Container & Kubernetes Networking

An Introduction

Agenda

1. Container Networking
    a. From scratch
    b. Docker
    c. CNI
2. Kubernetes Networking
    a. Networking Model
    b. Pod Network
    c. Service Network & DNS

What’s not covered

1. Network Policies
2. Ingress

whoami

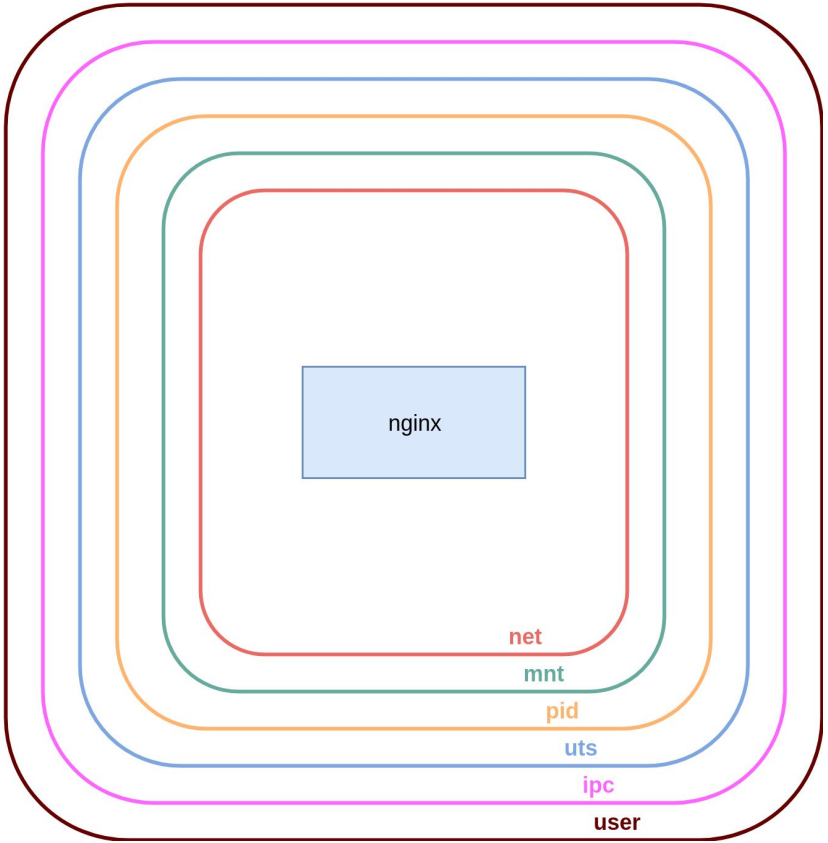
- Alexander Knipping, IT Systems Engineer @ noris network
  - Prometheus, Kubernetes
  - Cloud Native Adoption
- ❤️ Open Source
  - https://github.com/obitech
  - Contributor to Kubernetes (wg-component-standard) and M3DB
- Studying for my CKA 🤞

Container Networking

From Scratch

What’s a Container?

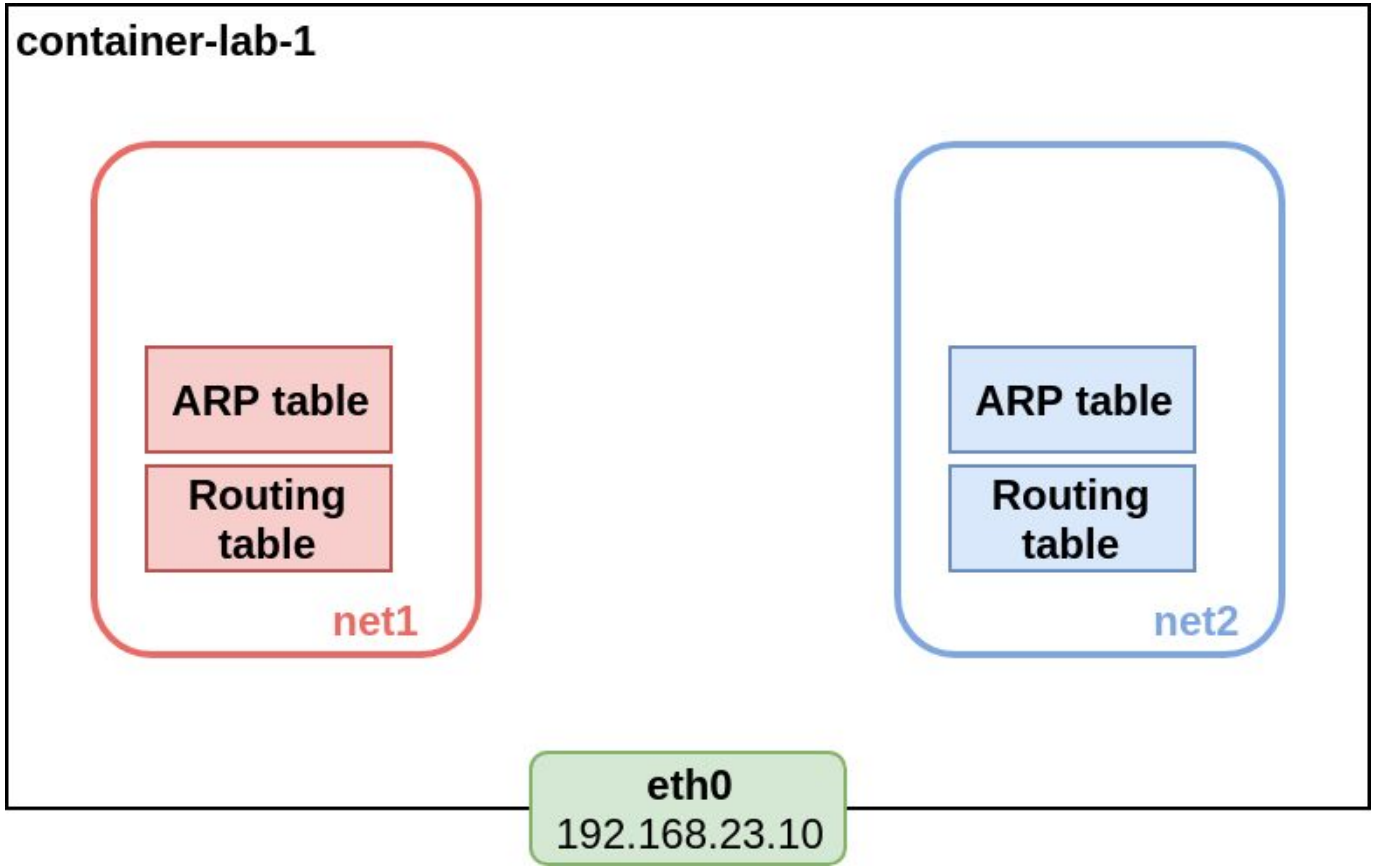
net	Network interfaces, IP addresses
mnt	Mountpoints
pid	Process IDs
uts	Hostname
ipc	Inter-process communication
user	User IDs



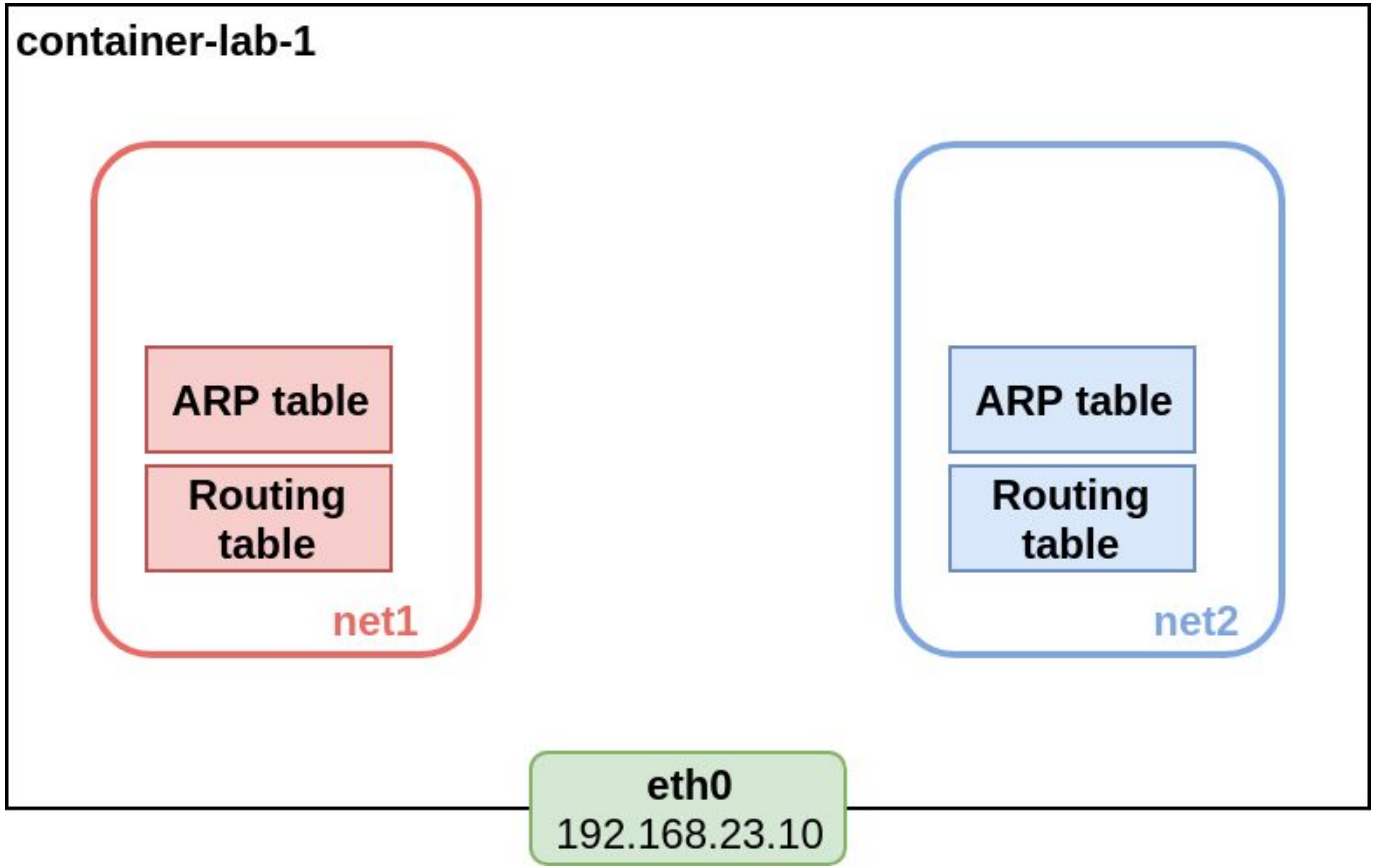
01_namespaces

Container Networking From Scratch

1. Create two network namespaces
2. Connect namespaces with each other and the host
3. Establish internal and outside connection


container-lab-1
ARP table
Routing table
net1
ARP table
Routing table
net2
eth0
192.168.23.10

02_netns


container-lab-1
ARP table
Routing table
net1
ARP table
Routing table
net2
eth0
192.168.23.10

Virtual Ethernet Devices

veth (4)

