KubeCon | CloudNativeCon

North America 2024

Understanding Kubernetes Networking in 30 minutes

James Strong
Ricardo Katz

James // strongjz // strongjz.bsky.social

- ❑ Solution Architect @ Isovalent part of Cisco
- ❑ Maintainer of Ingress NGINX
- ❑ Author Networking and Kubernetes
- ❑ ACloud Guru Instructor
- ❑ Gimli Cosplay Enthusiast

Ricardo // rikatz // @rkatz.xyz

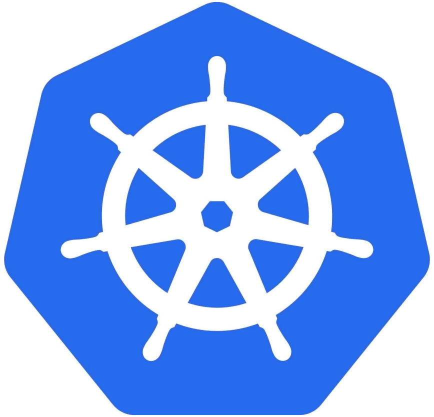
❑ Software Engineer @ VMware by Broadcom

❑ Creator of Kubepug

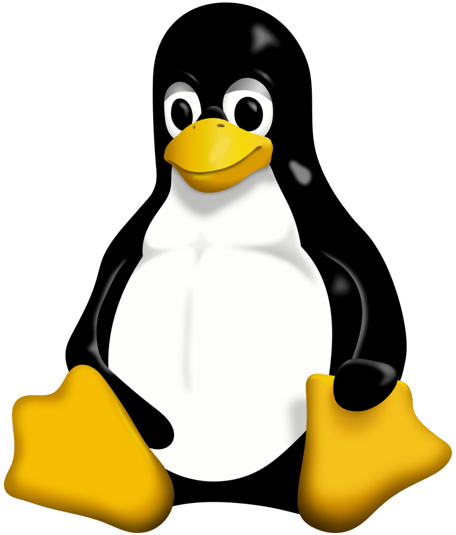
❑ Maintainer of Ingress NGINX

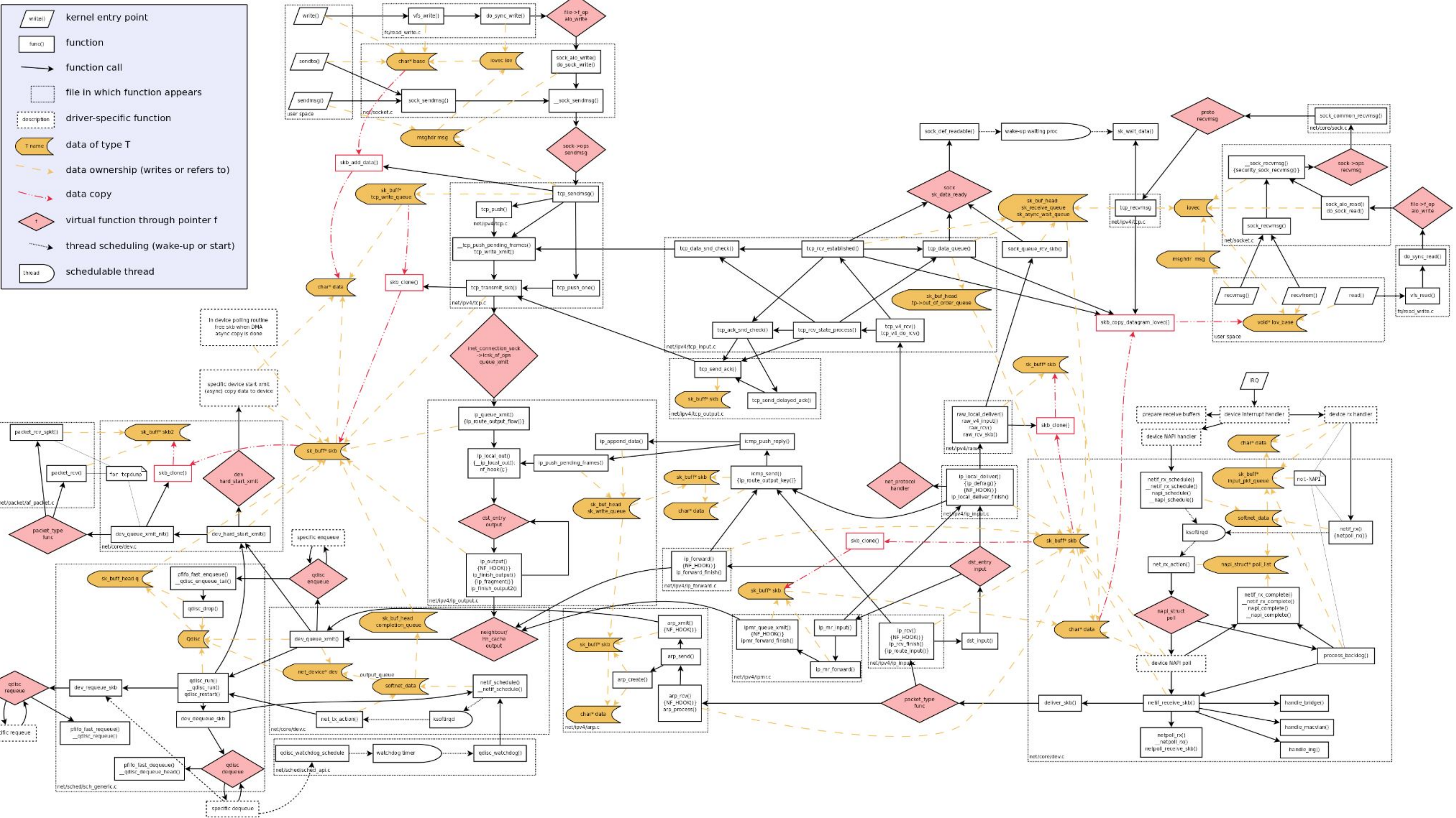
❑ Enthusiast

LEGO



is just

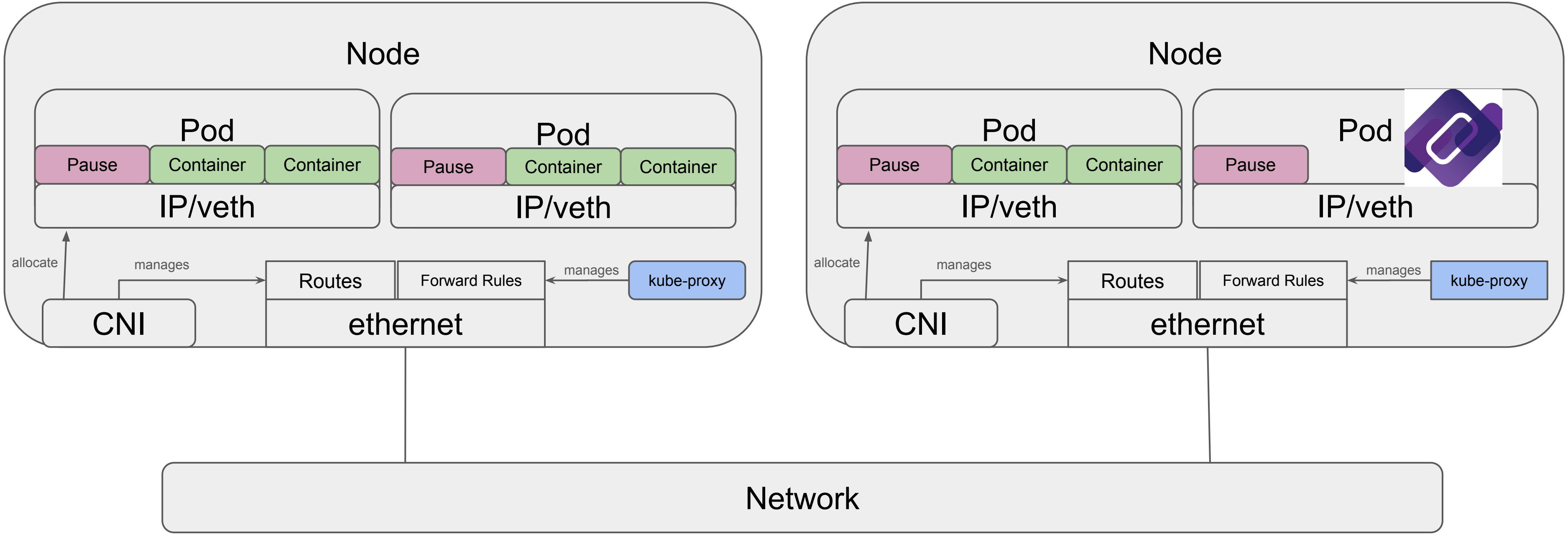



write()
kernel entry point
func()
function
function call
file in which function appears
description
driver-specific function
T name
data of type T
data ownership (writes or refers to)
data copy
f
virtual function through pointer f
thread scheduling (wake-up or start)
thread
schedulable thread
write()
sendto()
sendmsg()
user space
vfs_write()
do_sync_write()
fs/read_write.c
file->f_op
aio_write
char* base
iovec iov
sock_aio_write()
do_sock_write()
sock_sendmsg()
__sock_sendmsg()
net/socket.c
msghdr msg
sock->ops
sendmsg
skb_add_data()
tcp_sendmsg()
sk_buff*
tcp_write_queue
tcp_push()
net/ipv4/tcp.c
__tcp_push_pending_frames()
tcp_write_xmit()
tcp_transmit_skb()
tcp_push_one()
net/ipv4/tcp.c
skb_clone()
char* data
In device polling routine
free skb when DMA
async copy is done
specific device start xmit
(async) copy data to device
inet_connection_sock
->icsk_af_ops
queue_xmit
packet_rcv_spkt()
sk_buff* skb2
packet_rcv()
for tcpdump
skb_clone()
dev
hard_start_xmit
sk_buff* skb
net/packet/af_packet.c
packet_type
func
dev_queue_xmit_nit()
dev_hard_start_xmit()
net/core/dev.c
specific enqueue
qdisc
enqueue
sk_buff_head q
pfifo_fast_enqueue()
__qdisc_enqueue_tail()
qdisc_drop()
Qdisc
qdisc
requeue
dev_requeue_skb
qdisc_run()
__qdisc_run()
qdisc_restart()
dev_dequeue_skb
specific requeue
pfifo_fast_requeue()
__qdisc_requeue()
pfifo_fast_dequeue()
__qdisc_dequeue_head()
qdisc
dequeue
net/sched/sch_generic.c
specific dequeue
dev_queue_xmit()
sk_buff_head
completion_queue
net_device* dev
output_queue
softnet_data
netif_schedule()
__netif_schedule()
net_tx_action()
ksoftirqd
net/core/dev.c
qdisc_watchdog_schedule
watchdog timer
qdisc_watchdog()
net/sched/sched_api.c
ip_queue_xmit()
{ip_route_output_flow()}
ip_local_out()
{__ip_local_out();
nf_hook();}
ip_push_pending_frames()
ip_append_data()
icmp_push_reply()
dst_entry
output
ip_output()
{NF_HOOK()}
ip_finish_output()
{ip_fragment()}
ip_finish_output2()
net/ipv4/ip_output.c
neighbour/
hh_cache
output
sk_buff* skb
char* data
sk_buff_head
sk_write_queue
icmp_send()
{ip_route_output_key()}
arp_xmit()
{NF_HOOK()}
arp_send()
arp_create()
sk_buff* skb
char* data
arp_rcv()
{NF_HOOK()}
arp_process()
net/ipv4/arp.c
ip_forward()
{NF_HOOK()}
ip_forward_finish()
net/ipv4/ip_forward.c
skb_clone()
sk_buff* skb
ipmr_queue_xmit()
{NF_HOOK()}
ipmr_forward_finish()
ip_mr_input()
ip_mr_forward()
net/ipv4/ipmr.c
ip_rcv()
{NF_HOOK()}
ip_rcv_finish()
{ip_route_input()}
net/ipv4/ip_input.c
dst_input()
dst_entry
input
packet_type
func
deliver_skb()
tcp_data_snd_check()
tcp_rcv_established()
tcp_data_queue()
sock_queue_rcv_skb()
tcp_ack_snd_check()
tcp_rcv_state_process()
tcp_v4_rcv()
tcp_v4_do_rcv()
sk_buff_head
tp->out_of_order_queue
net/ipv4/tcp_input.c
tcp_send_ack()
sk_buff* skb
tcp_send_delayed_ack()
net/ipv4/tcp_output.c
sock_def_readable()
wake-up waiting proc
sk_wait_data()
sock
sk_data_ready
sk_buff_head
sk_receive_queue
sk_async_wait_queue
net_protocol
handler
raw_local_deliver()
raw_v4_input()
raw_rcv()
raw_rcv_skb()
net/ipv4/raw.c
sk_buff* skb
skb_clone()
ip_local_deliver()
{ip_defrag()}
{NF_HOOK()}
ip_local_deliver_finish()
net/ipv4/ip_input.c
sk_buff* skb
char* data
proto
recvmsg
sock_common_recvmsg()
net/core/sock.c
tcp_recvmsg
net/ipv4/tcp.c
iovec
__sock_recvmsg()
{security_sock_recvmsg()}
sock->ops
recvmsg
sock_aio_read()
do_sock_read()
file->f_op
aio_write
sock_recvmsg()
net/socket.c
msghdr msg
do_sync_read()
recvmsg()
recvfrom()
read()
vfs_read()
fs/read_write.c
skb_copy_datagram_iovec()
void* iov_base
user space
IRQ
prepare receive buffers
device interrupt handler
device rx handler
device NAPI handler
char* data
netif_rx_schedule()
__netif_rx_schedule()
napi_schedule()
__napi_schedule()
sk_buff*
input_pkt_queue
non-NAPI
softnet_data
ksoftirqd
netif_rx()
{netpoll_rx()}
net_rx_action()
napi_struct* poll_list
netif_rx_complete()
__netif_rx_complete()
napi_complete()
__napi_complete()
napi_struct
poll
device NAPI poll
process_backlog()
netif_receive_skb()
handle_bridge()
handle_macvlan()
handle_ing()
netpoll_rx()
__netpoll_rx()
netpoll_receive_skb()
net/core/dev.c

