



ons

NORTH AMERICA

OPEN NETWORKING //  
Enabling Collaborative  
Development & Innovation

# Packet Walk(s) In Kubernetes

Don Jayakody, Big Switch Networks

 @techjaya

 [linkedin.com/in/jayakody](https://linkedin.com/in/jayakody)

Hosted By

 THE LINUX FOUNDATION |  OLF NETWORKING



ONS  
NORTH AMERICA  
OPEN NETWORKING //  
Enabling Collaborative  
Development & Innovation

## Find out more:

- Video walkthroughs of this presentation will be available:
  - <https://www.youtube.com/user/BigSwitchNetworks>
- Repo for all the command outputs/PDF slides:  
<https://github.com/jayakody/ons-2019>



# About

## About Big Switch

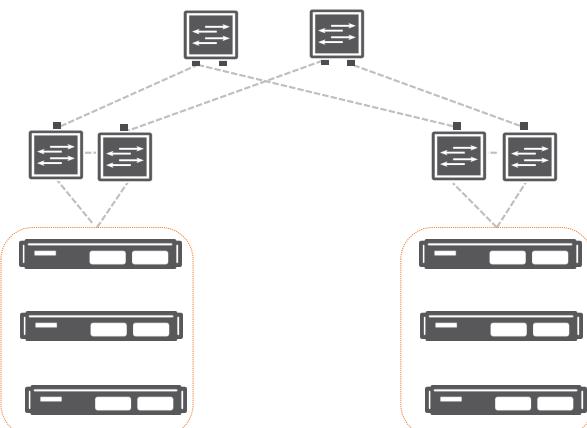
- We are in the business of “Abstracting Networks” (As one Big Switch) using Open Networking Hardware

## About Me

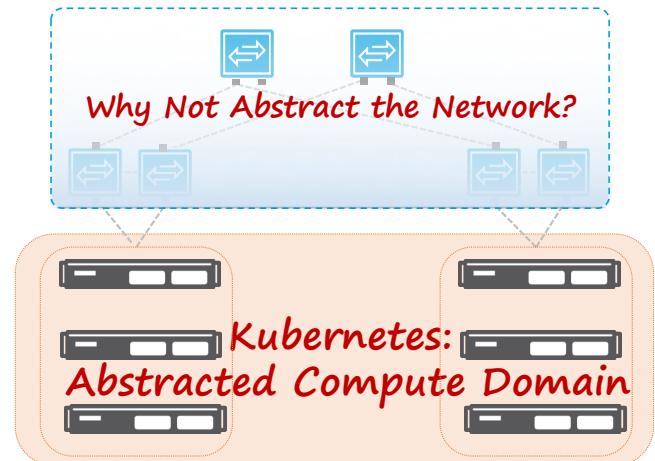
- Spent 4 years in Engineering building our products. Now in Technical Product Management



Legacy



Next-Gen



<http://www.youtube.com/watch?v=HIAxP0-M6SY&t=0m43s>

Hosted By

THE LINUX FOUNDATION | OLF NETWORKING



# Agenda

## Intro: K8S Networking

- Namespace/Pods/CNI?
- What's that "Pause" Container really do?
- Flannel: Intro / Packet Flows
- Exposing Services

## Calico: Networking

- Architecture
- IP-IP Mode (Route formation / Pod-to-pod communication / ARP Resolution/ Packet Flow)
- BGP Mode (Peering requirements/ Packet Flow)

## Cilium: Networking

- Architecture
- Overlay Network Mode (Configuration/ Pod-to-pod communication/ Datapath / ARP Resolution/ Packet Flow)
- Direct Routing Mode



Hosted By

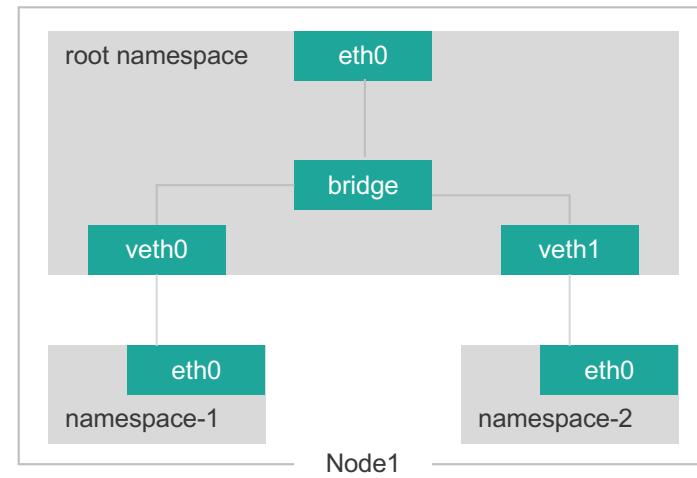
THE LINUX FOUNDATION | OFL NETWORKING



# K8S Networking: Basics

## Namespaces

- Linux kernel has 6 types of namespaces: pid, **net**, mnt, uts, ipc, user
- Network namespaces provide a brand-new network stack for all the processes within the namespace
- That includes **network interfaces**, **routing tables** and **iptables rules**





# K8S Networking: Basics

## Pods

- Lowest common denominator in K8S. Pod is comprised of one or more containers along with a “pause” container
- Pause container act as the “parent” container for other containers inside the pod. One of it’s primary responsibilities is to bring up the network namespace
- Great for the redundancy: Termination of other containers do not result in termination of the network namespace

```
[root@Master1 ~]# kubectl get pods --namespace web -o wide
NAME          READY   STATUS    AGE   IP           NODE
nginx-deployment-76bf4969df-5xzv7  1/1     Running  7m53s  172.31.155.17  Worker-1
```

```
[root@Worker-1 ~]# docker ps
CONTAINER ID   IMAGE          COMMAND
93490bfce728  docker.io/nginx@sha256  "nginx -q 'daemon off'"
2ef012ea5db0  k8s.gcr.io/pause:3.1   "/pause"
```





# K8S Networking: Basics

## Accessing Pod Namespaces

- Multiple ways to access pod namespaces
- 'kubectl exec --it'
- 'docker exec --it'
- nsenter ("namespace enter", let you run commands that are installed on the host but not on the container)

```
[root@worker-1 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND
5b54f2a44c3b        d8233ab899d4      "sleep 3600"
43e42c45522b        k8s.gcr.io/pause:3.1   "/pause"
CREATED              35 minutes ago
10 hours ago
NAMES                k8s_busybox_busybox0-6hc7c
k8s_POD_busybox0-6hc7c
```

```
[root@worker-1 ~]# docker inspect -f '{{.State.Pid}}' 21388
21388
[root@worker-1 ~]# nsenter -t 21388 -n ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qdisc 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
link/ether fa:4d:26:0b:4a:c7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 192.168.1.196/32 brd 192.168.1.196 scope global eth0
    valid_lft forever preferred_lft forever
inet6 fe80::f84d:26ff:fe0b:4ac7/64 scope link
    valid_lft forever preferred_lft forever
[root@worker-1 ~]#
```

```
[root@worker-1 ~]# docker inspect -f '{{.State.Pid}}' 8112
8112
[root@worker-1 ~]# nsenter -t 8112 -n ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qdisc 1000
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
17: eth0@if18: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
link/ether fa:4d:26:0b:4a:c7 brd ff:ff:ff:ff:ff:ff link-netnsid 0
inet 192.168.1.196/32 brd 192.168.1.196 scope global eth0
    valid_lft forever preferred_lft forever
inet6 fe80::f84d:26ff:fe0b:4ac7/64 scope link
    valid_lft forever preferred_lft forever
[root@worker-1 ~]#
```

Both containers belong to the same pod => Same Network Namespace => same 'ip a' output



Hosted By

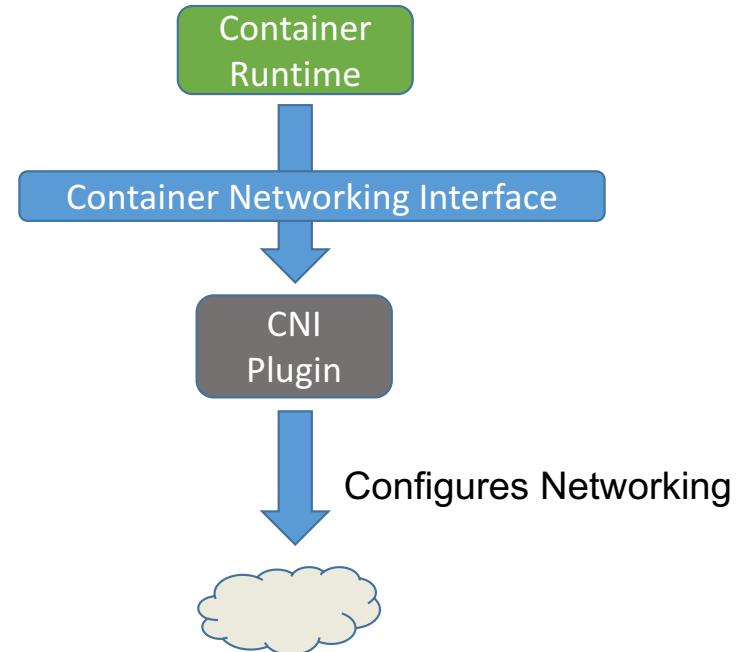
THE LINUX FOUNDATION | OLF NETWORKING



# K8S Networking: Basics

## Container Networking Interface : CNI

- Interface between container runtime and network implementation
- Network plugin implements the CNI spec. It takes a container runtime and configures (attach/detach) it to the network
- CNI plugin is an executable (in: /opt/cni/bin)
- When invoked it reads in a JSON config & Environment Variables to get all the required parameters to configure the container with the network



Credit: <https://www.slideshare.net/weaveworks/introduction-to-the-container-network-interface-cni>

Hosted By

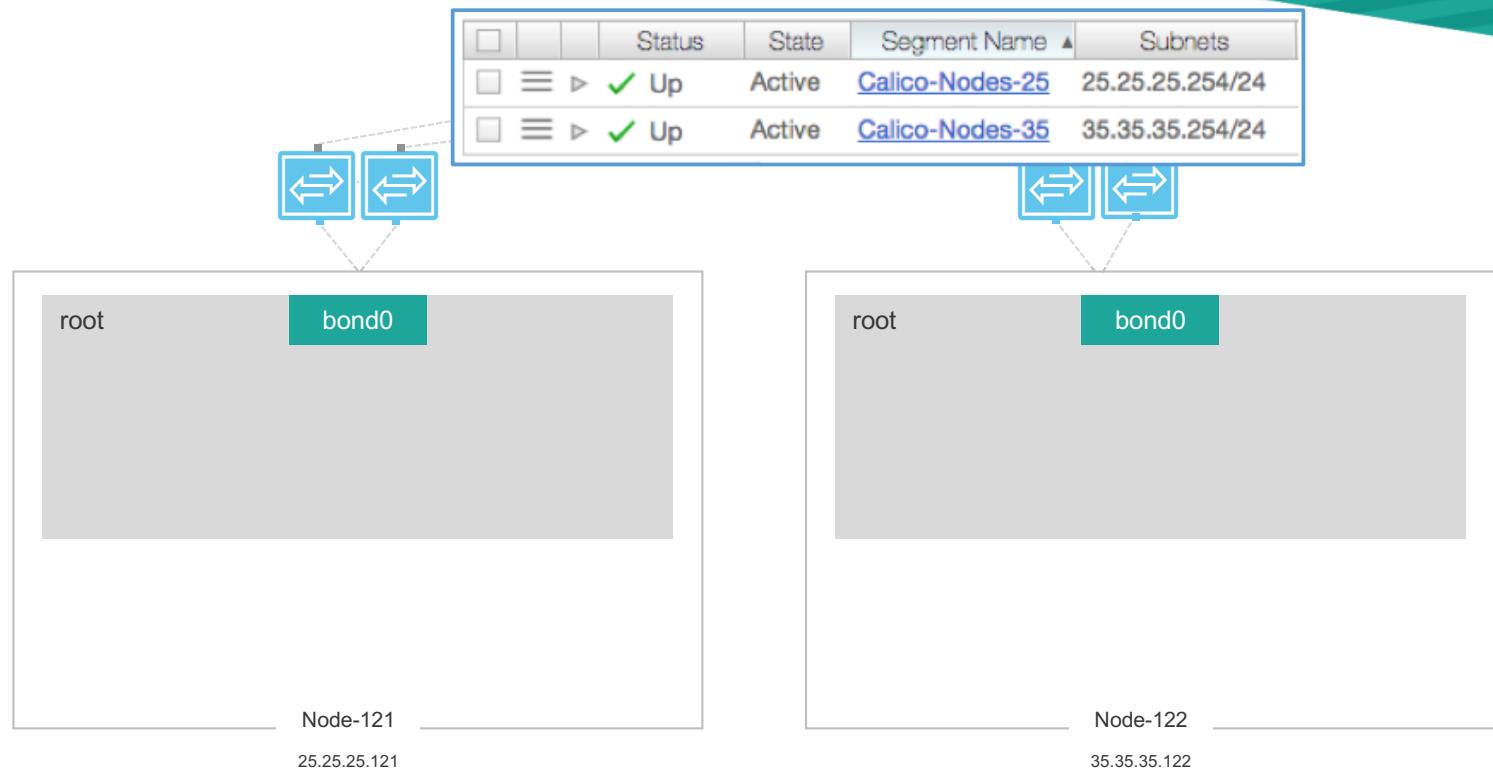
THE LINUX FOUNDATION | OLF NETWORKING



# Topology

## Topology Info

- Node IP belongs to 2 different subnets: 25.25.25.0/24 & 35.35.35.0/24
- Gateway is configured with .254 in the network for each network segment
- Bond0 interface is created by bonding two 10G interfaces

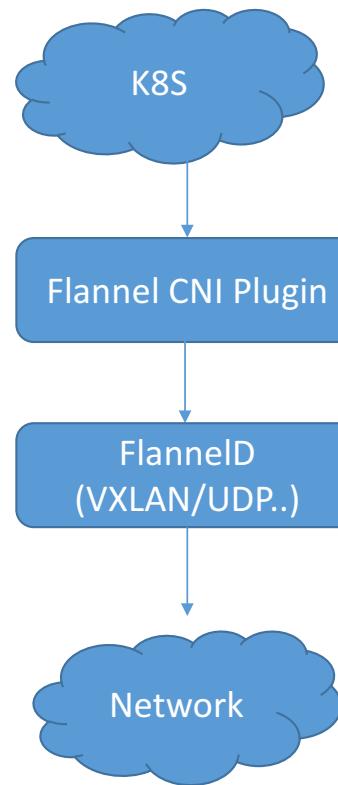




# Flannel

## Intro

- To make networking easier, Kubernetes does away with port-mapping and assigns a unique IP address to each pod
- If a host cannot get an entire subnet to itself things get pretty complicated
- Flannel aims to solve this problem by creating an overlay mesh network that provisions a subnet to each server



