KubeCon | CloudNativeCon

Europe 2020

Virtual

Communication is Key – Understanding Kubernetes Networking

Jeff Poole – Vivint Smart Home

What I plan to cover

- BRIEF intro to general networking (Ethernet + IP encapsulation), network namespaces, and container networking
- Kubernetes Services
- Flannel (with VXLAN encapsulation)
- Calico (with IP-over-IP encapsulation)

Who this is for

- Anyone who sets up clusters and needs to know how they work
- Developers who want to understand the limitations and advantages of how networking works in Kubernetes
- Operations and Network Engineers who need to debug either things running in the cluster that use networking (just about everything) or cluster networking in general

Note: I expect a working knowledge of basic networking and linux tools

Environment setup

If you want to follow along, you will need to clone this repo:
`git clone https://github.com/korvus81/k8s-net-labs.git`

You also **either** need to install:
- Docker https://docs.docker.com/get-docker/
- Footloose https://github.com/weaveworks/footloose#install

Or have a working Vagrant + VirtualBox setup.

You can probably do most of the demonstrations on other Kubernetes clusters, but this will let you have the environment I use. I use k3s in Docker for the demos (stood up with Footloose) because it was the lightest weight Kubernetes solution I could come up with that looks like a real cluster from a networking perspective.

Demo: Environment Setup

Encapsulation in Networking

Encapsulation in Networking

To get us all on the same page, let's talk about how encapsulation works in networking.

I'm going to assume we are talking about **IPv4** over **Ethernet**

Encapsulation in Networking

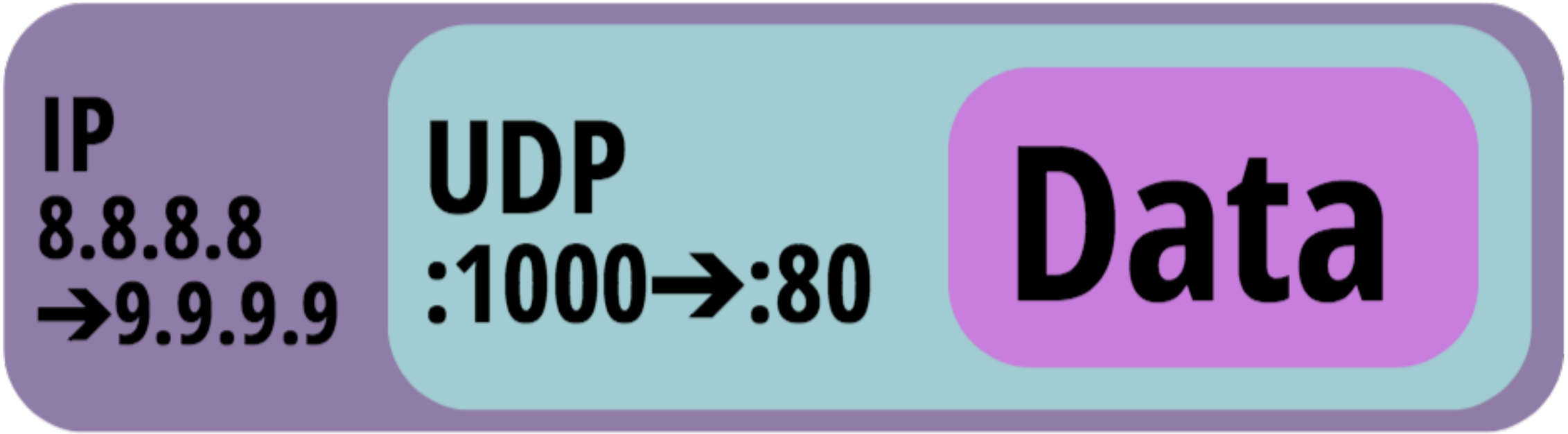
To send a packet via, say, UDP, we get the data:

Encapsulation in Networking

We stick a (layer 4) UDP header on it:

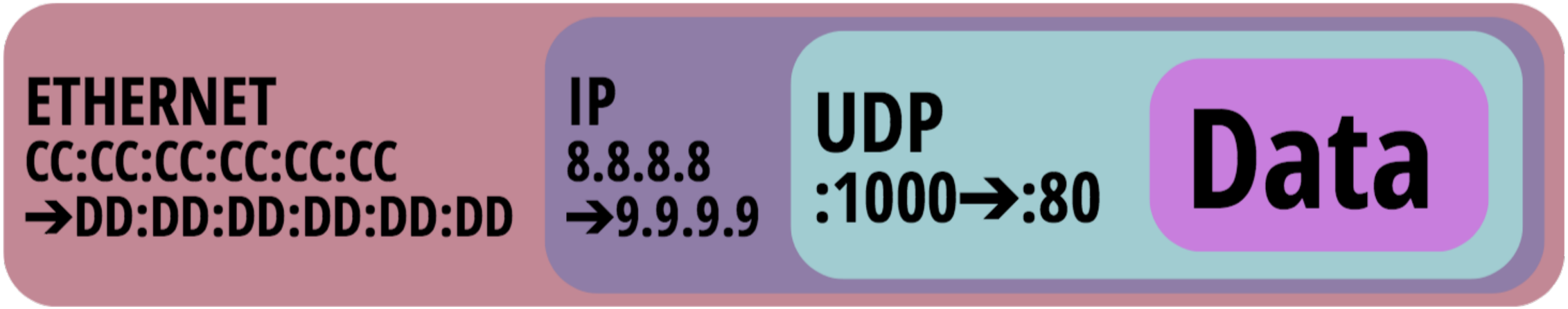
Encapsulation in Networking

We then stick a (layer 3) IP header on it:



Encapsulation in Networking

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



The next hop on the local network (addressed by the Ethernet header) will either:
- consume the IP packet
- replace the Ethernet header with one addressed to the next hop

Docker / Container Networking

Docker / Container Networking

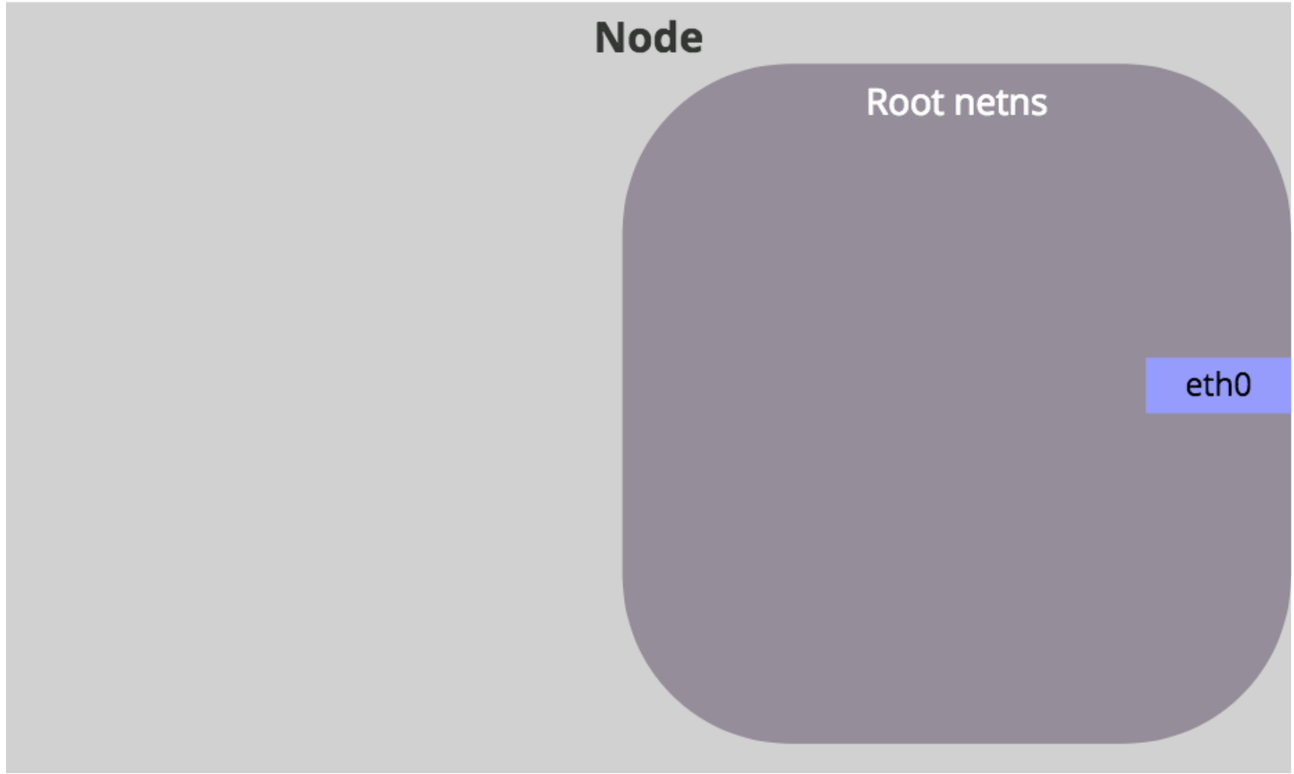
Docker allows you to run a process with various forms of isolation from the host operating system.

