# Kubernetes Networking

## What to expect in the talk

- SHORT and BRIEF intro to general networking(Ethernet + IP encapsulation)
- Docker networking
- IPs in Kubernetes
  - Node Address
  - Pod address
  - Service address

## How the Demo is set up

- Learn thru doing:
  - git clone git@github.com:IBM/mvs-oss-presentation.git
  - cd pavel-malinov/kubernetes-networking

Requirements for this talk:

1. Docker
2. Footloose

Or just have fun and explore on other cluster setups. I plan to use k3s. KIND and K3D will not be proper to use as they rely on docker for the networking and the interaction is a wonky.

## Network encapsulation

To make sure we are on the same page, let’s talk about how encapsulation works in networking

I am making few assumptions here:

- Only IPv4
- Ethernet

## Network encapsulation

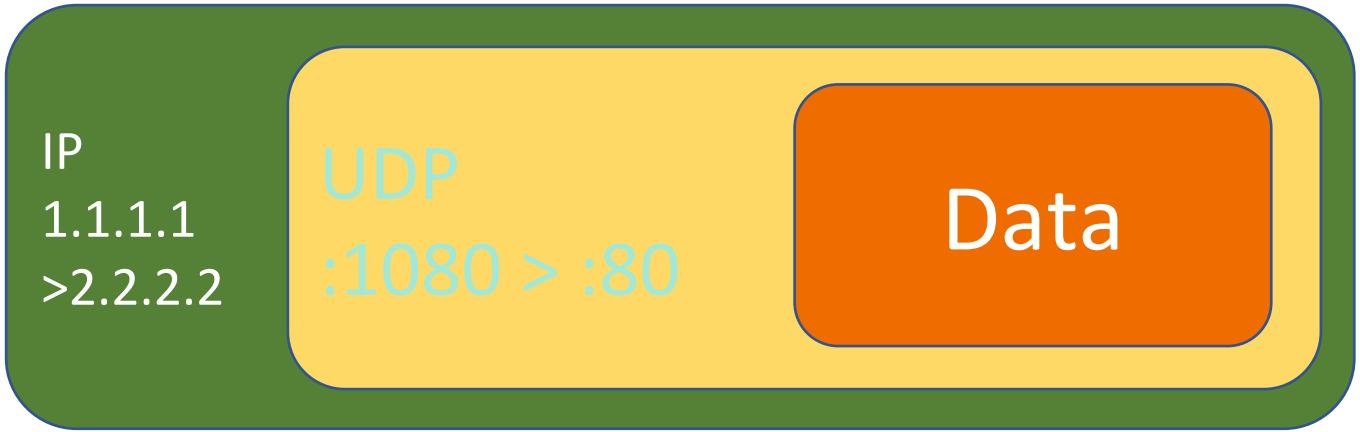
- To be able to send a packet we can use UDP or TCP we get the data:

## Network encapsulation

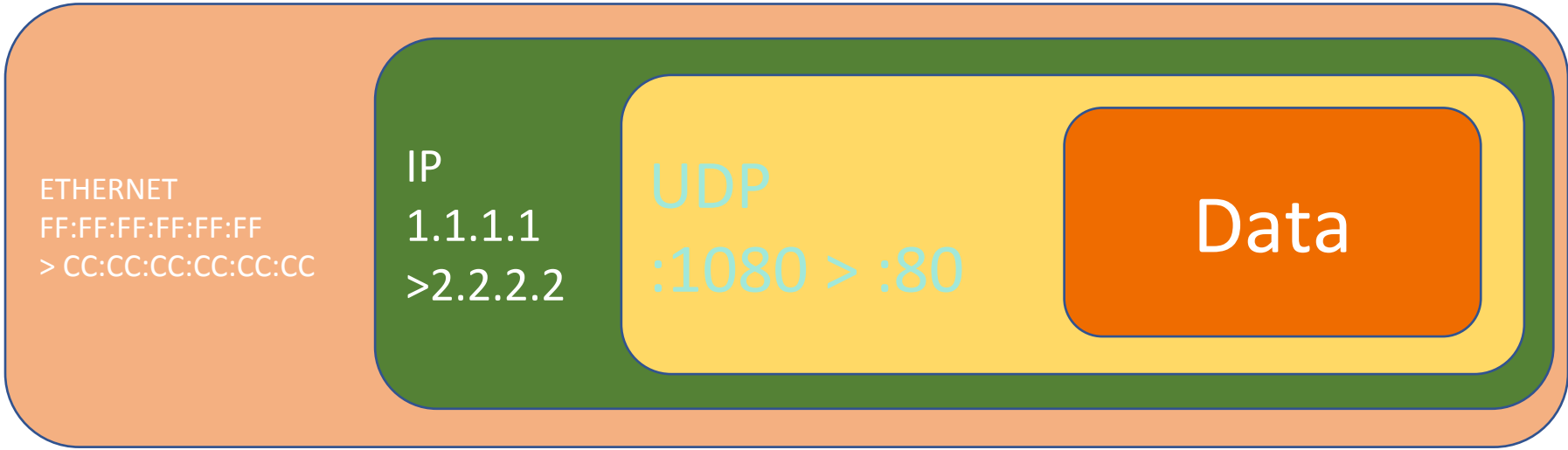
- We put a UDP header on it:

## Network encapsulation

- We then place a layer 3 IP header on it:



- Then we stick a layer 2 Ethernet header on top of the that(ip):




The next hop on the local network(addressed by the Ethernet header) will :

- Consume the IP Packet
- Replace the Ethernet header with the address of the next hop

## Container(Docker) Networking

Tools like Docker/Podman&CRI-O let you run a process isolated from the host.

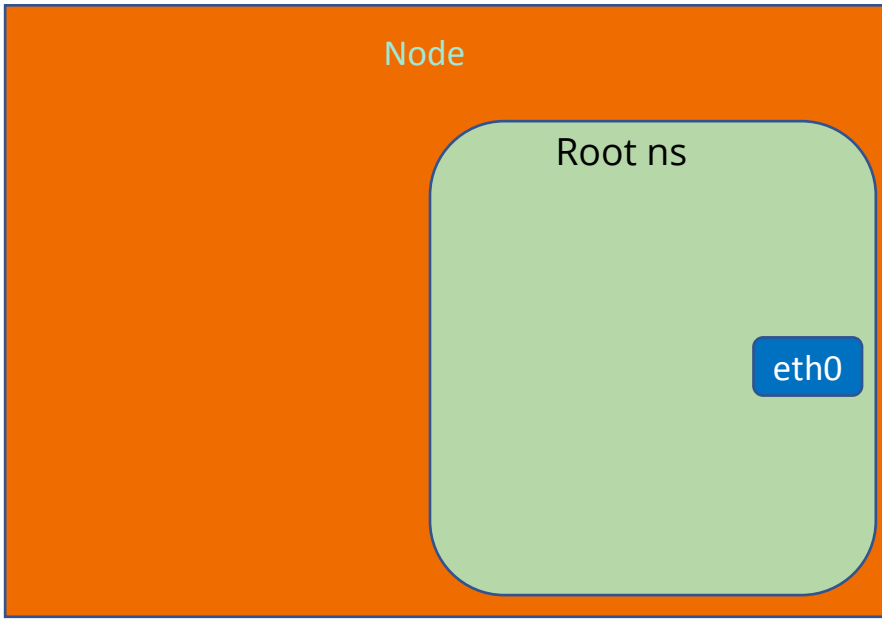
As we spoken already about other isolation previously, we will focus a bit on network isolation.

Containers are run in a network namespace == no access to the host network adapters by default.

Example will be with Docker.