Hosted By

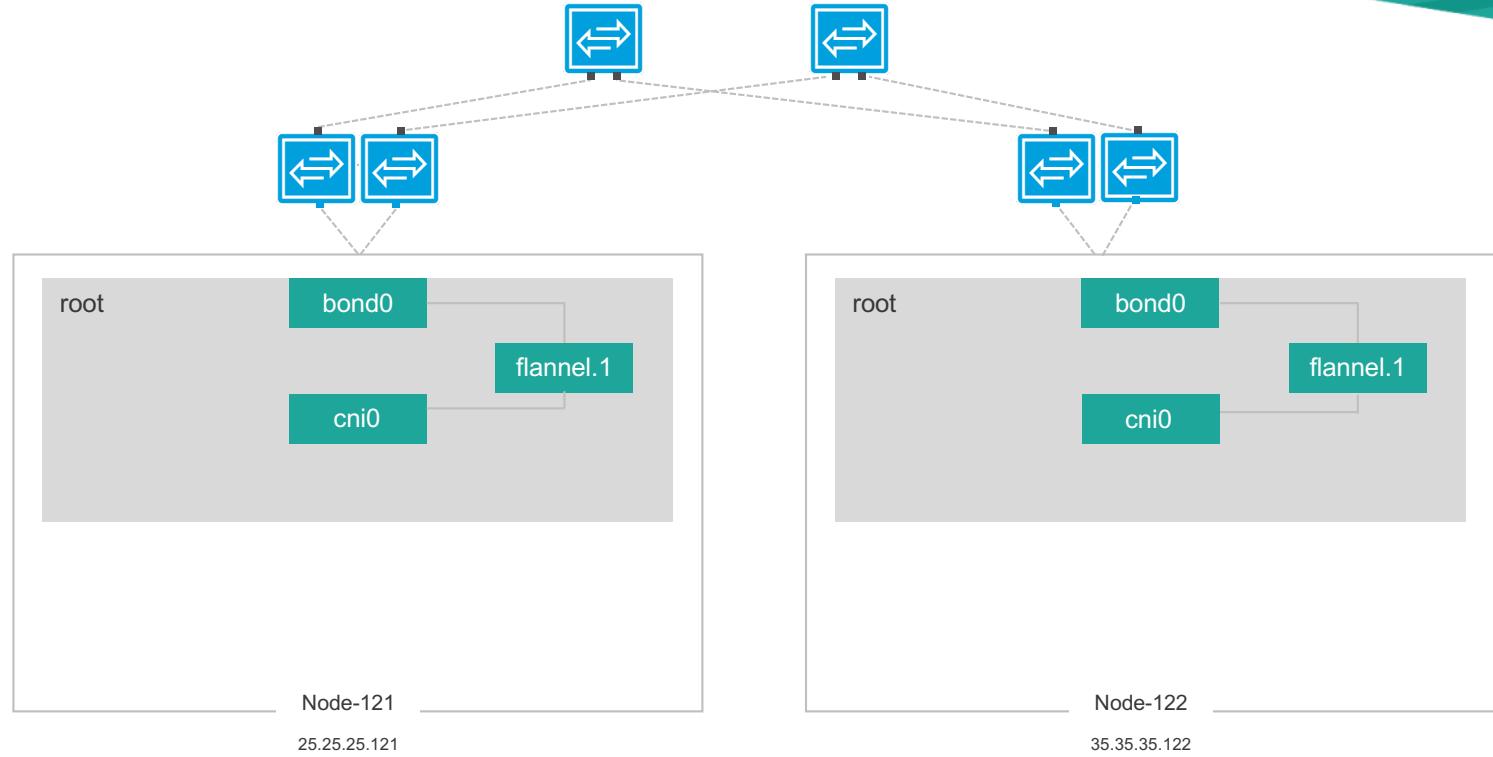
THE LINUX FOUNDATION | OLF NETWORKING



# Flannel

## Default Config

- “Flannel.1” Is the VXLAN interface
- CNI0 is the bridge



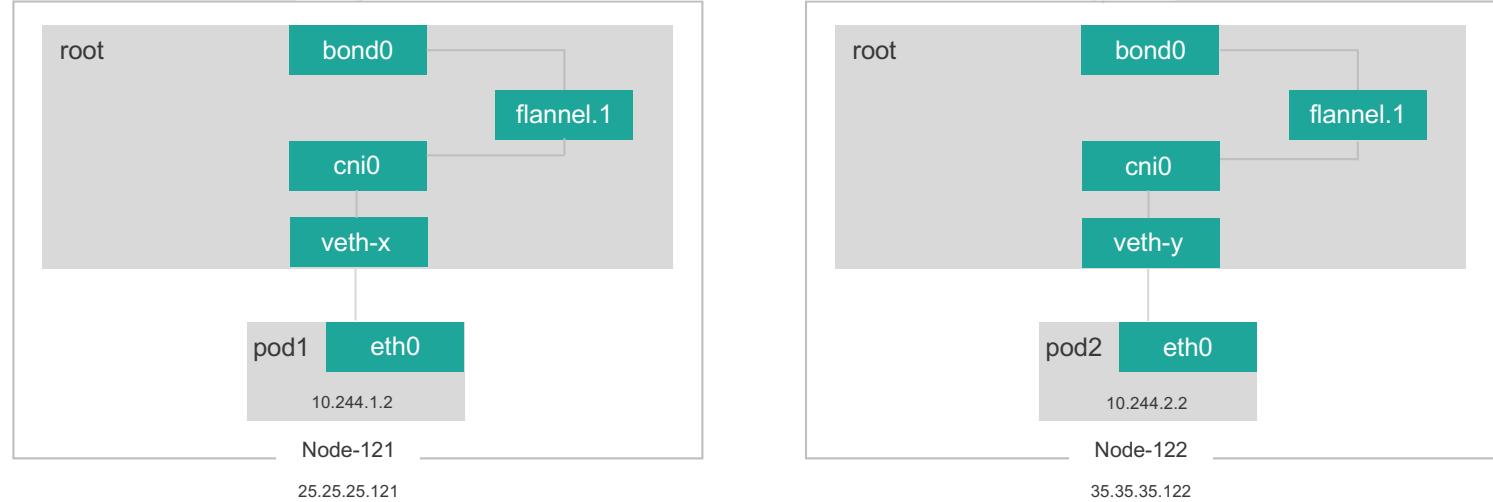


# Flannel

## Pod-to-Pod Communication

- Brought up 2 pods

```
$ kubectl get pods -n test -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE
busybox0-nzxgg 1/1     Running   10         10h   10.244.1.2   flannel-node-18121
busybox0-6qrqm 1/1     Running   9          10h   10.244.2.2   flannel-node-18122
```





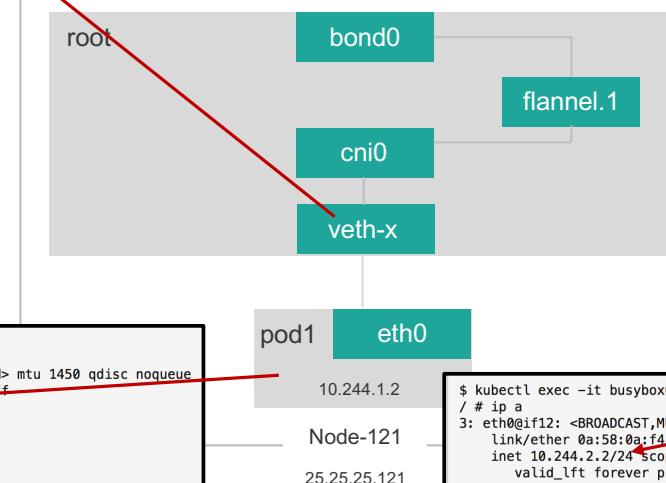
notice the index on veth.  
eg: "eth0@if12" on pod1 ns corresponds to 12<sup>th</sup> interface on the root ns

# Flannel

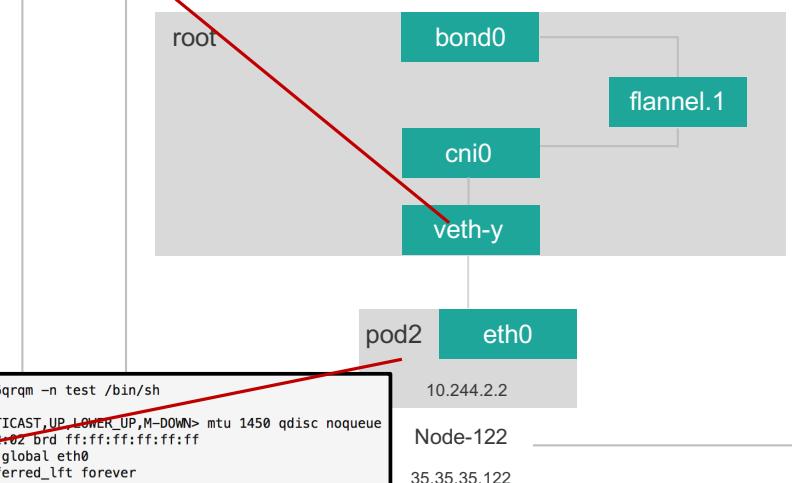
## Pod-to-Pod Communication

- VETH Interface with "veth-"  
got created on the root  
namespace
- Other end is attached to  
the pod namespace

```
$ kubectl exec -it busybox0-nzxgg -n test /bin/sh
/ # ip a
3: eth0@if12: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1450 qdisc noqueue
    link/ether 0a:58:0a:f4:01:02 brd ff:ff:ff:ff:ff:ff
    inet 10.244.1.2/24 brd global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::4c4e:5bff:fe16:dbf9/64 scope link
            valid_lft forever preferred_lft forever
/ #
```



```
[root@flannel-node-18122 ~]# ip a
11: cni0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UP qlen 1000
    link/ether 0a:58:0a:f4:02:01 brd ff:ff:ff:ff:ff:ff
    inet 10.244.2.1/24 brd global cni0
        valid_lft forever preferred_lft forever
        inet6 fe80::3c3b:29ff:fe29:286d/64 scope link
            valid_lft forever preferred_lft forever
12: veth047ac7bb0@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue master cni0 state UP
    link/ether 06:8c:15:09:9f:63 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::48c:15ff:fe09:9f63/64 scope link
        valid_lft forever preferred_lft forever
```



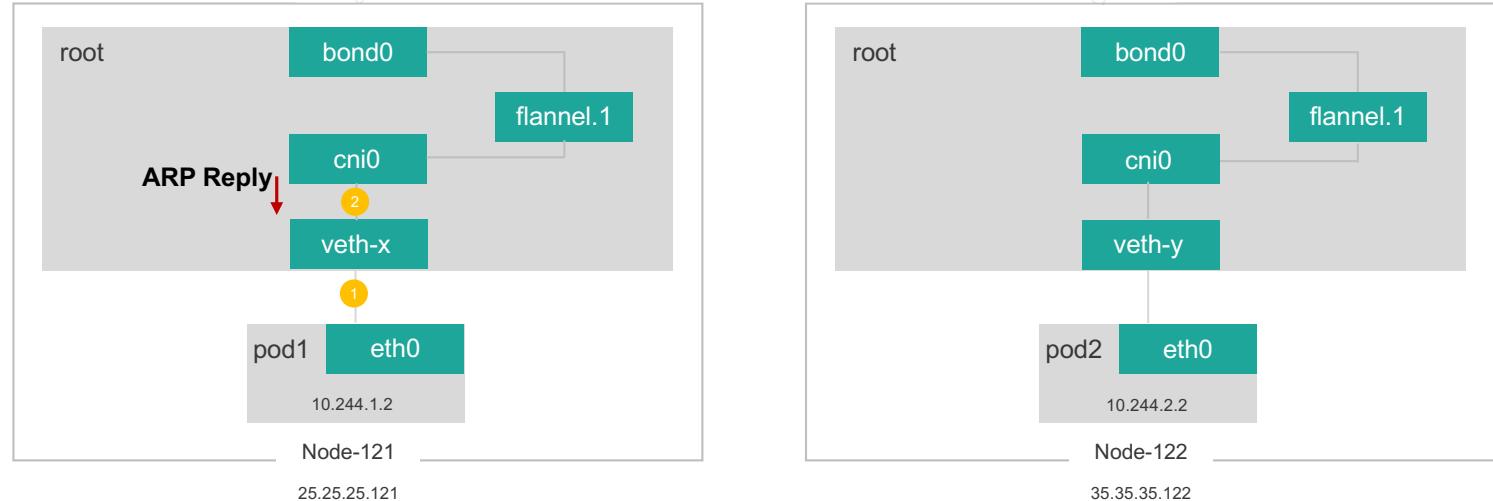


# Flannel

## ARP Handling

- 'cni0' bridge is replying to ARP requests from the pod

```
$ kubectl get pods -n test -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE
busybox0-nzxgg 1/1     Running   10         10h   10.244.1.2   flannel-node-18121
busybox0-6qrqm 1/1     Running   9          10h   10.244.2.2   flannel-node-18122
```