NAME

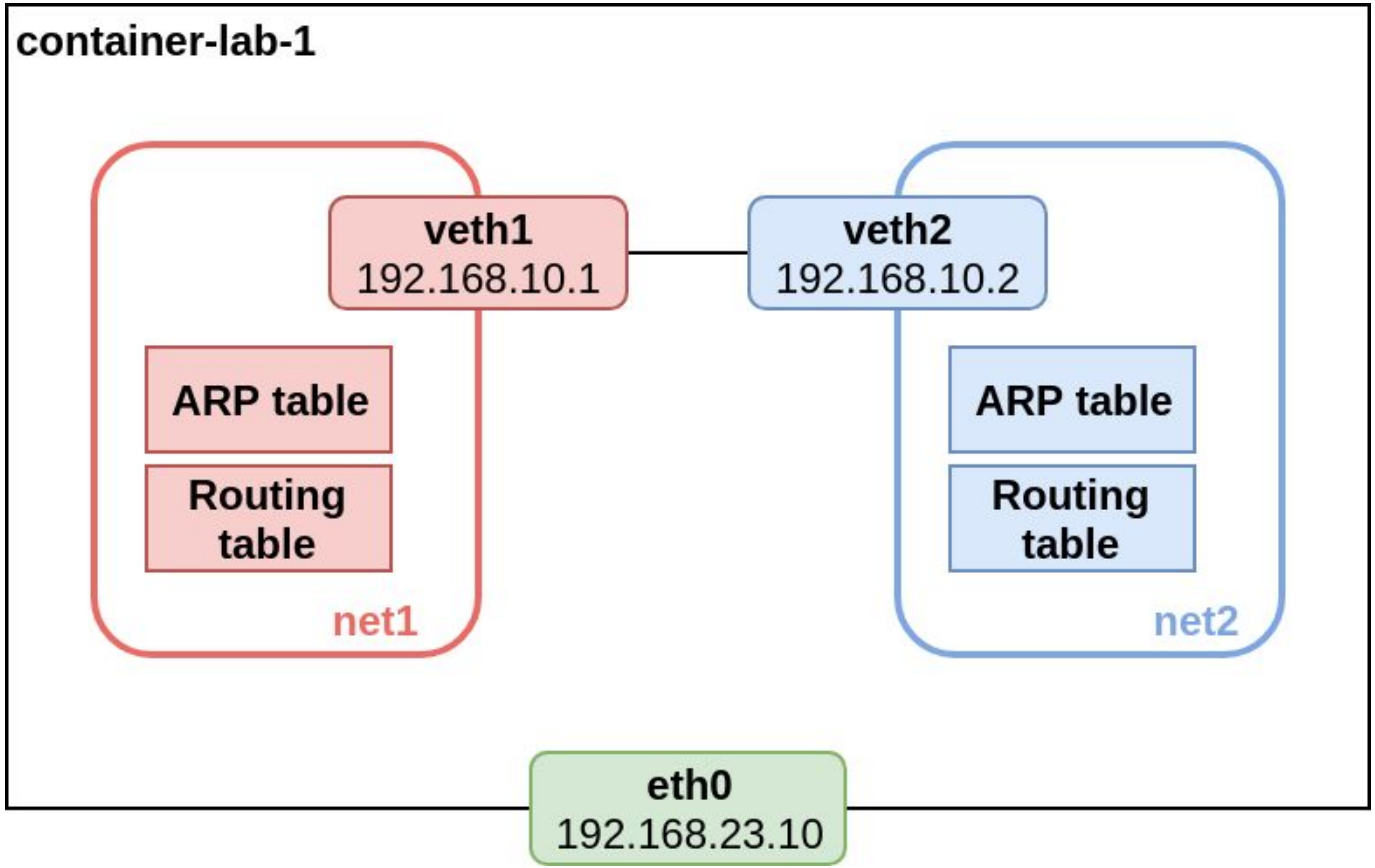
veth - Virtual Ethernet Device

DESCRIPTION

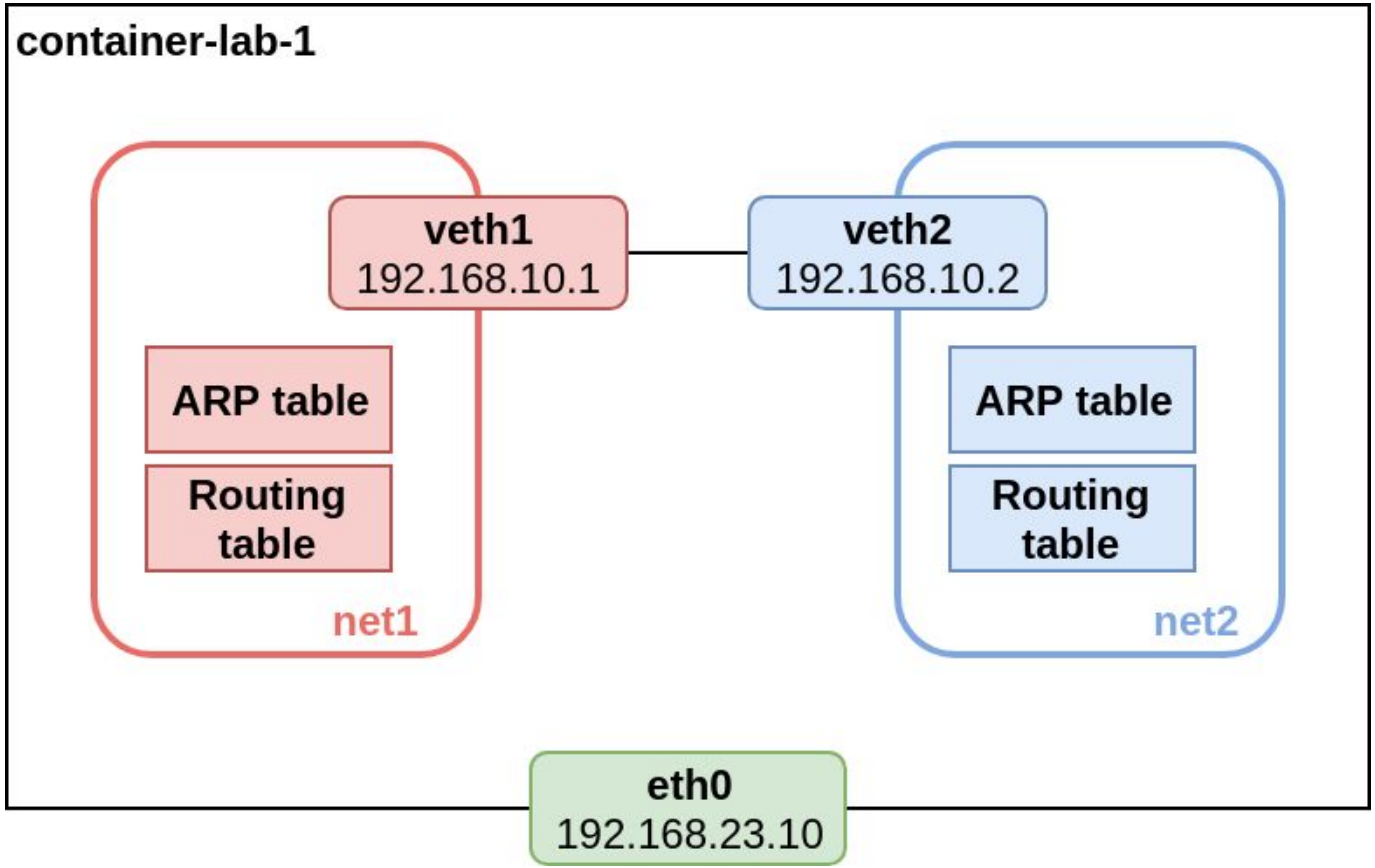
The **veth** devices are virtual Ethernet devices. They can act as tunnels between network namespaces to create a bridge to a physical network device in another namespace, but can also be used as standalone network devices.

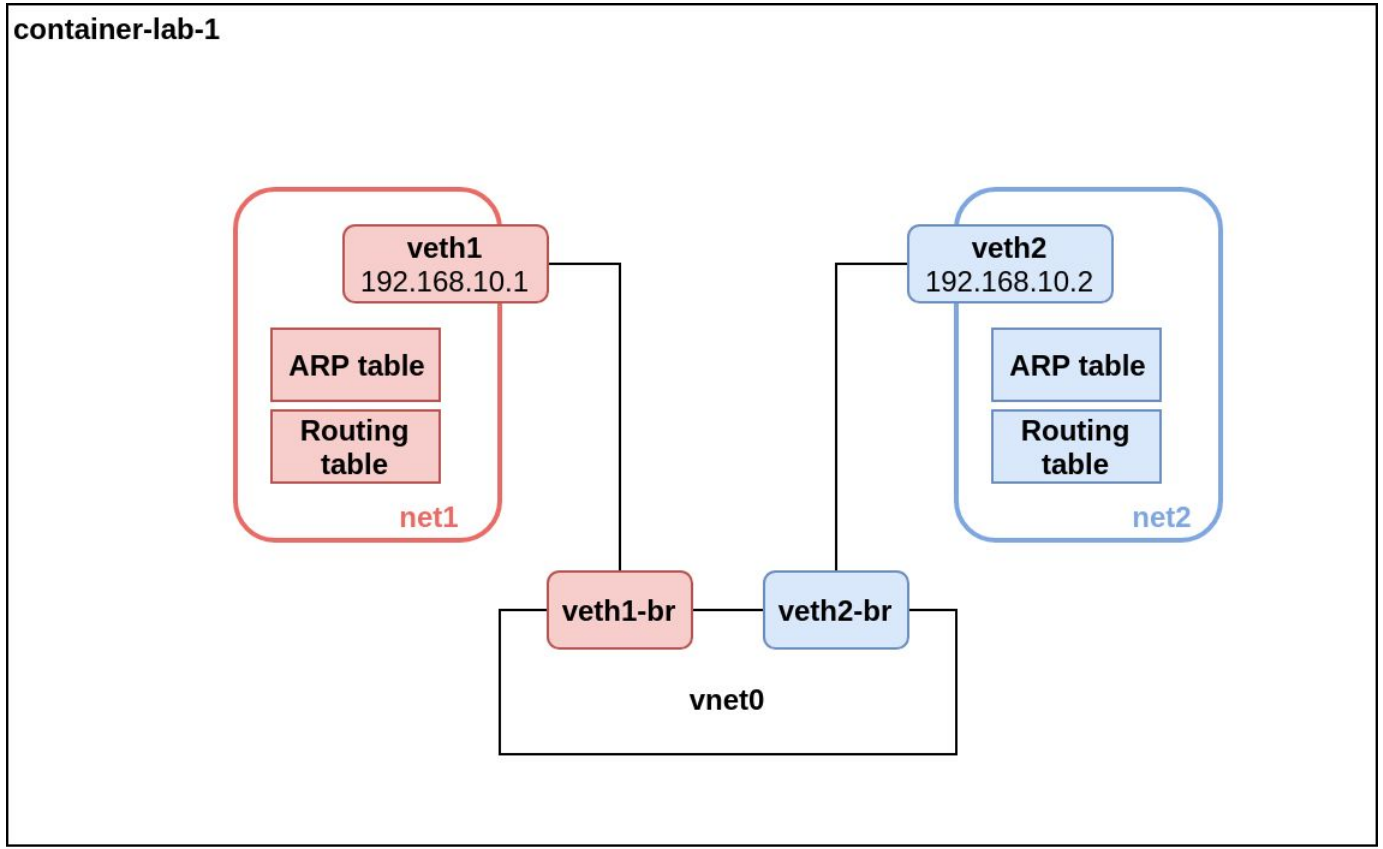
veth devices are always created in interconnected pairs. A pair can be created using the command:

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

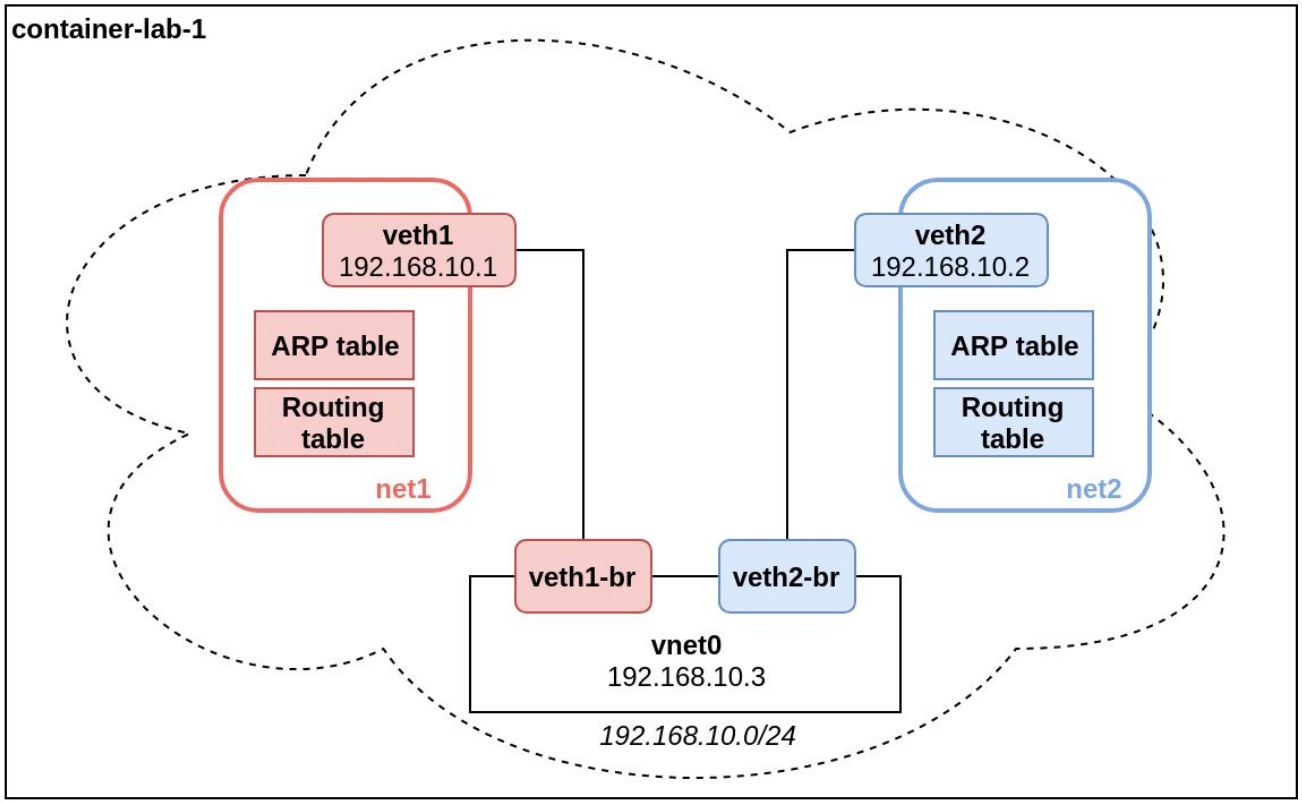

container-lab-1
veth1
192.168.10.1
veth2
192.168.10.2
ARP table
Routing table
net1
ARP table
Routing table
net2
eth0
192.168.23.10

03_netns_connected


container-lab-1
veth1
192.168.10.1
veth2
192.168.10.2
ARP table
Routing table
net1
ARP table
Routing table
net2
eth0
192.168.23.10

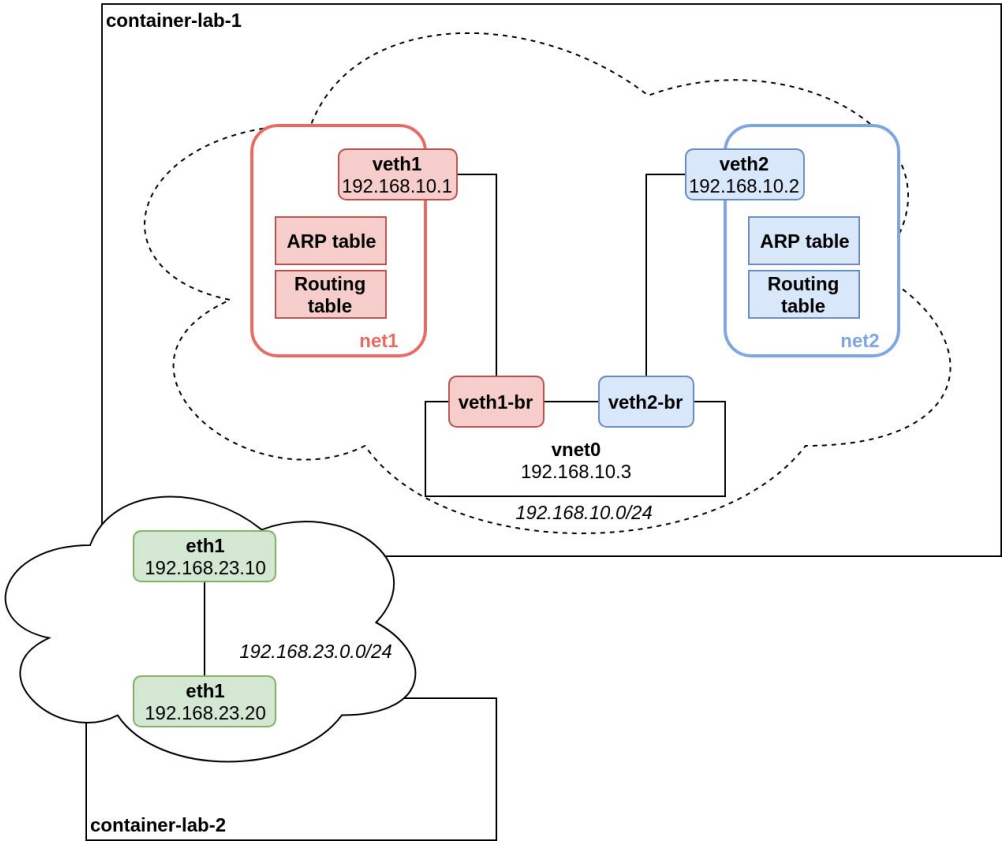

container-lab-1
veth1
192.168.10.1
ARP table
Routing table
net1
veth2
192.168.10.2
ARP table
Routing table
net2
veth1-br
veth2-br
vnet0

04_netns_bridge

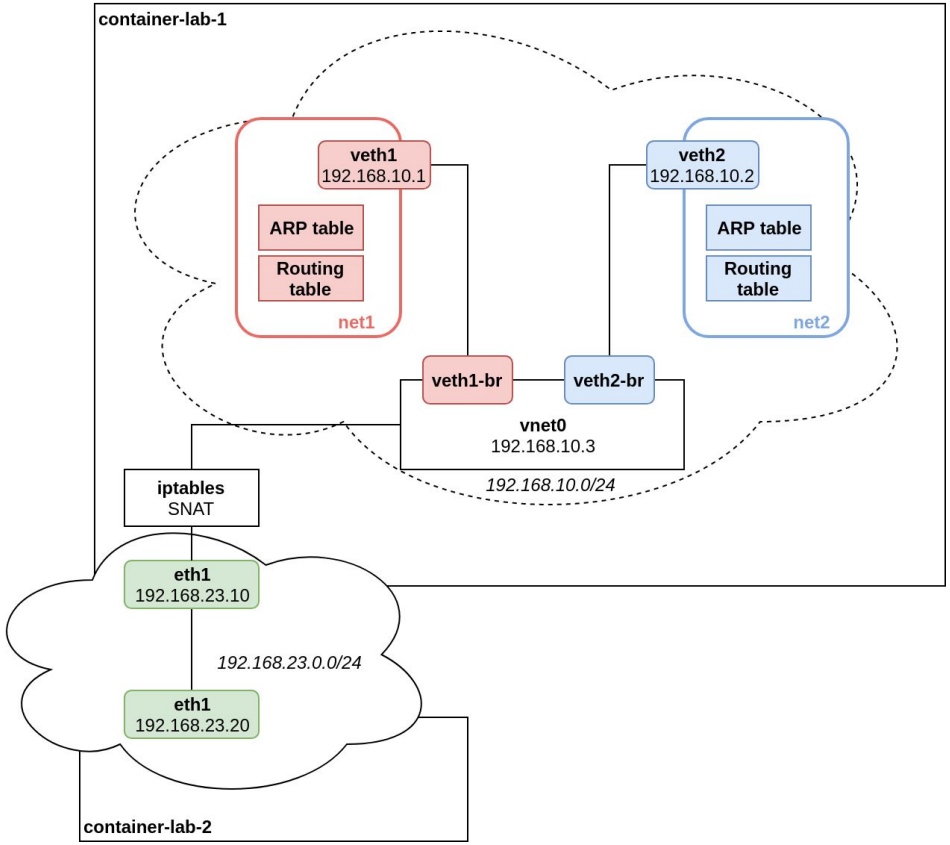

container-lab-1
veth1
192.168.10.1
ARP table
Routing table
net1
veth2
192.168.10.2
ARP table
Routing table
net2
veth1-br
veth2-br
vnet0
192.168.10.3
192.168.10.0/24

How can we reach `container-lab-2` from `net1`?