# Docker bridge mode

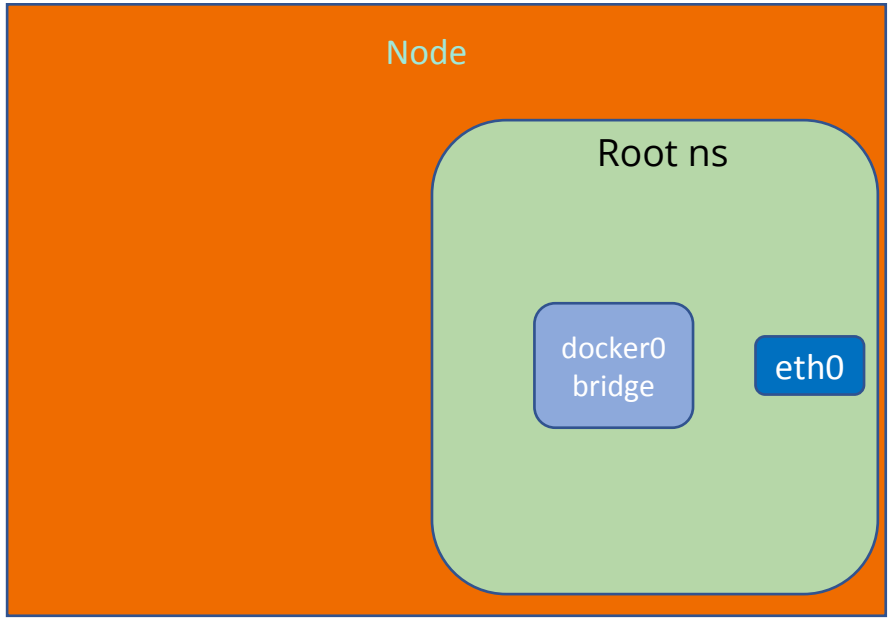
- “bridge mode” is the standard Docker networking mode.



# Docker bridge mode

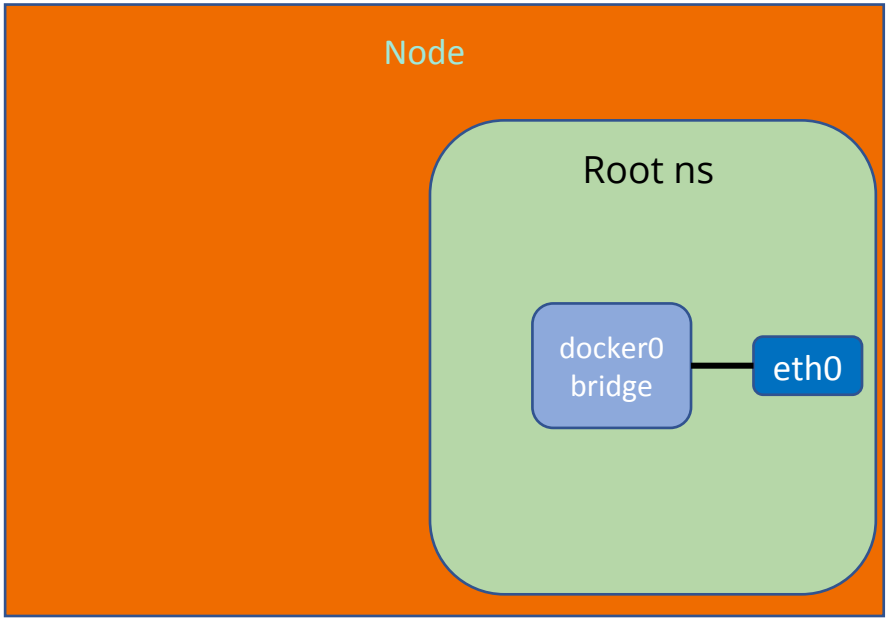
- Docker creates a bridge device (docker0*) and allocates a block of IPs (172.17.0.0/16 by default)
- The bride device acts like