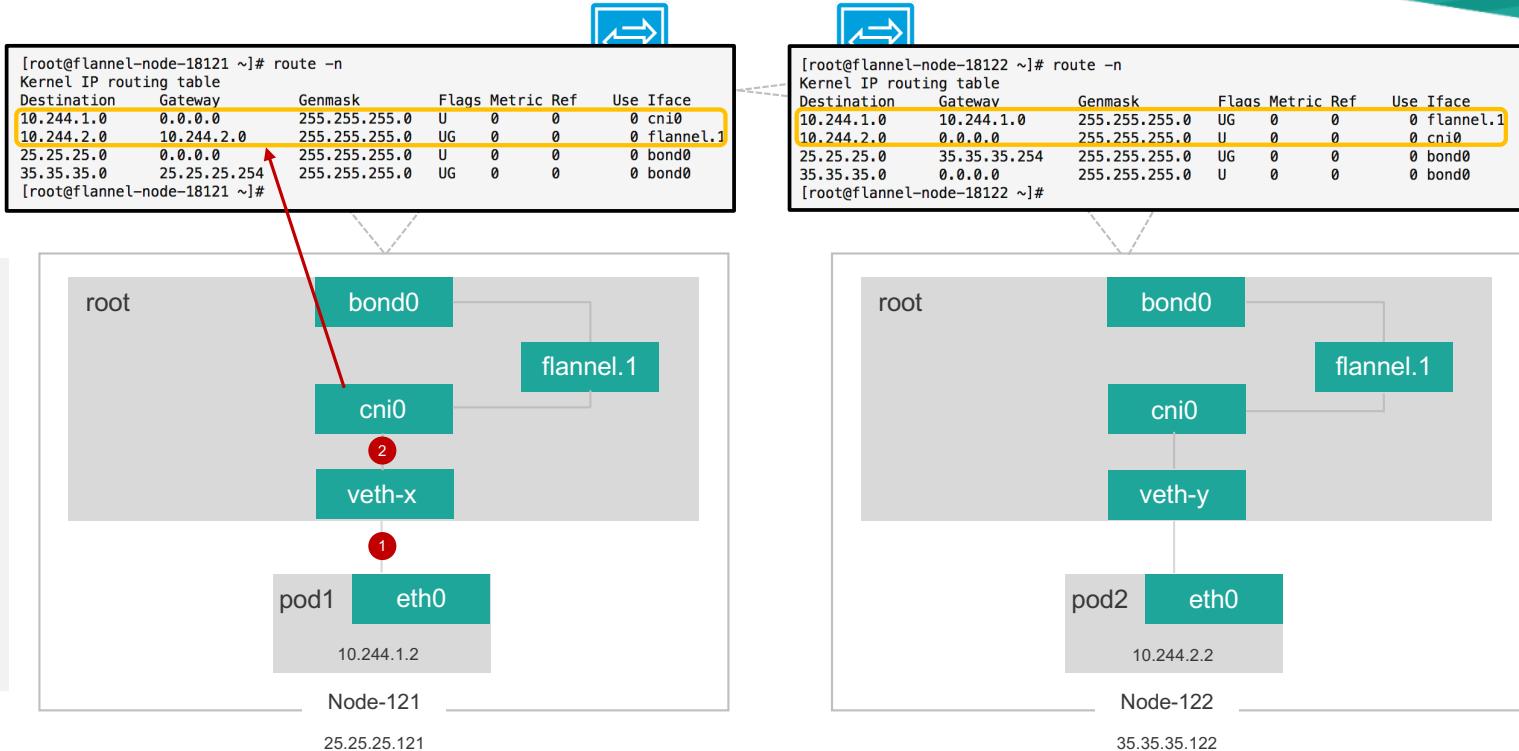




# Flannel

## Packet Flow

- Routing table lookup to figure out where to send the packet

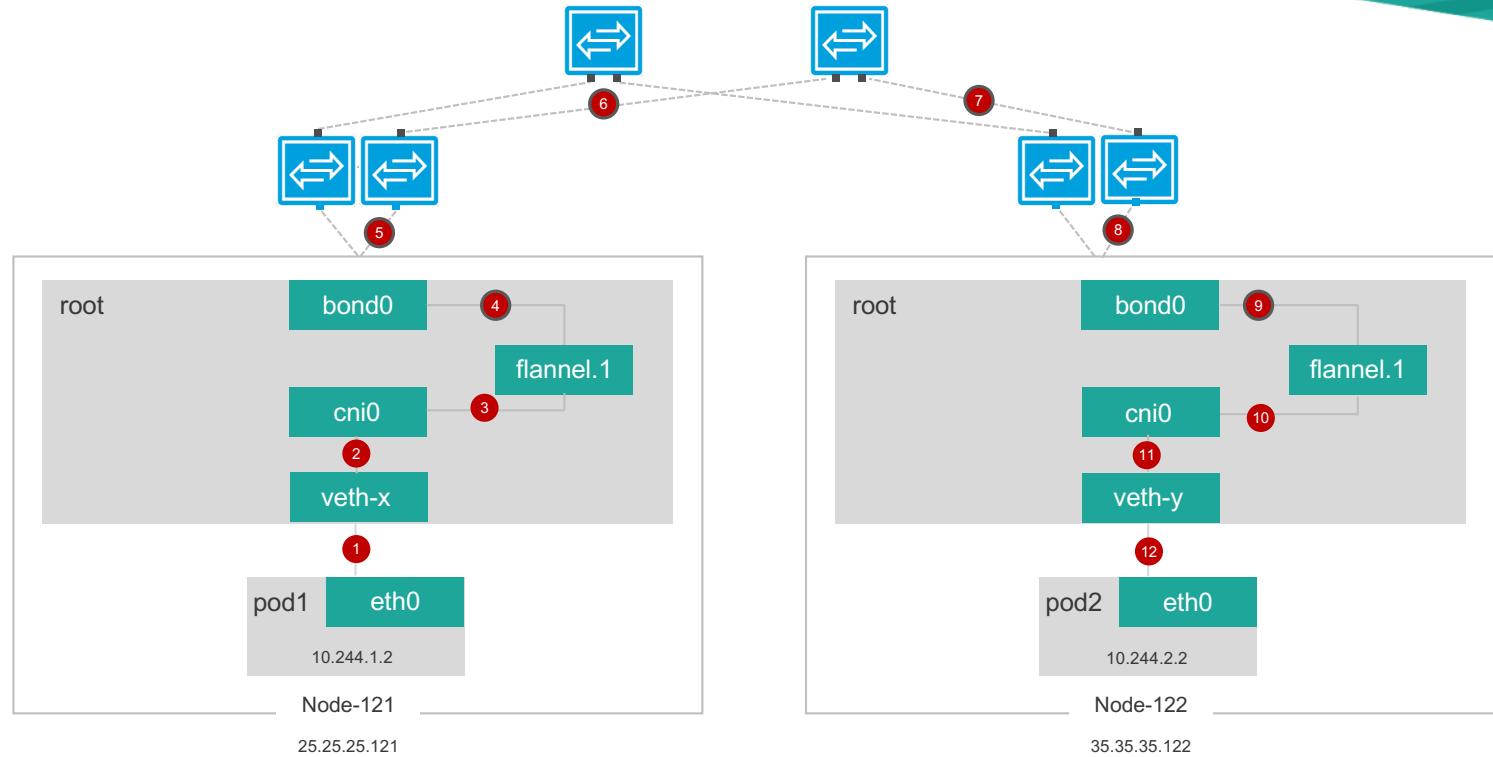




# Flannel

## Packet Flow

- Flannel.1 device does VXLAN encap/decap
- Traffic from pod1 is going through “flannel.1” device before exiting





# Calico

## Felix

- The primary Calico agent that runs on each machine that hosts endpoints.
- Responsible for programming routes and ACLs, and anything else required on the host

## Bird

- BGP Client: responsible of route distribution
- When Felix inserts routes into the Linux kernel FIB, Bird will pick them up and distribute them to the other nodes in the deployment



Hosted By

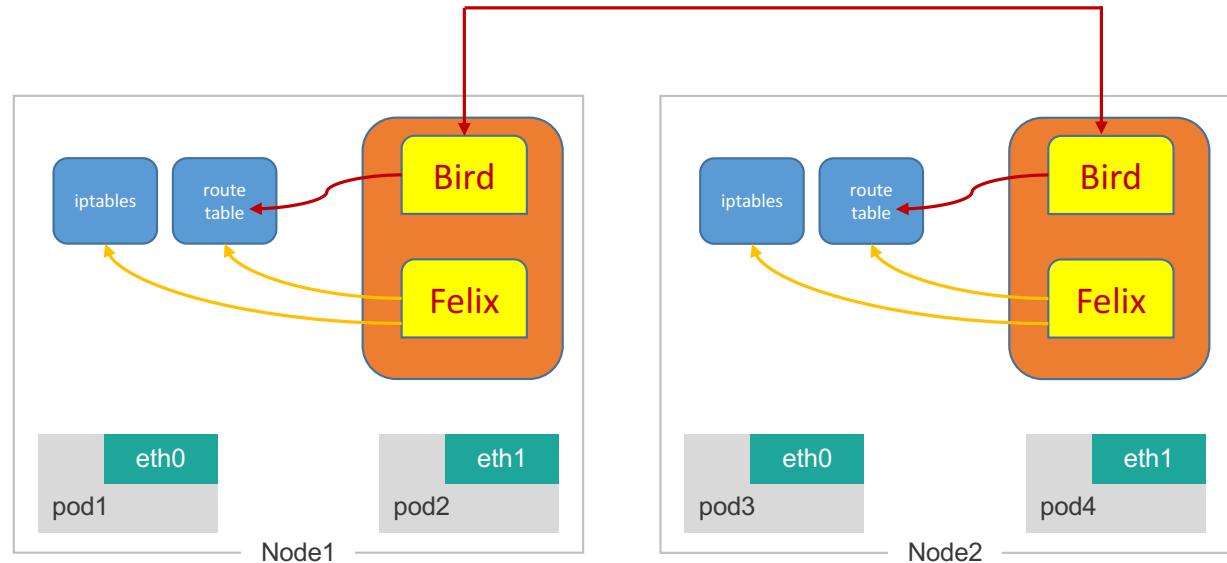
 THE LINUX FOUNDATION |  OFL NETWORKING



# Calico

## Architecture

- Felix's primary responsibility is to program the host's iptables and routes to provide the connectivity to pods on that host.
- Bird is a BGP agent for Linux that is used to exchange routing information between the hosts. The routes that are programmed by Felix are picked up by bird and distributed among the cluster hosts



\*etcd/confd components are not shown for clarity

Hosted By

THE LINUX FOUNDATION | OLF NETWORKING



# Calico

## Default Configuration

- Node-to-node mesh
- IP-IP encapsulation

```
[root@calico-node-18121 ~]# calicoctl node status  
Calico process is running.
```

IPv4 BGP status

PEER ADDRESS	PEER TYPE	STATE	SINCE	INFO
35.35.35.122	node-to-node mesh	up	2019-03-20	Established

root

bond0

iptables

route table

tunl0

```
[root@calico-node-18121 ~]# calicoctl get ippool -o wide  
NAME CIDR NAT IPIP MODE DISABLED  
default-ipv4-ippool 192.168.0.0/16 true Always false
```

Node-121

25.25.25.121

```
[root@calico-node-18122 ~]# calicoctl node status  
Calico process is running.
```

IPv4 BGP status

PEER ADDRESS	PEER TYPE	STATE	SINCE	INFO
25.25.25.121	node-to-node mesh	up	2019-03-20	Established

root

bond0

iptables

route table

tunl0

```
[root@calico-node-18122 ~]# calicoctl get ippool -o wide  
NAME CIDR NAT IPIP MODE DISABLED  
default-ipv4-ippool 192.168.0.0/16 true Always false
```

Node-122

35.35.35.122

Hosted By

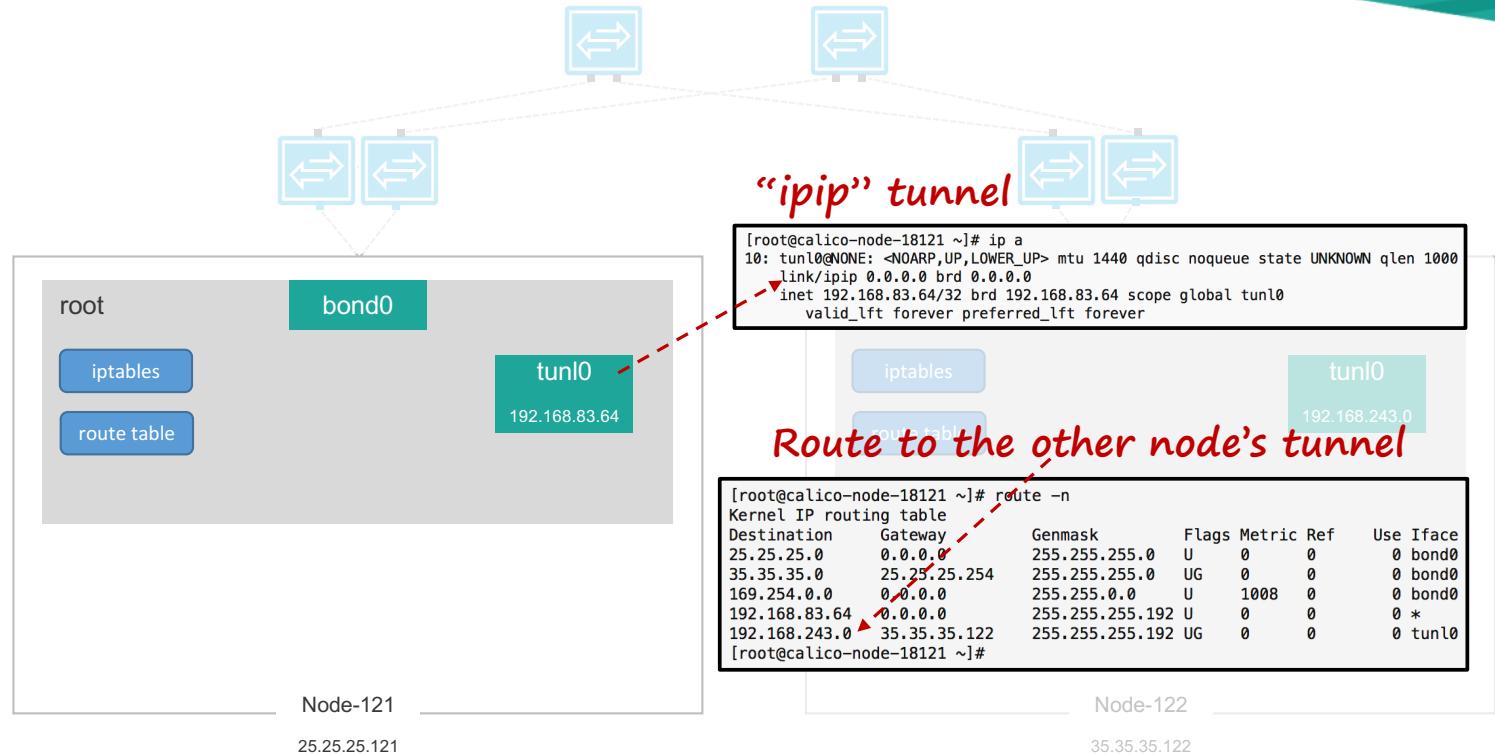
 THE LINUX FOUNDATION | OLF NETWORKING



# Calico

## Default Configuration

- Default: Node Mesh with IP-IP tunnel
- Route table has entries to all the other tunl0 interfaces through other node IPs





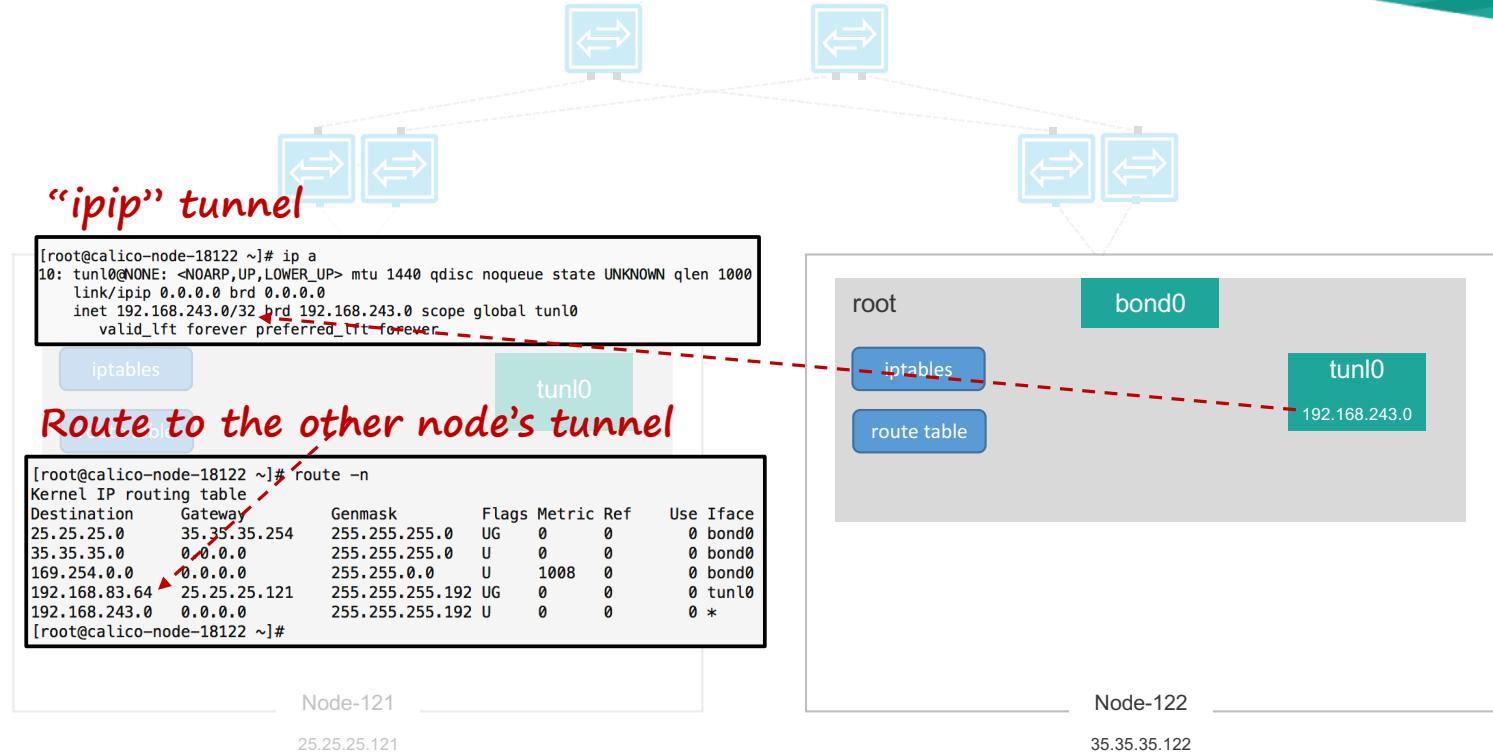
# ONS

NORTH AMERICA  
OPEN NETWORKING //  
Enabling Collaborative  
Development & Innovation

# Calico

## Default Configuration

- Default: Node Mesh with IP-IP tunnel
- Route table has entries to all the other tunl0 interfaces through other node IPs



Hosted By

THE LINUX FOUNDATION | OLF NETWORKING



# Calico

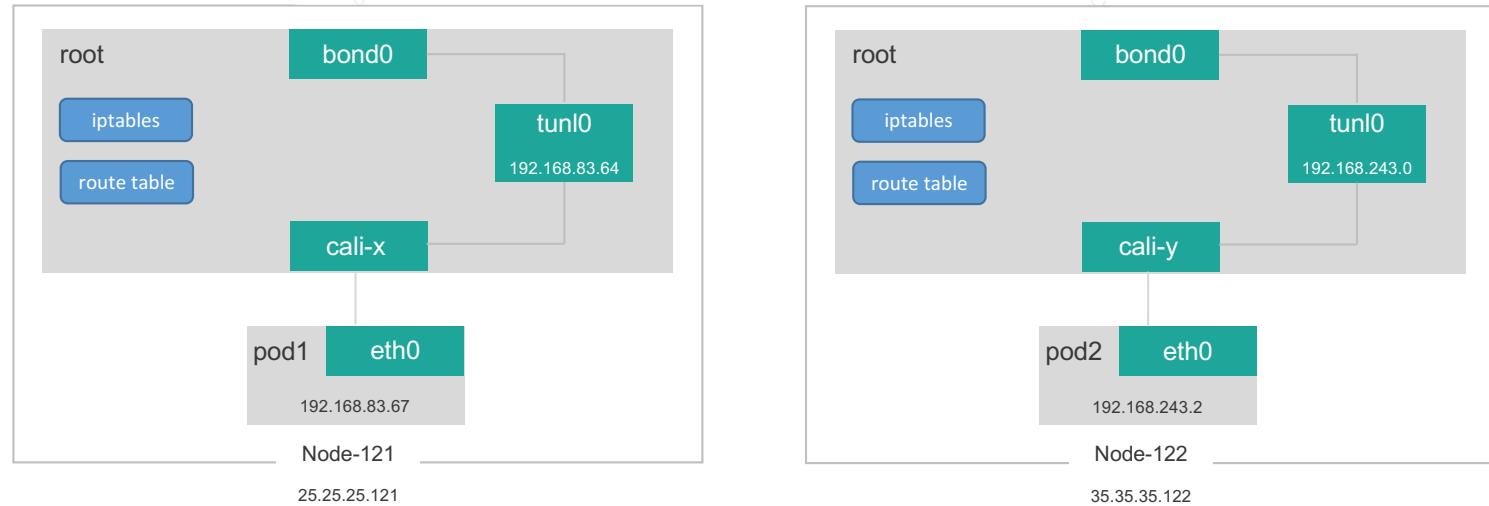
## Pod-to-Pod Communication

- Brought up 2 pods
- “calicoctl get wep” (“workloadendpoints”) shows the endpoints in calico end

```
$ kubectl get pods -n test -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP           NODE
busybox0-4ldqp 1/1     Running   0          13s   192.168.83.67  calico-node-18121
busybox0-c4bl8 1/1     Running   0          13s   192.168.243.2  calico-node-18122

$ calicoctl get wep -n test -o wide
NAMESPACE   WORKLOAD   NODE           NETWORKS   INTERFACE
test        busybox0-4ldqp  calico-node-18121  192.168.83.67/32
test        busybox0-c4bl8  calico-node-18122  192.168.243.2/32
```