- Source NAT !

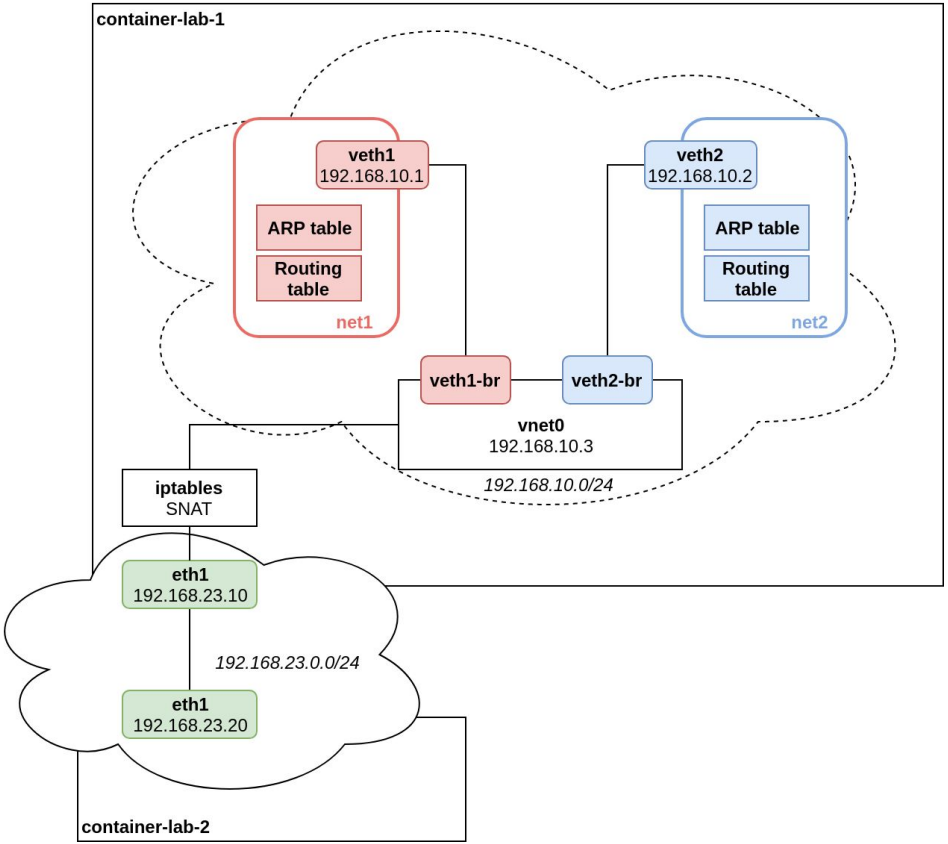


05_2host


container-lab-1
veth1
192.168.10.1
ARP table
Routing table
net1
veth2
192.168.10.2
ARP table
Routing table
net2
veth1-br
veth2-br
vnet0
192.168.10.3
192.168.10.0/24
iptables
SNAT
eth1
192.168.23.10
192.168.23.0/24
eth1
192.168.23.20
container-lab-2

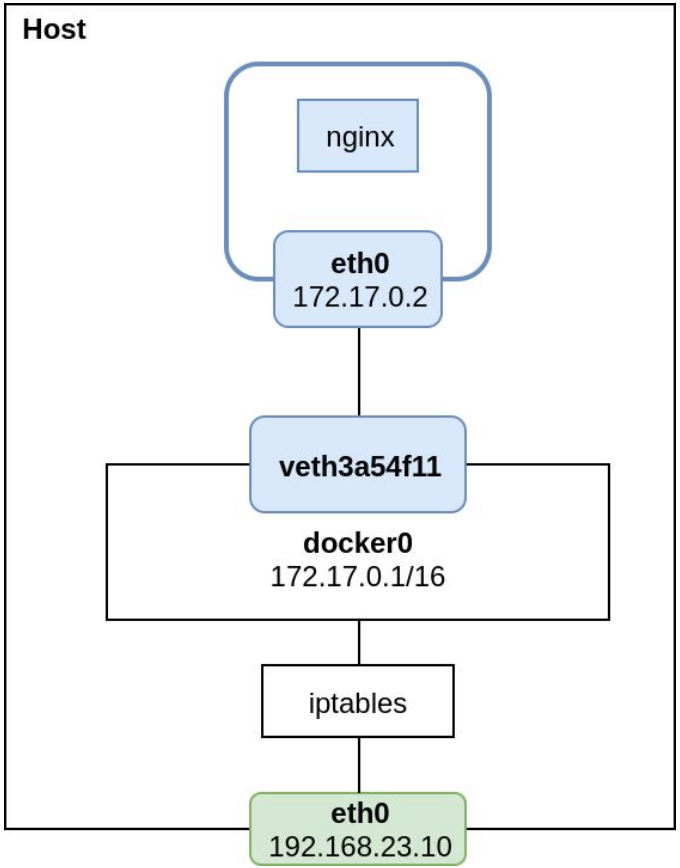
How can we reach `net1` from `container-lab-2`?

- Destination NAT with Port Mapping !



Container Networking

With Docker


Host
nginx
eth0
172.17.0.2
veth3a54f11
docker0
172.17.0.1/16
iptables
eth0
192.168.23.10

06_docker

Source NAT

```
-A POSTROUTING -s 172.17.0.0/16 ! -o docker0 -j MASQUERADE
```

For every packet which originates from the container network and is not sent to the docker0 interface, apply source NAT

Destination NAT

```
-A DOCKER ! -i docker0 -p tcp -m tcp --dport 8080 -j DNAT
--to-destination 172.17.0.2:80
```

For every packet that does not arrive over the docker0 interface and wants to reach port 8080, change the destination address to the container's on port 80

General Steps for Network Namespaces

1. Create a new network namespace
2. Create a bridge network
3. Create and assign a veth pair
4. Assign IP addresses
5. Setup NAT

Container Networking

With Container Networking Interface (CNI)

CNI

- Standard every container runtime needs to adhere to
- Specification of operations
- Plugins (executables) that implement the spec

CNI

- Standard every container runtime needs to adhere to
- Specification of operations
- Plugins (executables) that implement the spec

CNI Plugin

Overview

Each CNI plugin must be implemented as an executable that is invoked by the container management system (e.g. rkt or Kubernetes).

A CNI plugin is responsible for inserting a network interface into the container network namespace (e.g. one end of a veth pair) and making any necessary changes on the host (e.g. attaching the other end of the veth into a bridge). It should then assign the IP to the interface and setup the routes consistent with the IP Address Management section by invoking appropriate IPAM plugin.

Parameters

The operations that CNI plugins must support are:

- `ADD`: Add container to network
  - Parameters:
    - **Container ID**. A unique plaintext identifier for a container, allocated by the runtime. Must not be empty. Must start with a alphanumeric character, optionally followed by any combination of one or more alphanumeric characters, underscore (_), dot (.) or hyphen (-).
    - **Network namespace path**. This represents the path to the network namespace to be added, i.e. /proc/[pid]/ns/net or a bind-mount/link to it.
    - **Network configuration**. This is a JSON document describing a network to which a container can be joined. The schema is described below.
    - **Extra arguments**. This provides an alternative mechanism to allow simple configuration of CNI plugins on a per-container basis.
    - **Name of the interface inside the container**. This is the name that should be assigned to the interface created inside the container (network namespace); consequently it must comply with the standard Linux restrictions on interface names.
  - Result:
    - **Interfaces list**. Depending on the plugin, this can include the sandbox (eg, container or hypervisor) interface name and/or the host interface name, the hardware addresses of each interface, and details about the sandbox (if any) the interface is in.
    - **IP configuration assigned to each interface**. The IPv4 and/or IPv6 addresses, gateways, and routes assigned to sandbox and/or host interfaces.
    - **DNS information**. Dictionary that includes DNS information for nameservers, domain, search domains and options.
- `DEL`: Delete container from network
  - Parameters:
    - **Container ID**. as defined above

https://github.com/containernetworking/cni/blob/master/SPEC.md

CNI Spec

- **ADD**ing a container (= network namespace) to a network (= other container, machine, network device, etc).
- **DEL**eting a container from a network.
- **CHECK**ing that the container network's state is as desired.
- **VERSION** - report the version

CNI `bridge` Plugin

```
$ cat bridge.conf
{
    "cniVersion": "0.2.0",
    "name": "container_bridge",
    "type": "bridge",
    "bridge": "vnet0",
    "isDefaultGateway": true,
    "ipMasq": true,
    "ipam": {
        "type": "host-local",
        "ranges": [
            {
                "subnet": "192.168.10.0/24",
                "gateway": "192.168.10.3"
            },
        ],
        "routes": [
            { "dst": "0.0.0.0/0" },
        ]
    }
}
```

CNI `bridge` Plugin

```
CNI_COMMAND=ADD \
CNI_CONTAINERID=833aaaf9fd817b62d07515544039a
c3e12702f4e00f41732d77e42dc3b90a0c2 \
CNI_NETNS=/var/run/netns/833aaaf9fd81 \
CNI_IFNAME=eth0 \
CNI_PATH=$(pwd) \
    ./bridge bridge.conf
```

Summary

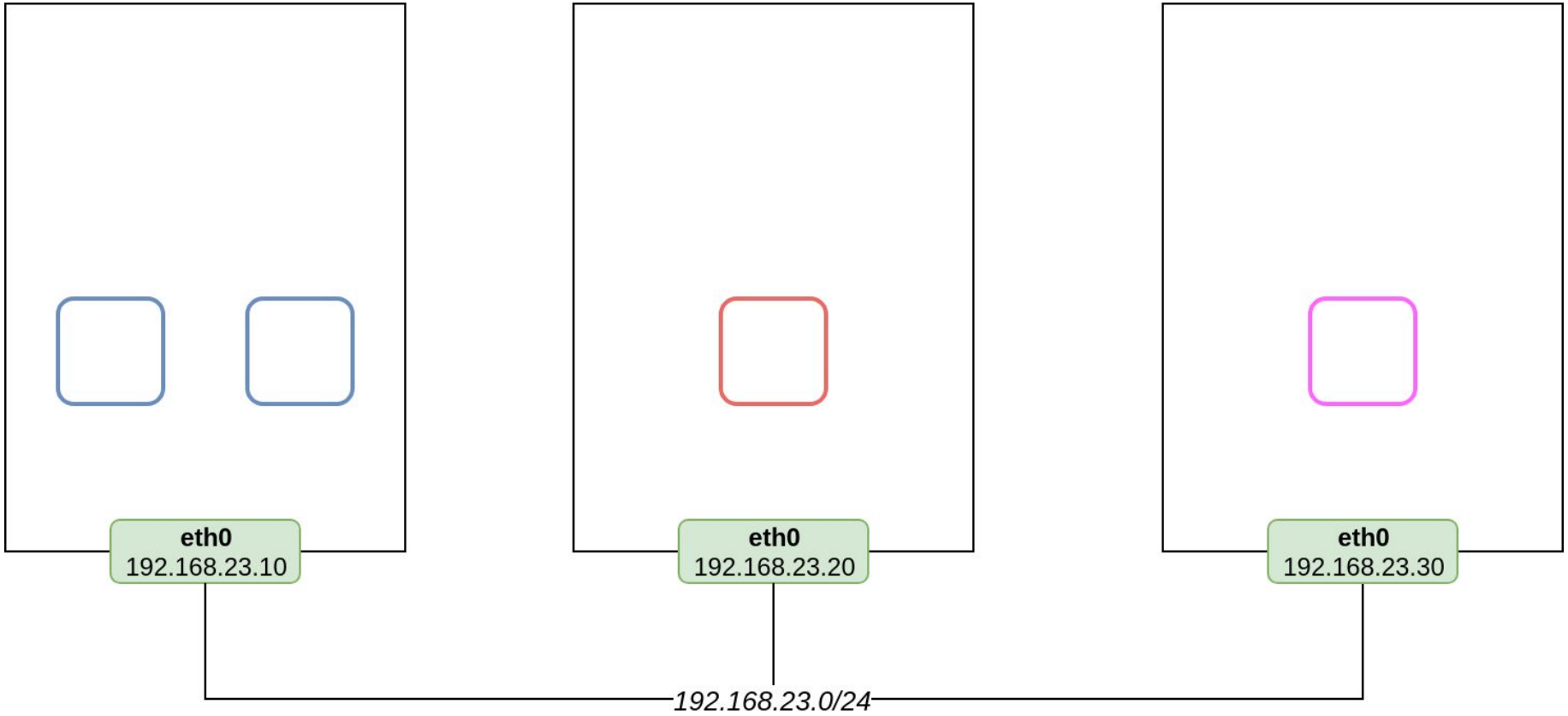
- Network namespace + veth pairs + bridge + routes + iptables
- From scratch with iproute2
- Via Docker
- Via CNI