For our purposes, we only really care about network isolation.

Docker containers are run in a **network namespace** -- this means they have no access to the host network adapters by default.

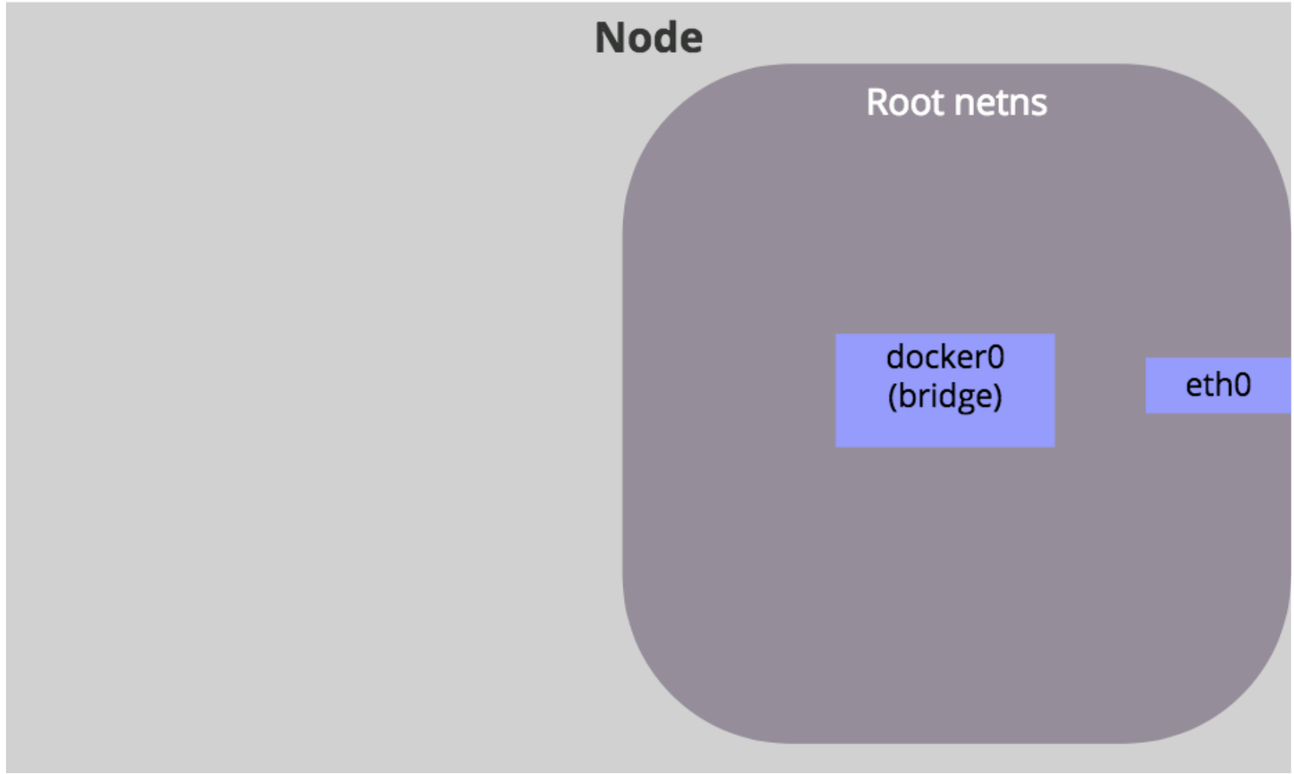
Docker bridge mode

- The standard Docker networking mode is "bridge mode"