*veth interfaces*



Hosted By

THE LINUX FOUNDATION | OLF NETWORKING





# ONS

NORTH AMERICA  
OPEN NETWORKING //  
Enabling Collaborative  
Development & Innovation

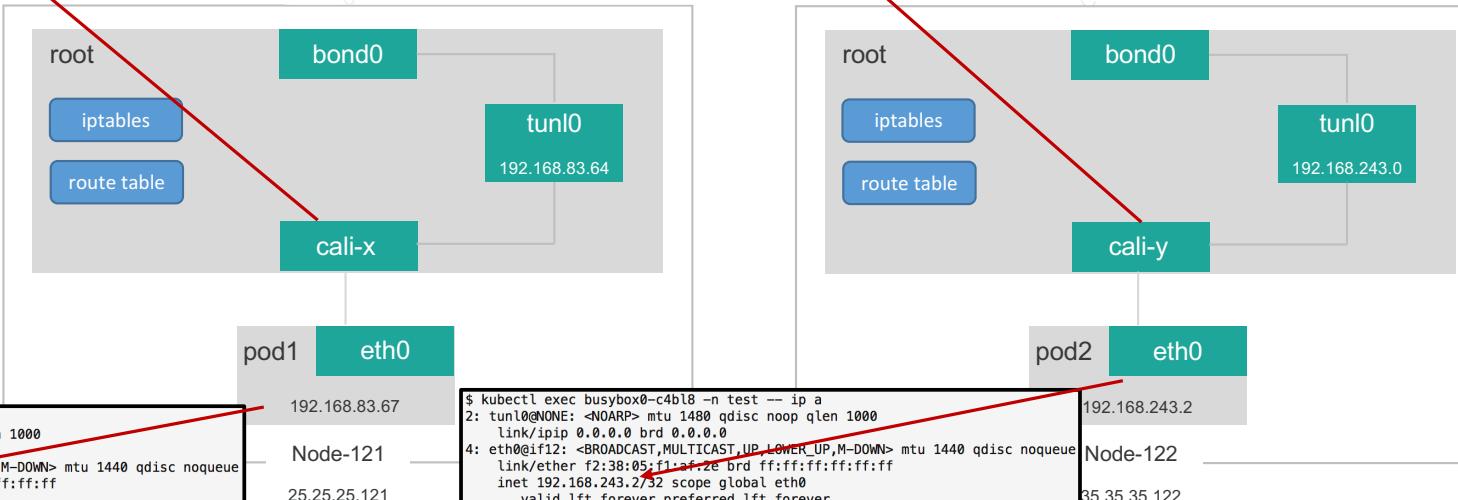
# Calico

## Pod-to-Pod Communication

- VETH Interface with “cali-xxx” got created on the root namespace
- Other end is attached to the pod namespace

```
[root@calico-node-18121 ~]# ip a
10: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state UNKNOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
    inet 192.168.83.64/32 brd 192.168.83.64 scope global tunl0
        valid_lft forever preferred_lft forever
13: cali47dc5a6ea0@if4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ecce:eff:feee:eeee/64 scope link
        valid_lft forever preferred_lft forever
```

```
[root@calico-node-18122 ~]# ip a
10: tunl0@NONE: <NOARP,UP,LOWER_UP> mtu 1440 qdisc noqueue state UNKNOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
    inet 192.168.243.0/32 brd 192.168.243.0 scope global tunl0
        valid_lft forever preferred_lft forever
12: cali596577171e6@if6: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1440 qdisc noqueue state UP
    link/ether ee:ee:ee:ee:ee:ee brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::ecce:eff:feee:eeee/64 scope link
        valid_lft forever preferred_lft forever
```

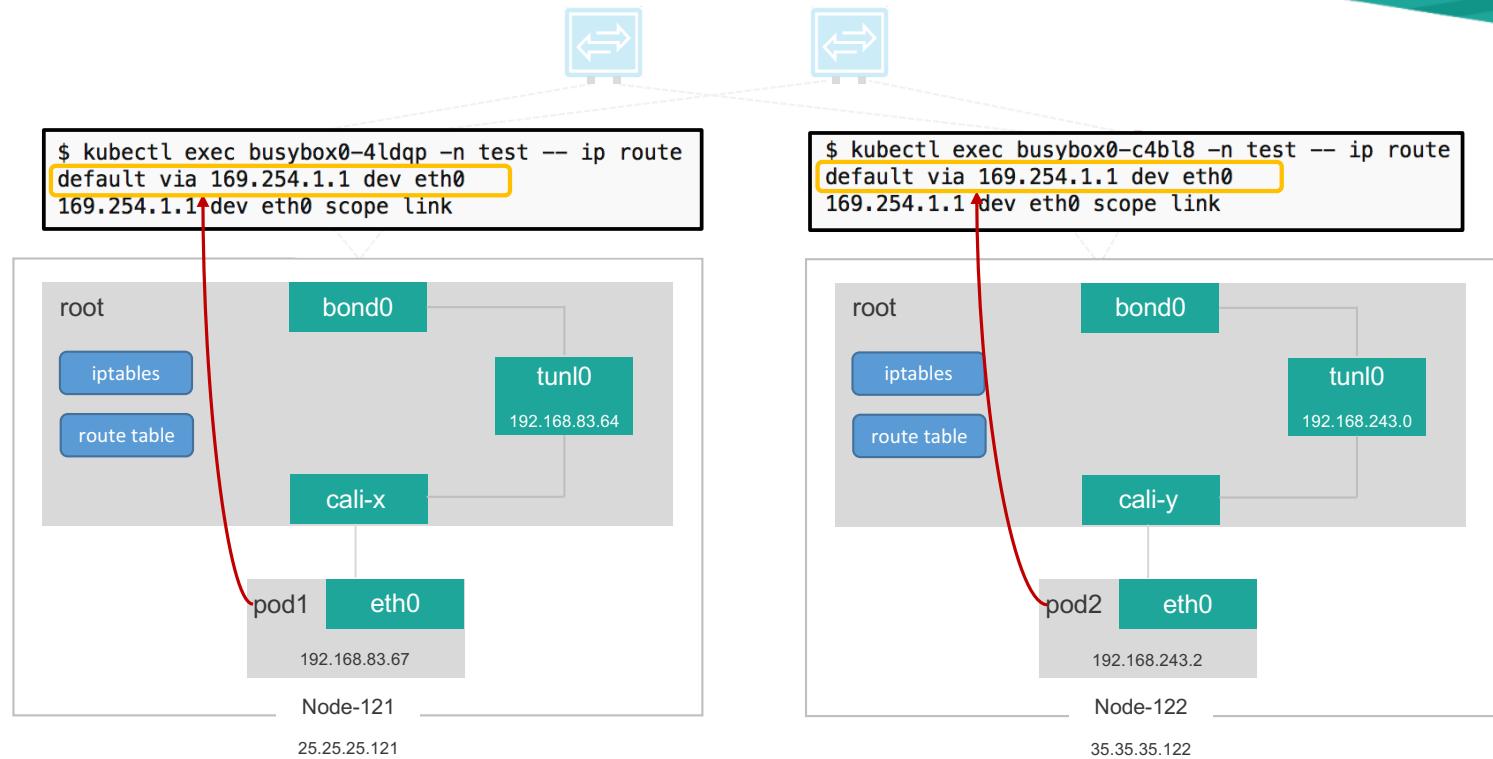




# Calico

## ARP Resolution

- How does ARP gets resolved?
- pod1 & pod2 default route is pointing to private IPv4 "169.254.1.1"

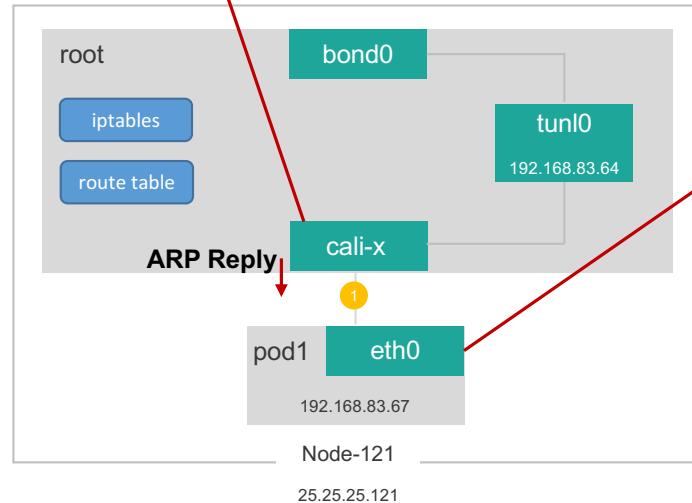




# Calico

## ARP Resolution

- Initiate ping from pod1-to pod2
- ARP request is send to default GW 169.254.1.1
- cali interface replies to ARP request with it's own MAC



② *tcpdump on root veth*

```
[root@calico-node-18121 ~]# tcpdump -i cali4e7dc5a6ea0 -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on cali4e7dc5a6ea0, link-type EN10MB (Ethernet), capture size 262144 bytes
20:44:31.575726 ARP, Request who-has 169.254.1.1 tell 192.168.83.67, length 28
20:44:31.575746 ARP, Reply 169.254.1.1 is-at ee:ee:eeee:ee:ee:ee, length 28
20:44:31.575752 IP 192.168.83.67 > 192.168.243.2: ICMP echo request, id 5632, seq 0, length 64
20:44:31.575931 IP 192.168.243.2 > 192.168.83.67: ICMP echo reply, id 5632, seq 0, length 64
```

③ *root veth is replying with its own MAC*

```
[root@calico-node-18121 ~]# ifconfig cali4e7dc5a6ea0
cali4e7dc5a6ea0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1440
inet6 fe80::eece:eeff:feee:eeee brd fe80::ff:fece:ffff scopeid 0x20<link>
      ether ee:ee:ee:ee:ee:ee txqueuelen 0 (Ethernet)
```

① *Pinging from pod1 to pod2*

```
$ kubectl exec busybox0-4ldqp -n test -- ping 192.168.243.2 -c 1
PING 192.168.243.2 (192.168.243.2): 56 data bytes
64 bytes from 192.168.243.2: seq=0 ttl=62 time=0.381 ms
--- 192.168.243.2 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.381/0.381/0.381 ms
$
```

```
$ kubectl exec busybox0-4ldqp -n test -- arp -n
? (169.254.1.1) at ee:ee:ee:ee:ee:ee [ether] on eth0
$
```

Hosted By

THE LINUX FOUNDATION | OLF NETWORKING



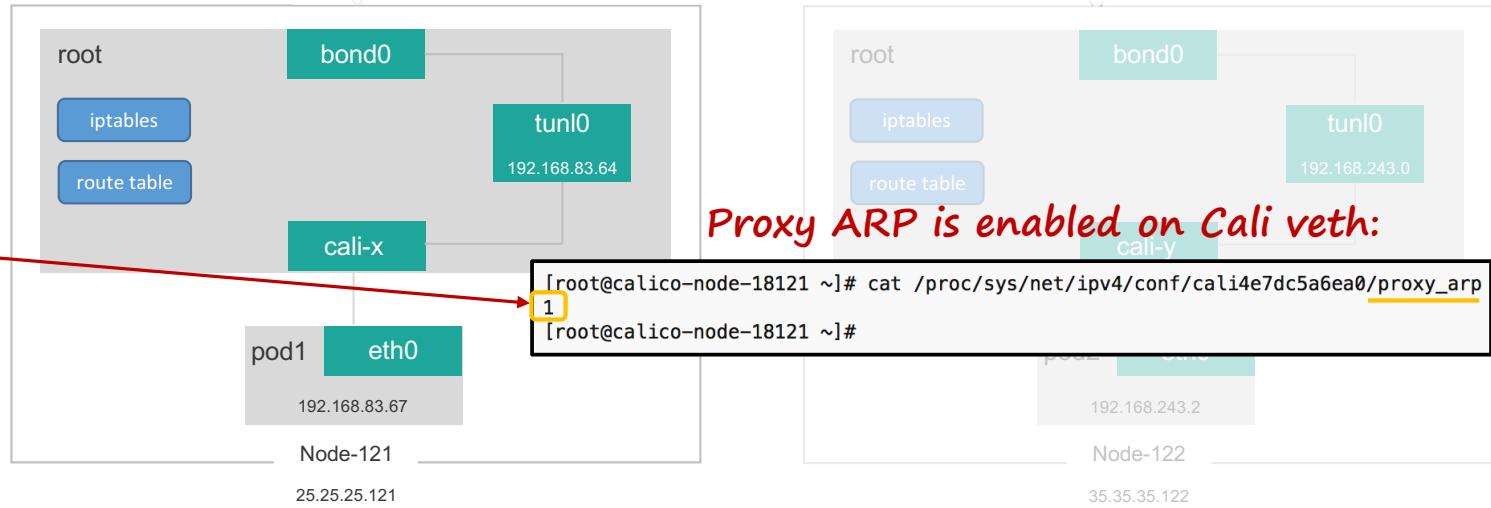
# Calico

## ARP Resolution

- Why “Cali-” VETH Interfaces replies to the ARP request?
- Reason: Proxy-ARP is enabled on the veth interface on root namespace
- Network is only learning the Node IP/MACs, not pod macs

No pod-MAC/IPs will be learned on the network. Only Node-IPs

Endpoints						
Tenant VPC	Segment	MAC	Name	Source	State	IP Addresses
Calico-VPC	Calico-Nodes-25	a0:36:9f:e2:fd:e4	—	Learned	Active	25.25.25.121 (learned) fe80::a236:9fff:fee2:fd:e4 (learned-L2 Only)
Calico-VPC	Calico-Nodes-35	a0:36:9f:e3:05:d0	—	Learned	Active	35.35.35.122 (learned) fe80::a236:9fff:fee3:5d0 (learned-L2 Only)
Filtered Results						Learned





# Calico

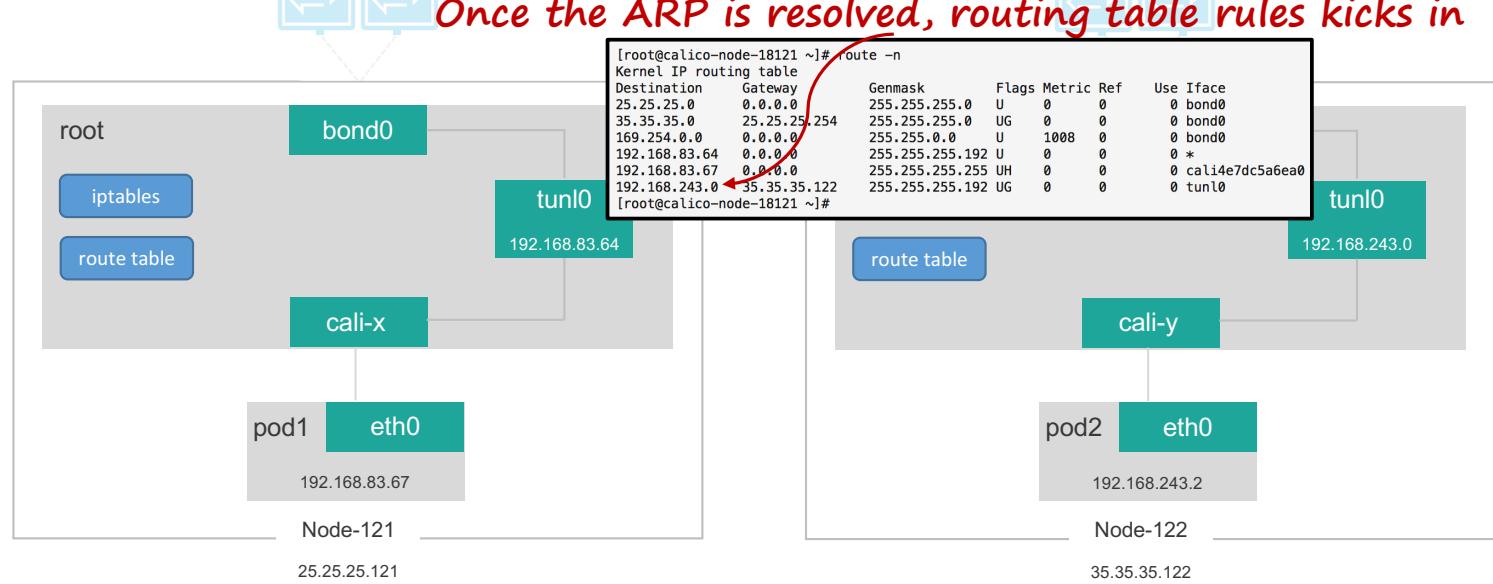
## Packet Forwarding

- After ARP Resolution packet gets forwarded according to the routing table entries
- Packets will get encapsulated via IP-IP