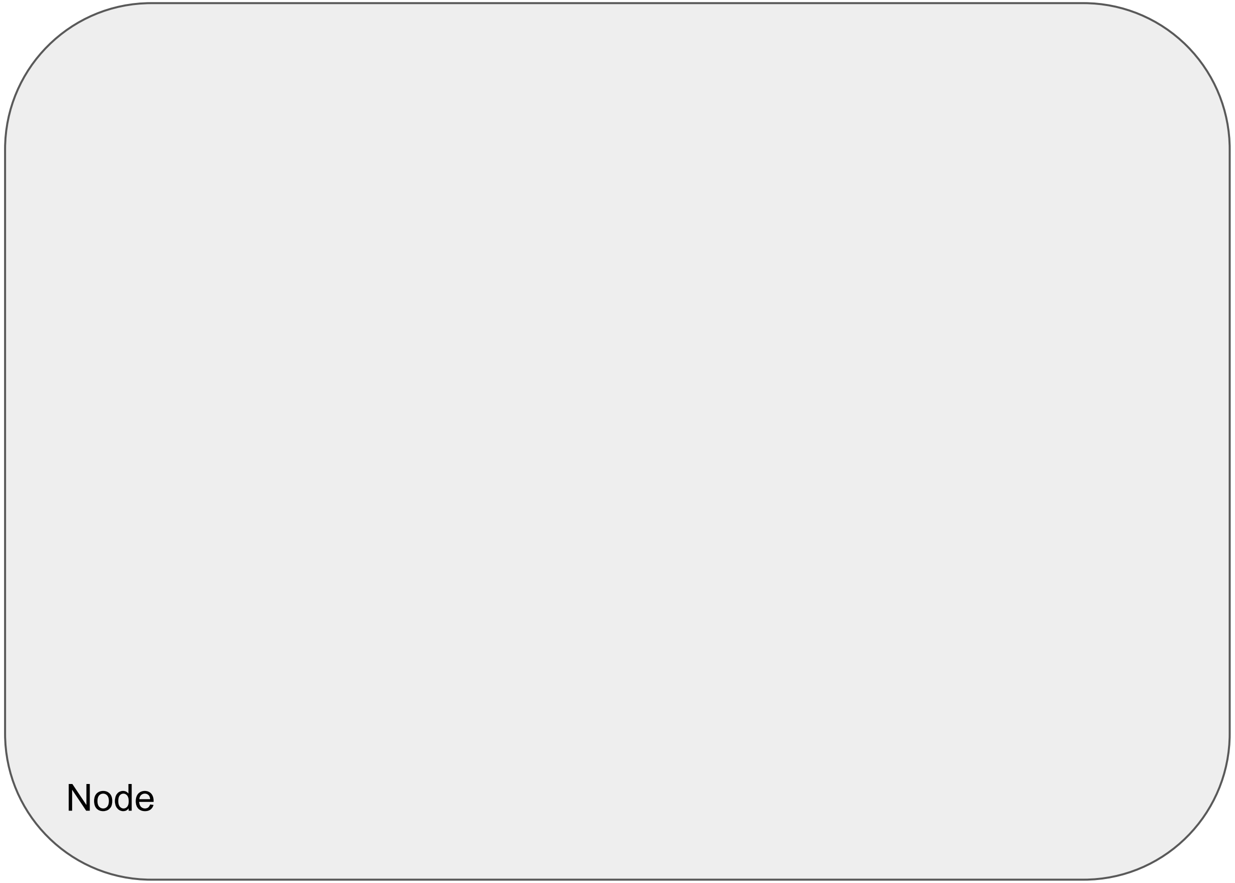
Agenda

- Linux Networking
- Pods
- Container Network Interface
- Building Kubernetes abstractions

