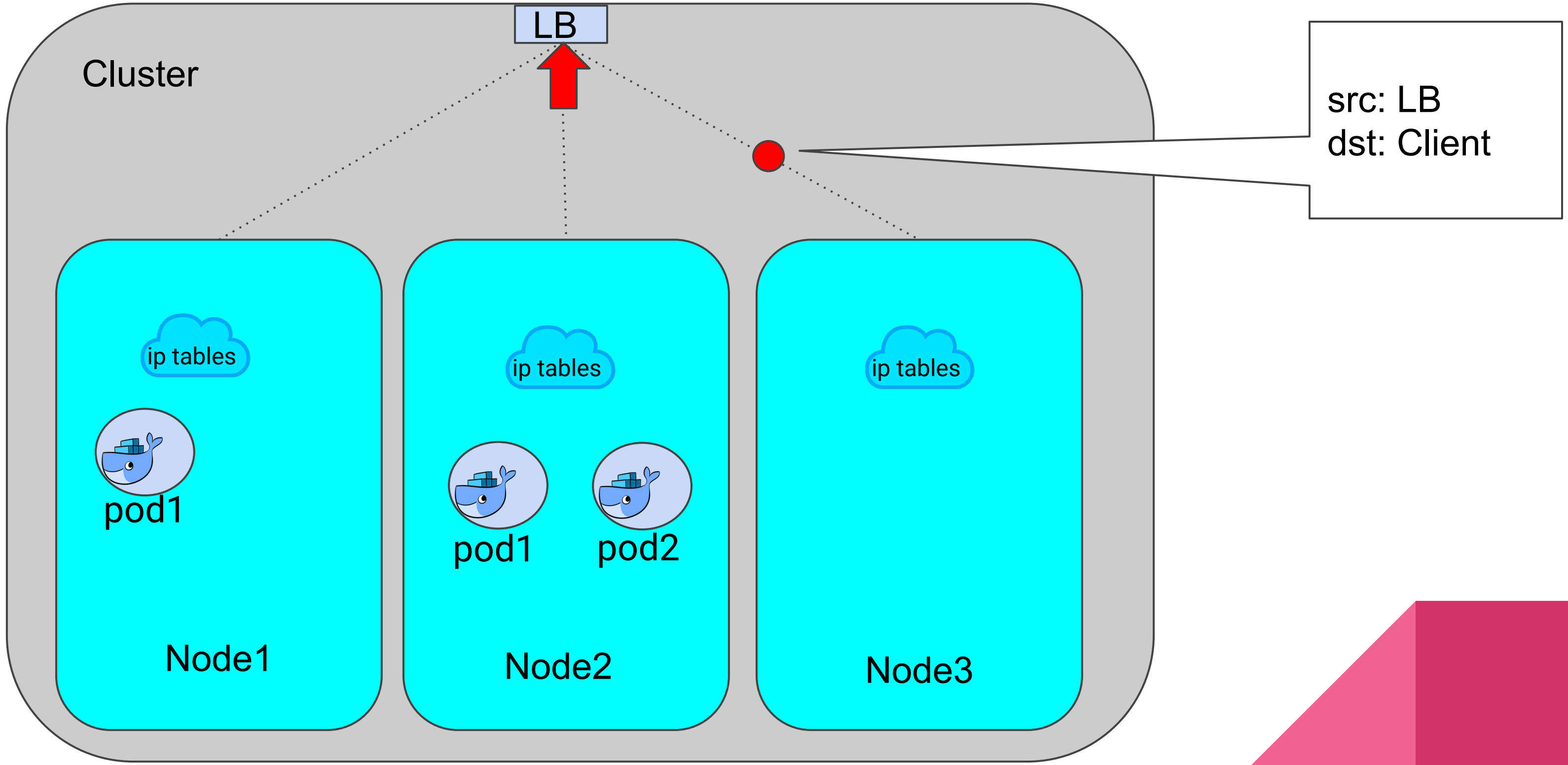
Life of packet: external to service(L4 case)



~~src: pod2~~
src: LB
~~dst: Node3~~
dst: Client

NAT

Life of packet: external to service(L4 case)



Network Policy

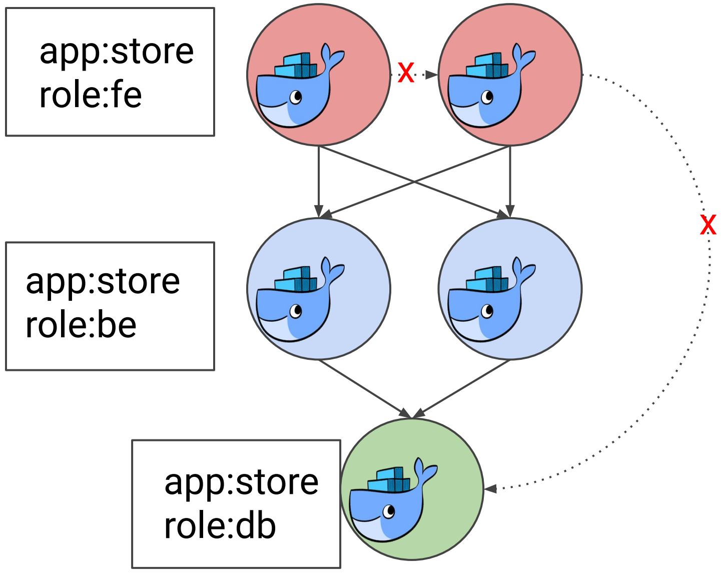
Network Policy

like “firewall”, regulate inbound and outbond for pods

grouped by pods selector

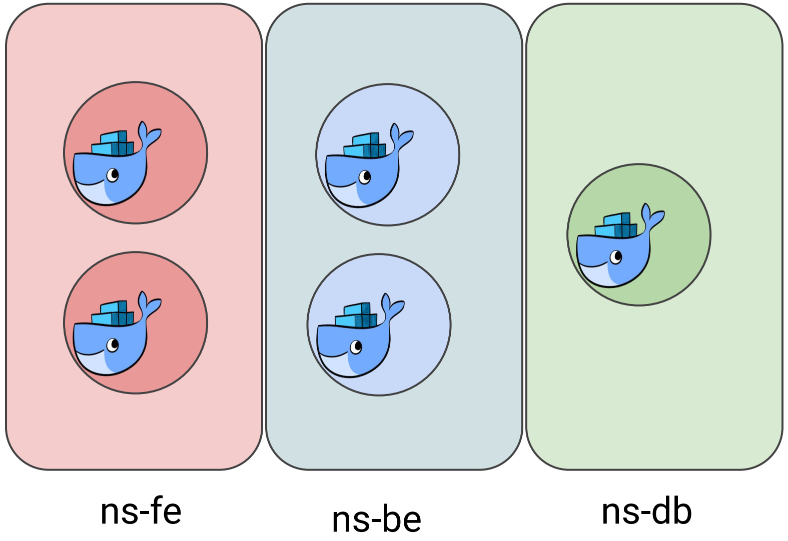
default: allow all

policies can be stacked. order is important



Namespace

private area to manage object(pods, service, policies)



Network Policy

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

```yaml
ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
egress:
  - to:
    - ipBlock:
        cidr: 10.0.0.0/24
    ports:
    - protocol: TCP
      port: 597
```

need to specify

- subject pods
- rule ingress/egress
- open port

Custom Network Policy

- CNI that support additional feature in Network Policy
  - calico
  - romana
  - weavenet
  - openshift
  - .....

Thats All...

Pahamify is Hiring..

pahamify.com/career

or

Contact:
career@pahamify.com

JS