Notice the 2 IP headers: IP-IP protocol

```
[root@calico-node-18121 ~]# tcpdump -i bond0 -nn proto 4
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on bond0, link-type EN10MB (Ethernet), capture size 262144 bytes
22:39:00.890339 IP 25.25.25.121 > 35.35.35.122: IP [REDACTED] ICMP echo request, id 4864, seq 0, length 64 (ipip-proto-4)
22:39:00.890502 IP 35.35.35.122 > 25.25.25.121: IP [REDACTED] ICMP echo reply, id 4864, seq 0, length 64 (ipip-proto-4)
```

Once the ARP is resolved, routing table rules kicks in

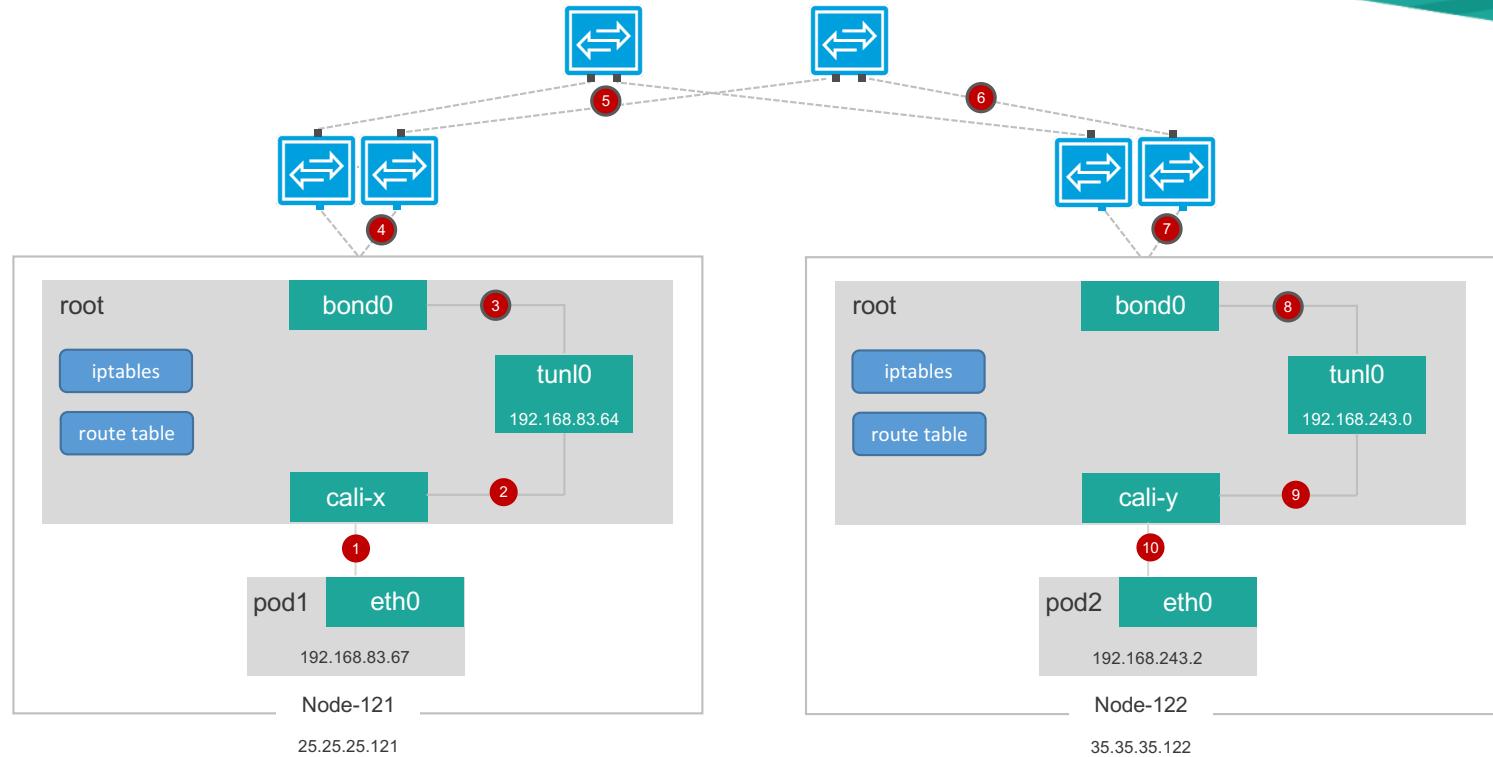




# Calico

## Packet Forwarding

- After ARP Resolution packet gets forwarded according to the routing table entries
- Packets will get encapsulated via IP-IP

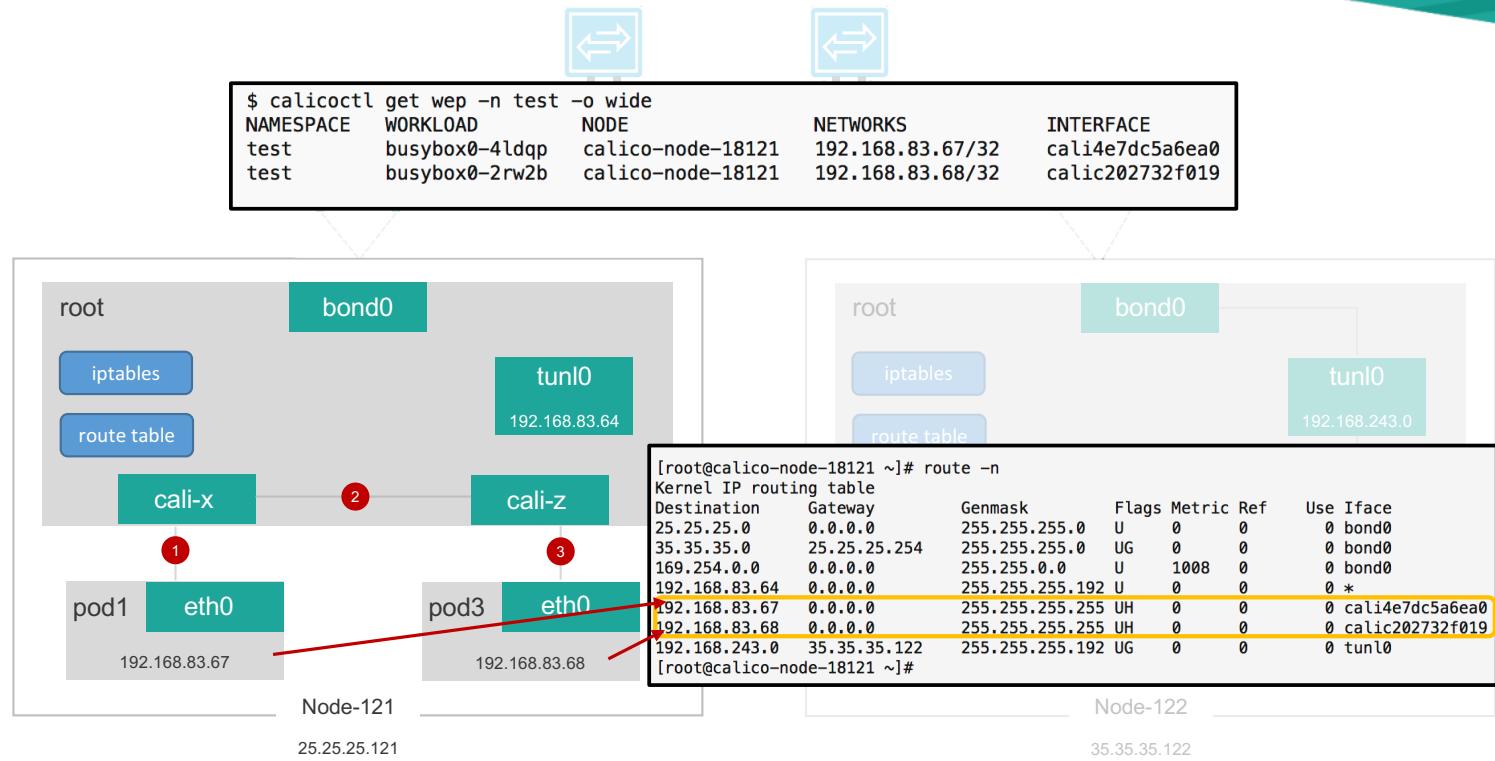




# Calico

## Packet Forwarding in the same host

- /32 routes are present in the routing table for all the containers
- Packets get forwarded to the appropriate calico veth interface based on the routing rules





# Calico

## Non IP-IP

- Disable IP-IP Mode

```
$ cat changeIPPool.yaml
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: default-ipv4-ipool
spec:
  cidr: 192.168.0.0/16
  ipipMode: Never
  natOutgoing: false
```

```
$ calicoctl get ippool -o wide
NAME          CIDR  NAT  IPIPMode  DISABLED
default-ipv4-ipool  192.168.0.0/16  true  Always  false
```

```
$ calicoctl apply -f changeIPPool.yaml
Successfully applied 1 'IPPool' resource(s)
```

```
$ calicoctl get ippool -o wide
NAME          CIDR  NAT  IPIPMode  DISABLED
default-ipv4-ipool  192.168.0.0/16  false  Never  false
```

Node-121

25.25.25.121

Node-122

35.35.35.122

Hosted By

THE LINUX FOUNDATION | OLF NETWORKING

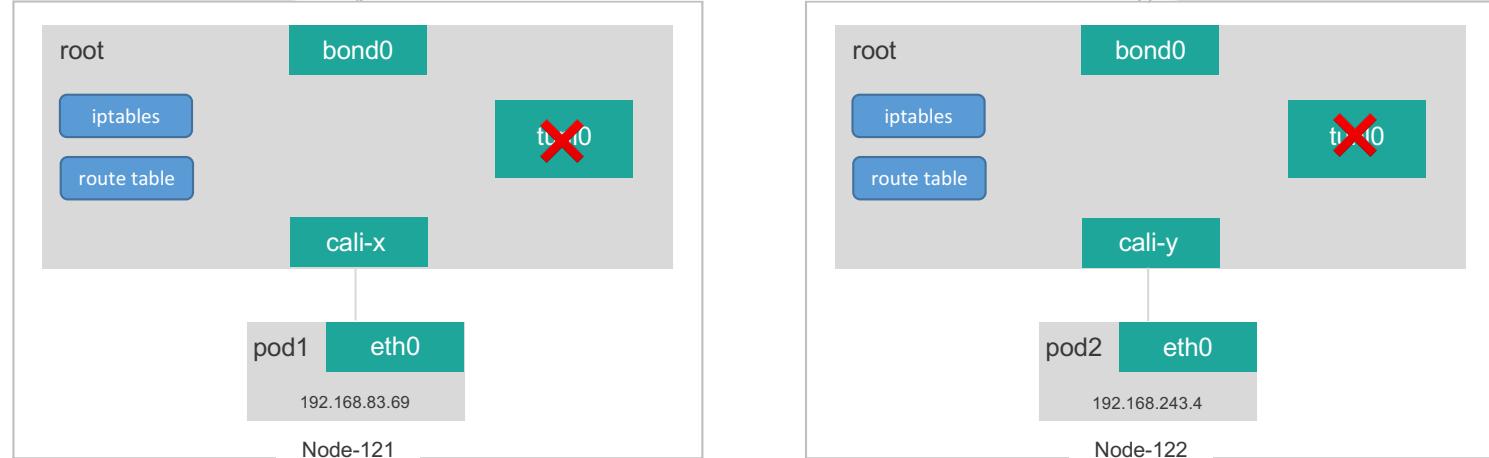




# Calico

## Non IP-IP

- Disable IP-IP Mode. tunl0 interface is not present anymore
- Routes are pointing to the bond0 interface
- Bring up 2 pods as before



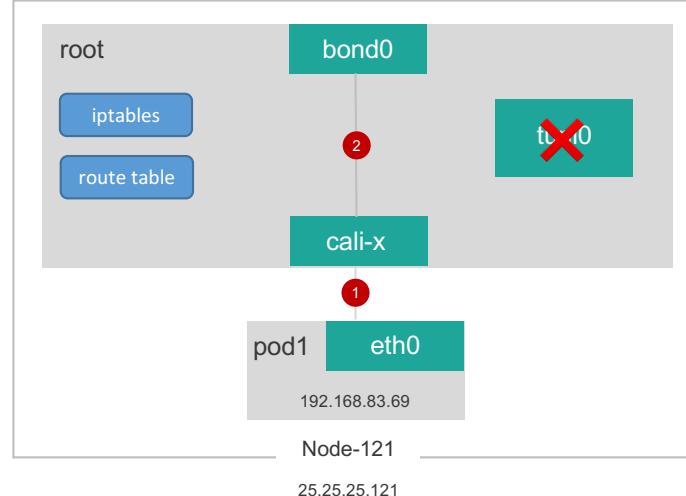
```
$ calicoctl get wep -n test -o wide
NAMESPACE WORKLOAD NODE NETWORKS INTERFACE
test busybox0-nvzth calico-node-18121 192.168.83.69/32 cali12c689f4d96
test busybox0-wcm2w calico-node-18122 192.168.243.4/32 calia75062cc7be
```



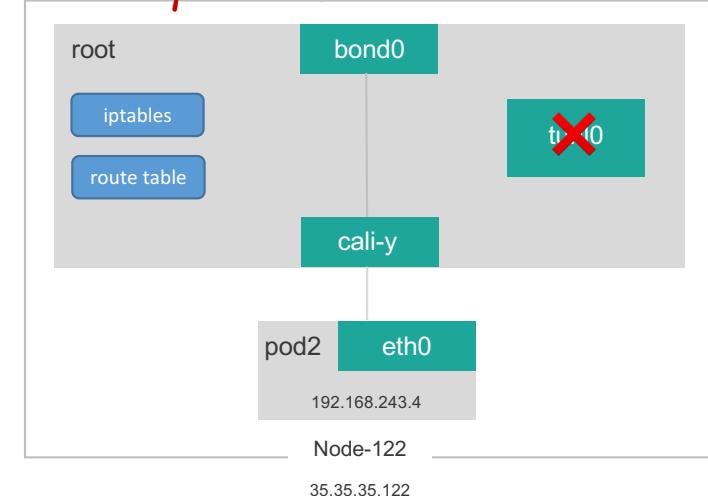
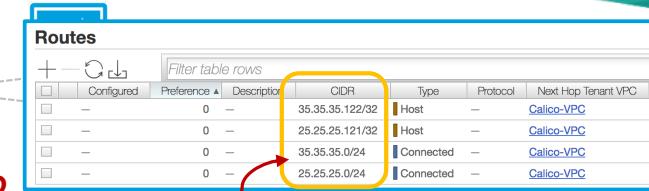
# Calico

## Non IP-IP

- Ping from pod1 to pod2 is unsuccessful
- Reason: Routes are not advertised to the Network