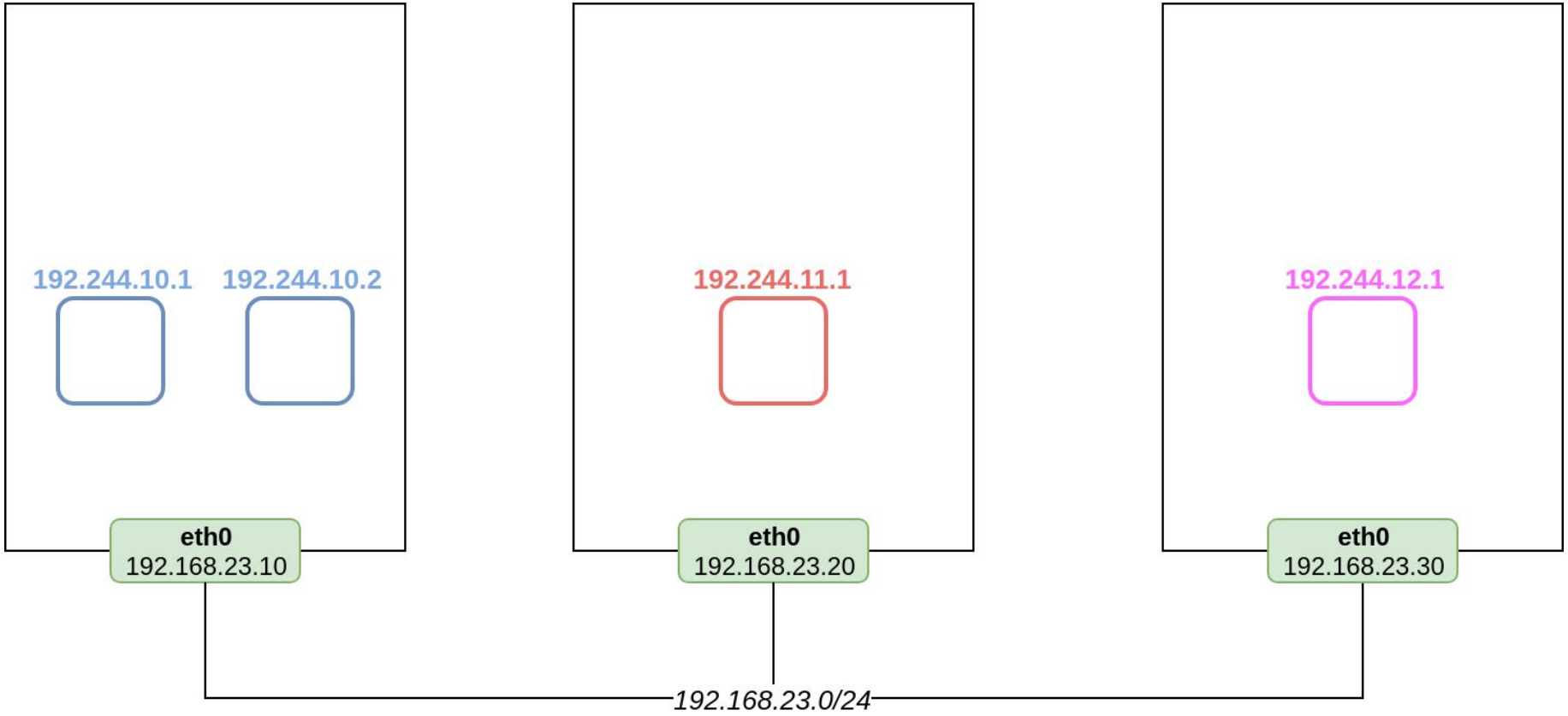
Kubernetes Networking

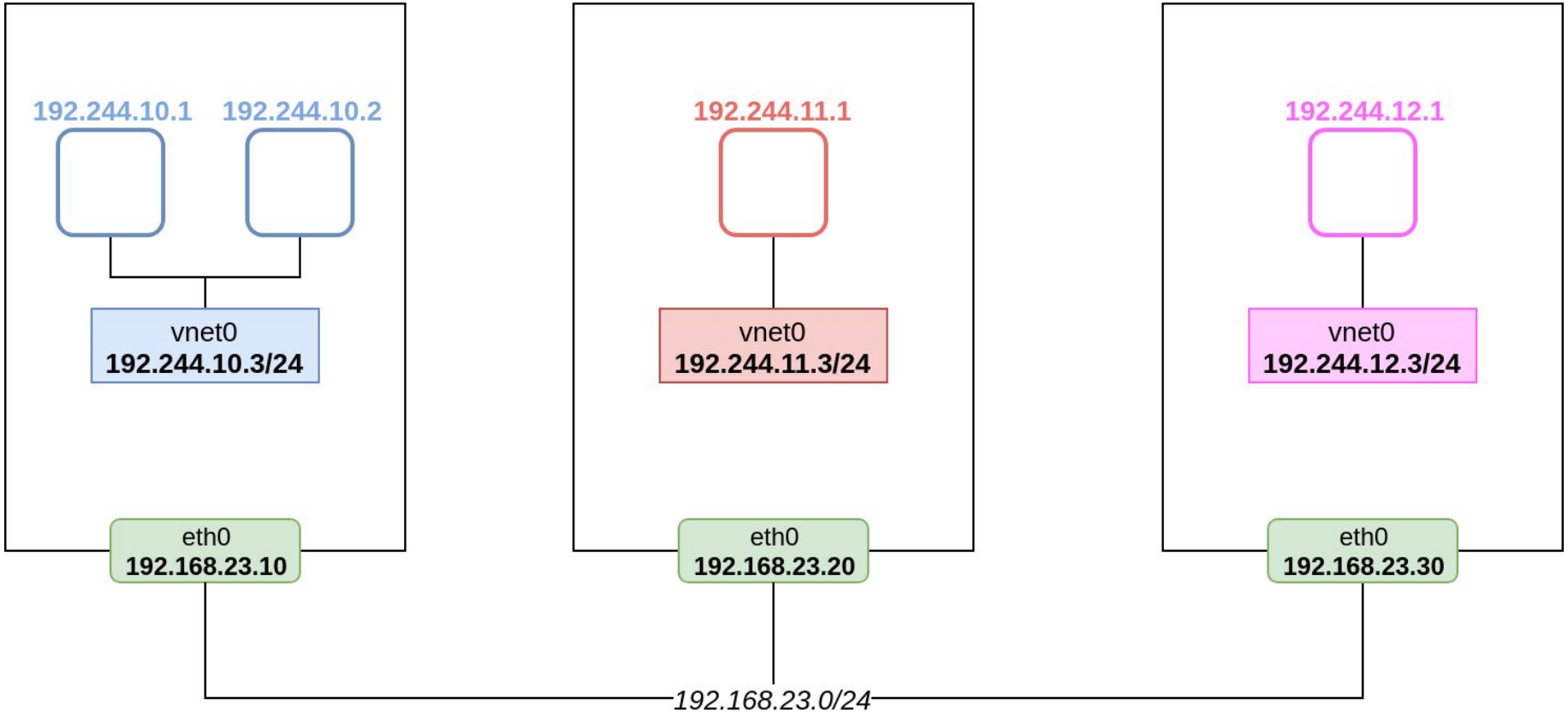
Pod Network

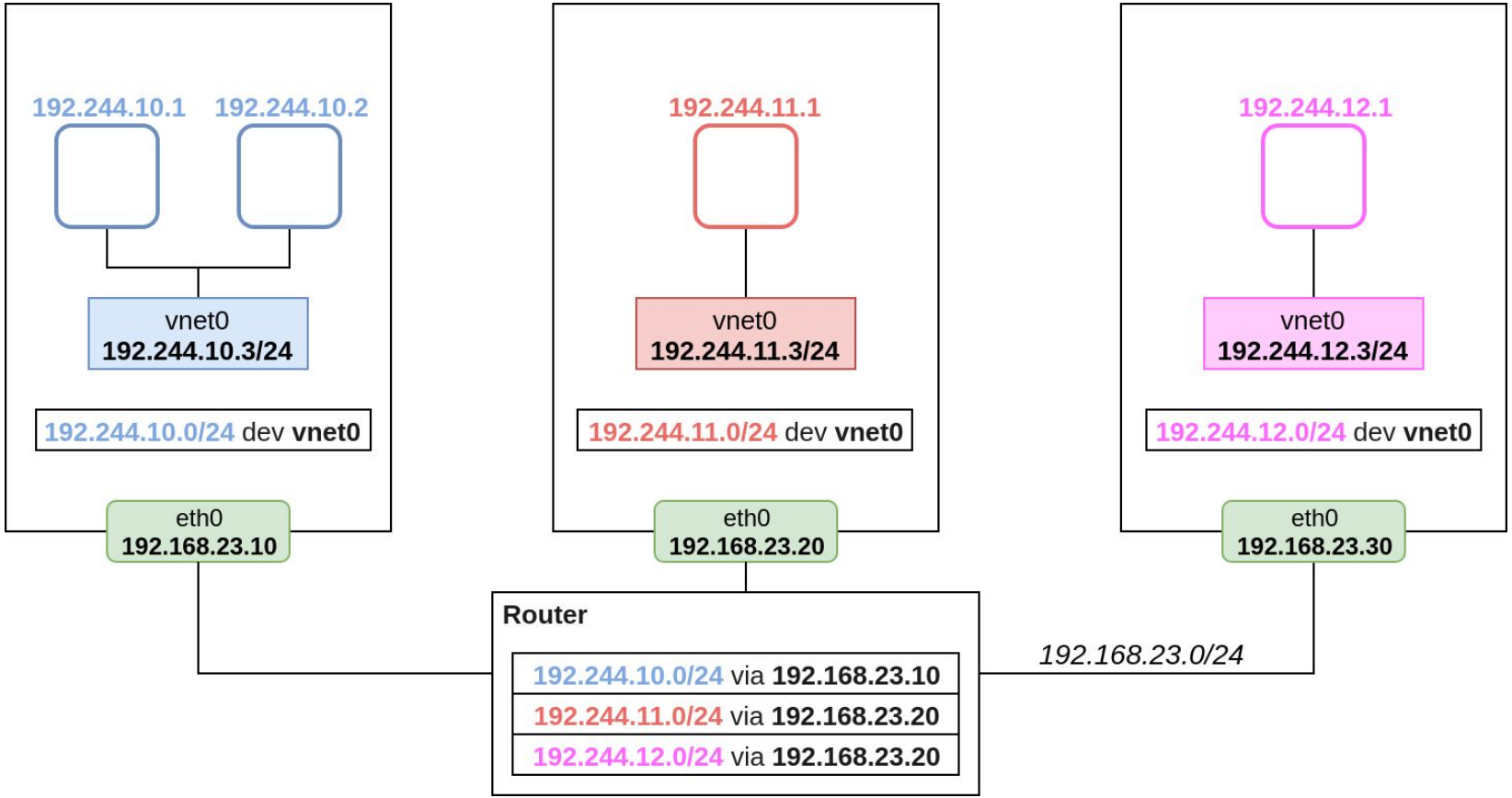
Kubernetes Network Model

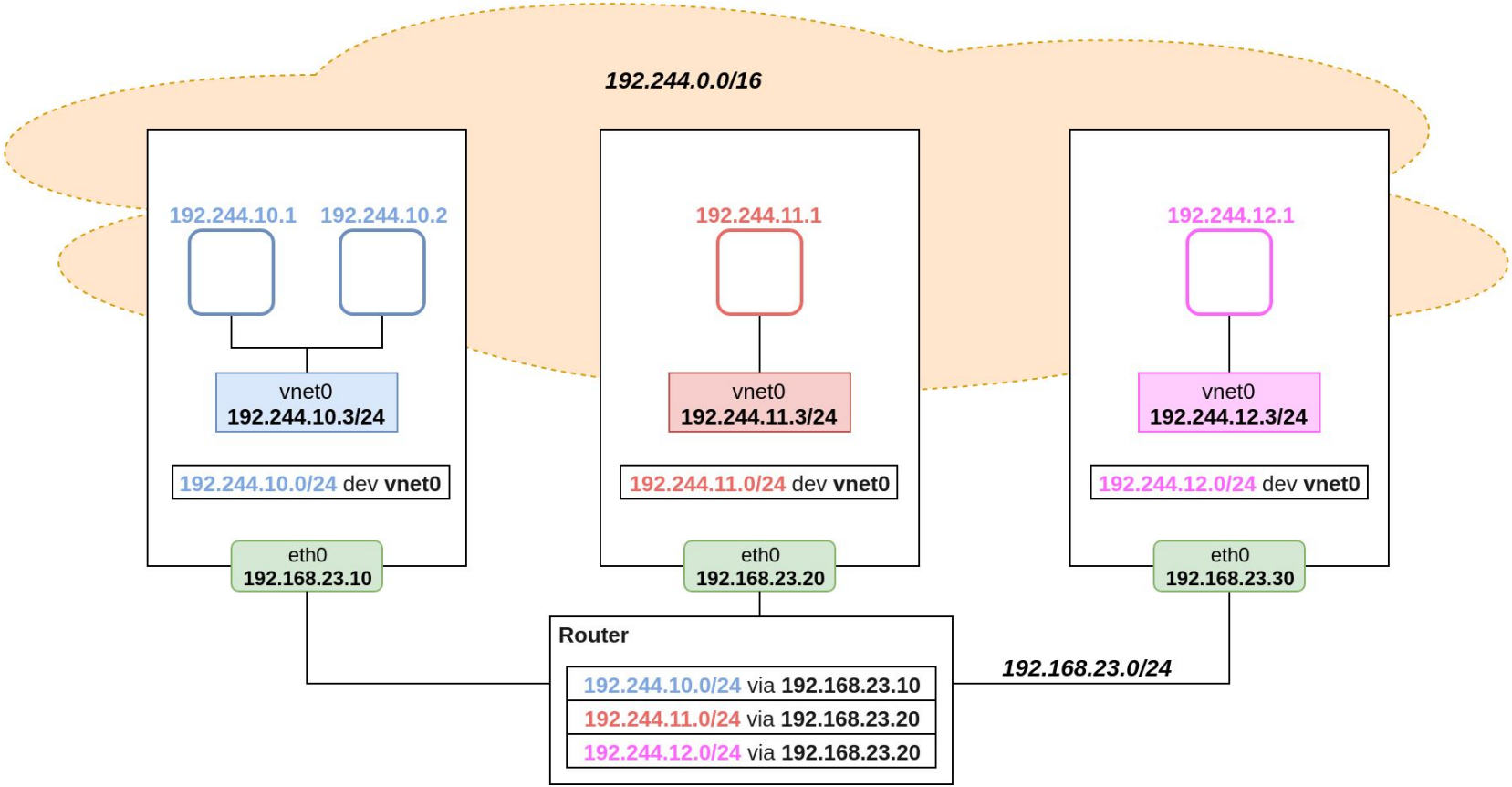
1. Every Pod has its own IP address
2. Every Pod can reach every other Pod without NAT


eth0
192.168.23.10
eth0
192.168.23.20
eth0
192.168.23.30
192.168.23.0/24


192.244.10.1
192.244.10.2
192.244.11.1
192.244.12.1
eth0
192.168.23.10
eth0
192.168.23.20
eth0
192.168.23.30
192.168.23.0/24

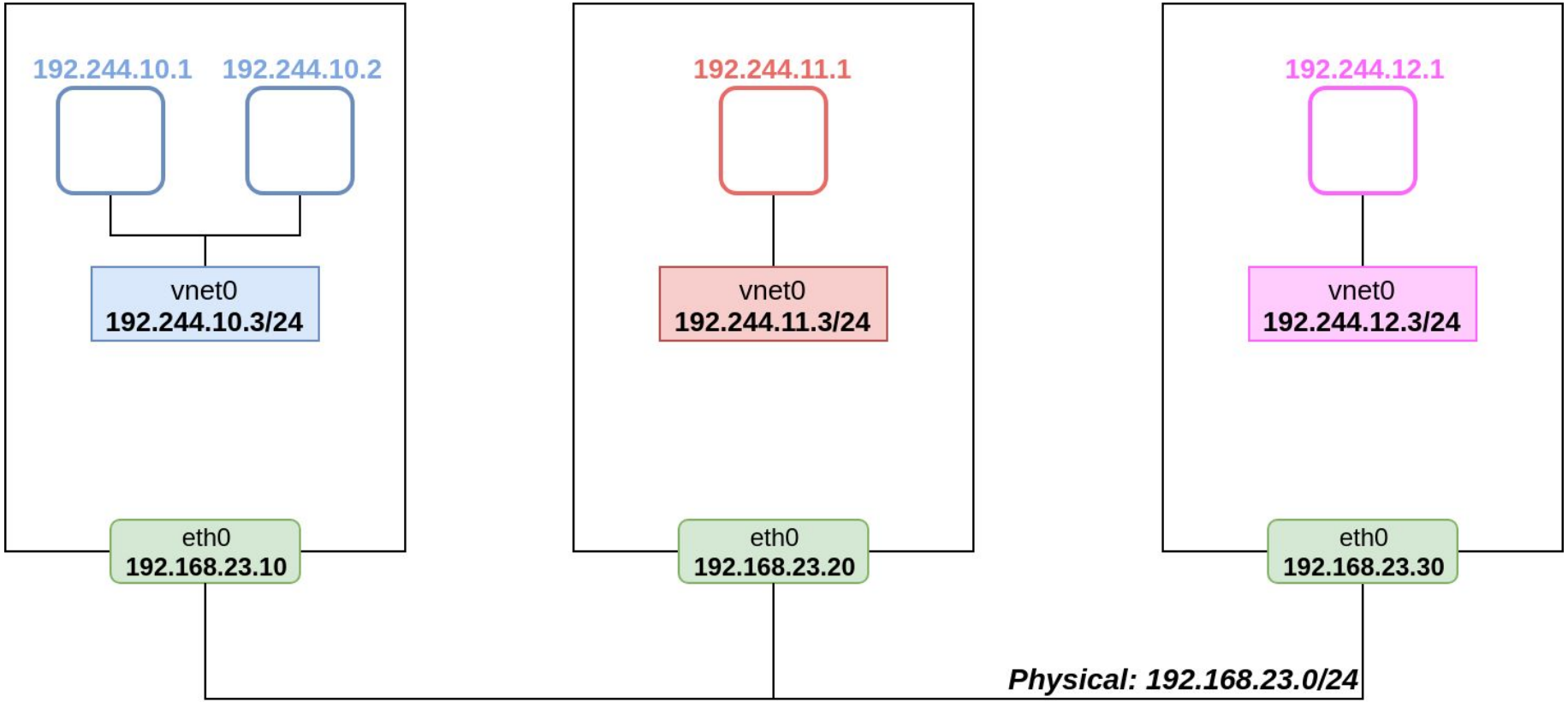

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
eth0
192.168.23.30
192.168.23.0/24

