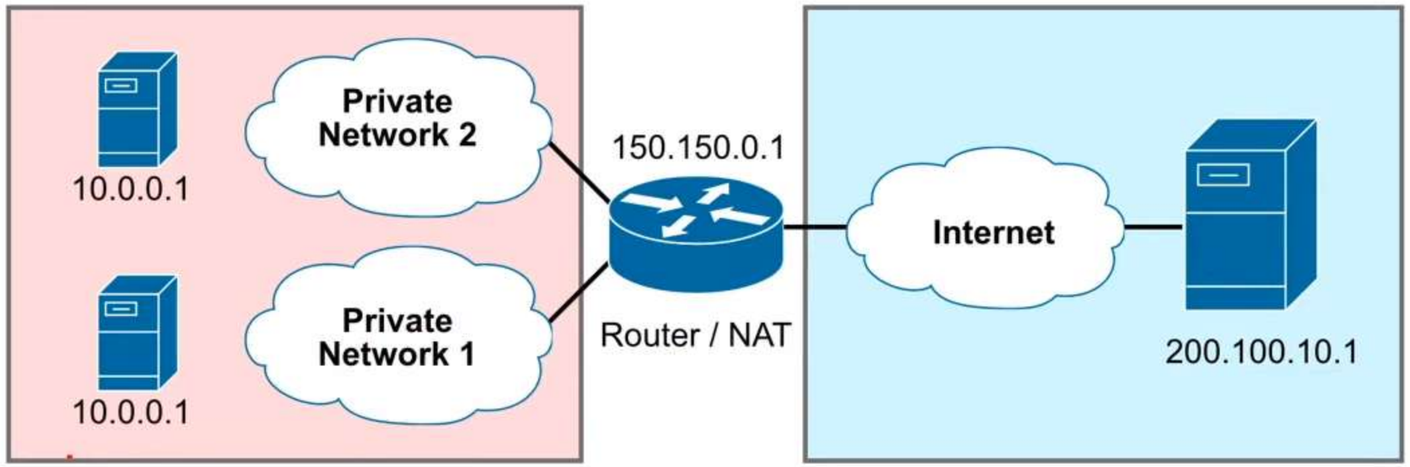
Network Address Translation (NAT)

What is Network Address Translation (NAT)?
a method of mapping an IP address space into another by modifying network address information in the IP header of packets while they are in transit across a traffic routing device



Eth0 and Network Namespace

What is an Ethernet Device?

An Ethernet Device is a software and/or hardware technology that allows a server to communicate on a computer network. A **Network Interface Card (NIC)** are commonly used to establish a wired connection to a network.

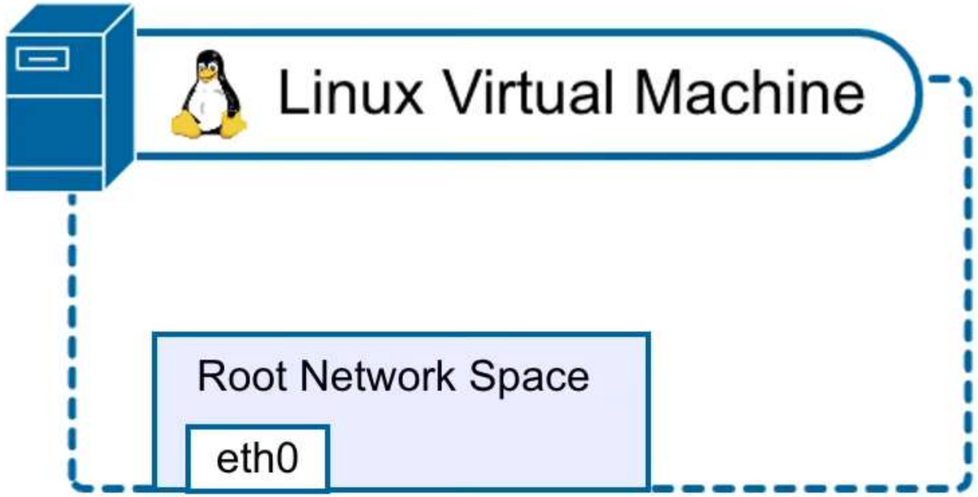
Cloud Service Providers (CSPs), have Virtual NICs for your Virtual Machines (VMs) to connect to the Virtual Network

What is a Network Namespace?

eth0 represents the first Ethernet interface attached to your Virtual Machine.