Kubernetes Networking

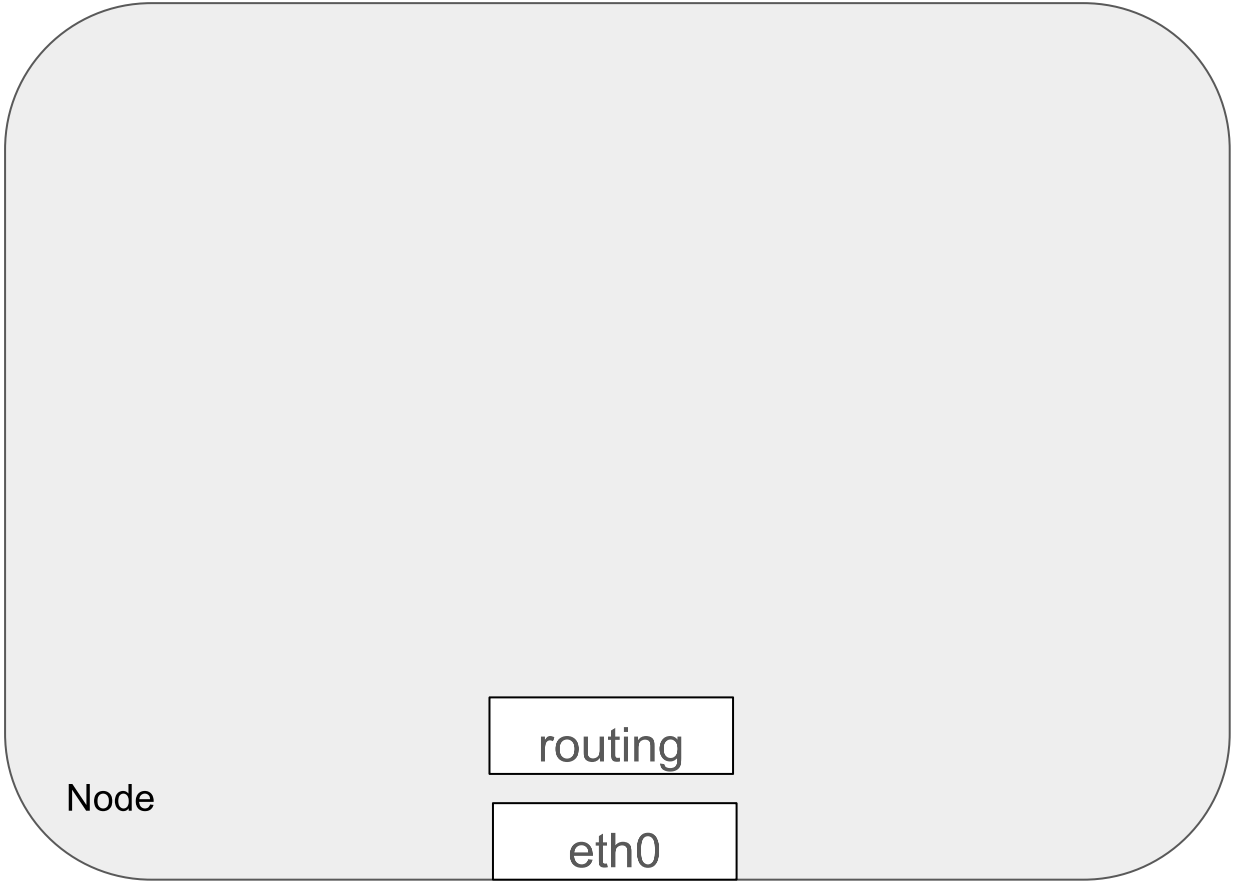


Linux Networking

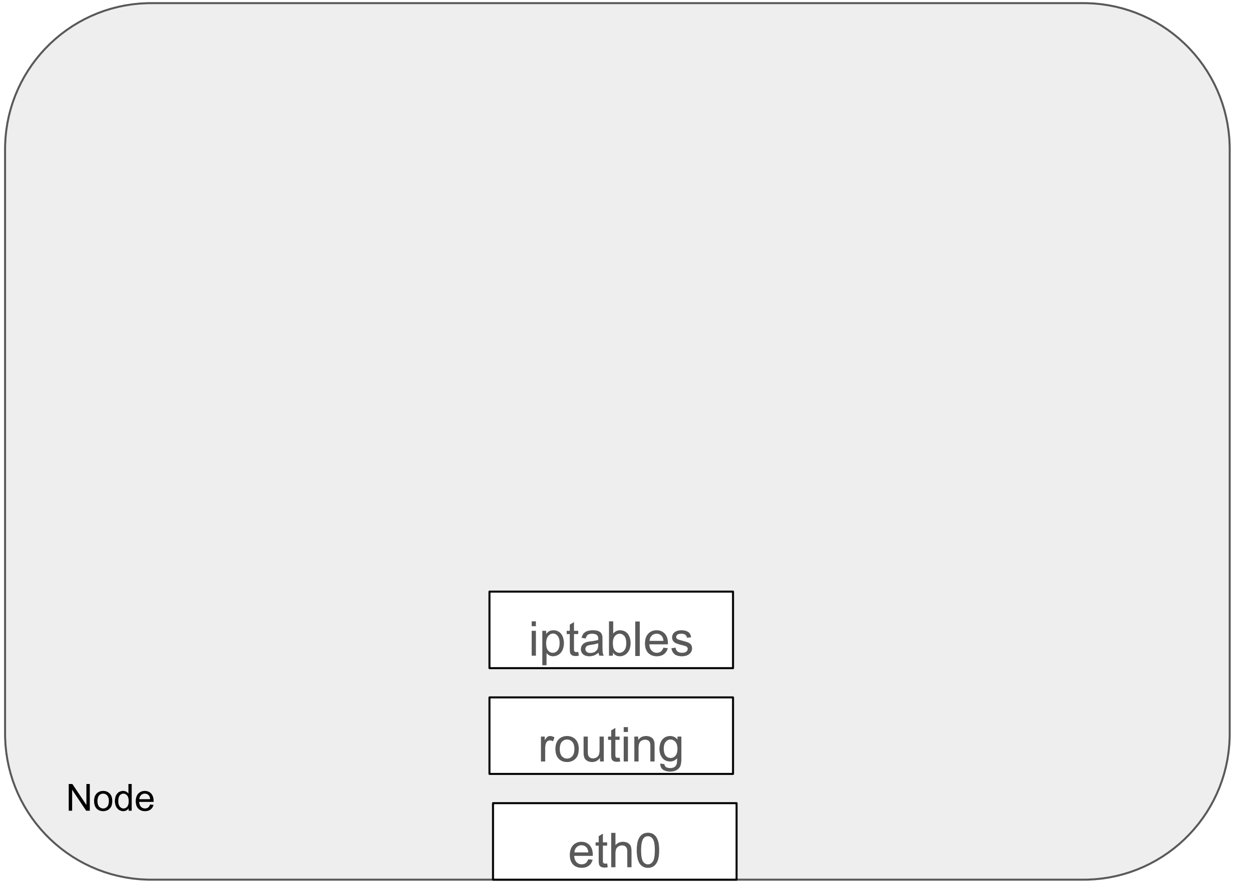


Linux Networking

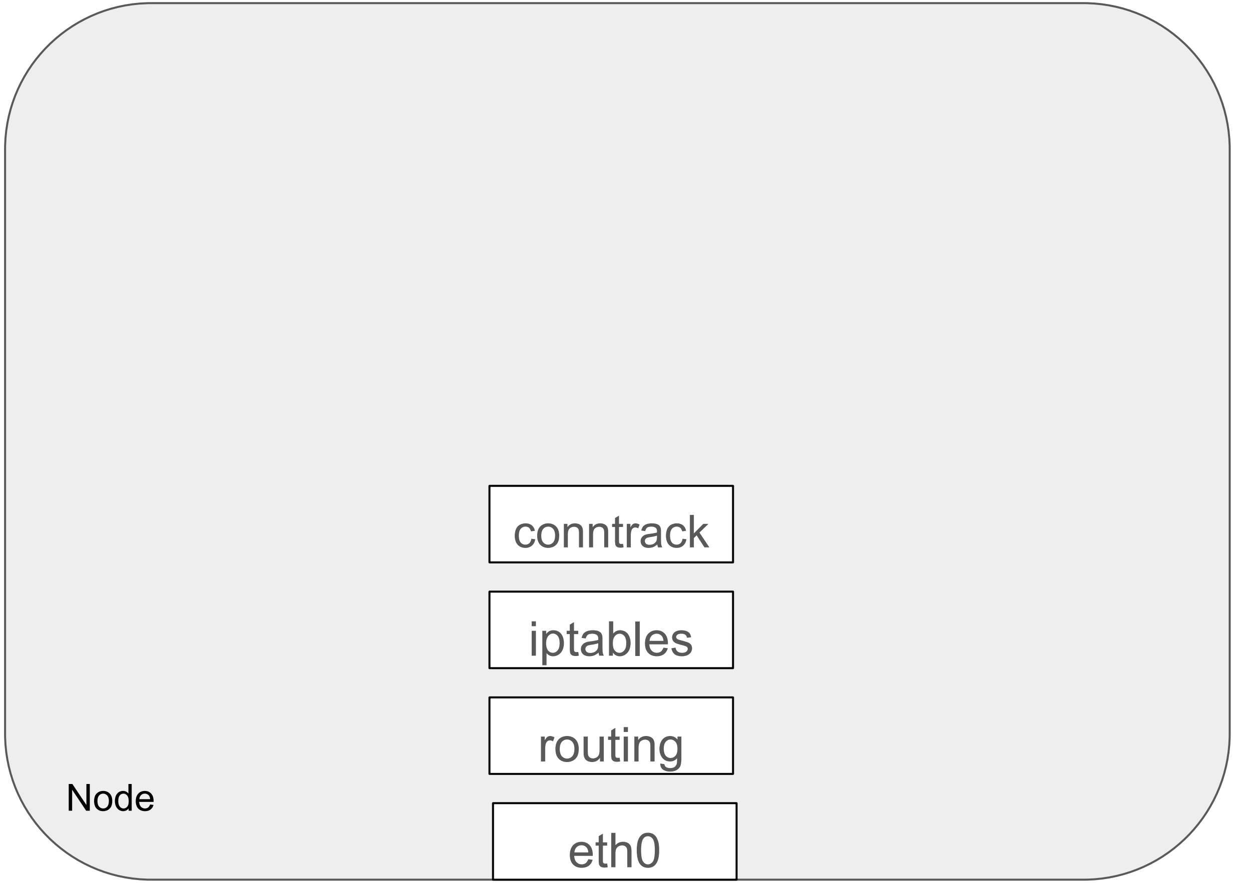
Linux Networking



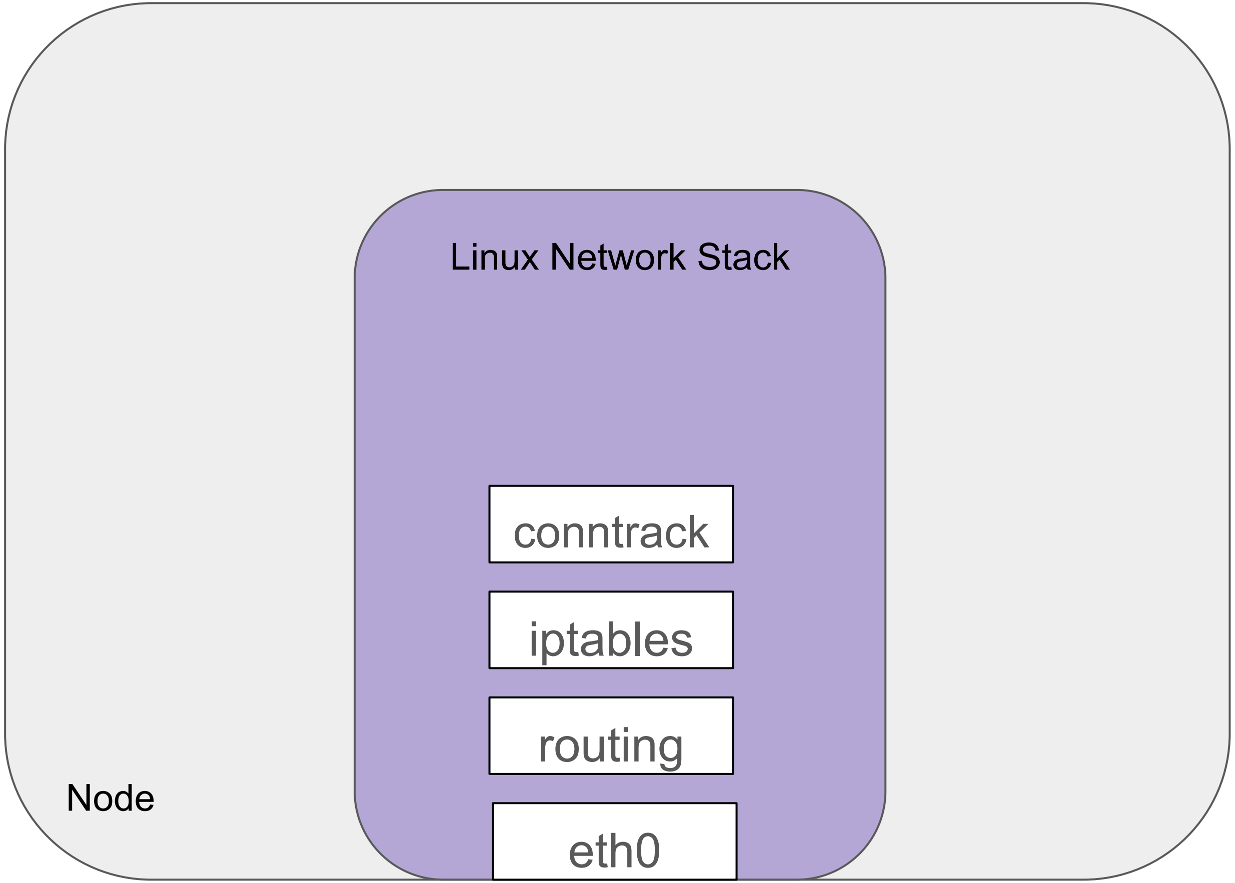
Linux Networking



Linux Networking



Linux Networking



Linux Network Stack

conntrack

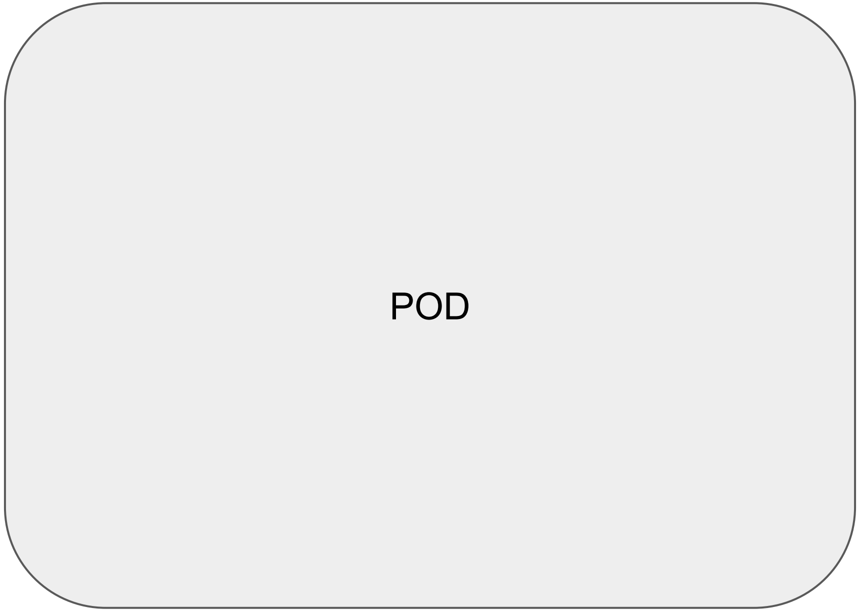
iptables

routing

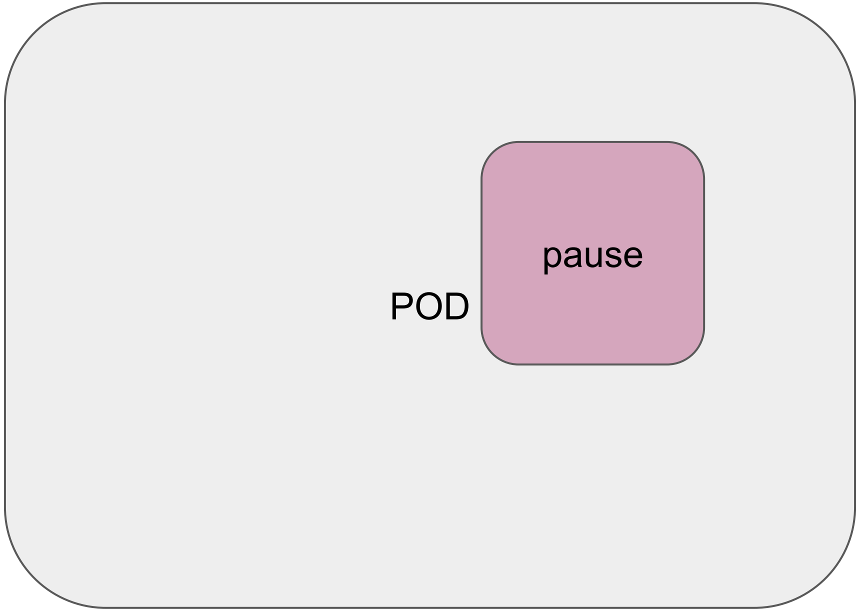
eth0

Node

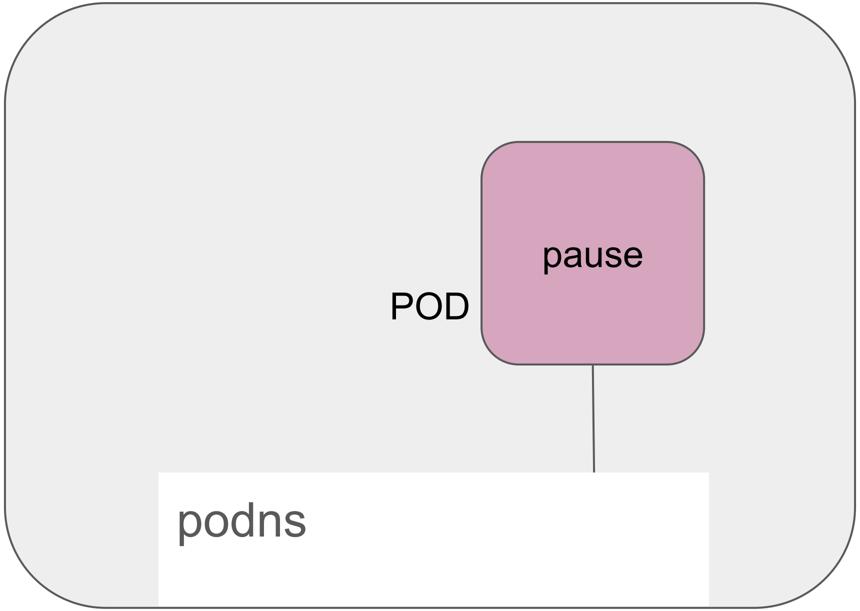
What’s in a Pod?



What’s in a Pod?

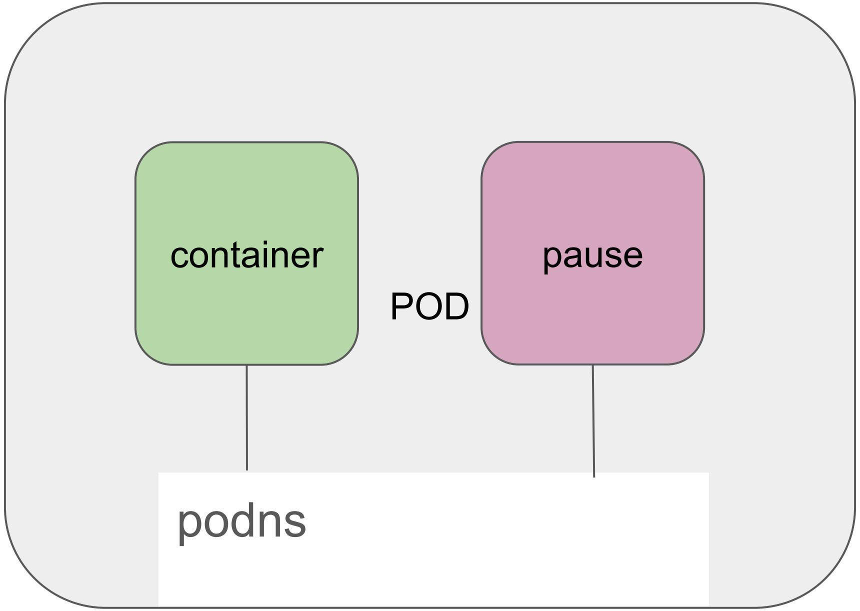


What’s in a Pod?