Ethernet switch but in software



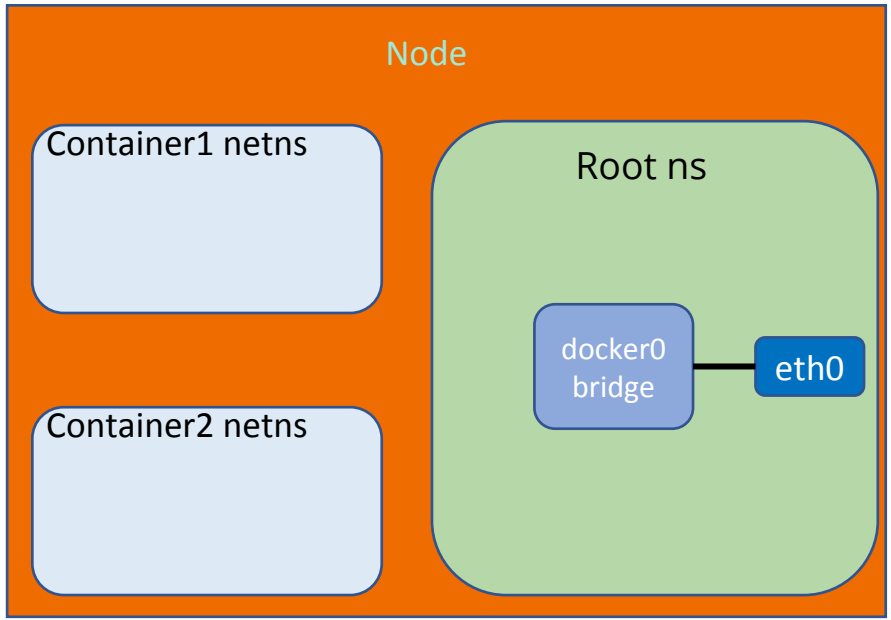
## Docker bridge mode

- The bridge gets attached to the host network interface



## Docker bridge mode

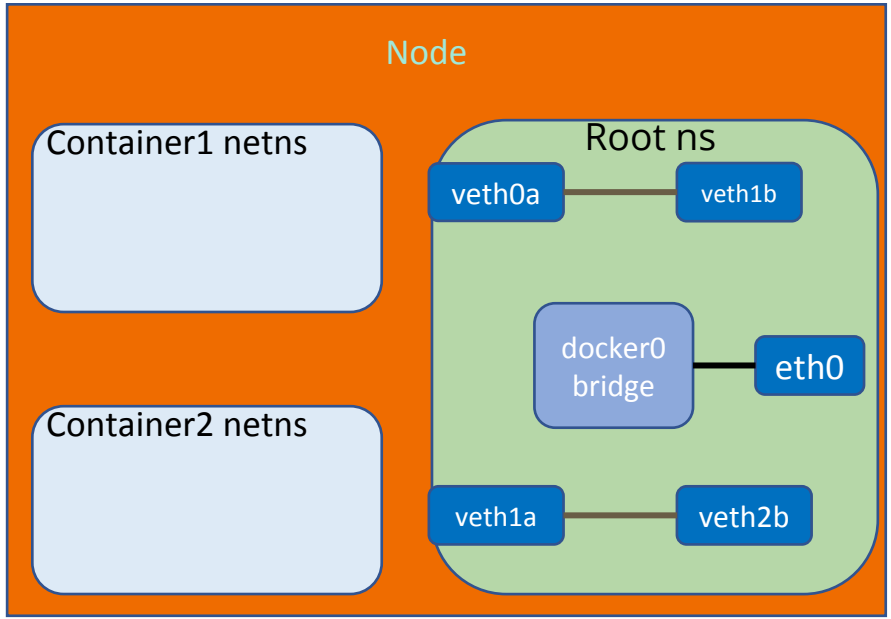
- A new namespace is created for each container created



## Docker bridge mode

Docker creates VETH pair( virtual ethernet device). Think of it as two network devices with a pipe between them.