A network interface is an abstraction on top of the ethernet interface to provide a logical networking stack with its own routes, firewall rules, and network devices

Linux by default has one Network Namespace called the Root Network Space and this is what programs will use by default.



Eth0 and Network Namespace

To observe network devices attached to a linux virtual machine you use the **ifconfig** command.

```
Session ID: brown-laptop-0587bdce3842e9e35          Instance ID: i-07eba66923d9fa554
[ec2-user@ip-172-31-82-19 ~]$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 9001
        inet 172.31.82.19  netmask 255.255.240.0  broadcast 172.31.95.255
        inet6 fe80::106f:4aff:fe1b:9b13  prefixlen 64  scopeid 0x20<link>
        ether 12:6f:4a:1b:9b:13  txqueuelen 1000  (Ethernet)
        RX packets 53021  bytes 72653845 (69.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 23977  bytes 2018753 (1.9 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 24  bytes 1944 (1.8 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 24  bytes 1944 (1.8 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

[ec2-user@ip-172-31-82-19 ~]$
```

To create modify or view Network Namespaces the **ip netns** command cand be used.

```
Session ID: brown-laptop-06df3855e35907f62          Instance ID: i-07el
[ec2-user@ip-172-31-82-19 ~]$ sudo ip netns add ns1
[ec2-user@ip-172-31-82-19 ~]$ ip netns
ns1
[ec2-user@ip-172-31-82-19 ~]$
```

Cluster Networking

Kubernetes has the following opinions about cluster networking:

- all Pods can communicate with all other Pods without using NAT
- all Nodes can communicate with all Pods without using NAT.
- the IP that a Pod sees itself as, is the same IP that others see it as

NATs are and can be used in Kubernetes, even though of the above contradiction

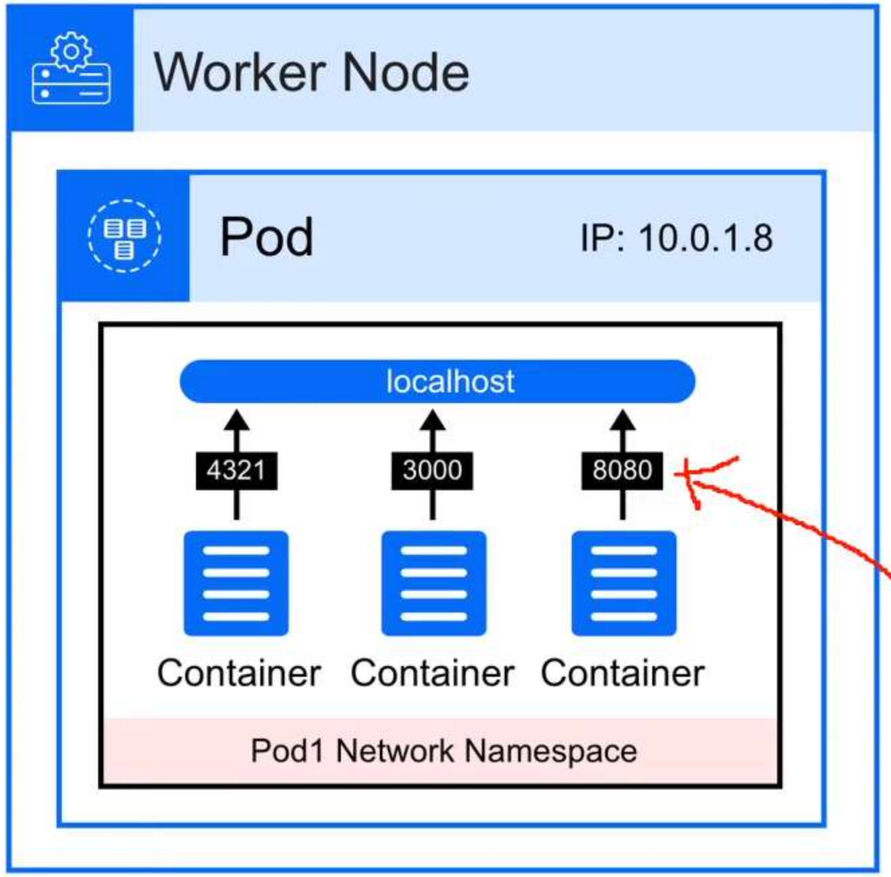
There are 4 broad types of network communication for clusters:

1. Container-to-Container
2. Pod-to-Pod
3. Pod-to-Service
4. External-to-Service

Watch the CNCF tech talk **Life of a Packet** by Michael Rubin for detailed technical information about Cluster Networking

Container to Container Networking

Containers all in the same pod have the same IP address and port space.
Containers can communicate with each other via **localhost** via different ports




```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
spec:
  selector:
    matchLabels:
      run: my-nginx
  replicas: 2
  template:
    metadata:
      labels:
        run: my-nginx
    spec:
      containers:
      - name: my-nginx
        image: nginx
        ports:
        - containerPort: 80
```

Within the manifest file **containerPort** is used to define the local port for the container

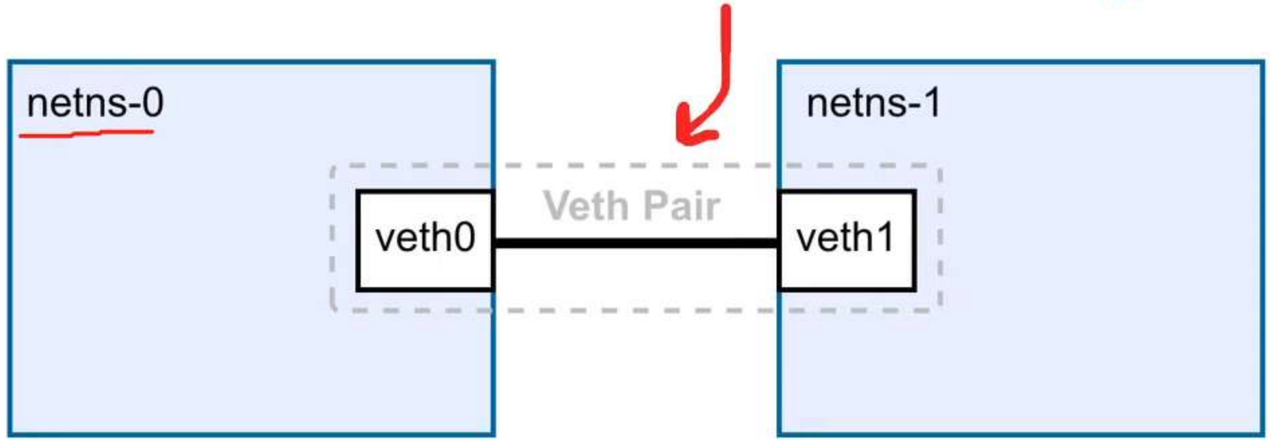
Virtual Ethernet Devices (Veths)

veth devices are Virtual Ethernet devices.

They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.