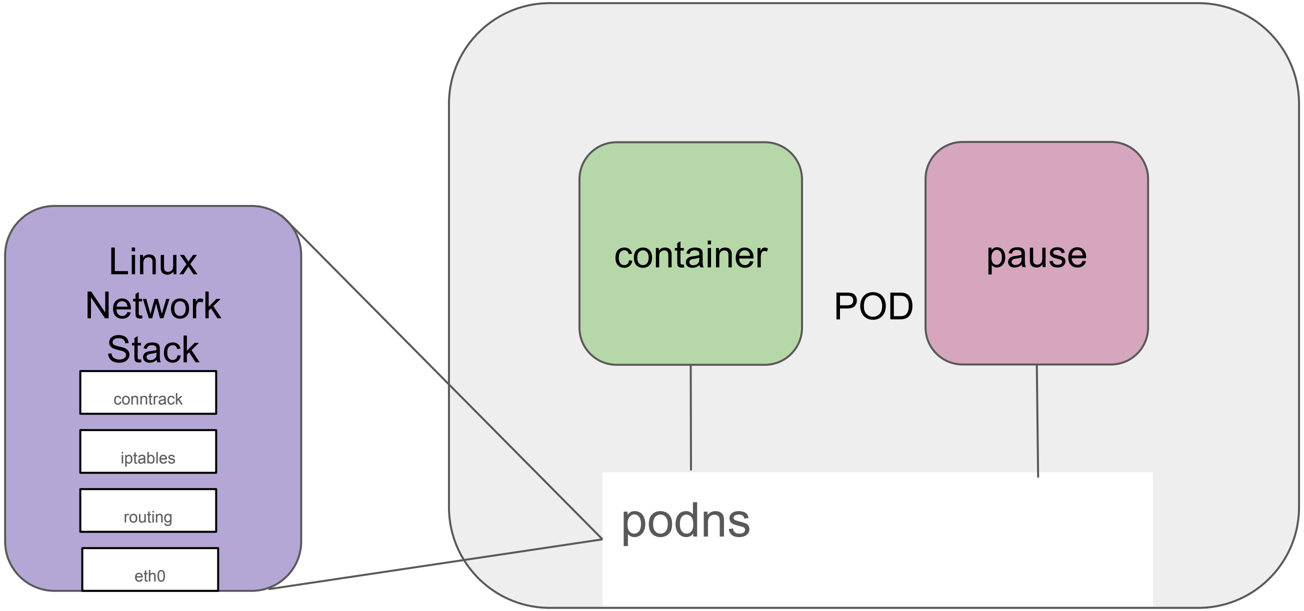


```
ip netns add podns
```

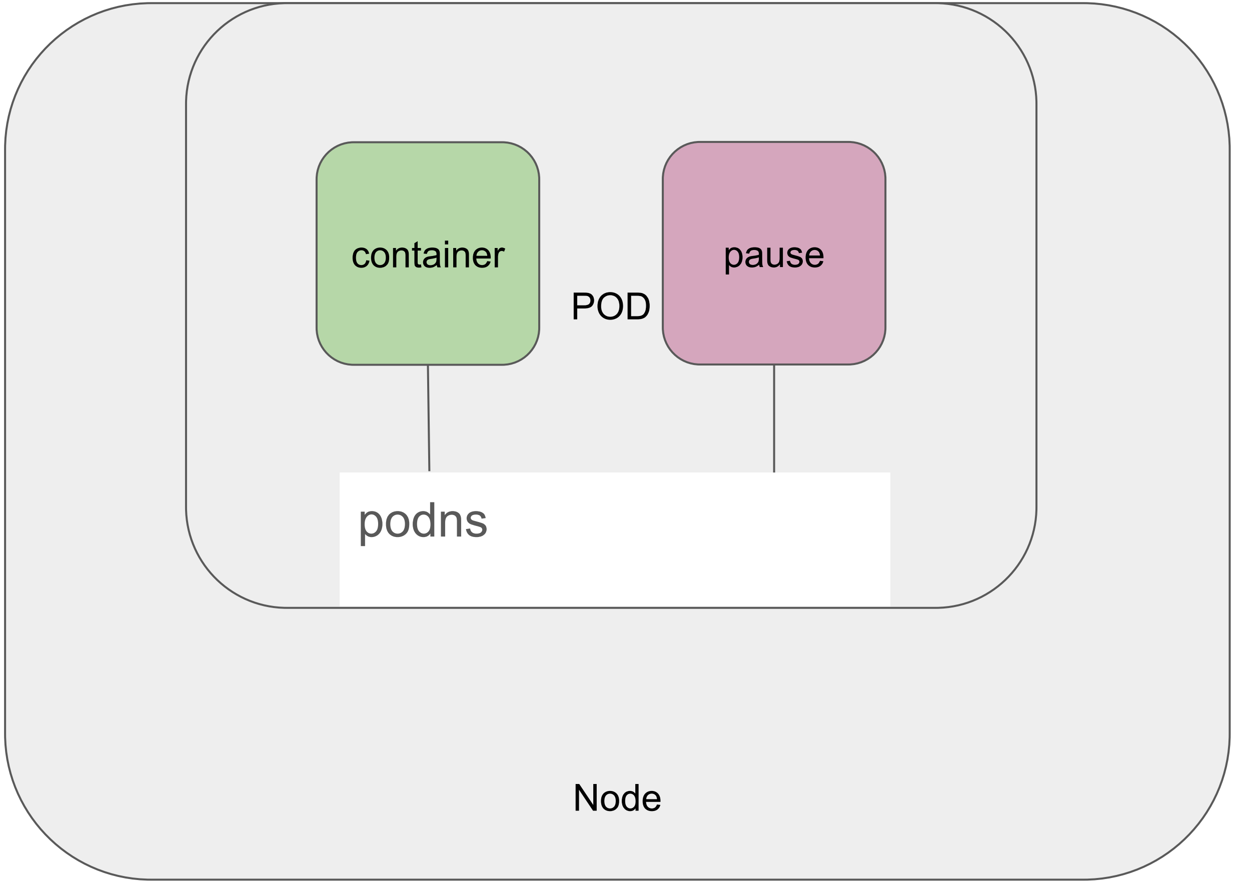
What's in a Pod?



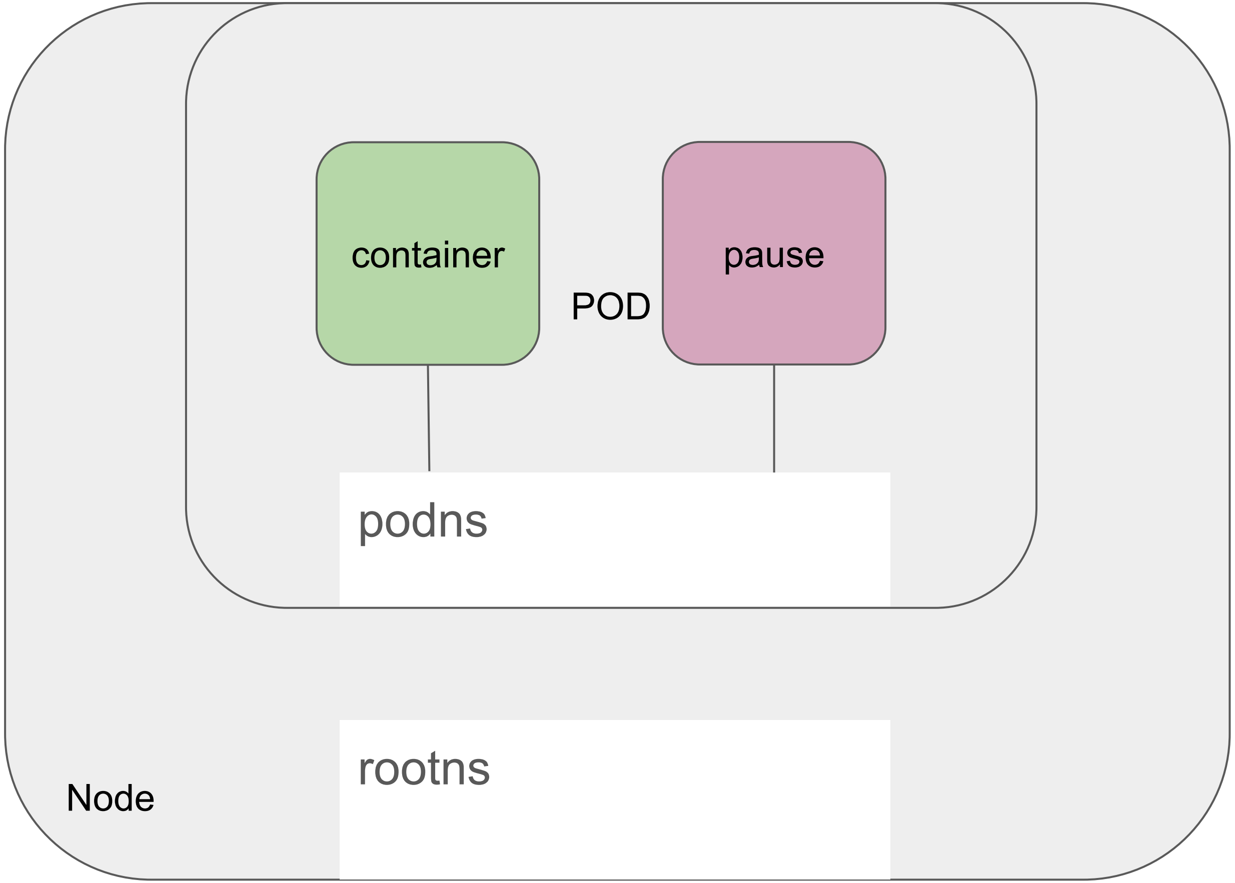
What’s in a Pod?



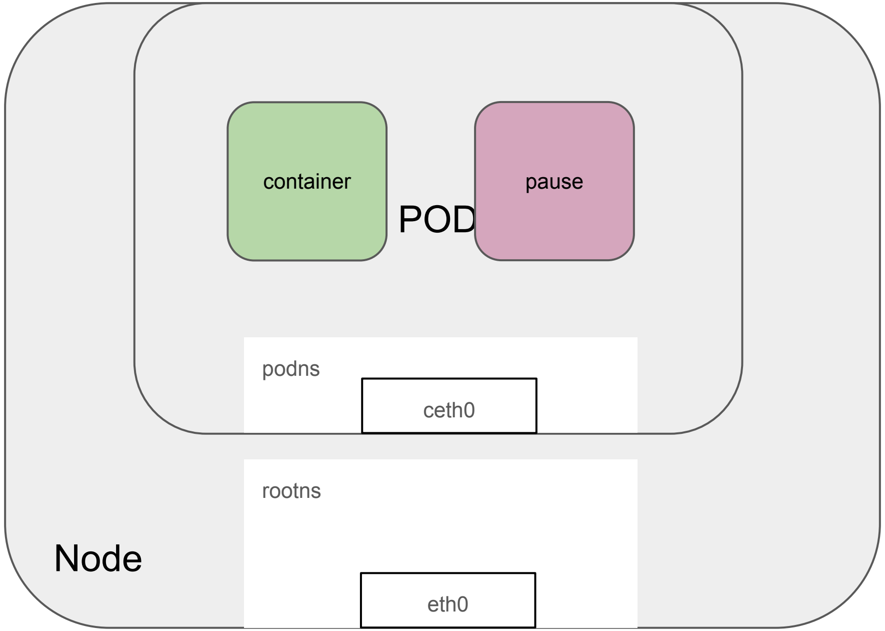
What’s in a Pod?



What’s in a Pod?

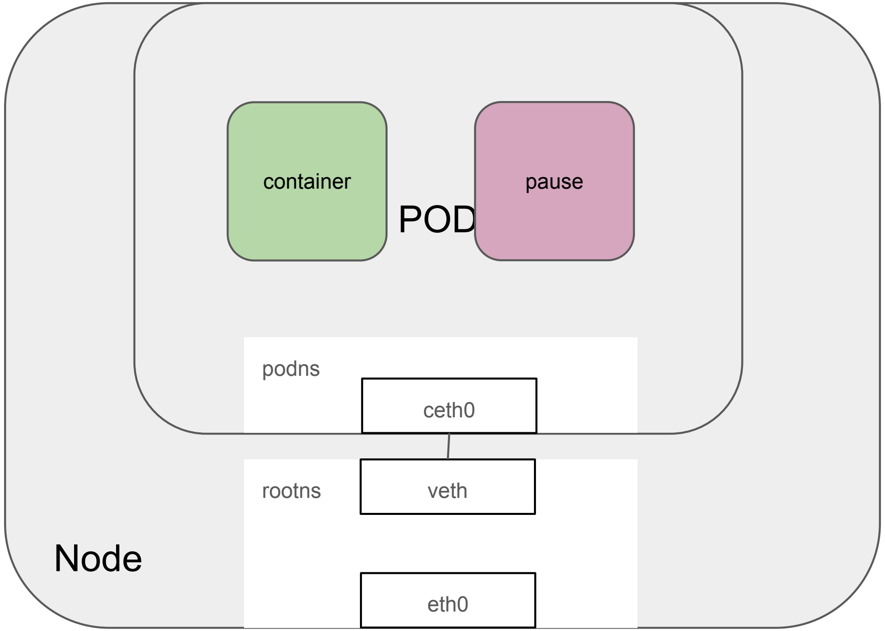


What’s in a Pod?



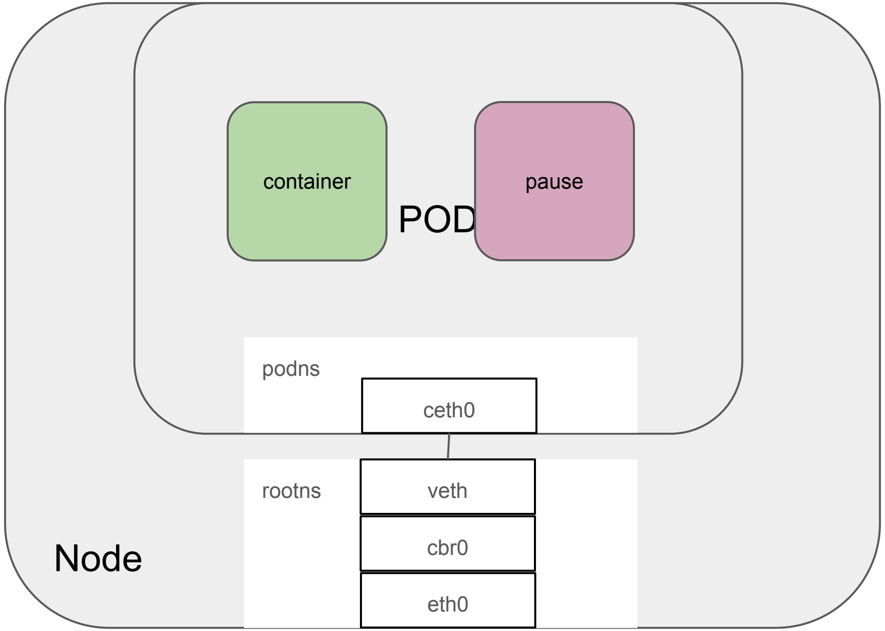
```
ip link set ceth0 netns podns
```

What’s in a Pod?



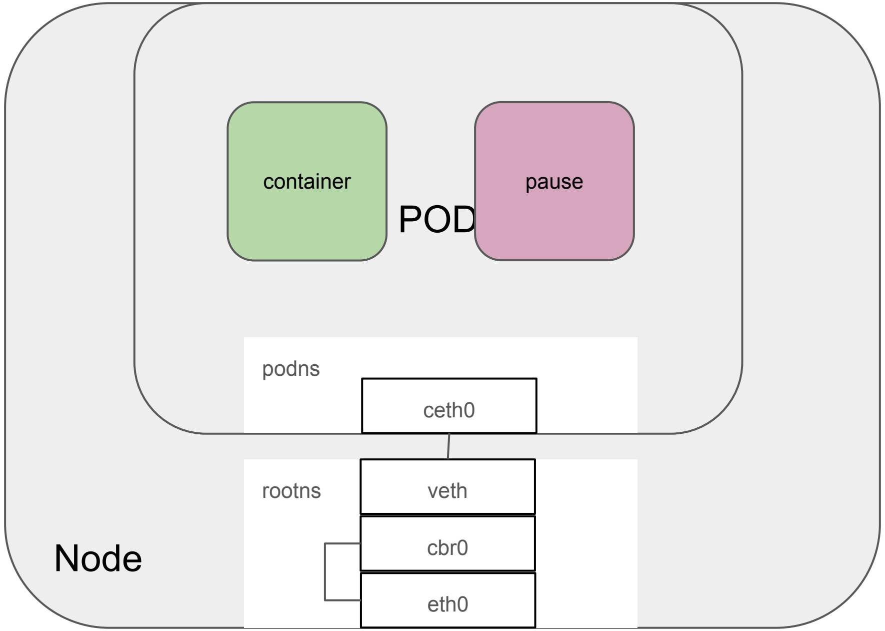
```
ip link add veth type veth peer name ceth0
```

What’s in a Pod?