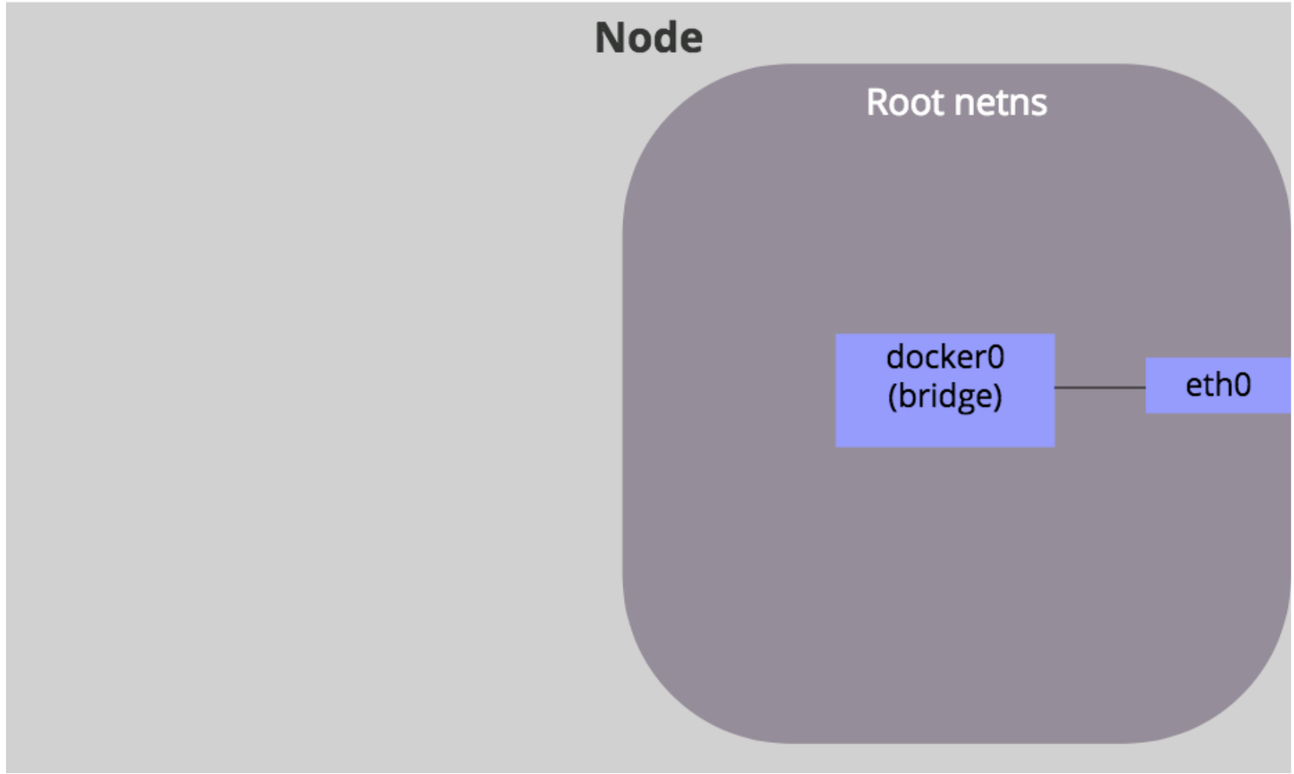
Docker bridge mode

- Docker creates a bridge device (`docker0` unless you specify otherwise) and allocates a block of IPs (172.17.0.0/16 by default)
- This bridge device operates like an Ethernet switch, running in software