Packets on one device in the pair are immediately received on the other device

veth devices are always created in **interconnected pairs**



You use the **ip link** command to create veth pairs

```
ip link add <p1-name> netns <p1-ns> type veth peer <p2-name> netns <p2-ns>
```

Pod to Pod Same Node Networking

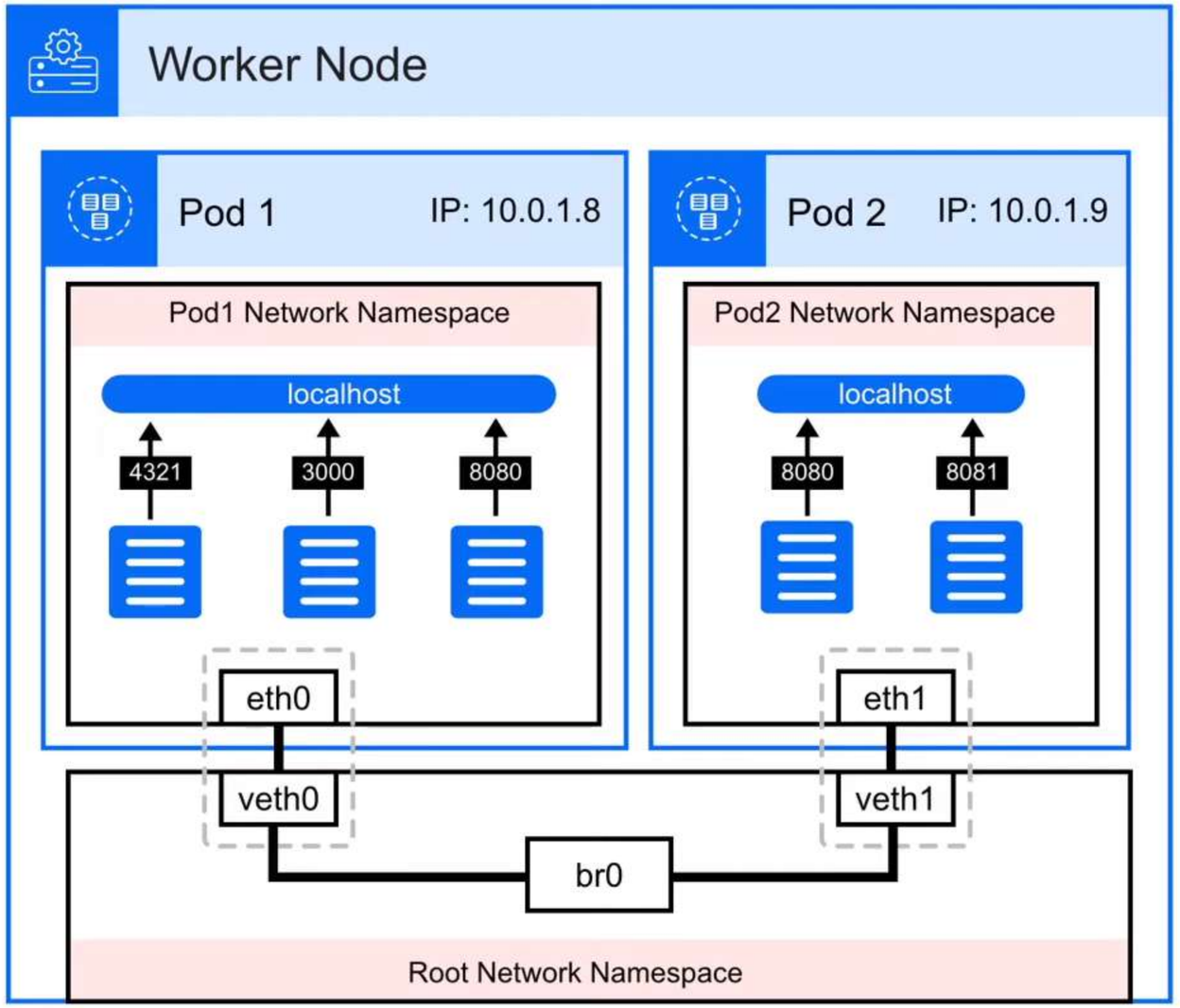
For Pod to Pod communication on the same on the node Veth is used to communication from the Pod Network Namespace to the Root Network Namespace.