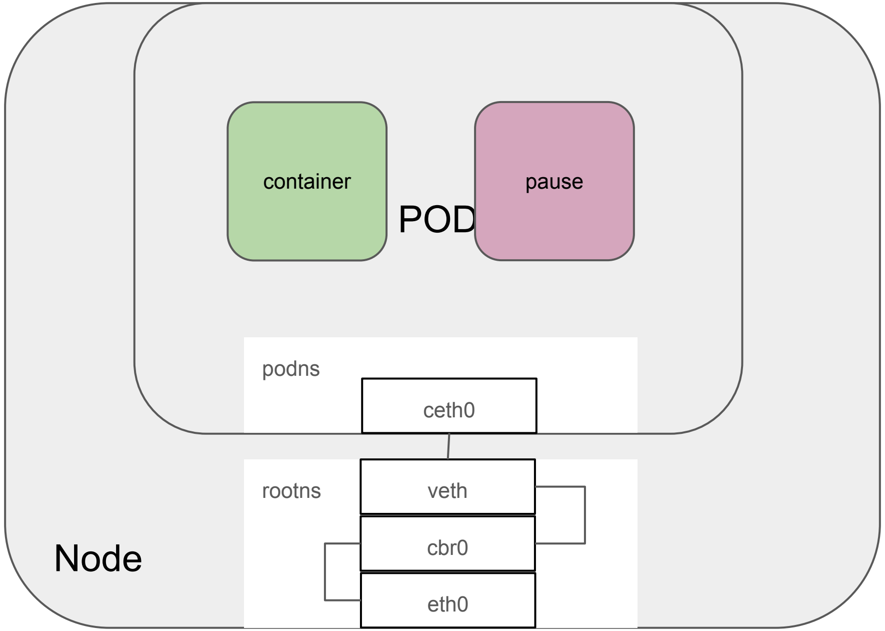
```
ip link add cbr0 type bridge
ip link set br0 up
```

What's in a Pod?



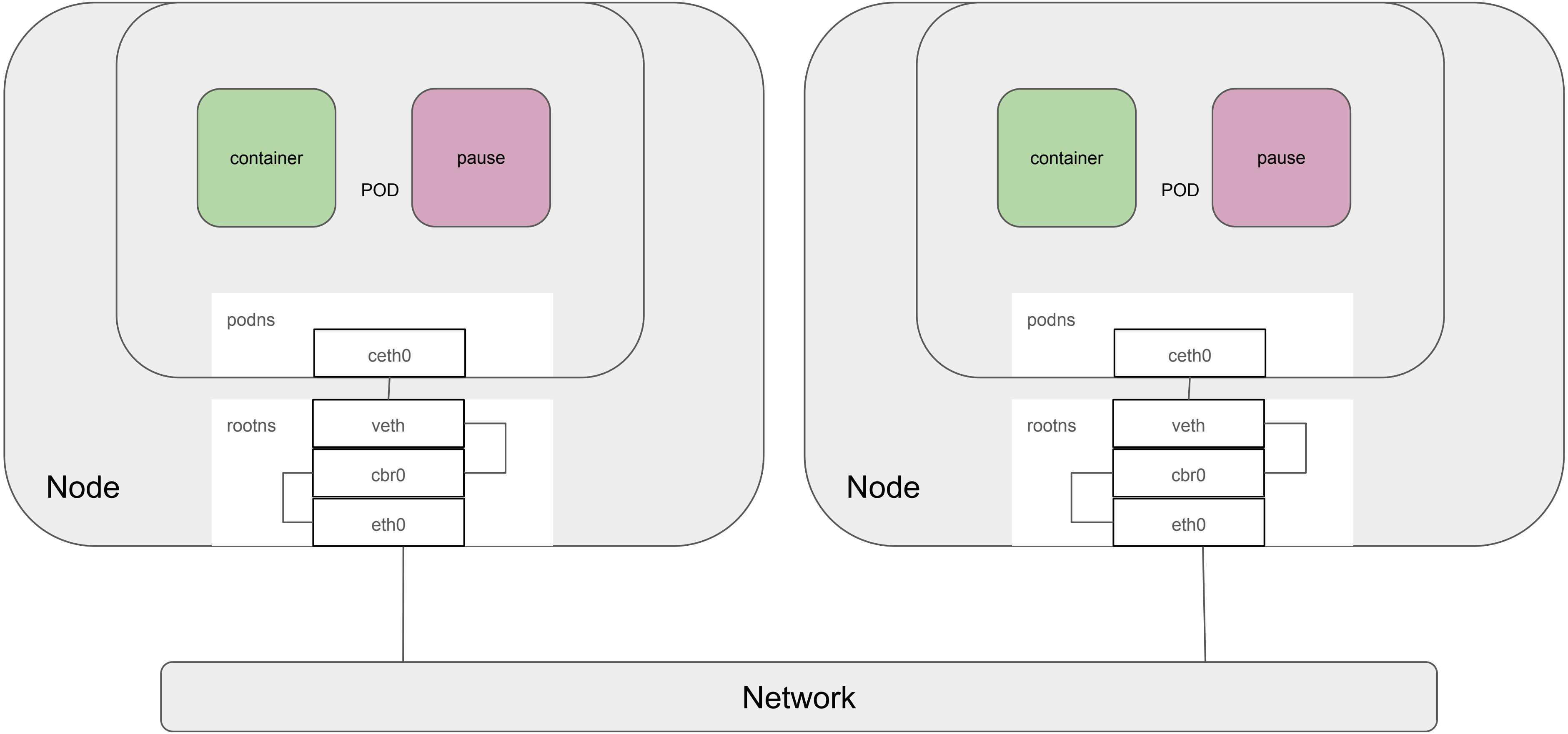
```
ip link set veth master cbr0
```

What’s in a Pod?



```
ip link set veth0 up
ip addr add 172.18.0.11/16 dev veth0
```

What’s in a Pod?



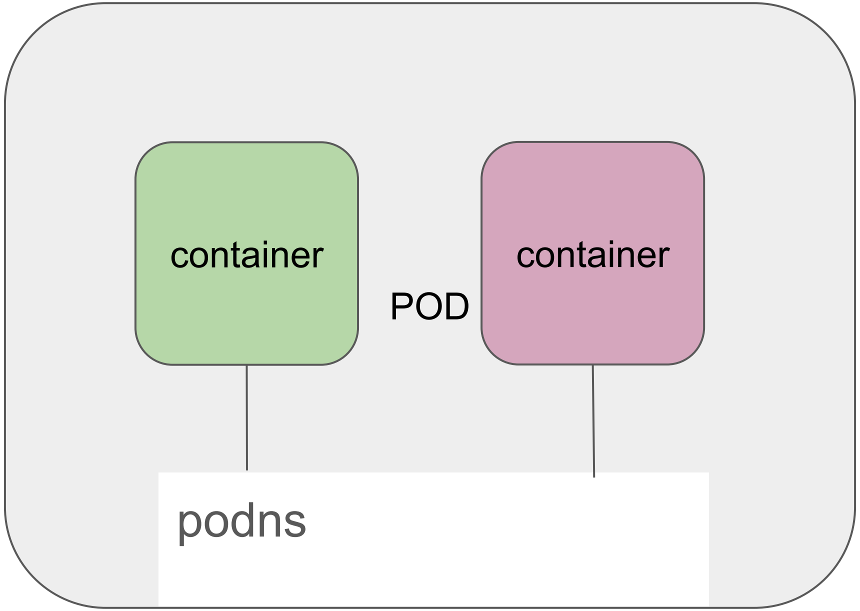
Kubernetes Networking Fight Club Rules

Kubernetes Networking Fight Club Rules

1. Highly-coupled container-to-container communications: this is solved by Pods and `localhost` communications.

Kubernetes Networking Fight Club Rules

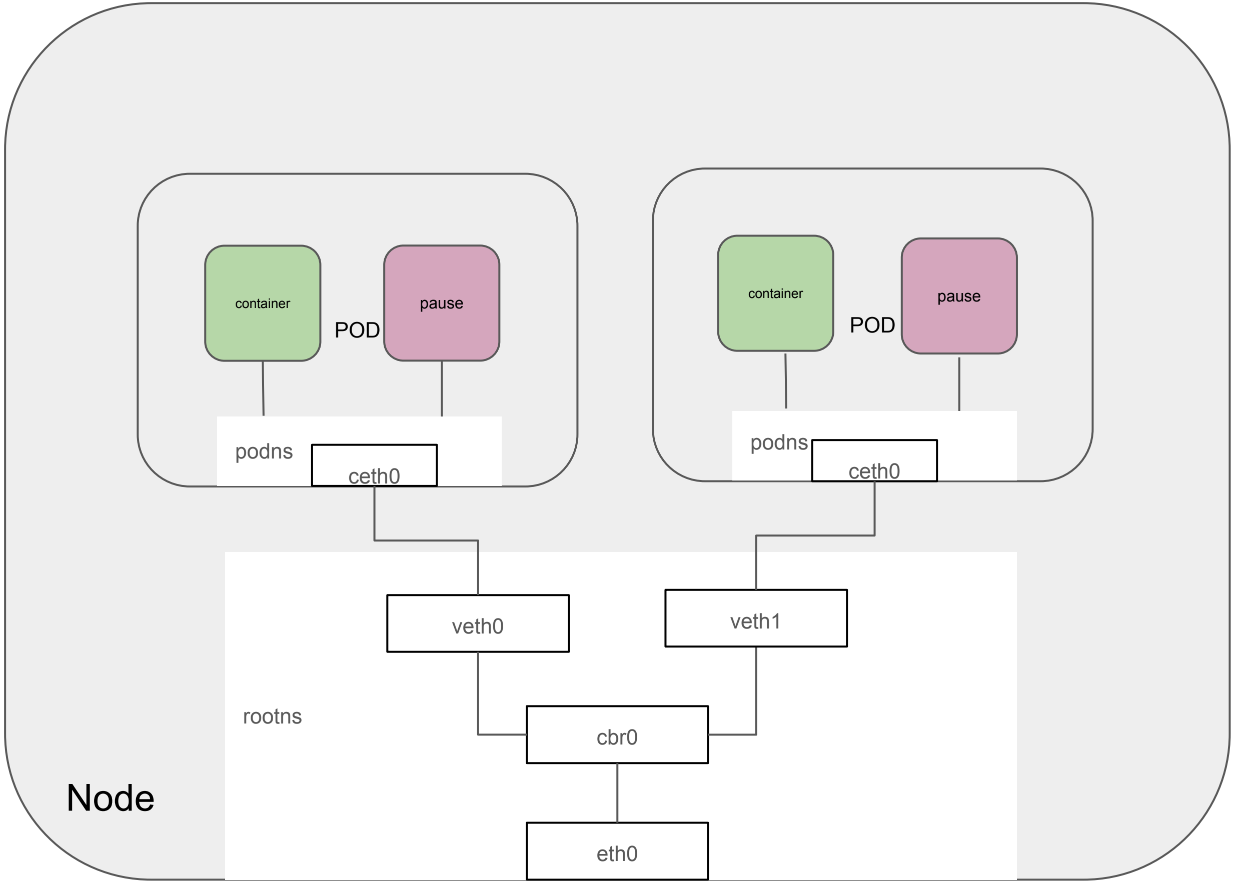
1. Highly-coupled container-to-container communications: this is solved by Pods and `localhost` communications.