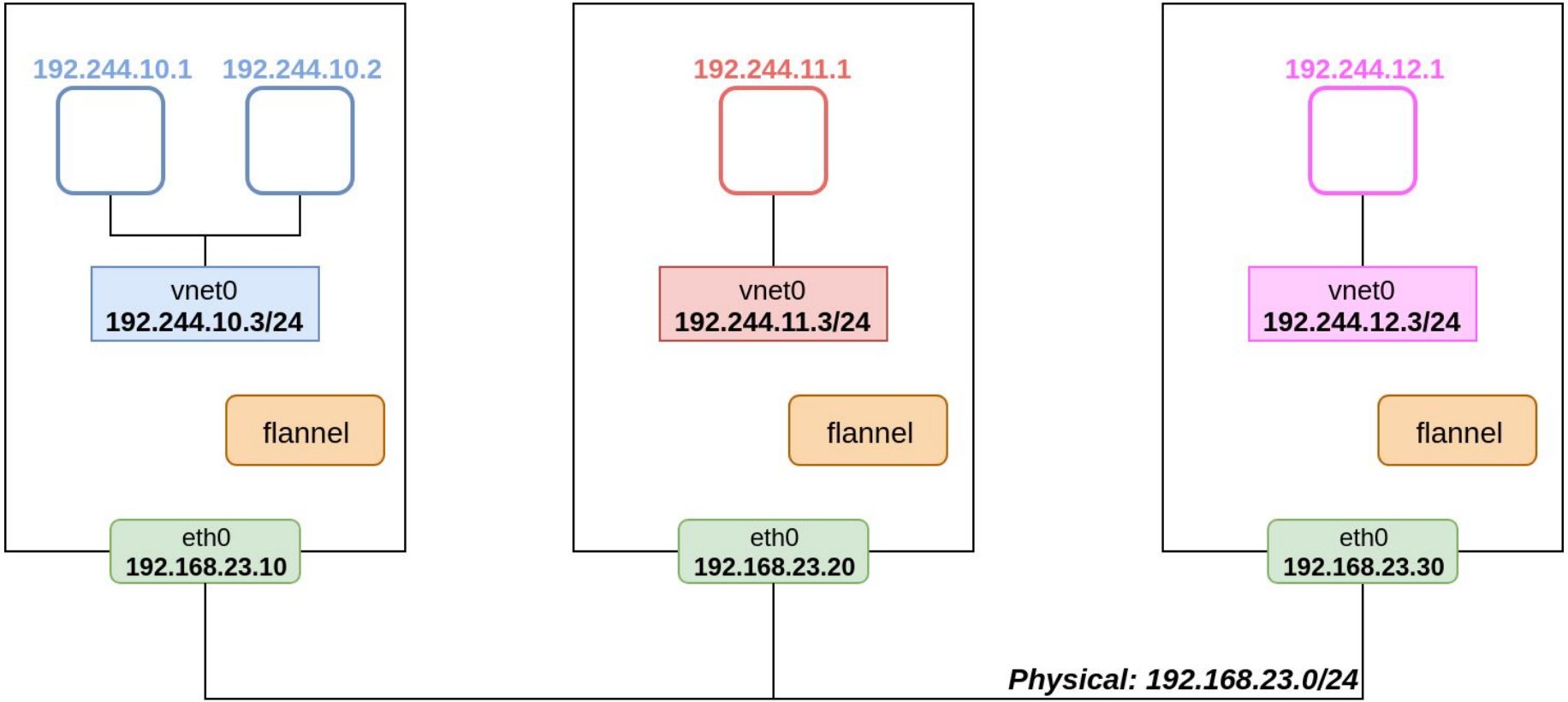

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
192.244.10.0/24 dev vnet0
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
192.244.11.0/24 dev vnet0
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
192.244.12.0/24 dev vnet0
eth0
192.168.23.30
Router
192.244.10.0/24 via 192.168.23.10
192.244.11.0/24 via 192.168.23.20
192.244.12.0/24 via 192.168.23.20
192.168.23.0/24

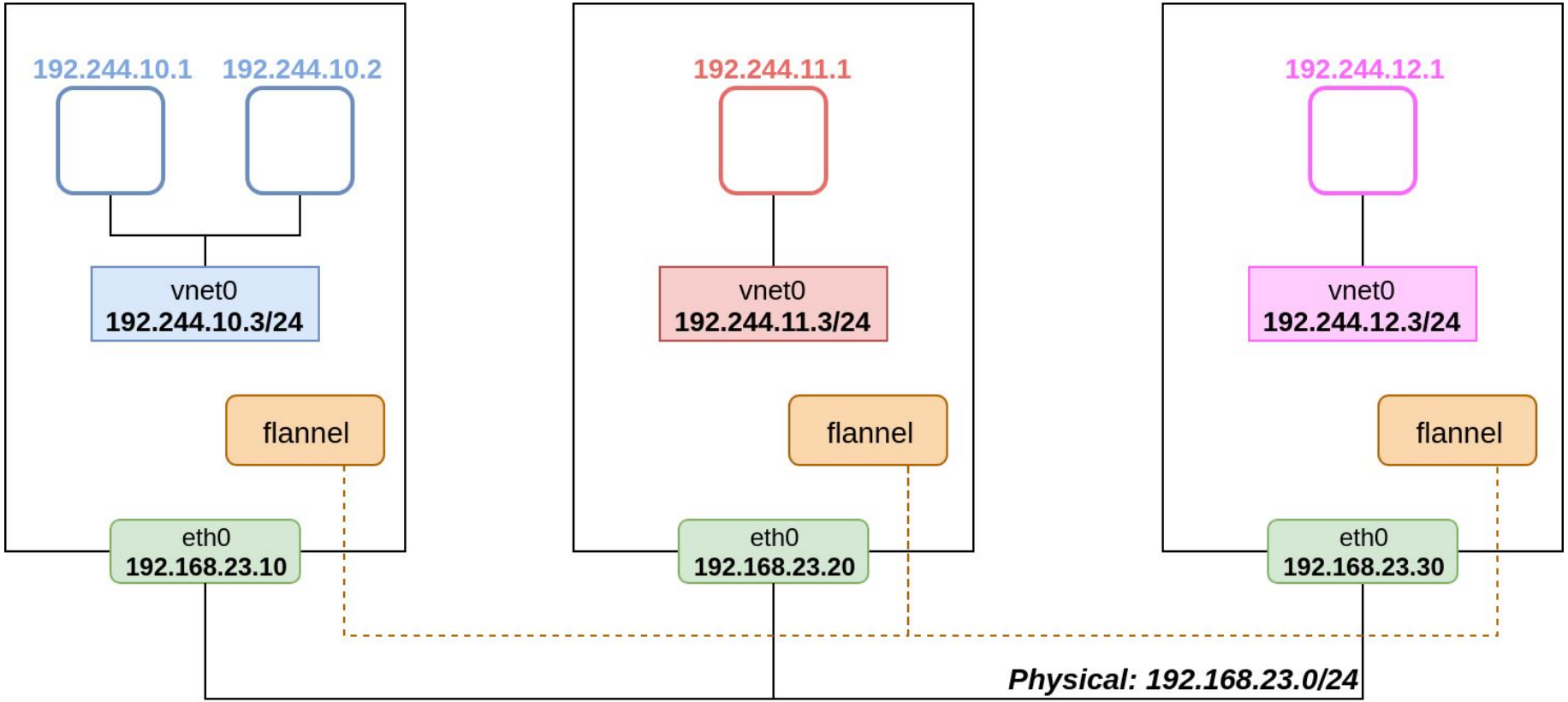

192.244.0.0/16
192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
192.244.10.0/24 dev vnet0
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
192.244.11.0/24 dev vnet0
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
192.244.12.0/24 dev vnet0
eth0
192.168.23.30
Router
192.244.10.0/24 via 192.168.23.10
192.244.11.0/24 via 192.168.23.20
192.244.12.0/24 via 192.168.23.20
192.168.23.0/24

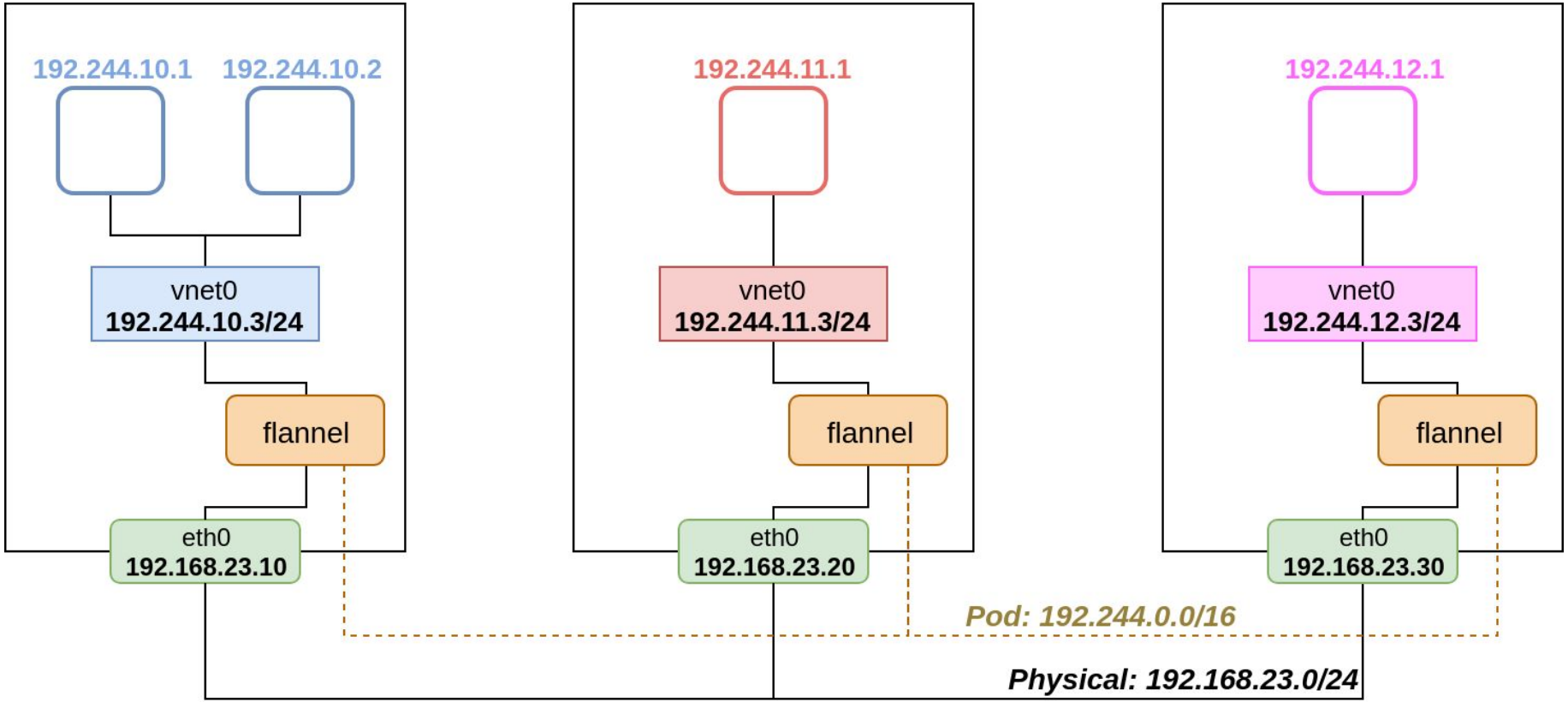
Overlay Networks

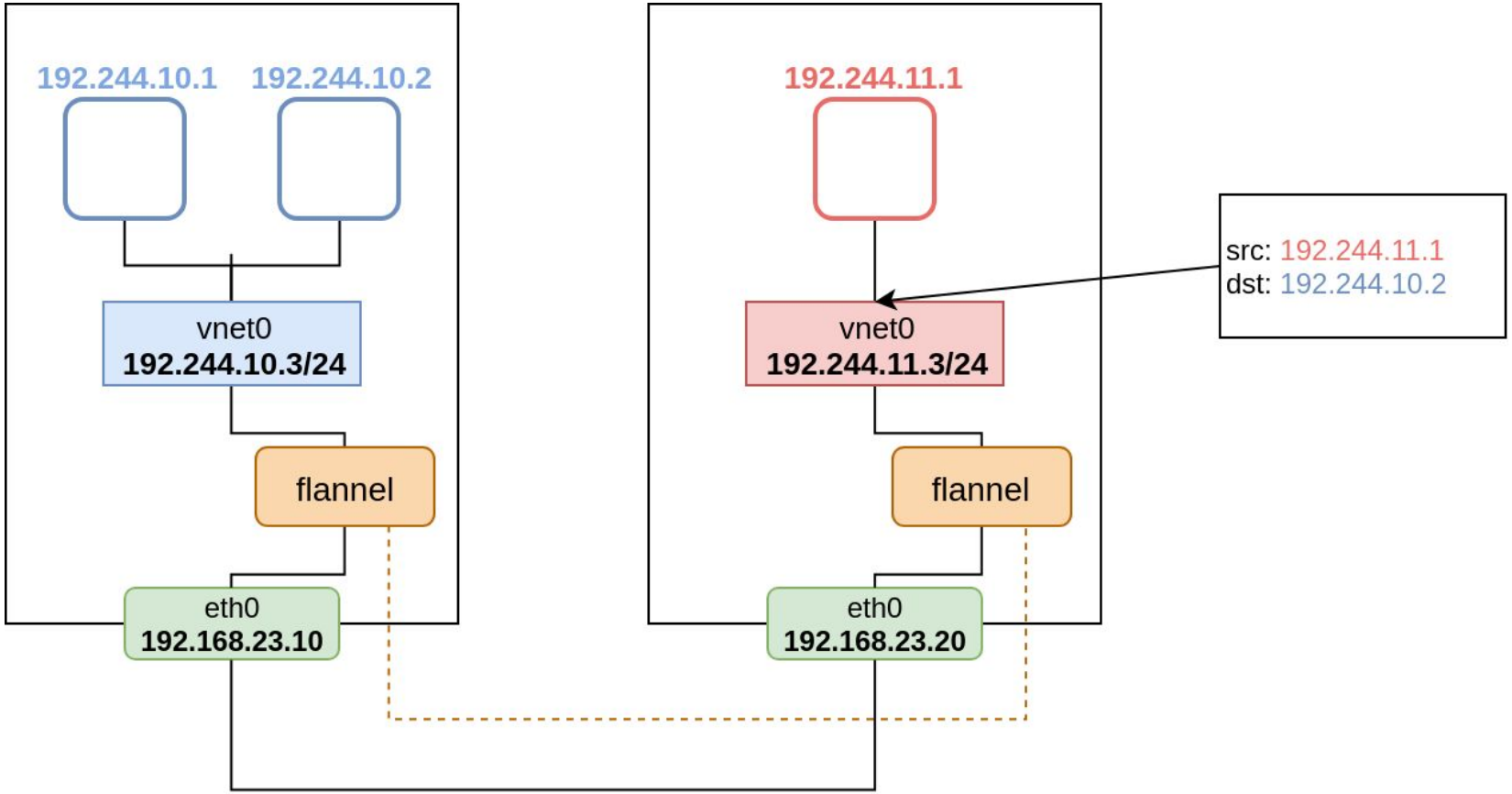
Flannel, WeaveNet, Calico, etc.