In the Root Network Namespace a **bridge** is used allow all Pod Network Namespaces to talk to other pods.

Pods can see all other pods, and communicate using their IP addresses.

Routing allows multiple networks to communicate independently and yet remain separate using a Router

Bridging connects two separate networks as if they were a single network using a Bridge



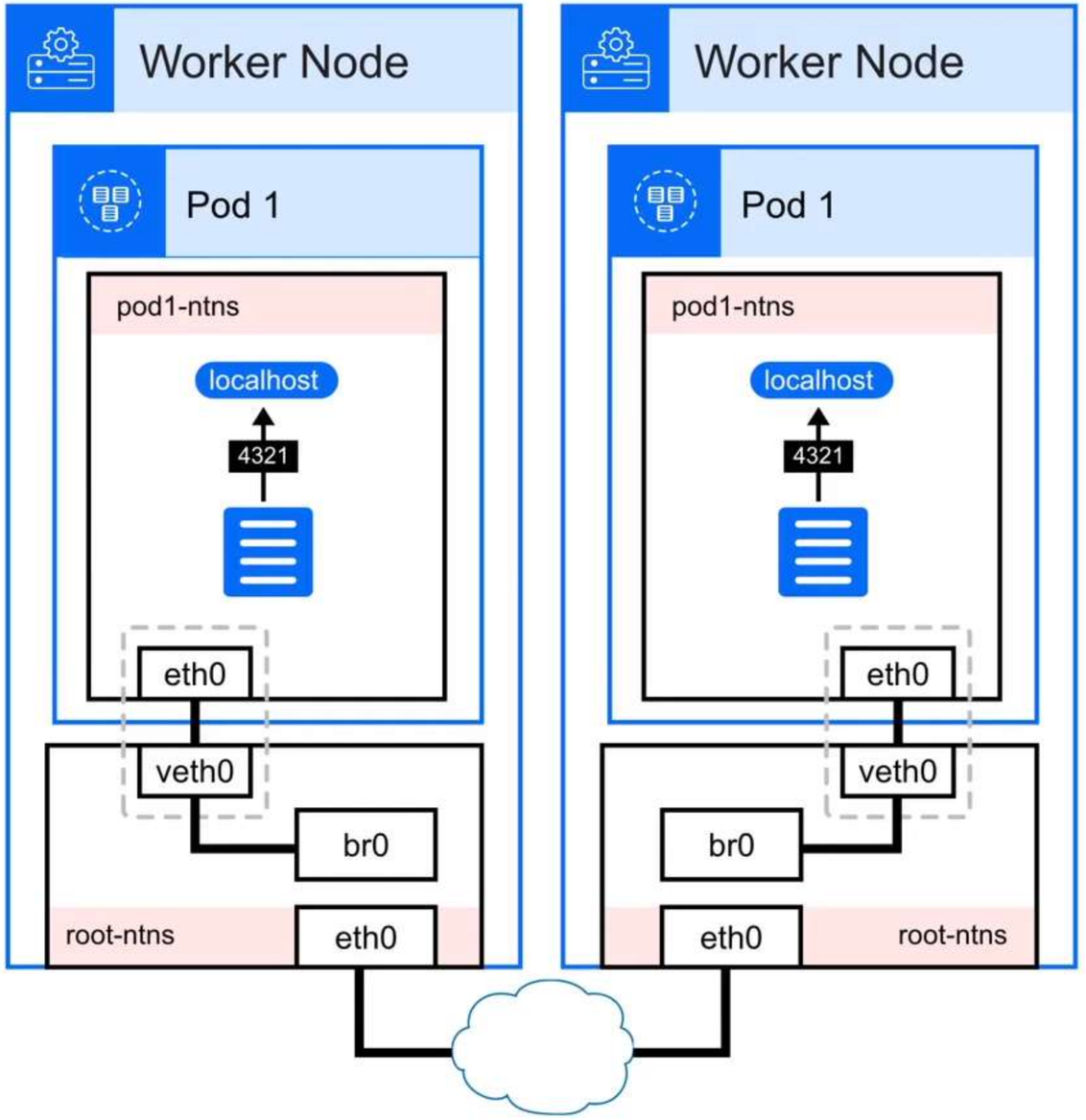
Pod to Pod Across Node Networking

Pods can communicate to other Pods running on other worker nodes

How pods can communicate pods on other nodes is network specific to the scenario and will vary.