Kubernetes Networking Fight Club Rules

1. Highly-coupled container-to-container communications: this is solved by Pods and `localhost` communications.

2. Pod-to-Pod communications: All Pods can communicate with other Pods

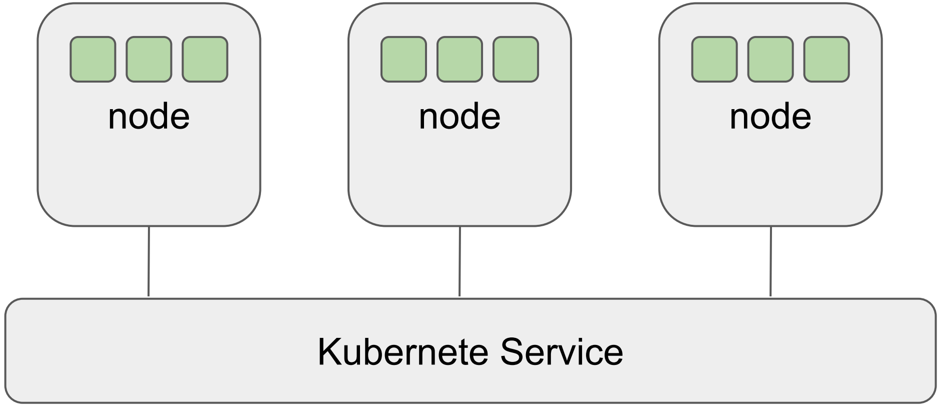


Kubernetes Networking Fight Club Rules

1. Highly-coupled container-to-container communications: this is solved by Pods and `localhost` communications.

2. Pod-to-Pod communications: All Pods can communicate with other Pods with their IP addresses

3. Pod-to-Service communications: this is covered by Services.



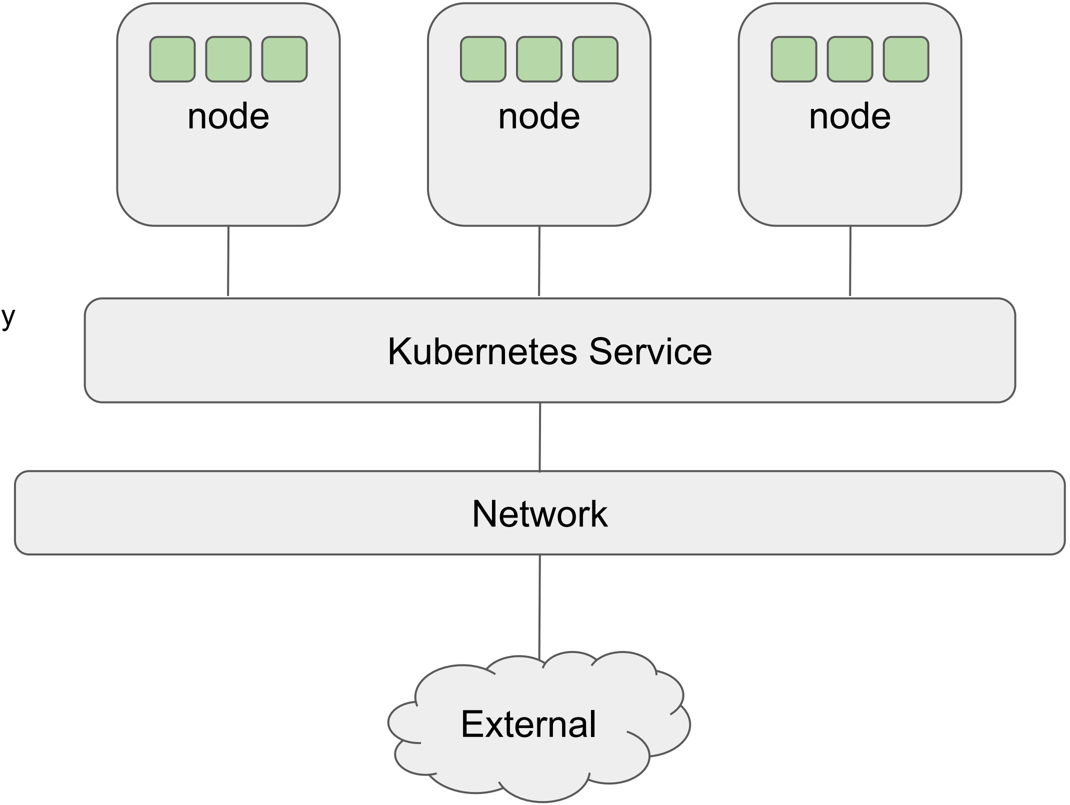
Kubernetes Networking Fight Club Rules

1. Highly-coupled container-to-container communications: this is solved by Pods and `localhost` communications.

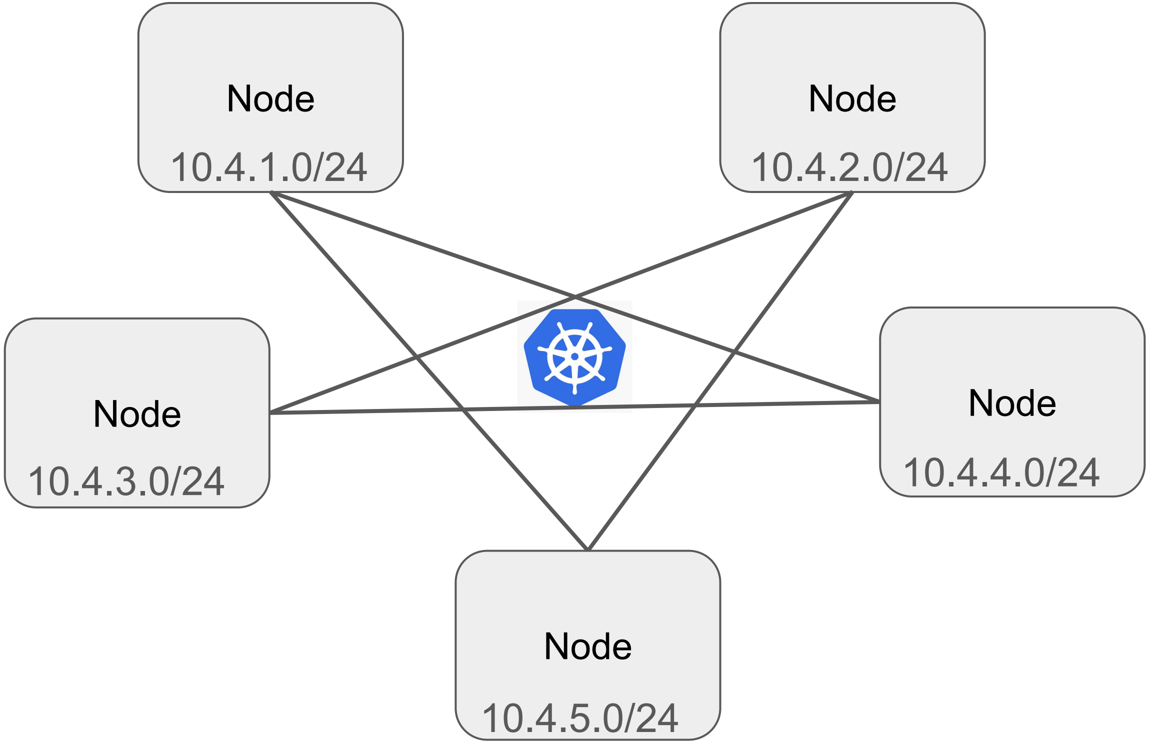
2. Pod-to-Pod communications: All Pods can communicate with other Pods

3. Pod-to-Service communications: this is covered by Services.

4. External-to-Service communications: this is also covered by Services.



Pod CIDR

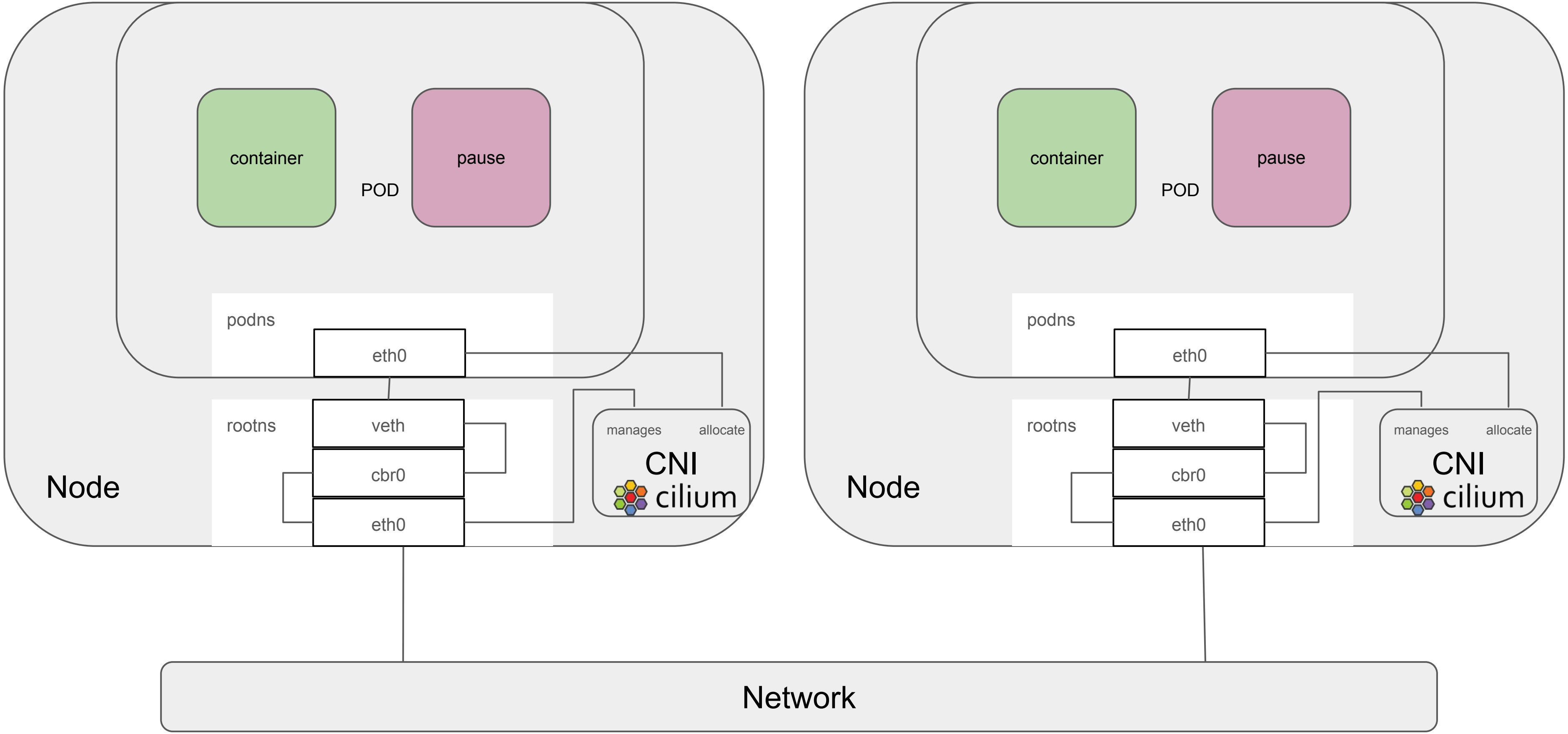

Node
10.4.1.0/24
Node
10.4.2.0/24
Node
10.4.3.0/24
Node
10.4.4.0/24
Node
10.4.5.0/24

Pod Demo

Container Network Interface

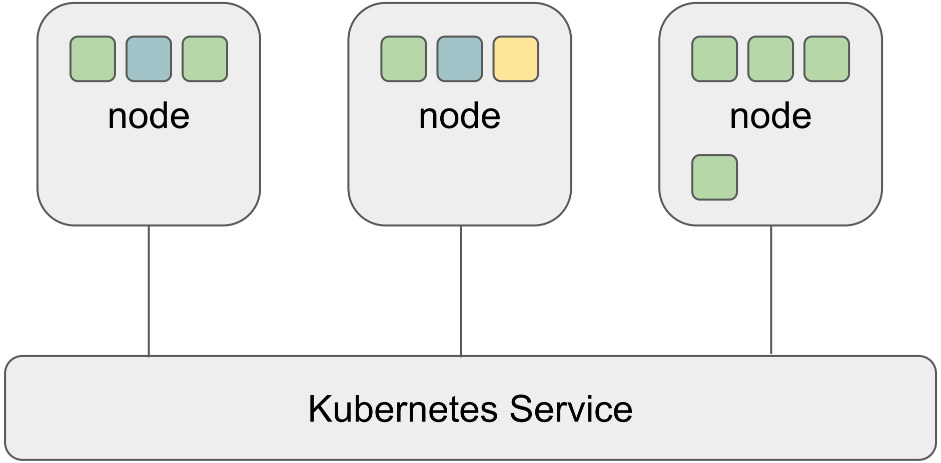
- Separate Software install
- Standard way to manage Network interfaces
- Lots of Options
  - Cilium
  - Kuberouter
  - Flannel
- https://github.com/containernetworking/cni
- A CNI plugin is required to implement the Kubernetes network model.

Container Network Interface

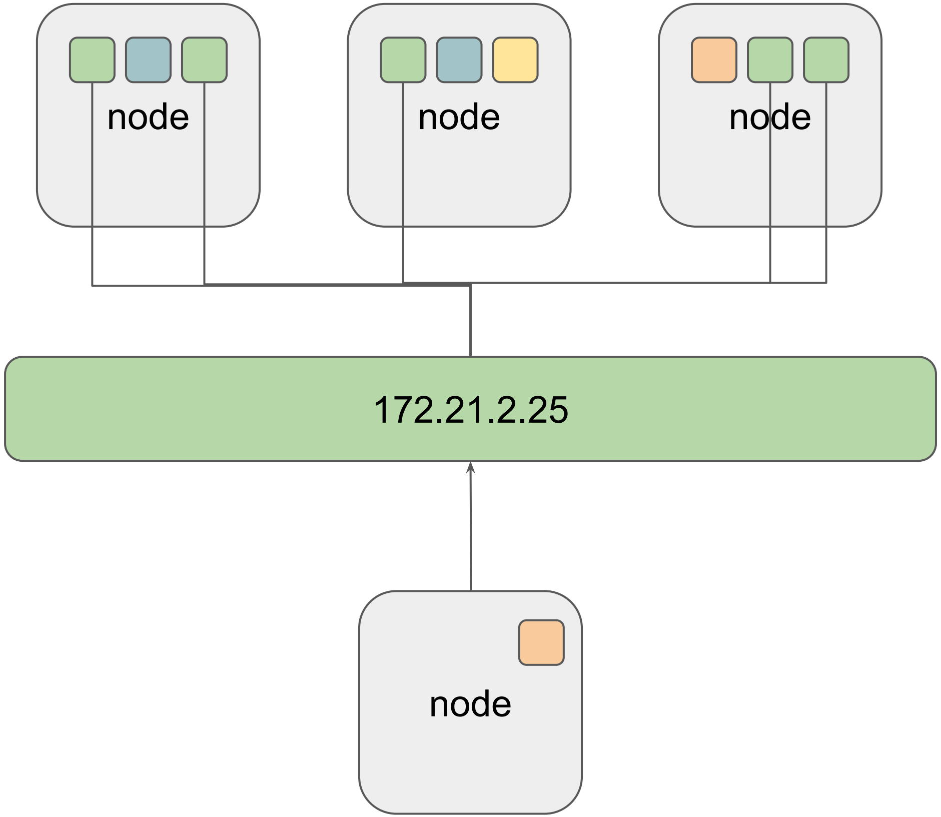


Services

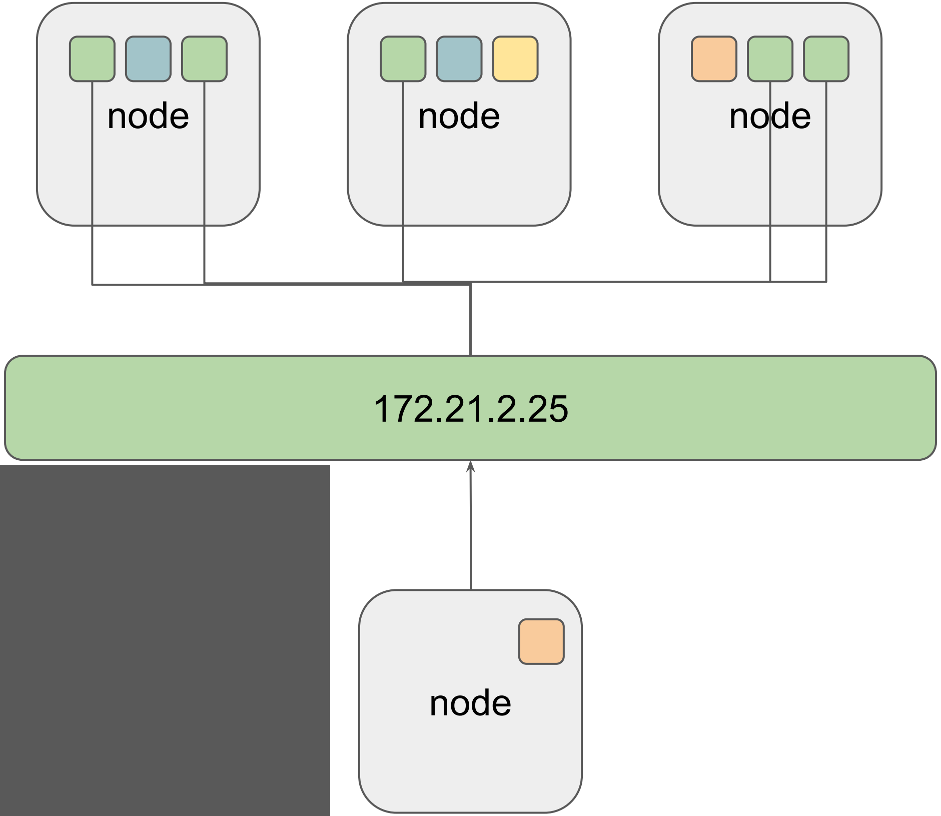
- Cluster IP
- Nodeport
- ExternalName
- Load Balancer
- Headless