*Packets won't go  
Reason: No routes to the pod networks*

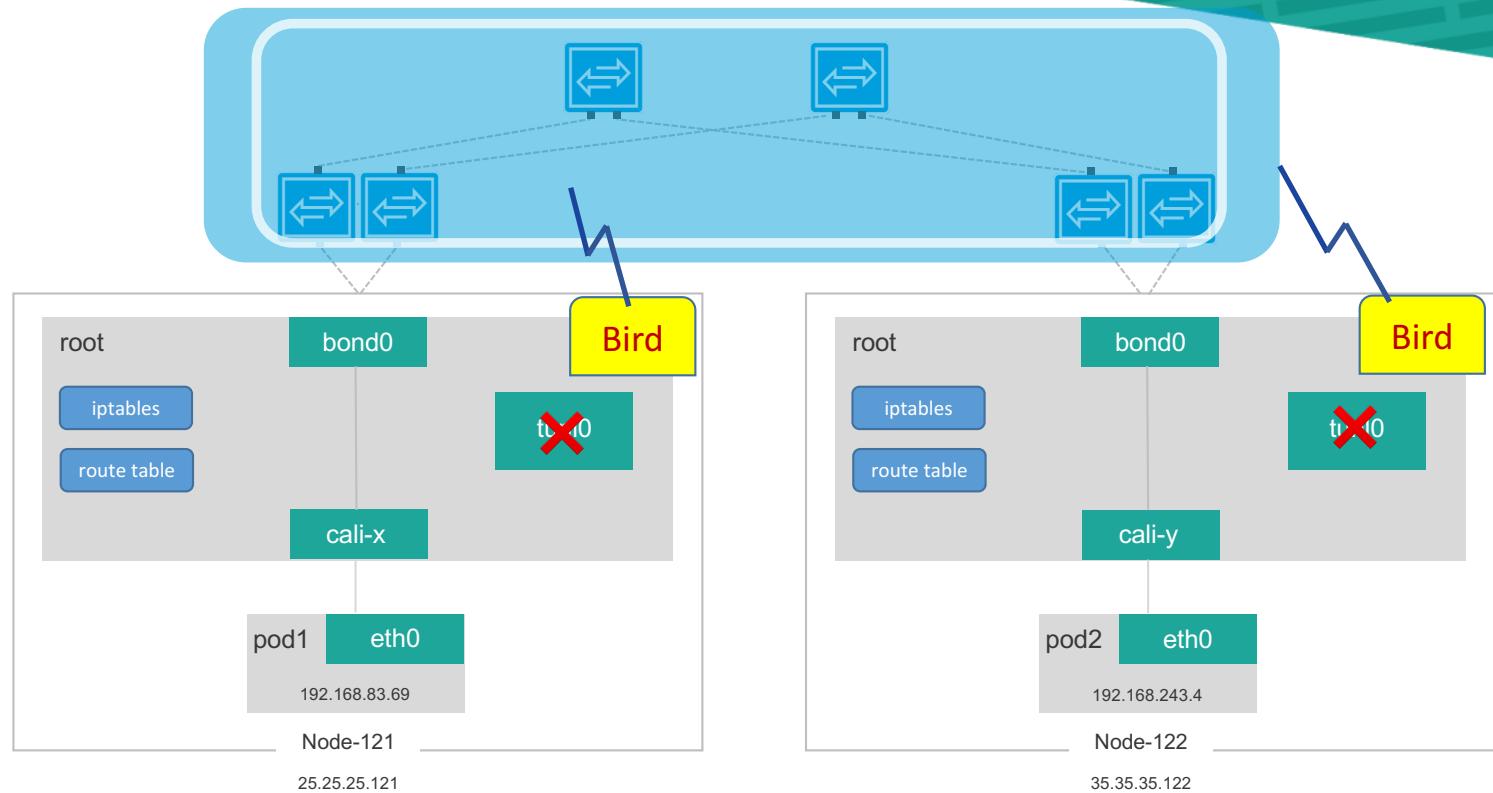




# Calico

## BGP mode

- Need the Calico nodes to peer with the network fabric





# Calico

## BGP Mode

- Create a global BGP configuration
- Create the network as a BGP Peer (\*\*Assuming an abstracted cloud network. Config will vary depending on vendor)

Create a BGP configuration with AS:63400

```
$ cat defaultNodeMesh.yaml
apiVersion: projectcalico.org/v3
kind: BGPConfiguration
metadata:
  name: default
spec:
  logSeverityScreen: Info
  nodeToNodeMeshEnabled: true
  asNumber: 63400

$ calicoctl apply -f defaultNodeMesh.yaml
Successfully applied 1 'BGPConfiguration' resource(s)

$ calicoctl get bgpconfig -o wide
NAME      LOGSEVERITY      MESHENABLED      ASNUMBER
default   Info             true            63400
```

```
[root@calico-node-18121 ~]# calicoctl node status
Calico process is running.

IPv4 BGP status
+-----+-----+-----+-----+-----+
| PEER ADDRESS | PEER TYPE | STATE | SINCE | INFO |
+-----+-----+-----+-----+-----+
| 35.35.35.122 | node-to-node mesh | up | 05:00:33 | Established |
| 2.2.2.2 | global | up | 05:02:40 | Established |
+-----+-----+-----+-----+-----+


IPv6 BGP status
No IPv6 peers found.

[root@calico-node-18121 ~]#
```

Create a "global" BGP Peer

```
$ cat bgpGlobalPeer.yaml
apiVersion: projectcalico.org/v3
kind: BGPPeer
metadata:
  name: bgppeer-bcf
spec:
  peerIP: 2.2.2.2
  asNumber: 63400

$ calicoctl create -f bgpGlobalPeer.yaml
Successfully created 1 'BGPPeer' resource(s)

$ calicoctl get bgppeer
NAME      PEERIP      NODE
bgppeer-bcf  2.2.2.2  (global)  ASN
63400
```

```
[root@calico-node-18122 ~]# calicoctl node status
Calico process is running.

IPv4 BGP status
+-----+-----+-----+-----+-----+
| PEER ADDRESS | PEER TYPE | STATE | SINCE | INFO |
+-----+-----+-----+-----+-----+
| 25.25.25.121 | node-to-node mesh | up | 05:00:31 | Established |
| 2.2.2.2 | global | up | 05:04:19 | Established |
+-----+-----+-----+-----+-----+


IPv6 BGP status
No IPv6 peers found.

[root@calico-node-18122 ~]#
```

Both Nodes are peering with the global BGP Peer

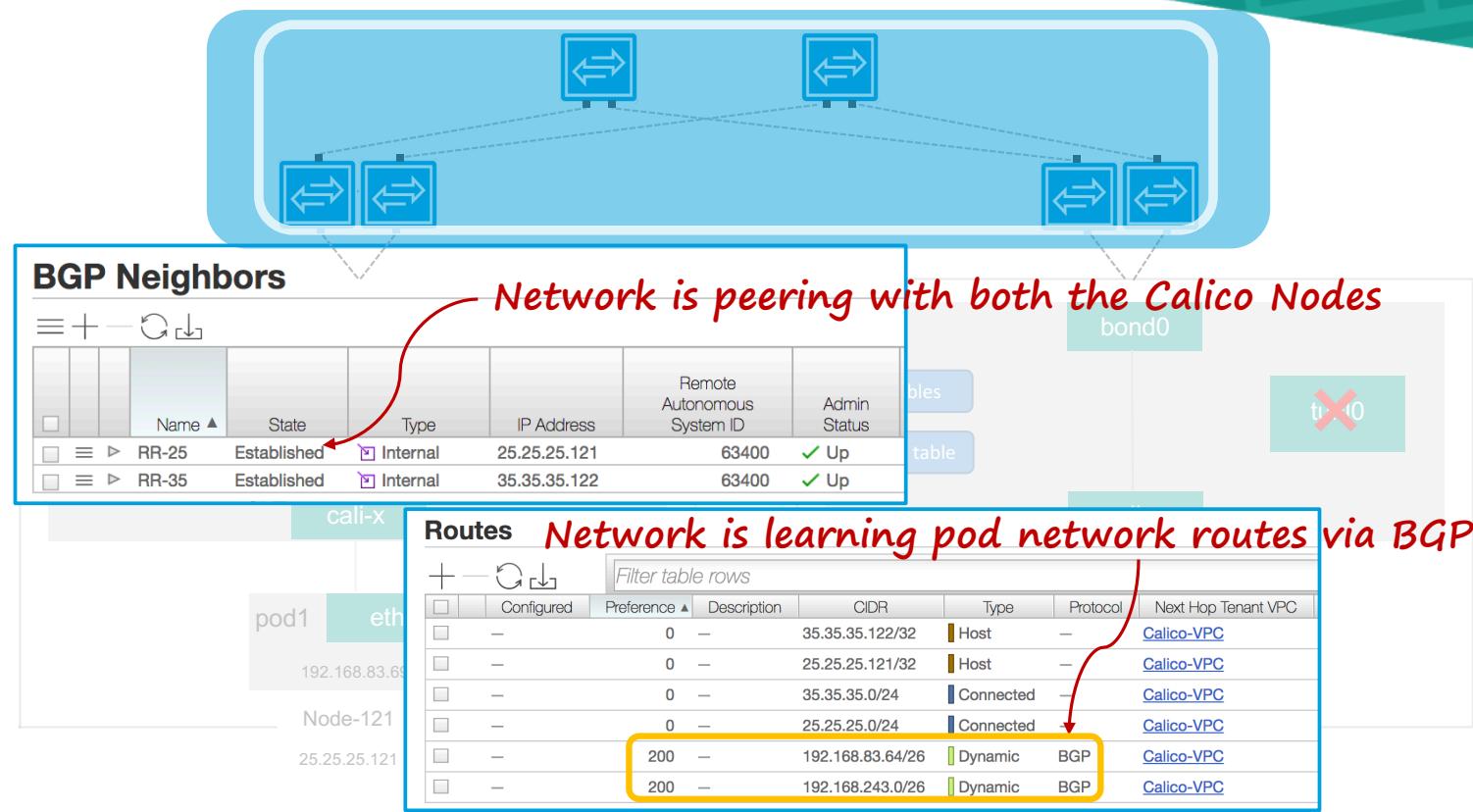
Hosted By



# Calico

## BGP Mode

- Configure the Calico nodes as BGP Neighbors on the network
- As a result network will get to learn about 192.168.83.64/26 & 192.168.243.0/26 routes

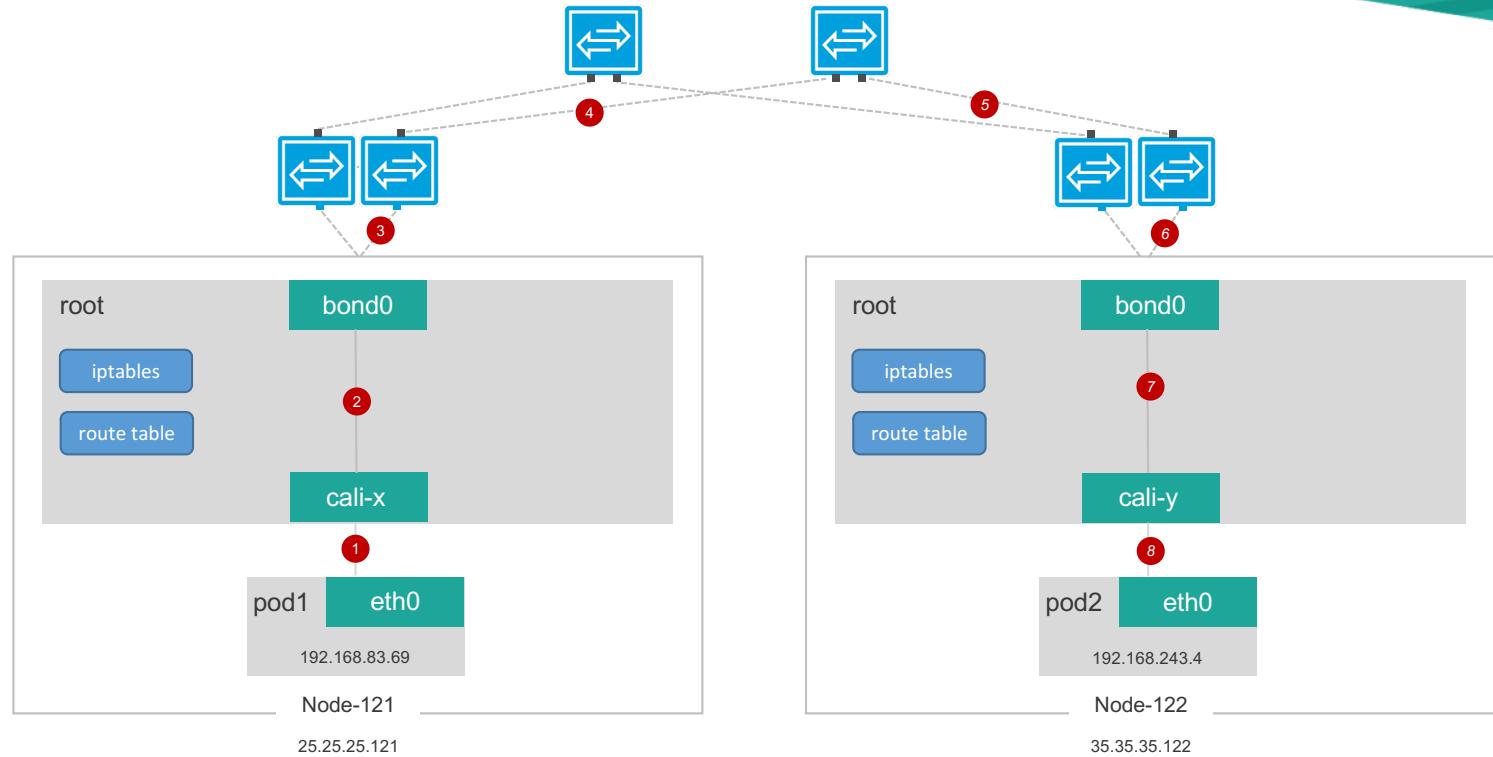




# Calico

## BGP Mode

- Packets goes across the network without any encapsulations



Hosted By

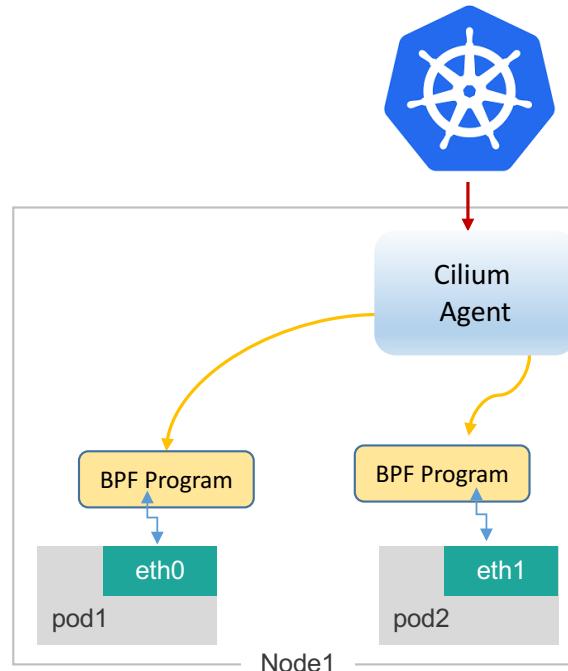
THE LINUX FOUNDATION | OLF NETWORKING



# Cilium

## Architecture

- Cilium Agent, Cilium CLI Client, CNI Plugin will be running on every node
- Cilium agent compiles BPF programs and make the kernel runs these programs at key points in the network stack to have visibility and control over all network traffic in/out of all containers
- Cilium interacts with the Linux kernel to install BPF program which will then perform networking tasks and implement security rules



\*etcd/monitor components are not shown for clarity

# Cilium: Networking

## Overlay Network Mode

- All nodes form a mesh of tunnels using the UDP based encapsulation protocols: VXLAN (default) or Geneve
- Simple: Only requirement is cluster nodes should be able to reach each other using IP/UDP
- Auto-configured: Kubernetes is being run with the "--allocate-node-cidrs" option, Cilium can form an overlay network automatically without any configuration by the user

## Direct/Native Routing Mode

- In direct routing mode, Cilium will hand all packets which are not addressed for another local endpoint to the routing subsystem of the Linux kernel
- Packets will be routed as if a local process would have emitted the packet
- Admins can use routing daemon such as Zebra, Bird, BGPD. The routing protocols will announce the node allocation prefix via the node's IP to all other nodes.