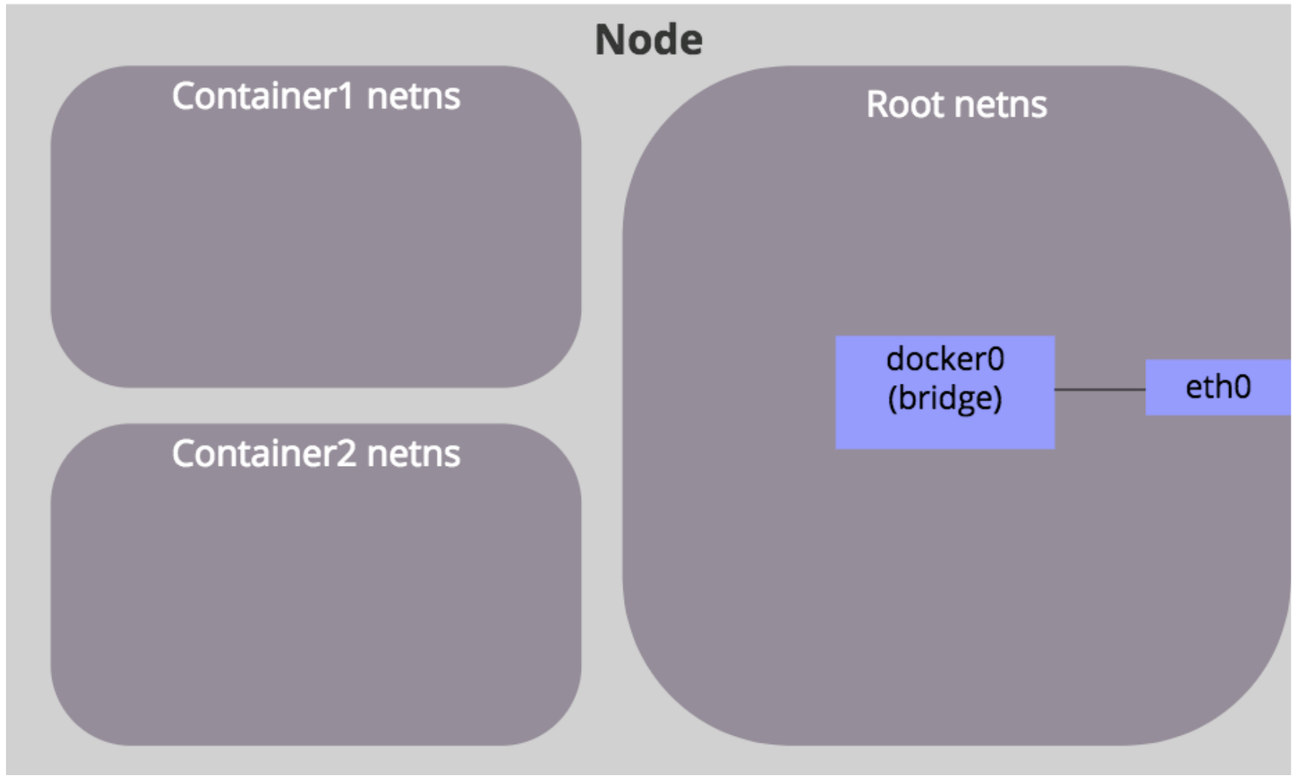
Docker bridge mode

- The bridge gets attached to the host network interface



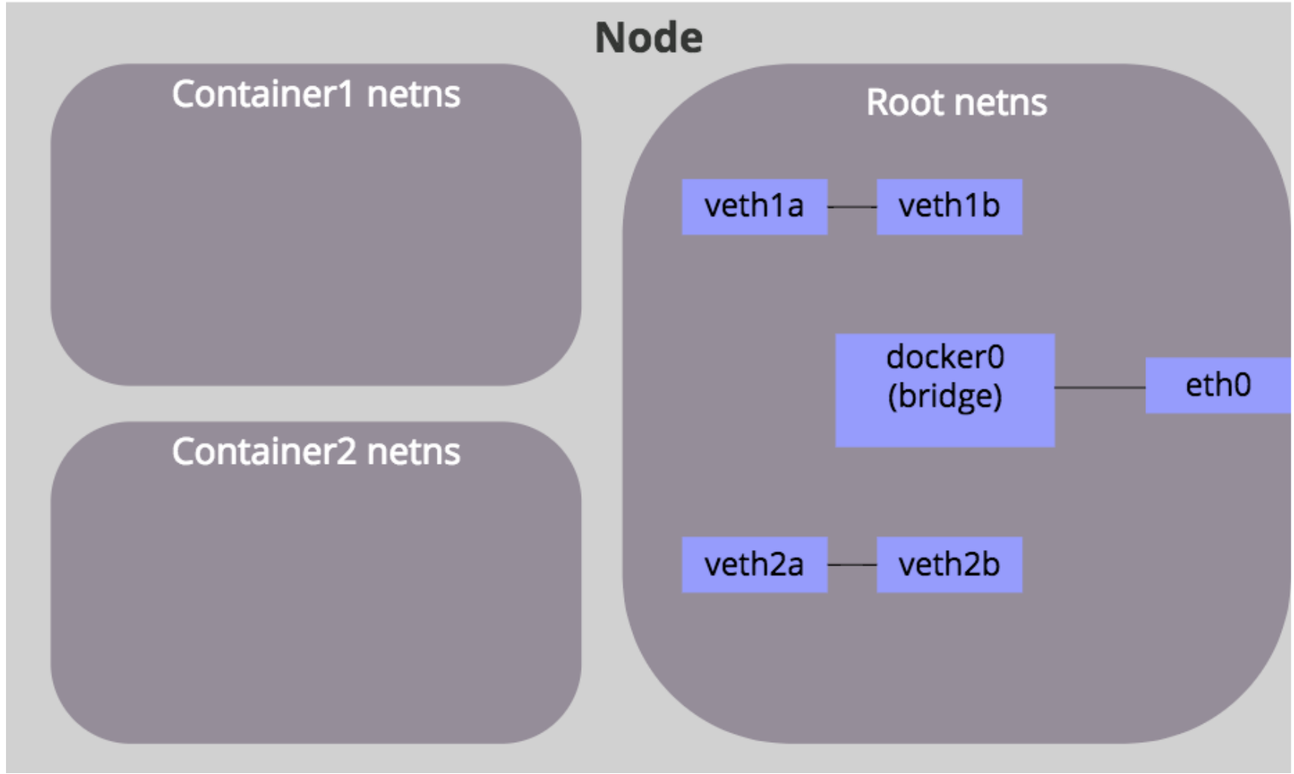
Docker bridge mode

- Docker creates a new network namespace for each container



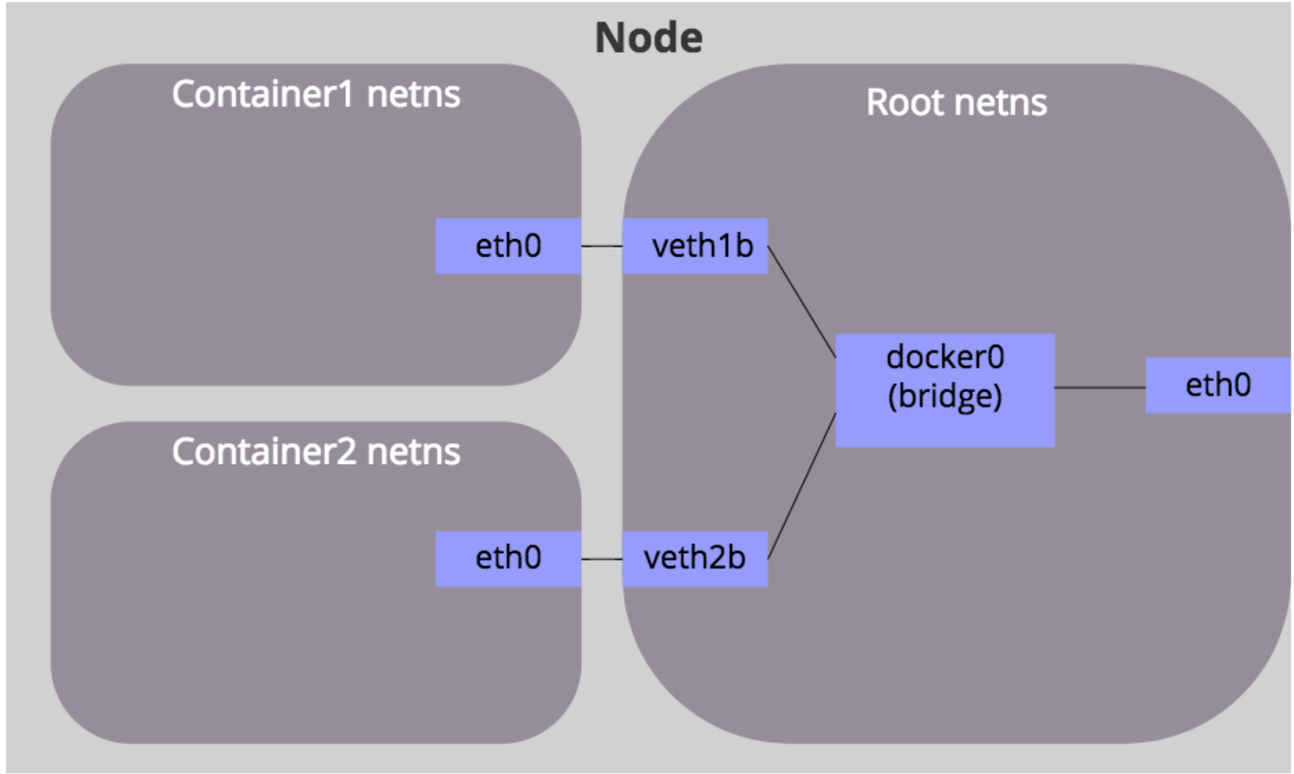
Docker bridge mode

- Docker creates a VETH pair (virtual ethernet device). This is like two network devices with a pipe between them. It attaches one of the devices to the `docker0` bridge and the other it moves into the container's network namespace and names it `eth0` within that namespace.



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 IP range.



Docker bridge mode

- Traditionally, Docker wouldn't expose those container IPs to anything off of the host the containers are on
- Getting traffic to and from containers from other machines is one of the problems Kubernetes needs to solve

Demo: Container Networking

IPs in Kubernetes

IPs in Kubernetes

We are going to talk about three main groups of IP addresses

- Node addresses
- Pod addresses
- Service addresses

Node addresses

- Each node needs an IP address
- This is used for nodes to talk to each other and exists before Kubernetes is set up
- This is outside of the scope of Kubernetes and is assigned by some outside process (DHCP, manual configuration, magically by a cloud provider, etc)

Pod addresses

- To review, in Kubernetes a **pod** consists of one or more **containers** sharing the same **network namespace**