Services CIDR

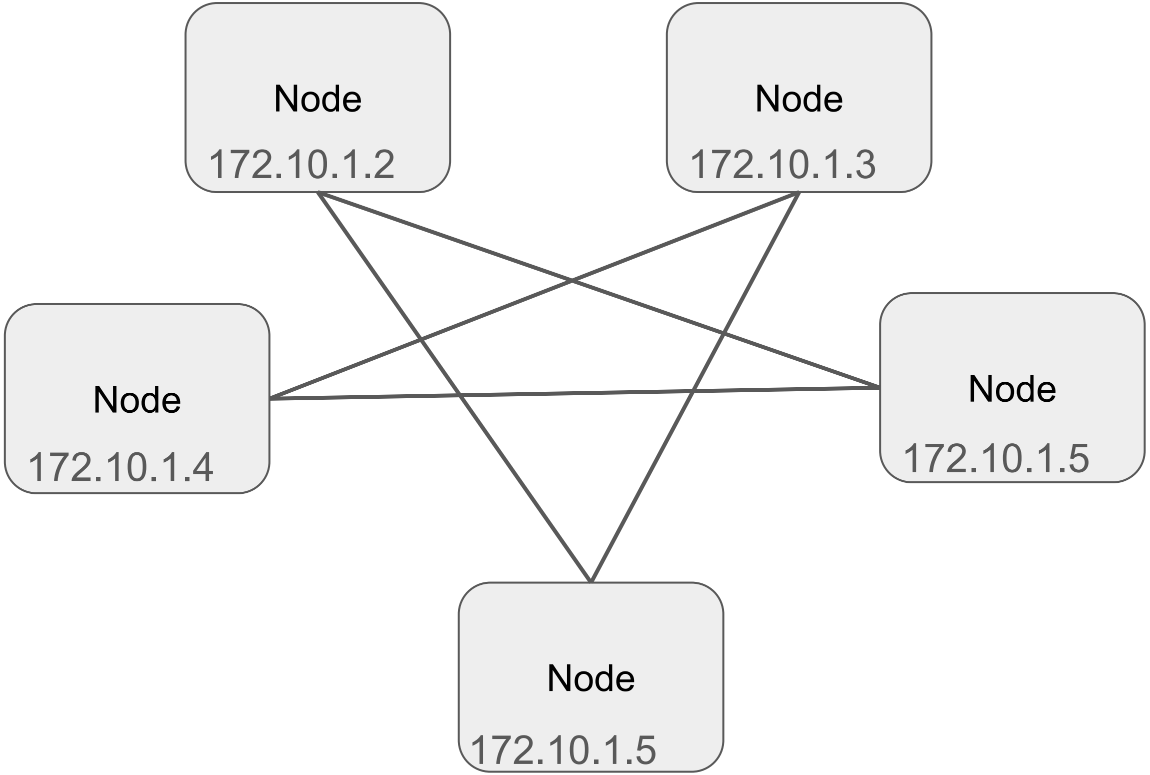


Services - ClusterIP

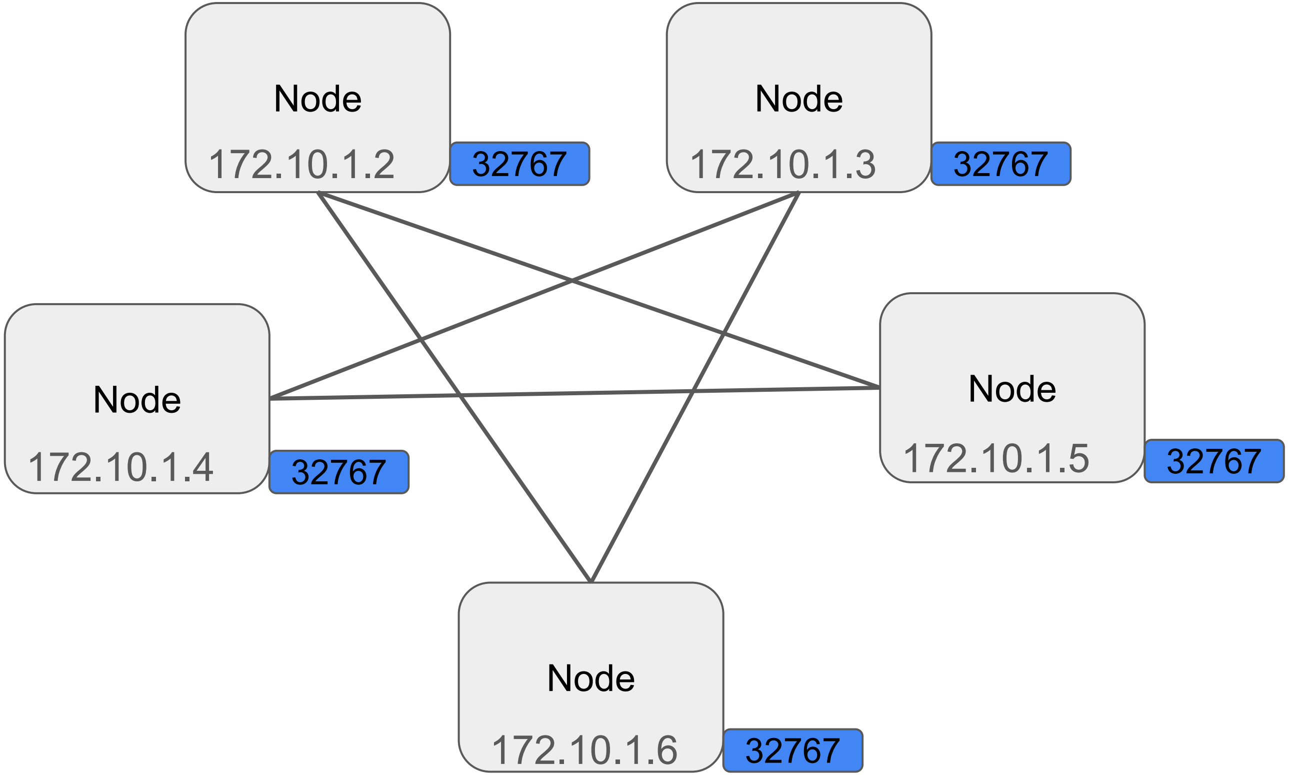


```
iptables \
  --table nat \
  --append APP-SVC-HTTP \
  --destination 172.21.2.25 \
  --protocol tcp \
  --match tcp \
  --dport 8080 \
  --jump DNAT \
  --to-destination 10.0.0.11:8080
```

Services - NodePort



Services - NodePort

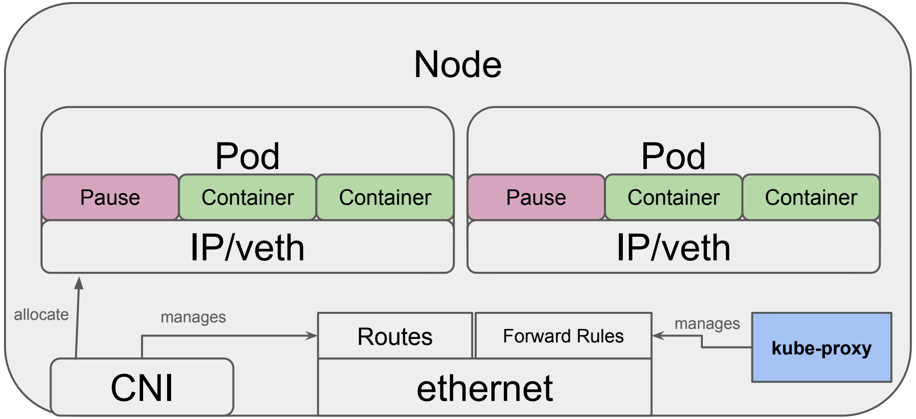


KubeCon | CloudNativeCon
North America 2024

Services Demo

Kube Proxy

- Kubernetes network proxy runs on each node
- Maintains network rules on nodes to implement Services
- Uses the operating system packet filtering layer (iptables/nftables)
- Routes traffic between nodes in a cluster
- Service-to-Pod mapping to work, you need continuous re-mapping



```
iptables -t nat -L PREROUTING
```