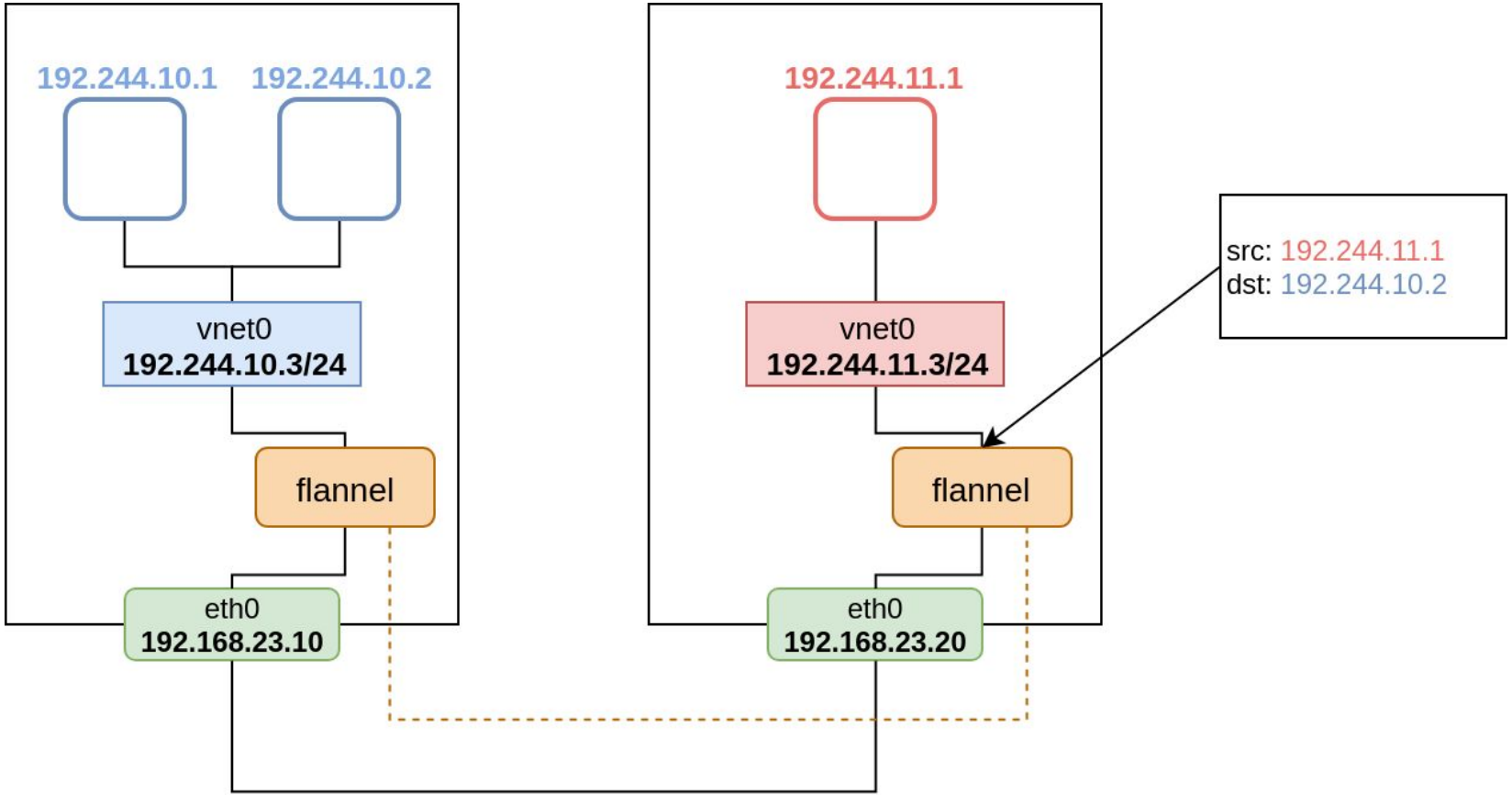

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
192.244.11.1
vnet0
192.244.11.3/24
192.244.12.1
vnet0
192.244.12.3/24
eth0
192.168.23.10
eth0
192.168.23.20
eth0
192.168.23.30
Physical: 192.168.23.0/24

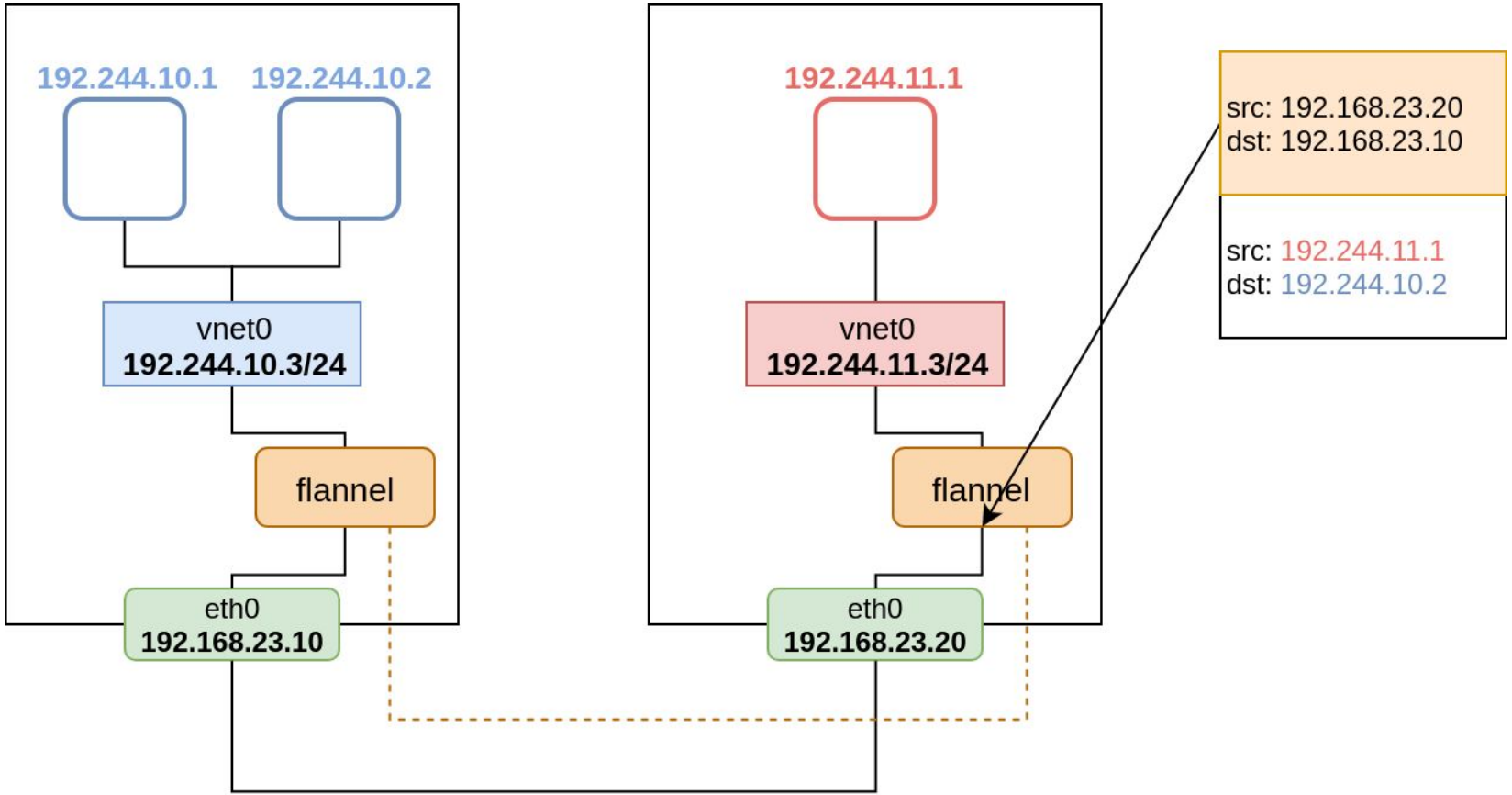

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
flannel
eth0
192.168.23.30
Physical: 192.168.23.0/24

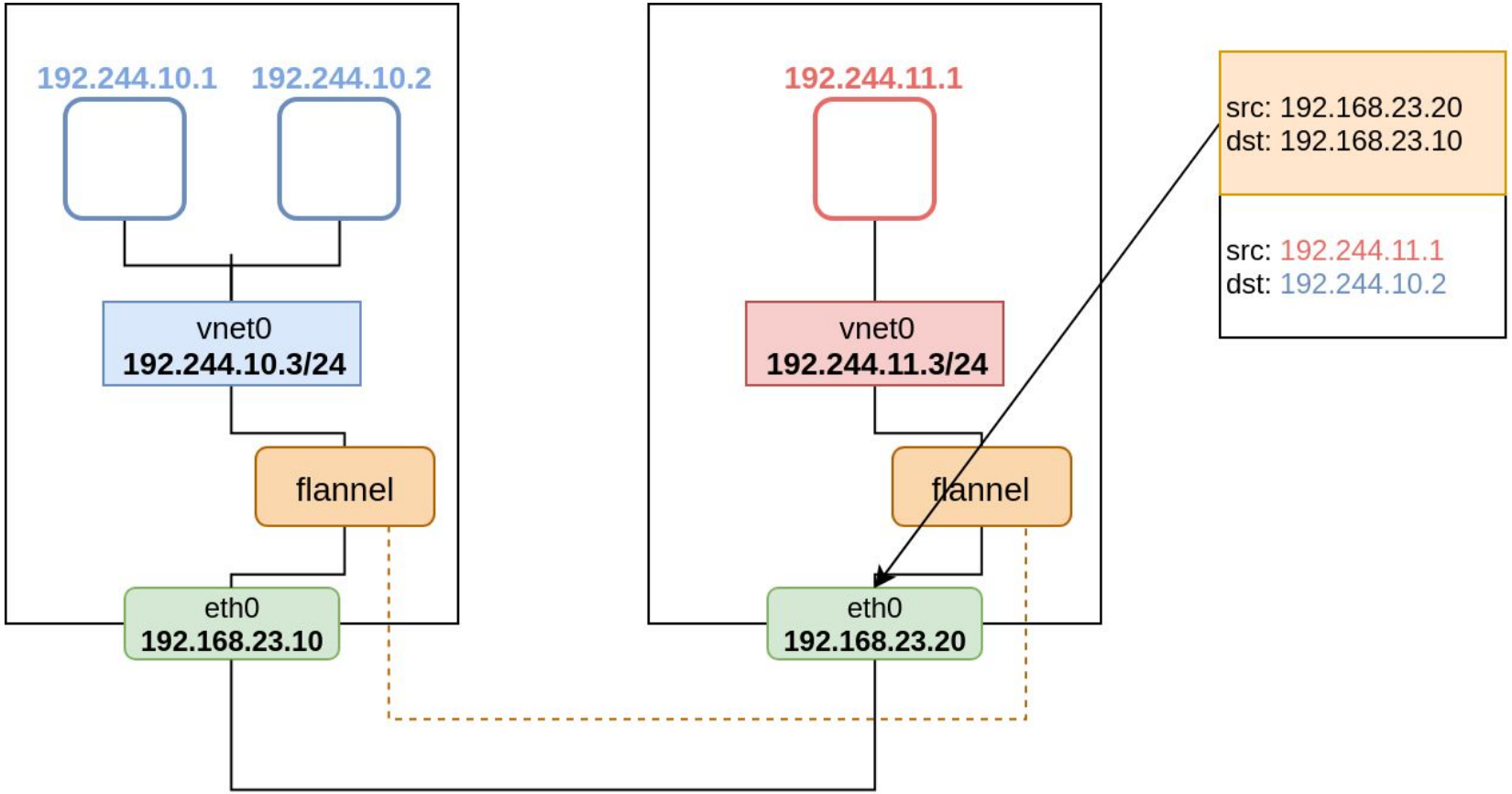

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
flannel
eth0
192.168.23.30
Physical: 192.168.23.0/24

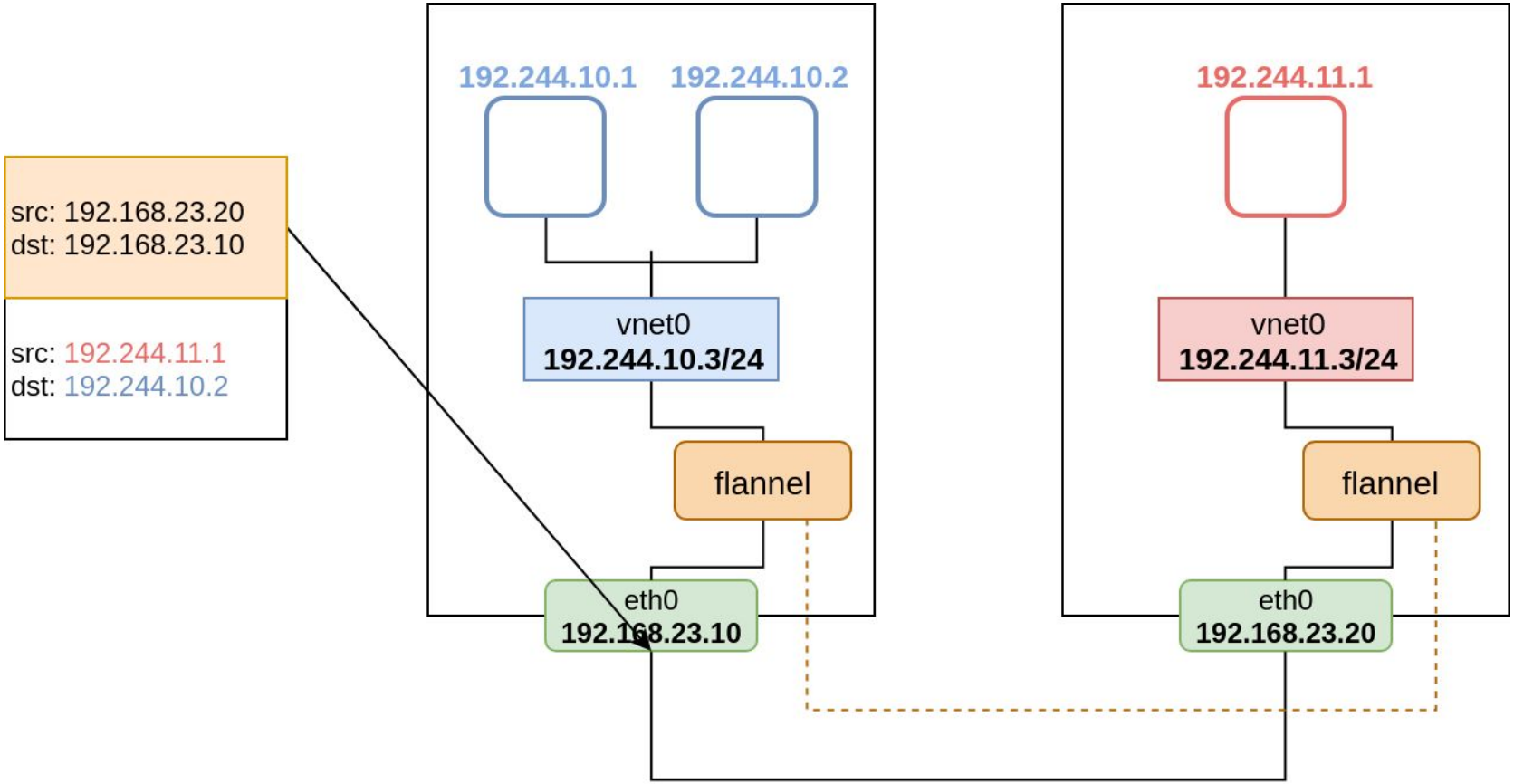

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
192.244.12.1
vnet0
192.244.12.3/24
flannel
eth0
192.168.23.30
Pod: 192.244.0.0/16
Physical: 192.168.23.0/24