- In the Kubernetes network model, every pod receives it's own IP address
- These are allocated through the IPAM functionality of the CNI (Container Network Interface) plugins you are using
  - The most basic involves assigning a subnet (IP address range) to each node to give out to the pods on that node
  - Networking plugins often do something fancier, such as dynamically allocating IP ranges
- The `kube-apiserver` process will be started with a flag that looks like `--cluster-cidr=172.16.0.0/16` -- this will determine the range all pod IPs should be in, since that is the range of IP addresses "within the cluster"

Service addresses

- A Kubernetes **service** is an abstraction over a set of pods
- All non-headless services have a `ClusterIP` assigned to them
- `ClusterIP`s are handed out from a pool based on a flag to `kube-apiserver` that looks like `--service-cluster-ip-range=172.15.100.0/23`
- The apiserver process handles this, regardless of your networking plugin, and tells the `kubelet` processes about what IPs are assigned to what services, as well as the `endpoints` which are IPs of pods behind that service

Demo: Service IPs

Flannel

Flannel

Flannel is one of the earlier network plugins, and a decent choice for small clusters. It's also the default for k3s, which makes it a good place to start.

Flannel runs at layer 2 (Ethernet) in the networking stack, so all pods can talk via Ethernet (as opposed to only IP).

Flannel -- IPAM

Flannel uses a pod subnet statically assigned to each Kubernetes node, so pod IP allocation decisions are all local to the node

Flannel -- Encapsulation

The default encapsulation for Flannel is VXLAN, which involves wrapping a **Layer 2 Ethernet packet** inside a UDP packet

Demo: Flannel

Calico

Calico

Calico is easily the most common networking plugin, with Tigera (the company behind it) claiming that it is in use to some degree (at least for network policies) in most cloud provider Kubernetes environments.

Calico runs at layer 3 (IP) in the networking stack, so only IP traffic can be encapsulated, and everything is routed.

Calico -- IPAM

Calico uses a dynamic subnet allocation scheme, with either the Kubernetes API server or it's own etcd cluster.

Calico -- Encapsulation

The default encapsulation for Calico is IP-in-IP, which involves wrapping a **Layer 3 IP packet** inside an extra IP header. This is very low overhead, but can only encapsulate IP packets.

Demo: Calico

Wrap up

- This has been a quick tour through container networking, service routing, Flannel, and Calico
- Kubernetes networking is a **HUGE** topic, so any talk has to only cover a small slice
- ...but hopefully this gets you started
- While this talk has been pre-recorded, I should be available to answer questions
- Or you can reach me after the talk at jeff@jeffpoole.net

Thanks!