KubeCon | CloudNativeCon

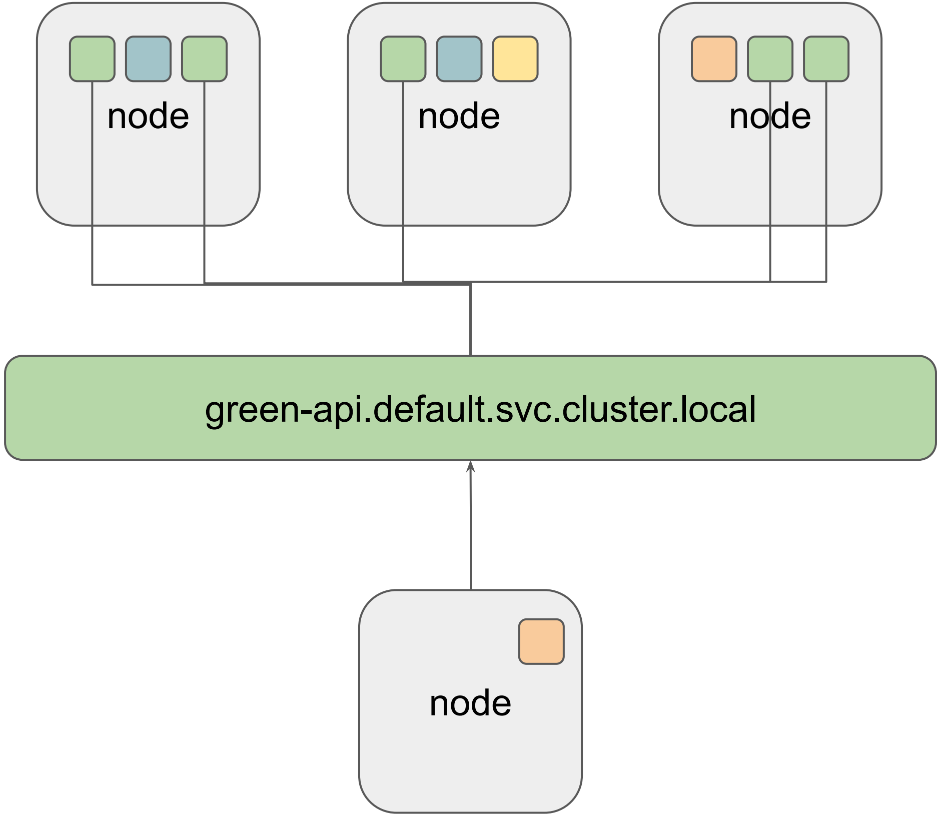
North America 2024

Kube-proxy Demo

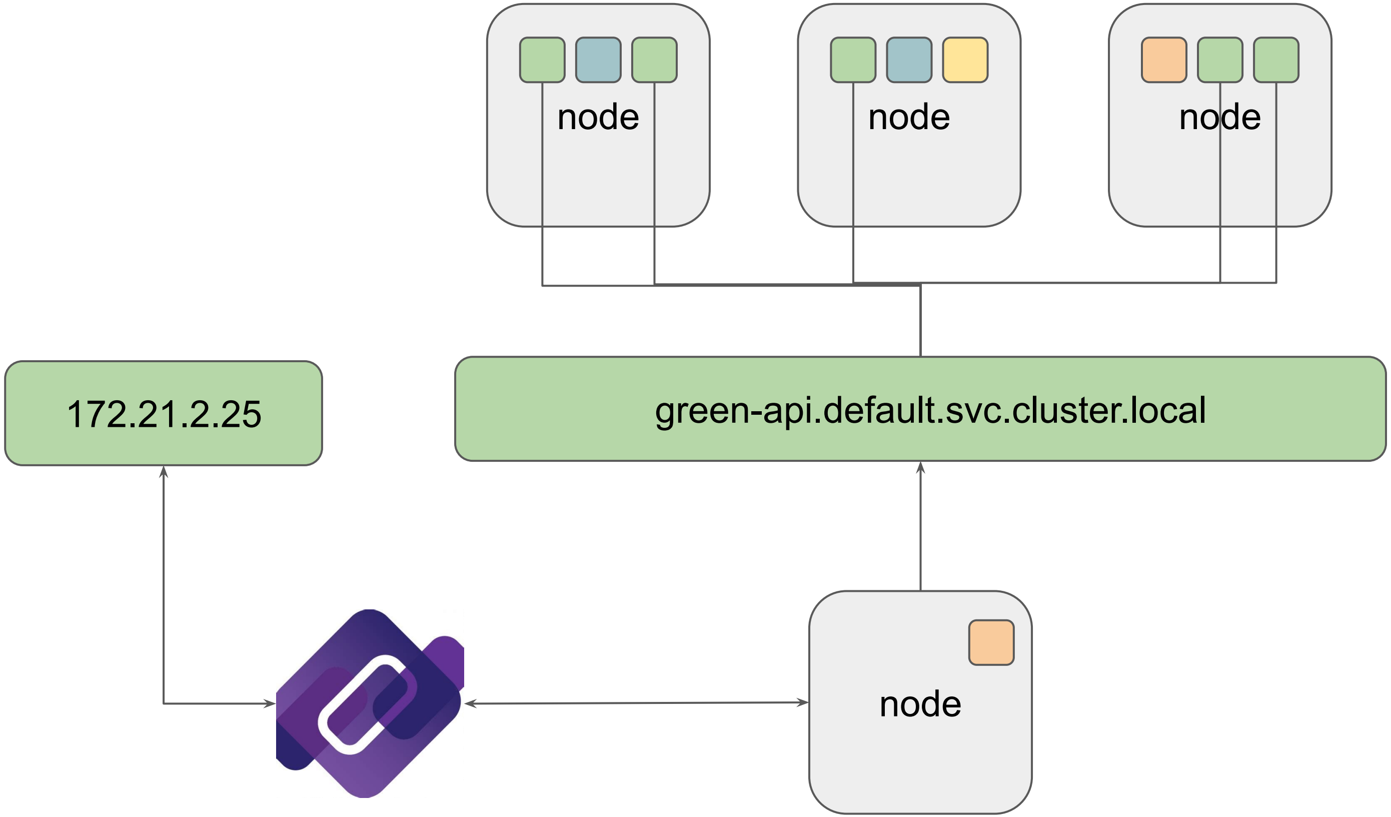
What about a name instead of an IP?

- Services can be resolved from it's name, like mydb.somens.svc.cluster.local
- CoreDNS that runs on the cluster is responsible of doing it
- Usually, CoreDNS can be reached on your cluster by…a Service!
- And by convention, CoreDNS service IP will always be the 10th IP of your Service IP range
    - eg.: If service range is 10.96.0.0/16, CoreDNS service IP will be 10.96.0.10
    - Kubernetes API server will be always reachable inside the cluster with the first IP of the range - 10.96.0.1

DNS



DNS

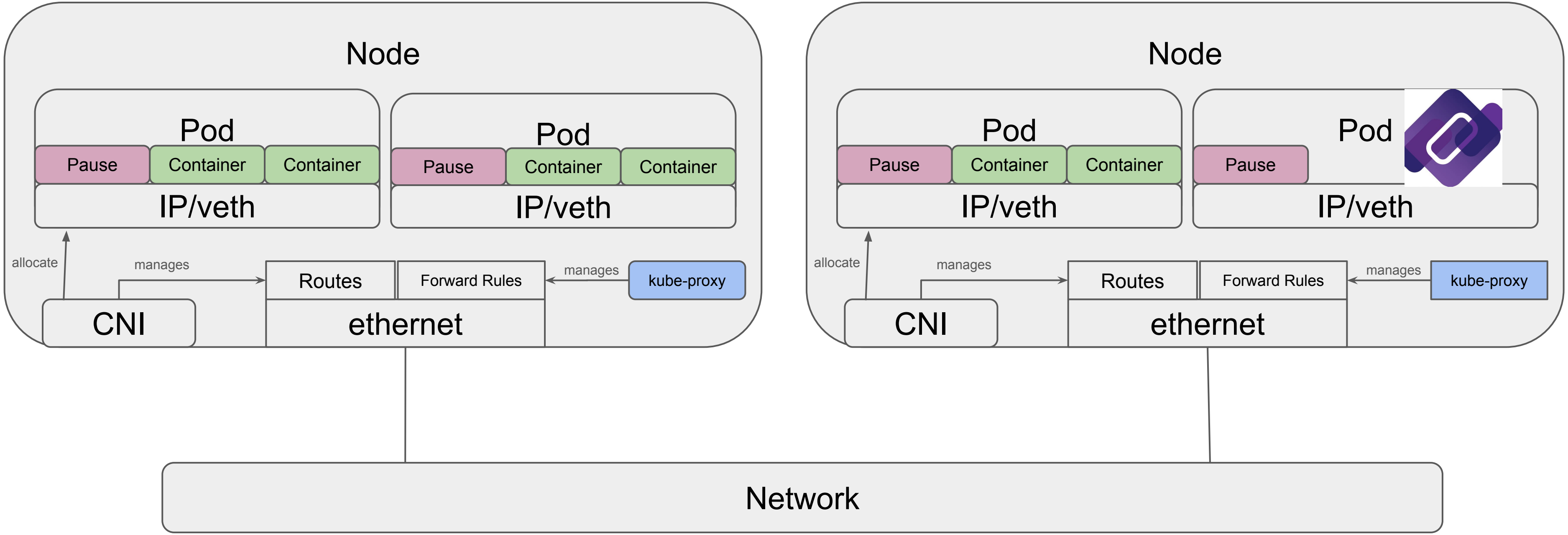

node
node
node
172.21.2.25
green-api.default.svc.cluster.local
node

KubeCon CloudNativeCon

North America 2024

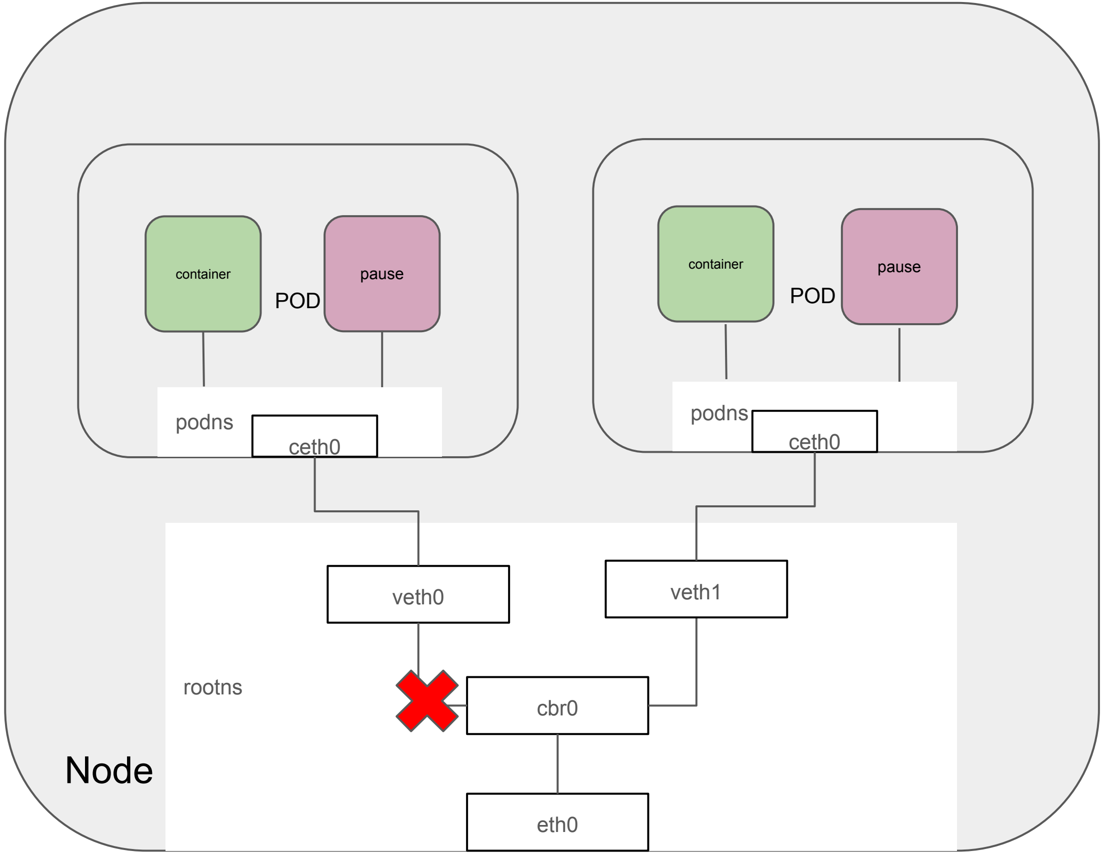
DNS Demo

How it looks like in the end



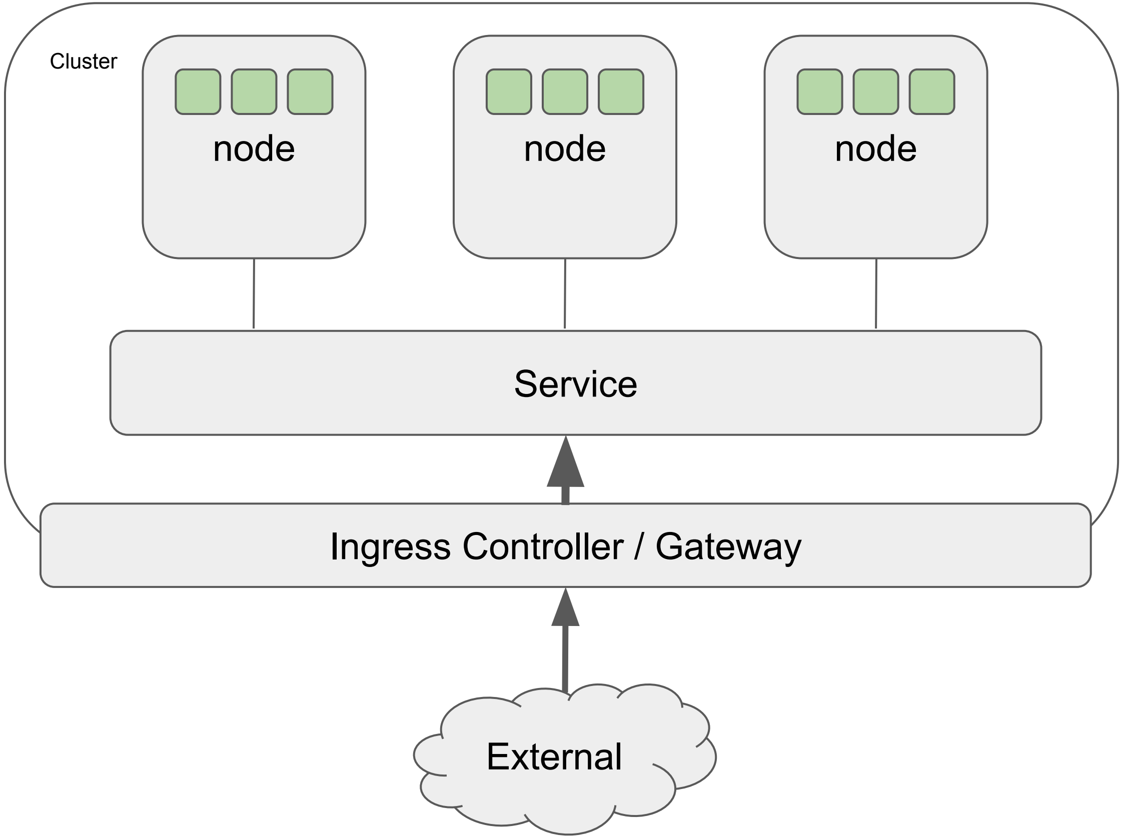
Other components to consider

- Network Policy - Another component that will create "firewall rules" on your node to control the traffic

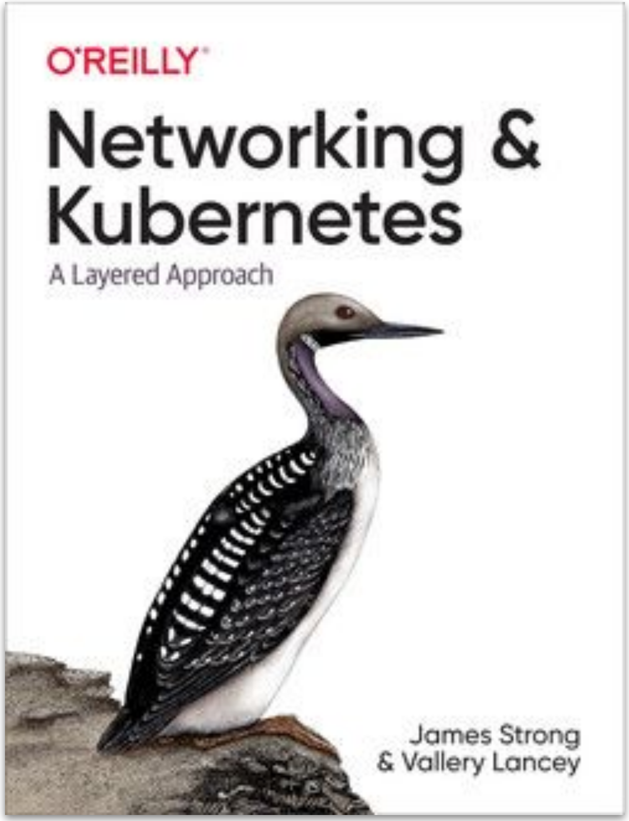


Other components to consider

- Ingress Controllers and Gateway API Controllers - Manages Pods that will do more complex traffic ingresses to the cluster



Resources


O'REILLY

Networking & Kubernetes

A Layered Approach

James Strong & Vallery Lancey

- Cilium Networking Labs
- Kubernetes Documentation
- Certified Kubernetes Administrator: Networking Part 1 with Marino Wijay
- Kevin Sookocheff A Guide to the Kubernetes Networking Model
- The Kubernetes Network Guide

Thank you

Presentation Survey

Gateway API Survey