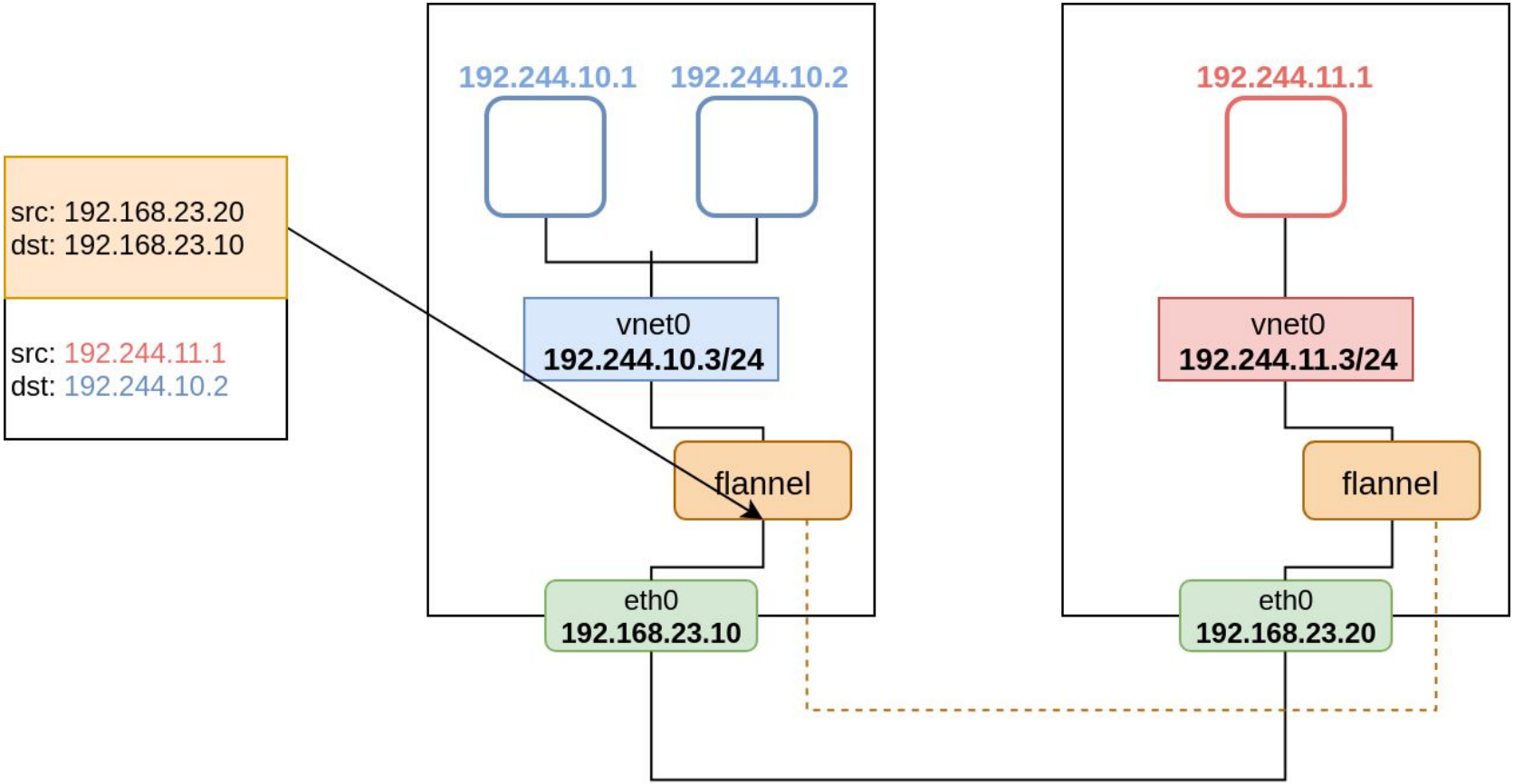

192.244.10.1
192.244.10.2
192.244.11.1
vnet0
192.244.10.3/24
vnet0
192.244.11.3/24
src: 192.244.11.1
dst: 192.244.10.2
flannel
flannel
eth0
192.168.23.10
eth0
192.168.23.20

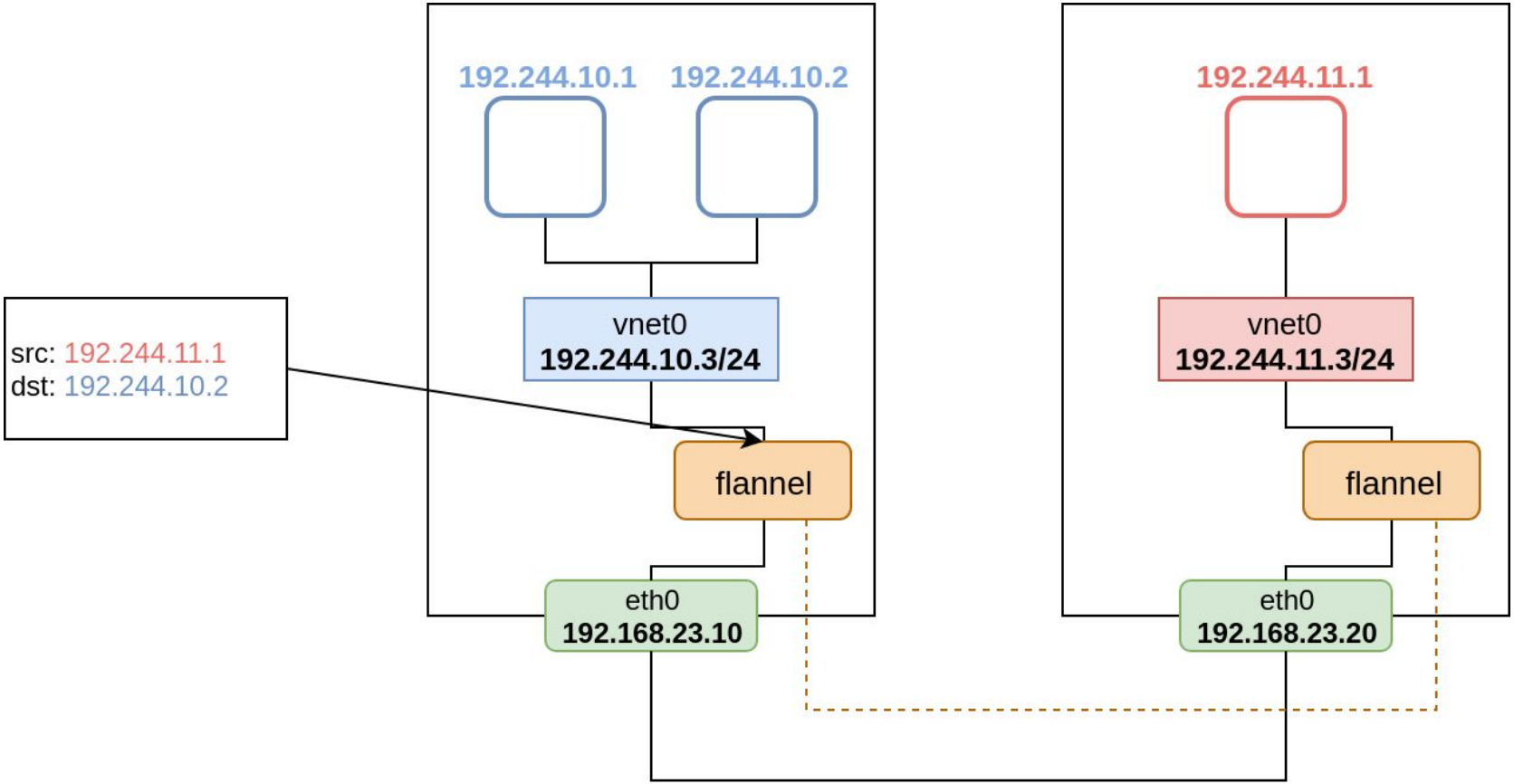

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
src: 192.244.11.1
dst: 192.244.10.2

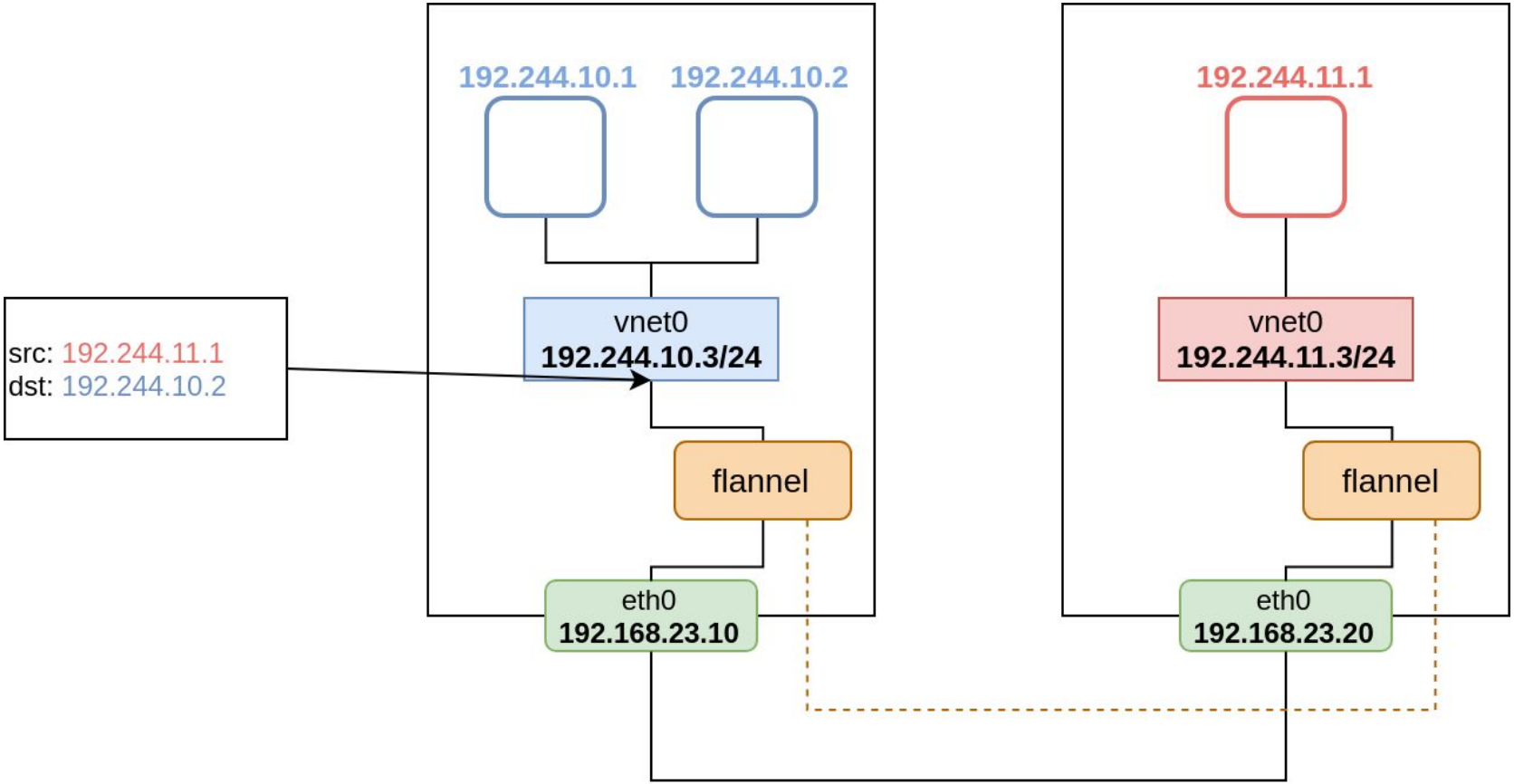

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
src: 192.168.23.20
dst: 192.168.23.10
src: 192.244.11.1
dst: 192.244.10.2

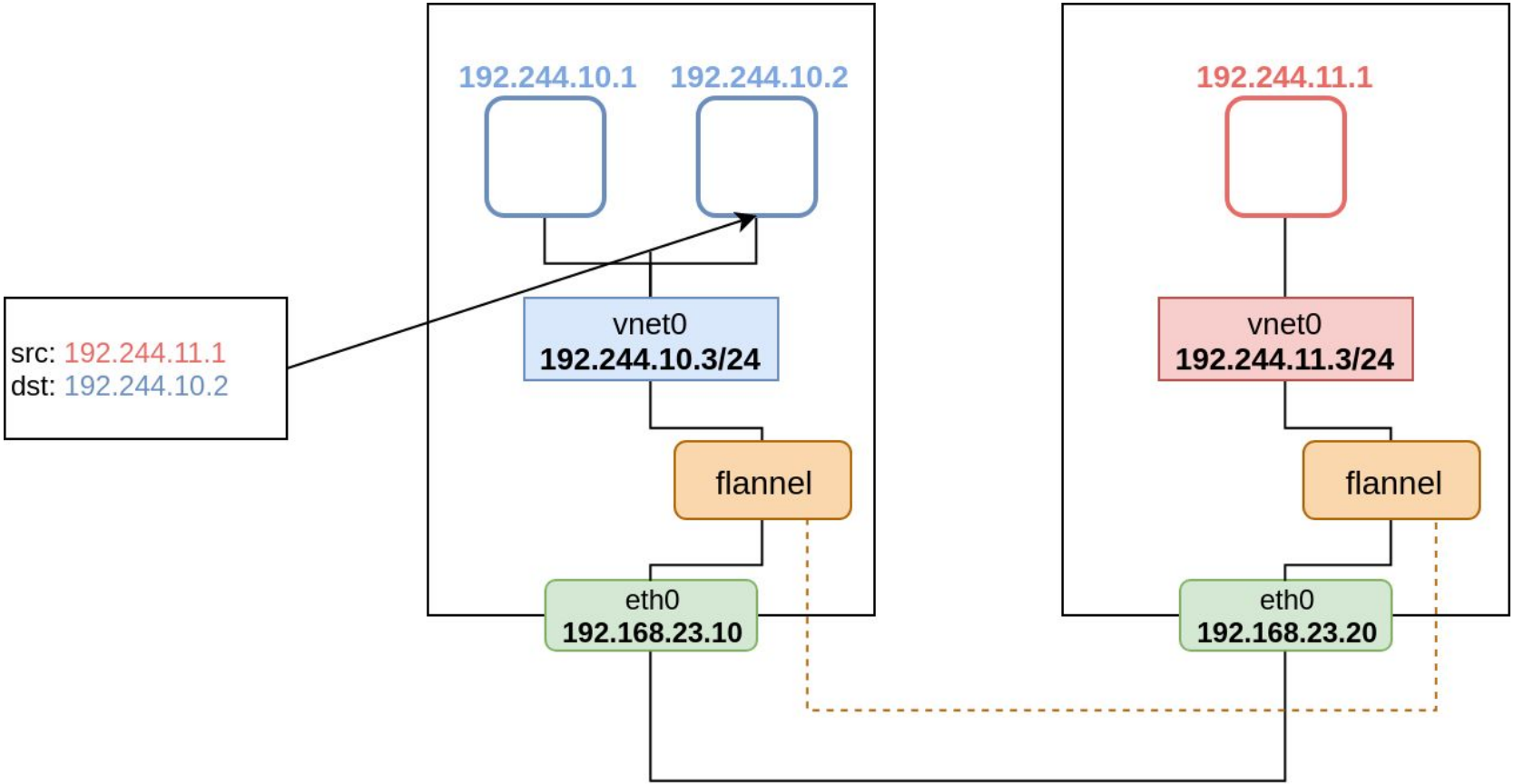

192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20
src: 192.168.23.20
dst: 192.168.23.10
src: 192.244.11.1
dst: 192.244.10.2


src: 192.168.23.20
dst: 192.168.23.10
src: 192.244.11.1
dst: 192.244.10.2
192.244.10.1
192.244.10.2
vnet0
192.244.10.3/24
flannel
eth0
192.168.23.10
192.244.11.1
vnet0
192.244.11.3/24
flannel
eth0
192.168.23.20