In the case of AWS, they have their own implementation of the Container Networking Interface (CNI)

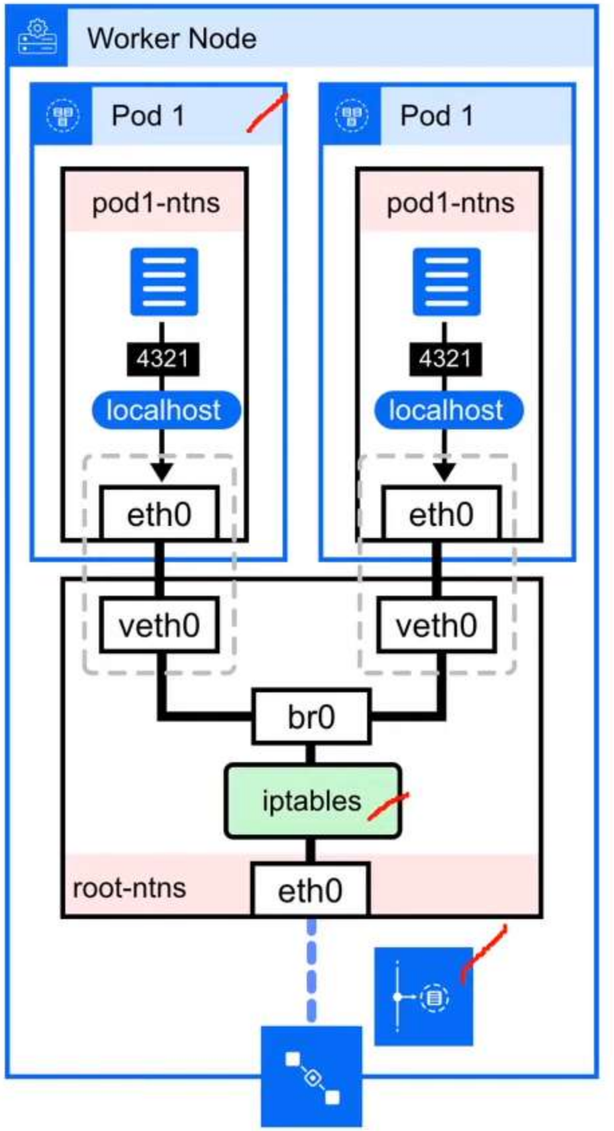
Amazon VPC Container Network Interface (CNI) plugin for Kubernetes managed node to node communicate leverage AWS Virtual Private Cloud (VPC)



Pods to Service Networking

When a pod dies its IP Address changes and this can make communication hard if you are relying on the IP address for communication.

A Service creates a virtualized IP (static IP) and then uses iptables which is installed on the Node to Network Address Translation (NAT) and Load Balancing to other pods.



Ingress/Egress From Internet to Cluster

Egress

How pod traffic exits to the internet will be network specific.
So in the case of AWS pods use the Amazon VPC Container Network Interface (CNI) plugin to be able to talk to your Virtual Private Cloud (VPC) and then egress out to the Internet Gateway (IGW) via route tables.

Ingres

For traffic to reach a pod, it travels to a service.
From there a Service could be using:

- K8s Service with Type Load Balancer
  - This will work with Cloud Controller Manager to implement a solution that works with a T4 (UDP/TCP) Load Balancer
- K8s Ingress
  - It will use a Ingress Controller to work with a Cloud Service Provider load balancer eg. T4 or T7 (application load balancer)