Hosted By

 THE LINUX FOUNDATION |  OFL NETWORKING



# ONS

NORTH AMERICA  
OPEN NETWORKING //  
Enabling Collaborative  
Development & Innovation

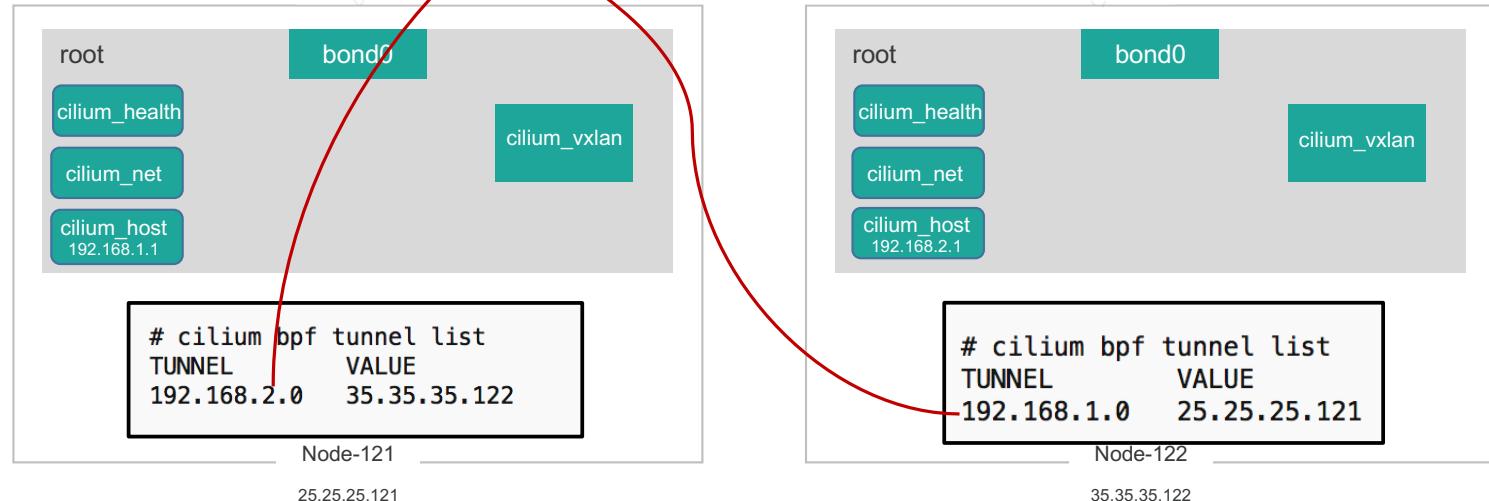
# Cilium

## Default Configuration

- Overlay Networking Mode
- VXLAN encapsulation
- Both VETH/IPVLAN is supported (Higher Performance gains with IPVLAN)

```
[root@Node-18121 ~]# ip a
11: cilium_host@cilium_net: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether ea:24:6e:6a:97:d8 brd fffff:ffff:ffff:ff:ff
        inet 192.168.1.1/32 brd cilium_host
            valid_lft forever preferred_lft forever
            inet6 fe80::e824:6eff:fe6a:97d8/64 scope link
                valid_lft forever preferred_lft forever
12: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 86:67:7a:f6:58:c1 brd fffff:ffff:ffff:ff
        inet6 fe80::8467:7aff:fe6f:58c1/64 scope link
            valid_lft forever preferred_lft forever
```

```
[root@cilium-node-18122 ~]# ip a
11: cilium_host@cilium_net: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether 46:4a:6c:a4:51:27 brd ff:ff:ff:ff:ff:ff
        inet 192.168.2.1/32 brd cilium_host
            valid_lft forever preferred_lft forever
            inet6 fe80::444a:6cff:fea4:5127/64 scope link
                valid_lft forever preferred_lft forever
12: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 72:cb:a4:c7:2a:36 brd ff:ff:ff:ff:ff:ff
        inet6 fe80::70cb:a4ff:fe7:2a36/64 scope link
            valid_lft forever preferred_lft forever
```



Hosted By

THE LINUX FOUNDATION | OLF NETWORKING



# Cilium

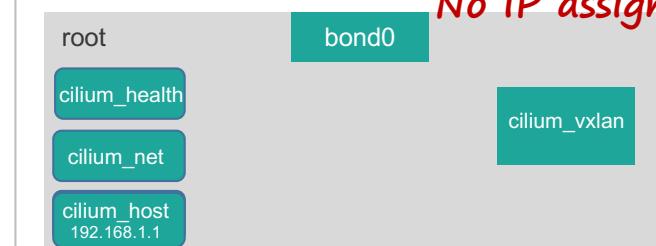
## Default Configuration

- "cilium\_vxlan" interface is in metadata mode. It can send & receive on multiple addresses
- Cilium addressing model allows to derive the node address from each container address.
- This is also used to derive the VTEP address, so you don't need to run a control plane protocol to distribute these addresses. All you need to have are routes to make the node addresses routable

```
[root@Node-18121 ~]# ip a
11: cilium_host@cilium_net: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether ea:24:6e:6a:97:d8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/32 brd ff:ff:ff:ff:ff:ff scope link cilium_host
        valid_lft forever preferred_lft forever
    inet6 fe80::8e24:6eff:fe6a:97d8/64 scope link
        valid_lft forever preferred_lft forever
12: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 86:67:7a:f6:58:c1 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::8467:7aff:fe76:58c1/64 scope link
        valid_lft forever preferred_lft forever
```

```
[root@cilium-node-18122 ~]# ip a
11: cilium_host@cilium_net: <BROADCAST,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether 46:4a:6c:a4:51:27 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/32 brd ff:ff:ff:ff:ff:ff scope link cilium_host
        valid_lft forever preferred_lft forever
    inet6 fe80::44a:6cff:fea4:5127/64 scope link
        valid_lft forever preferred_lft forever
12: cilium_vxlan: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 72:cba:4:c7:2:a3 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::70cba:4ff:fe7:c2a3/64 scope link
        valid_lft forever preferred_lft forever
```

'cilium\_vxlan' interface is in Metadata mode.  
No IP assigned to this interface



```
[root@Node-18121 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
35.35.35.0     25.25.25.254  255.255.255.0  UG   0      0      0 bond0
192.168.1.0    192.168.1.1   255.255.255.0  UG   0      0      0 cilium_host
192.168.1.1    0.0.0.0       255.255.255.255 UH   0      0      0 cilium_host
192.168.2.0    192.168.1.1   255.255.255.0  UG   0      0      0 cilium_host
```

Node-121

25.25.25.121

```
[root@cilium-node-18122 ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref  Use Iface
25.25.25.0     35.35.35.254  255.255.255.0  UG   0      0      0 bond0
192.168.1.0    192.168.2.1   255.255.255.0  UG   0      0      0 cilium_host
192.168.2.0    192.168.2.1   255.255.255.0  UG   0      0      0 cilium_host
192.168.2.1    0.0.0.0       255.255.255.255 UH   0      0      0 cilium_host
```

Node-122

35.35.35.122

Hosted By

THE LINUX FOUNDATION | OLF NETWORKING

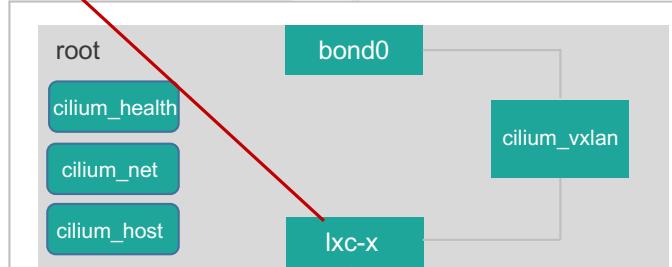


# Cilium

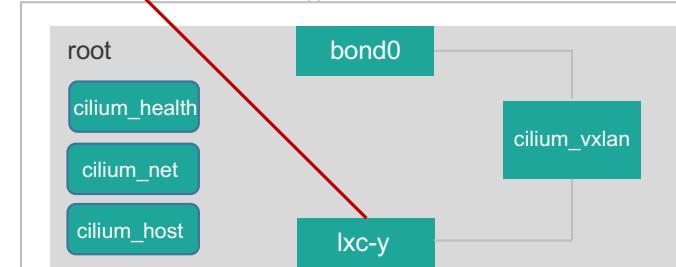
## Pod-to-Pod Communication

- VETH Interface with “lxc-xxx” got created on the root namespace
- Other end is attached to the pod namespace

```
[root@cilium-node-18121 ~]# ip a
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd :: scope host
        valid_lft forever preferred_lft forever
29: eth0@if30: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 79:bb:b4:c2 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.231/32 brd 192.168.1.231 scope global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::79bb:b4ff:fe00:231/64 scope link
            valid_lft forever preferred_lft forever
```



```
[root@cilium-node-18122 ~]# ip a
36: lxc70681372d77feif35: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue qlen 1000
    link/ether 9c:a0:38:46:be:21 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::9ca0:38ff:fe46:be21/64 scope link
        valid_lft forever preferred_lft forever
```



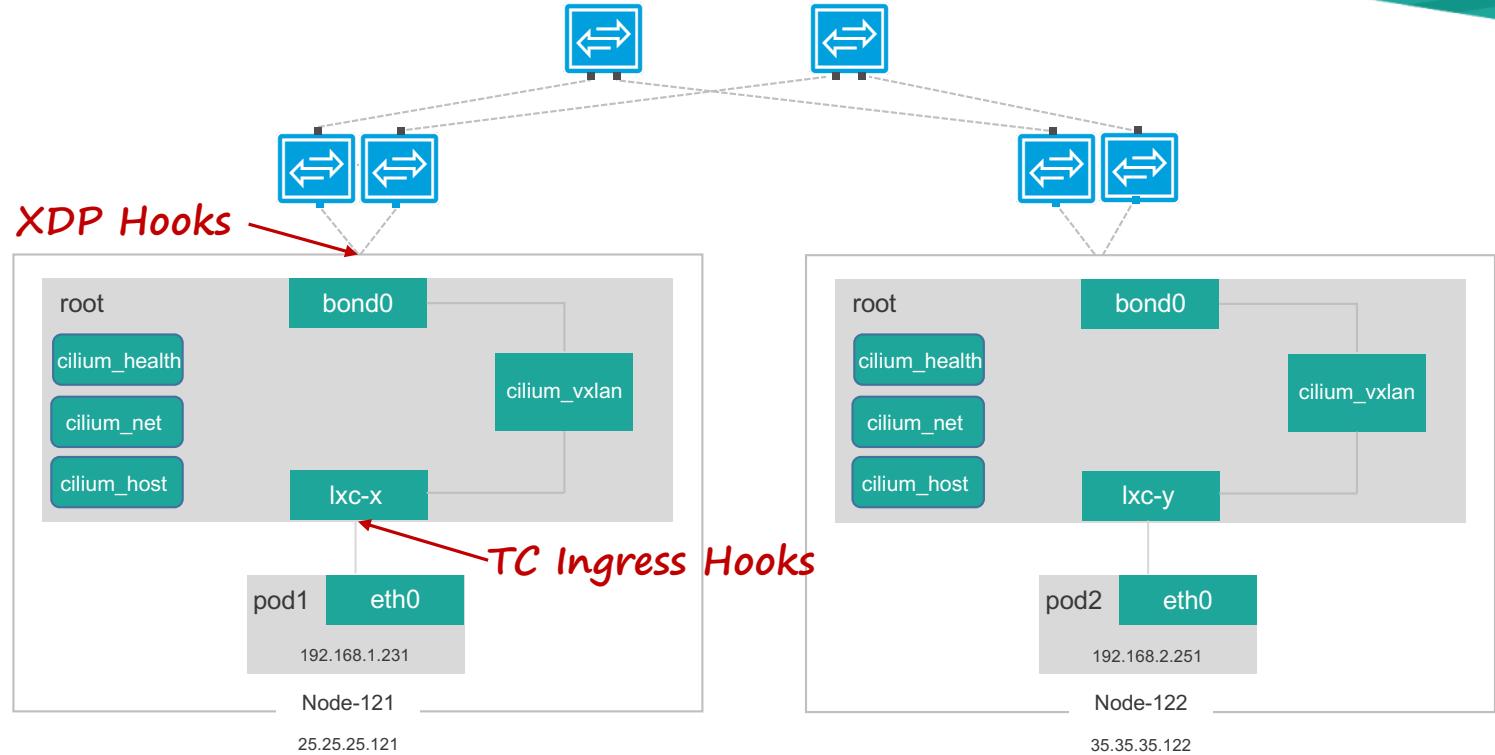
```
[root@cilium-node-18122 ~]# docker exec -it 730a700a0d7f /bin/sh
# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 brd :: scope host
        valid_lft forever preferred_lft forever
35: eth0@if36: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue
    link/ether 5e:00:00:26:95:ca brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.251/32 brd 192.168.2.251 scope global eth0
        valid_lft forever preferred_lft forever
        inet6 fe80::5e00:26ff:fe00:251/64 scope link
            valid_lft forever preferred_lft forever
```



# Cilium

## Datapath

- Cilium datapath uses eBPF hooks to load BPF programs
- XDP BPF hook is at the earliest point possible in the networking driver and triggers a run of the BPF program upon packet reception
- Traffic Control (TC) Hooks: BPF programs are attached to the TC Ingress hook of host side of the VETH pair for monitoring & enforcement





# Cilium

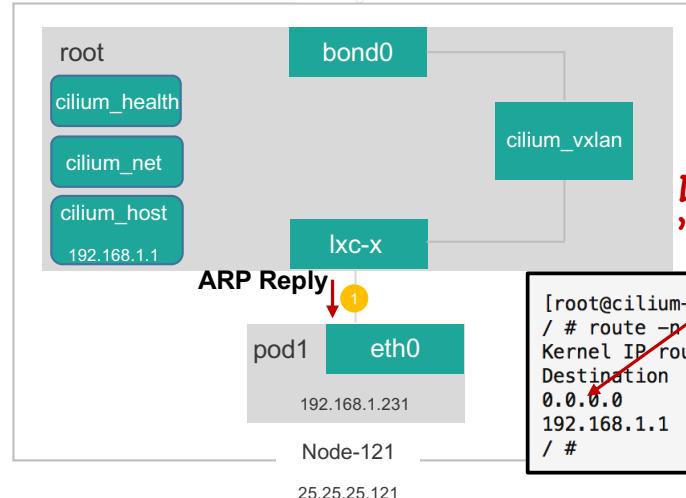
## ARP Resolution

- Default GW of the container is pointing to the IP of “cilium\_host”
  - BPF Program is installed to reply to the ARP request
  - LXC Interface MAC is used for the ARP reply

BPF program is responding to the ARP with VETH's (lxc-x) MAC

```
[root@Node-18121 ~]# tcpdump -i lxc97b73386ee5f -nn
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on lxc97b73386ee5f, link-type EN10MB (Ethernet), capture size 262144 bytes

01:11:02.484685 ARP, Request who-has 192.168.1.1 tell 192.168.1.22, length 28
01:11:02.484705 ARP, Reply 192.168.1.1 is-at ea:22:19:a6:88:22, length 28
```