It attaches one of the devices to the docker0 bridge and the other it places inside the container's network namespace and names, it eth0.



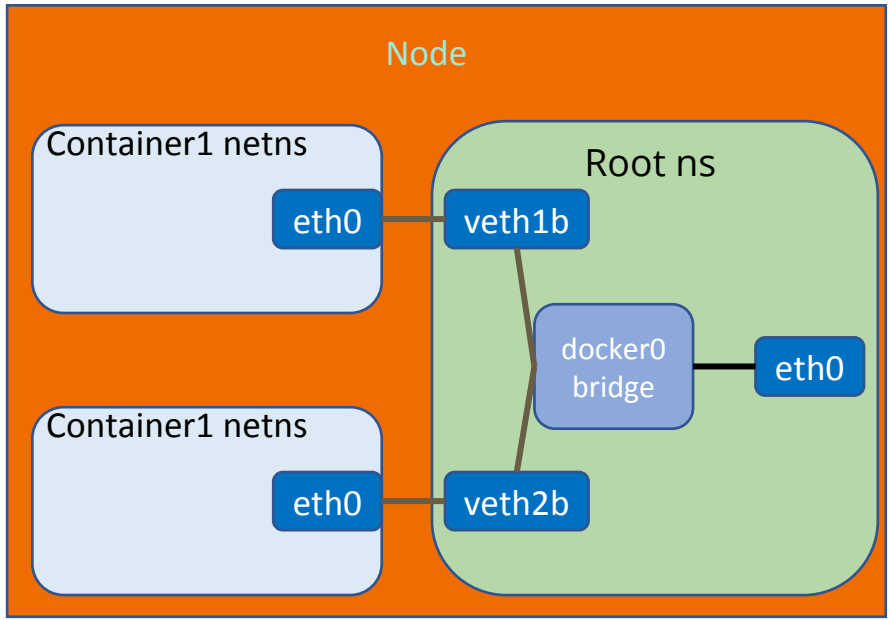
## Docker bridge mode

The eth0 interface inside the container is assigned an IP from the internal IP range docker has assigned to that

bridge(172.17.0.2-.254).

This allows containers to talk to each

Other via their private IPs



## Docker bridge mode

- Docker does not solve sending traffic from one container on a host, to another container in different host.

- To get traffic to and from containers on different hosts is a problem that Kubernetes solves

# Demo: Container Networking

# IPs in Kubernetes

## IPs In Kubernetes

This talk will focus on the three main pools of IP addresses  in Kubernetes:

- Node addresses
- Pod Addresses
- Service addresses

## Node addresses

Each node needs an IP address

It is used for nodes to talk to each other and exist before K8s is set up

This IPs are not managed by  Kuberentes,  they are set by external process like DHCP or magically by a cloud provider

## Pod addresses

A **pod** can have one or more containers and those **containers** do share the same **network namespace.**

It is a must in K8s network model for each pod to obtain an IP address.

This allocation is doe true the IPAM functionality of the CNI( Container network interface) plugin set on the cluster.

Most basic set up is to have a separate subnet for each node that is used to give out the pods its IP addresses.

Network plugins often do something fancier, such as dynamically allocation IP ranges.

The controller-manager process will start with a flag that looks like --cluster-cidr=10.244.0.0/16.

## Service addresses

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them (sometimes this pattern is called a micro-service)

All non-headless services have a ClusterIP assigned to them

ClusterIPs are handed out from a pool based on a flag for the kube-apiserver that looks like --service-cluster-ip-range=10.96.0.0/12

API-server handles this process without interaction with a CNI, it tells the kube-proxy about what IPs are assigned to what service, as well as the endpoints which are IPs of pods behind that service.

# Demo: Service IPs

# Wrap Up

This has been a quick tour through container networking, service routing

- Kubernetes networking is a HUGE topic, so any talk has to only cover a small slice
- ...but hopefully this gets you started