Default GW of the pod is pointing to 'cilium host' IP

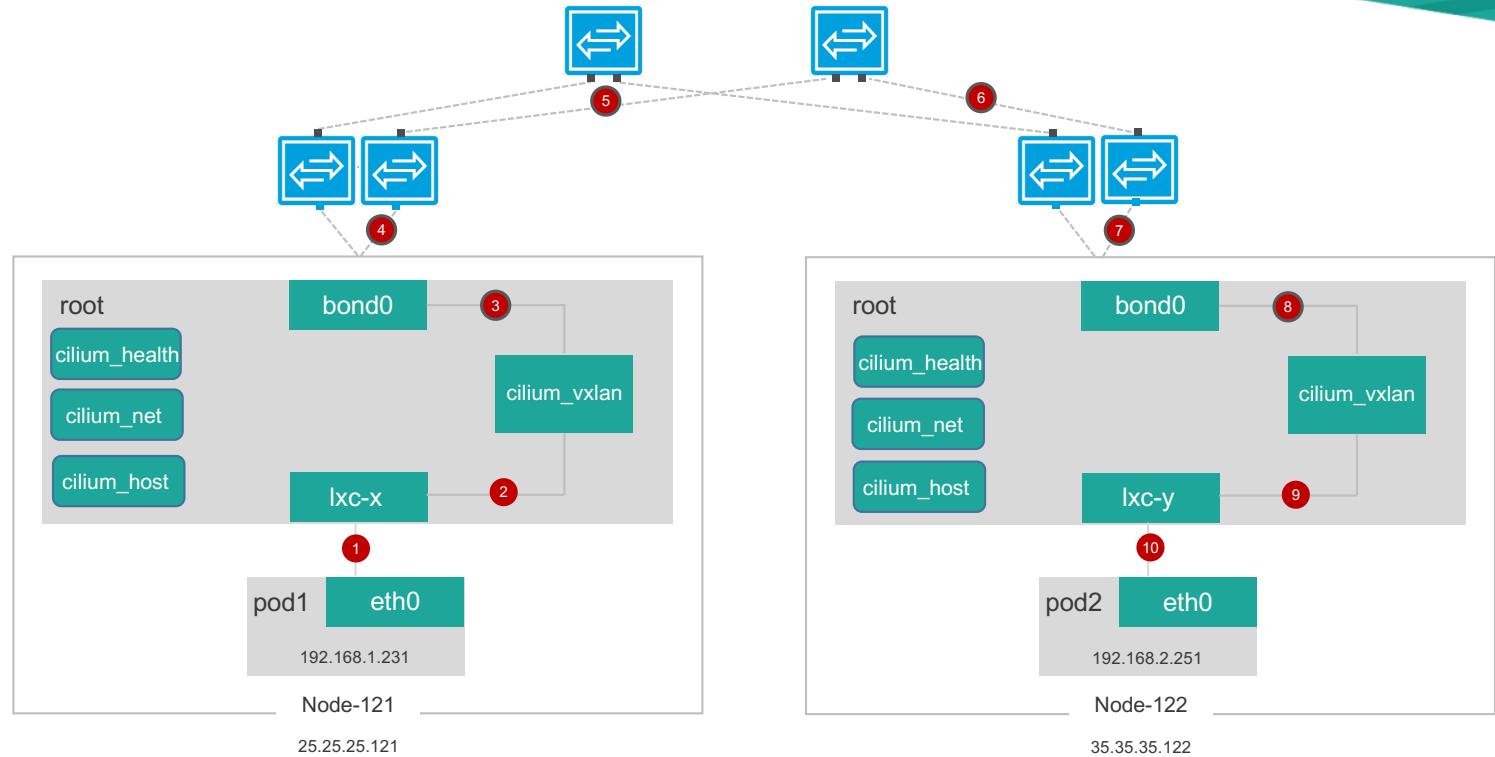
```
[root@cilium-node-18121 ~]# docker exec -it ed8ba415daac /bin/sh
/ # route -n
Kernel IP routing table
Destination      Gateway          Genmask        Flags Metric  Ref  Use Iface
0.0.0.0          192.168.1.1    0.0.0.0        UG      0      0      0 eth0
192.168.1.1    0.0.0.0        255.255.255.255 UH      0      0      0 eth0
/ #
```



# Cilium

## Packet Flow

- `cilium_vxlan` device does VXLAN encap/decap
- Traffic from `pod1` is going through “`cilium_vxlan`” device before exiting

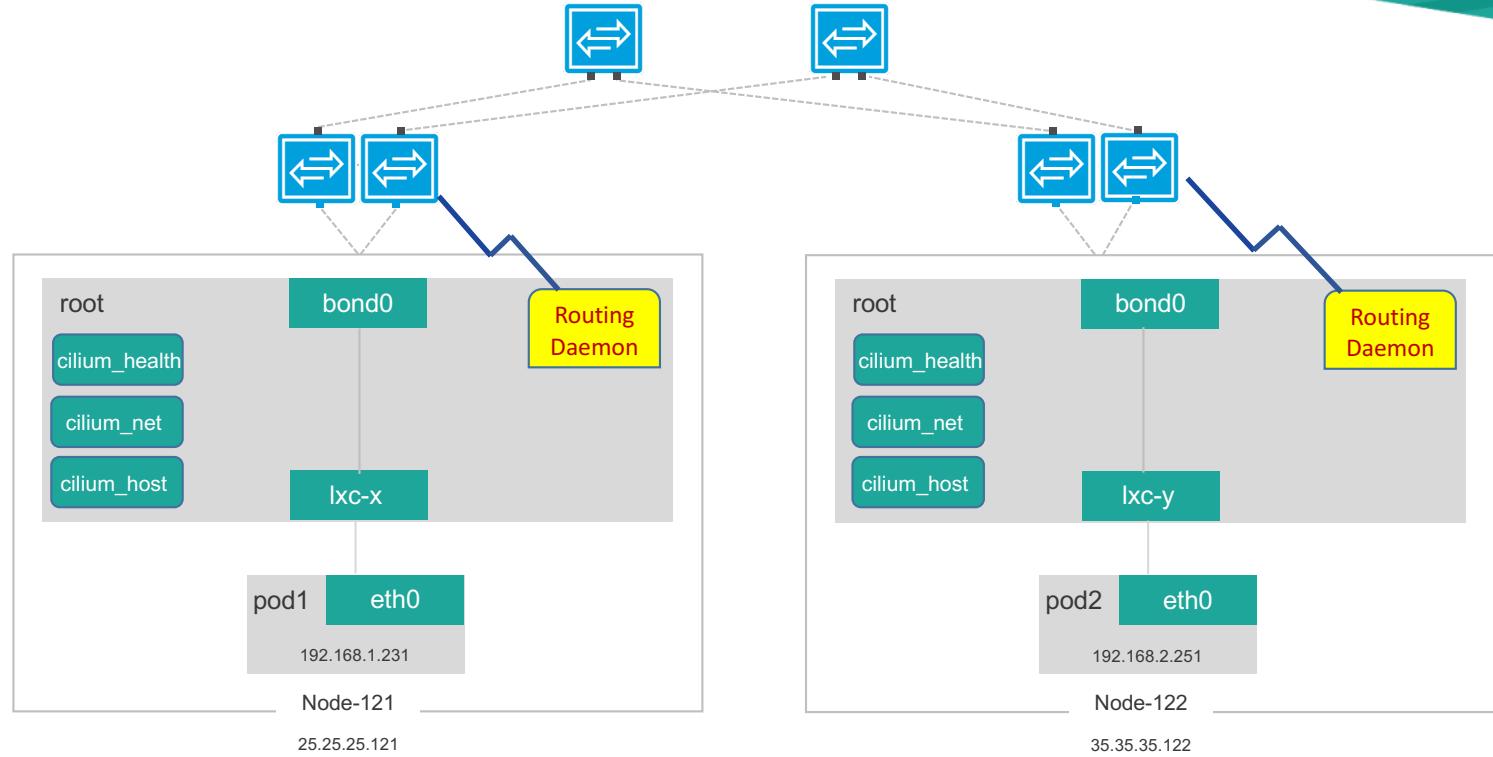




# Cilium

## Direct Routing

- No VXLAN/GENEVE overlays
- As an admin you are able to run your own flavor of routing daemon (Bird/BGPD/Zebra etc) to distribute routes
- You can make the Pod IP's routable as a result



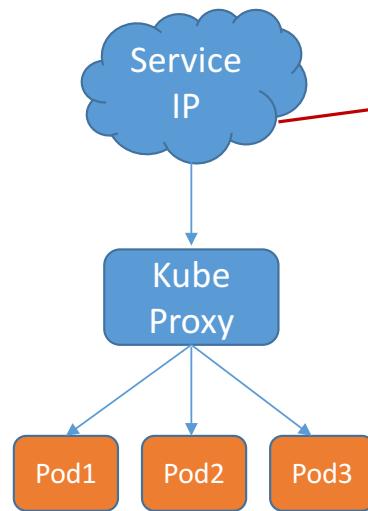


Yes! Back to the Basics...

# K8S Networking: Basics

## Services

- Pods are mortal
- Need a higher level abstractions: Services
- “Service” in Kubernetes is a conceptual concept. Service is not a process/daemon. Outside networks doesn’t learn Service IP addresses
- Implemented through Kube Proxy with IPTables rules



```
$ kubectl get services -n demo -o wide
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE   SELECTOR
hostnames  ClusterIP   10.96.13.117  <none>        80/TCP    23h   app=hostnames
```

```
$ kubectl describe service -n demo
Name:            hostnames
Namespace:       demo
Labels:          <none>
Annotations:    <none>
Selector:        app=hostnames
Type:           ClusterIP
IP:             10.96.13.117
Port:           default  80/TCP
TargetPort:      9376/TCP
Endpoints:      192.168.1.63:9376,192.168.2.171:9376,192.168.3.155:9376
Session Affinity: None
Events:          <none>
```

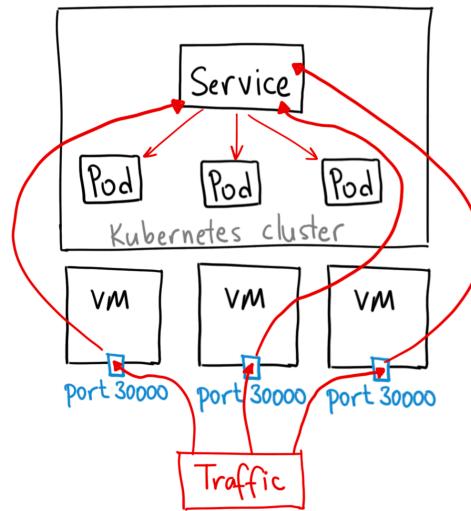


# K8S Networking: Basics

## Exposing Services

- If Services are an abstracted concept without any meaning outside of the K8S cluster how do we access?
  - NodePort / LoadBalancer / Ingress etc.

*NodePort: Service is accessed via 'NodeIP:port'*



Credit: <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>

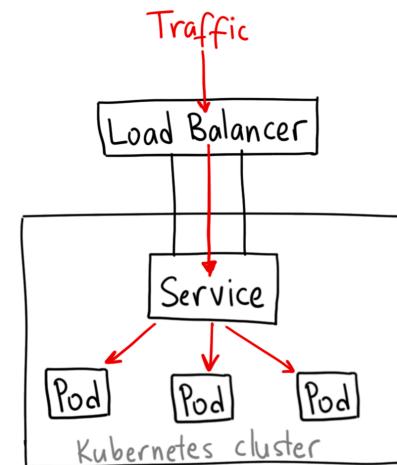


# K8S Networking: Basics

## Exposing Services

- Load Balancer: Spins up a load balancer and binds service IPs to Load Balancer VIP
- Very common in public cloud environments
- For baremetal workloads: '**MetalLB**' (Up & coming project, load-balancer implementation for bare metal K8S clusters, using standard routing protocols)

*LoadBalancer: Service is accessed via Loadbalancer*



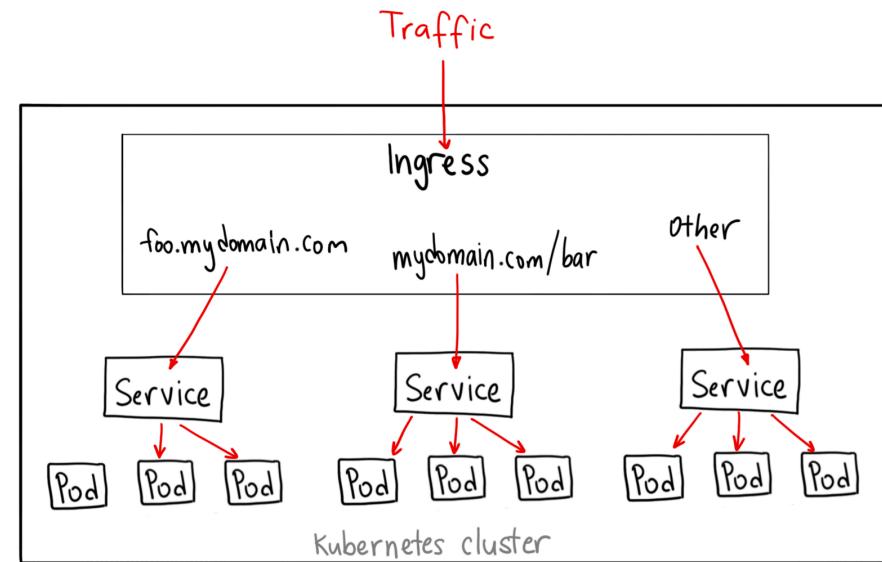
Credit: <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>



# K8S Networking: Basics

## Exposing Services

- **Ingress:** K8S Concept that lets you decide how to let traffic into the cluster
- Sits in front of multiple services and act as a "router"
- Implemented through an ingress controller (NGINX/HA Proxy)



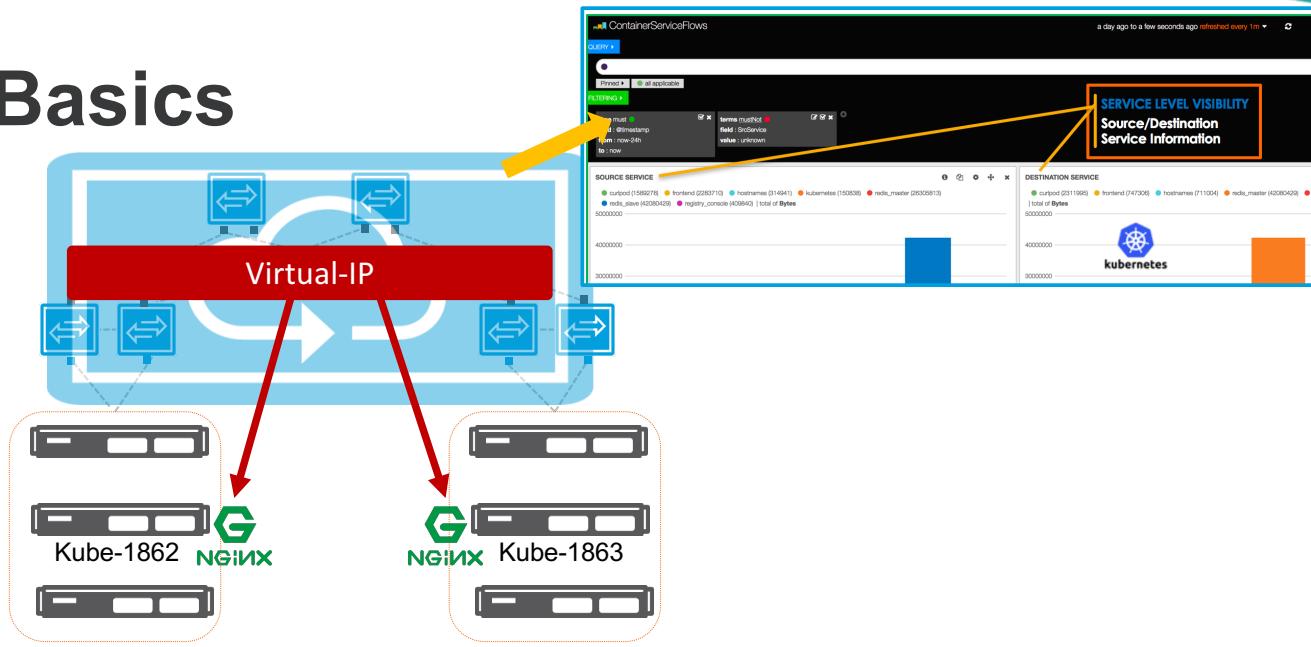
Credit: <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>



# K8S Networking: Basics

## Exposing Services

- **Ingress:** Network (“Abstracted Network”) can really help you out here
- Ingress controllers are deployed in some of your “public” nodes in your cluster
- Eg: **Big Cloud Fabric** (by Big Switch), can expose a Virtual IP in front of the Ingress Controllers and perform Load Balancing/Health Checks/Analytics



```
$ kubectl get pods --all-namespaces -o wide
kube-system      nginx-ingress-controller-5b95b96fb7-r6ltd  1/1   Running   10.2.18.62   kube-1862
kube-system      nginx-ingress-controller-5b95b96fb7-vj88x  1/1   Running   10.2.18.63   kube-1863
```



# Thanks!

- Repo for all the command outputs/PDF slides:  
**<https://github.com/jayakody/ons-2019>**
- Credits:
  - Inspired by Life of a Packet- Michael Rubin, Google (<https://www.youtube.com/watch?v=0Omvgd7Hg1I>)
  - Sarath Kumar, Prashanth Padubidri : Big Switch Engineering
  - Thomas Graf, Dan Wendlandt: Isovalent (Cilium Project)
  - Project Calico Slack Channel: special shout out to: Casey Davenport, Tigera
  - And so many other folks who took time to share knowledge around this emerging space through different mediums (Blogs/YouTube videos etc)