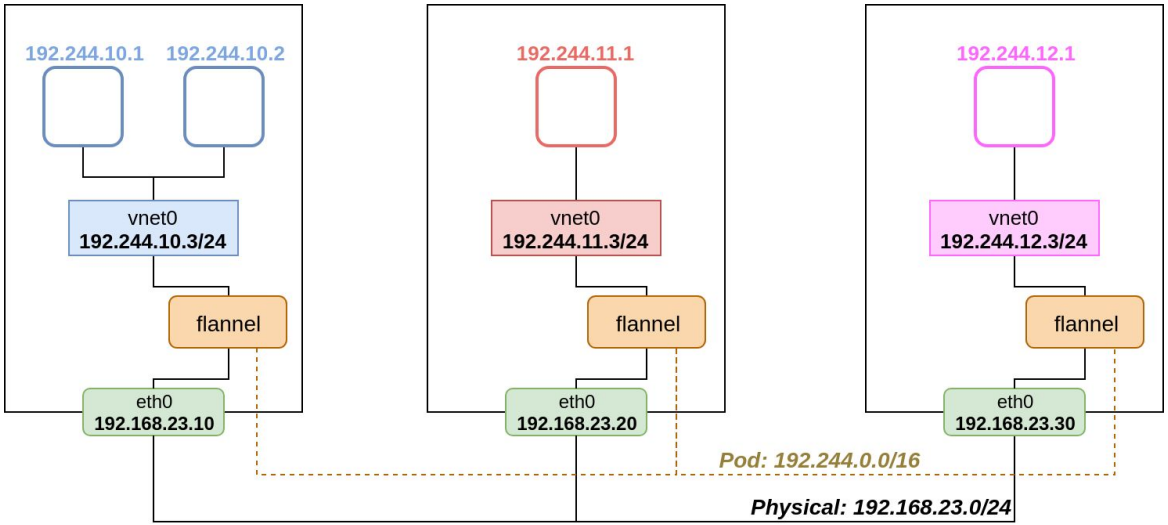

192.244.10.1
192.244.10.2
192.244.11.1
src: 192.168.23.20
dst: 192.168.23.10
src: 192.244.11.1
dst: 192.244.10.2
vnet0
192.244.10.3/24
vnet0
192.244.11.3/24
flannel
flannel
eth0
192.168.23.10
eth0
192.168.23.20


192.244.10.1
192.244.10.2
192.244.11.1
vnet0
192.244.10.3/24
vnet0
192.244.11.3/24
src: 192.244.11.1
dst: 192.244.10.2
flannel
flannel
eth0
192.168.23.10
eth0
192.168.23.20


192.244.10.1
192.244.10.2
192.244.11.1
src: 192.244.11.1
dst: 192.244.10.2
vnet0
192.244.10.3/24
vnet0
192.244.11.3/24
flannel
flannel
eth0
192.168.23.10
eth0
192.168.23.20


192.244.10.1
192.244.10.2
192.244.11.1
src: 192.244.11.1
dst: 192.244.10.2
vnet0
192.244.10.3/24
vnet0
192.244.11.3/24
flannel
flannel
eth0
192.168.23.10
eth0
192.168.23.20

How to deal with changes?

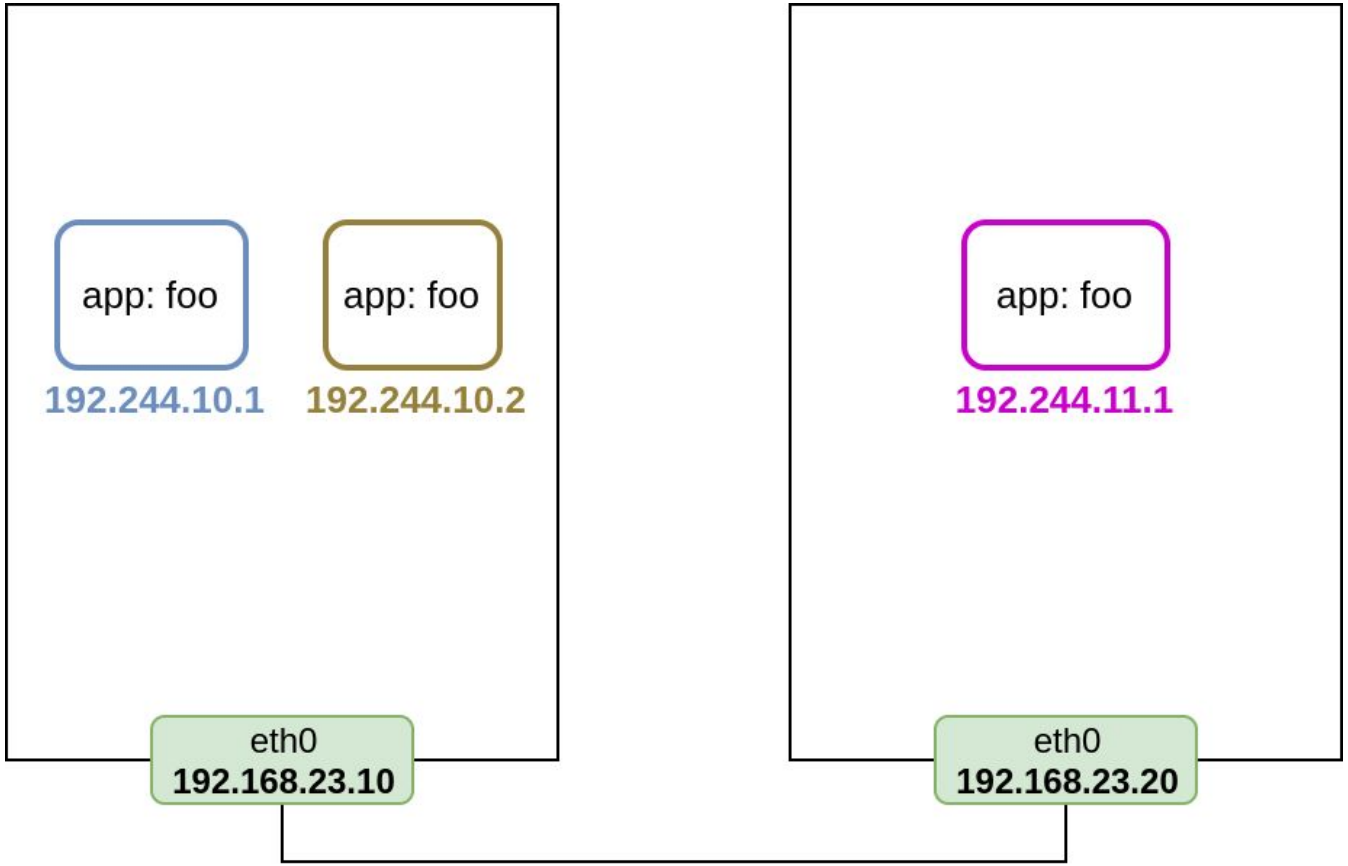


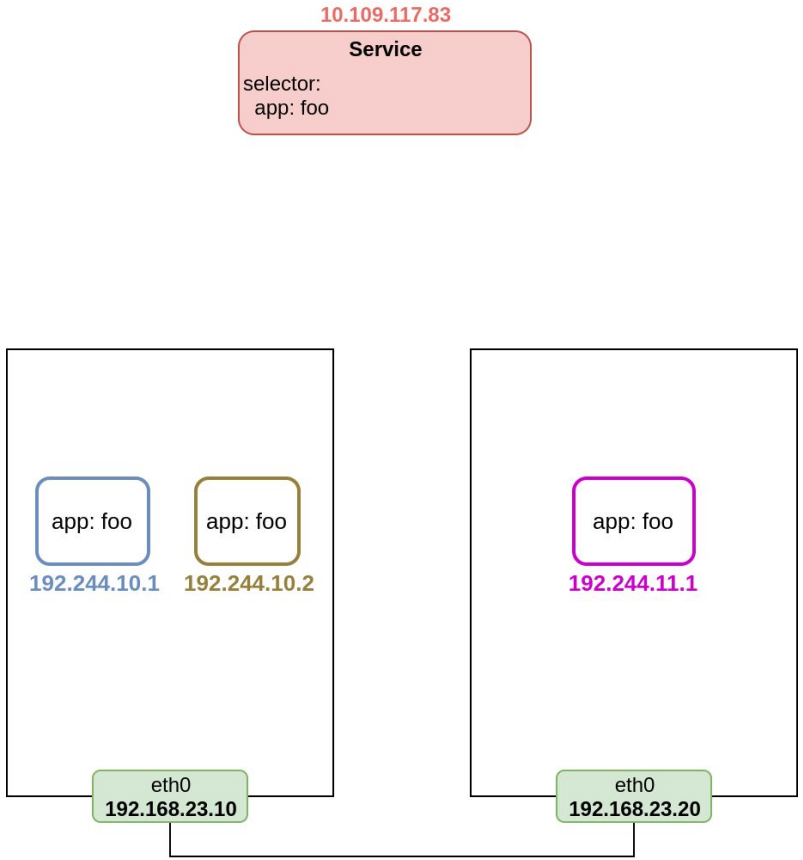
Kubernetes Networking

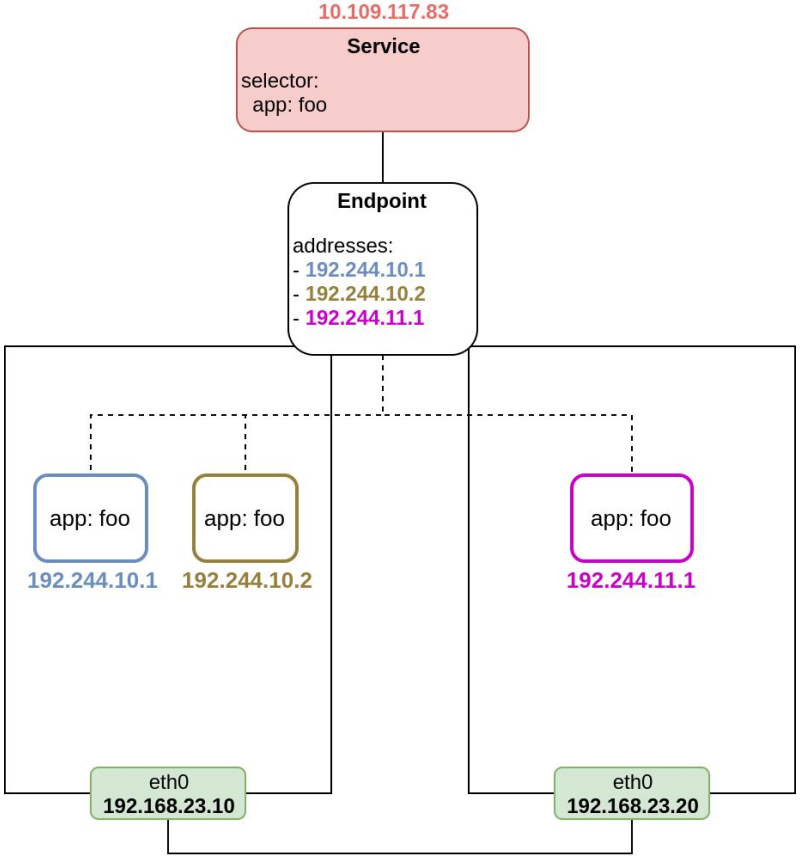
Service Network

Services

- Provide access to a group of Pods
  - Based on selectors
- Provide a static virtual IP (or Cluster IP)
  - Pod IPs can change, virtual IP stays stable
- Provide DNS A Record
- Provide basic load balancing between backends


app: foo
app: foo
192.244.10.1
192.244.10.2
app: foo
192.244.11.1
eth0
192.168.23.10
eth0
192.168.23.20


10.109.117.83
Service
selector:
app: foo
app: foo
192.244.10.1
app: foo
192.244.10.2
app: foo
192.244.11.1
eth0
192.168.23.10
eth0
192.168.23.20


10.109.117.83
Service
selector:
app: foo
Endpoint
addresses:
- 192.244.10.1
- 192.244.10.2
- 192.244.11.1
app: foo
192.244.10.1
app: foo
192.244.10.2
app: foo
192.244.11.1
eth0
192.168.23.10
eth0
192.168.23.20

Services are just a concept

- Cluster IP never gets assigned to an interface
- Kube-proxy sets iptables rules

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

```
NAME                  READY   STATUS    RESTARTS   AGE    IP           NODE                   NOMINATED NODE
READINESS GATES
nginx-5c7588df-5ft7g  1/1     Running   0          2m4s   10.233.3.2   container-lab-node-2   <none>
<none>
nginx-5c7588df-72crz  1/1     Running   0          2m4s   10.233.1.4   container-lab-node-1   <none>
<none>
nginx-5c7588df-zlrpm  1/1     Running   0          2m4s   10.233.2.3   container-lab-node-3   <none>
<none>
```

```
$ kubectl get svc nginx
NAME    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
nginx   ClusterIP   10.101.186.205   <none>        80/TCP    2m43s
```

```
$ kubectl describe svc nginx
Name:              nginx
Namespace:         default
Labels:            <none>
Annotations:       kubectl.kubernetes.io/last-applied-configuration:
                     {"apiVersion":"v1","kind":"Service","metadata":{"annotations":
{},"name":"nginx","namespace":"default"},"spec":{"ports":[{"port":80,"protoc...
Selector:          app=nginx
Type:              ClusterIP
IP:                10.101.186.205
Port:              <unset>  80/TCP
TargetPort:        80/TCP
Endpoints:         10.233.1.4:80,10.233.2.3:80,10.233.3.2:80
Session Affinity:  None
Events:            <none>
```

```
$ kubectl describe ep nginx
Name:         nginx
Namespace:    default
Labels:       <none>
Annotations:  <none>
Subsets:
  Addresses:          10.233.1.4,10.233.2.3,10.233.3.2
  NotReadyAddresses:  <none>
  Ports:
    Name     Port  Protocol
    ----     ----  --------
    <unset>  80    TCP

Events:  <none>


NAME                   READY   STATUS    RESTARTS   AGE    IP           NODE                   NOMINATED NODE
READINESS GATES
nginx-5c7588df-5ft7g   1/1     Running   0          2m4s   10.233.3.2   container-lab-node-2   <none>
<none>
nginx-5c7588df-72crz   1/1     Running   0          2m4s   10.233.1.4   container-lab-node-1   <none>
<none>
nginx-5c7588df-zlrpm   1/1     Running   0          2m4s   10.233.2.3   container-lab-node-3   <none>
<none>
```

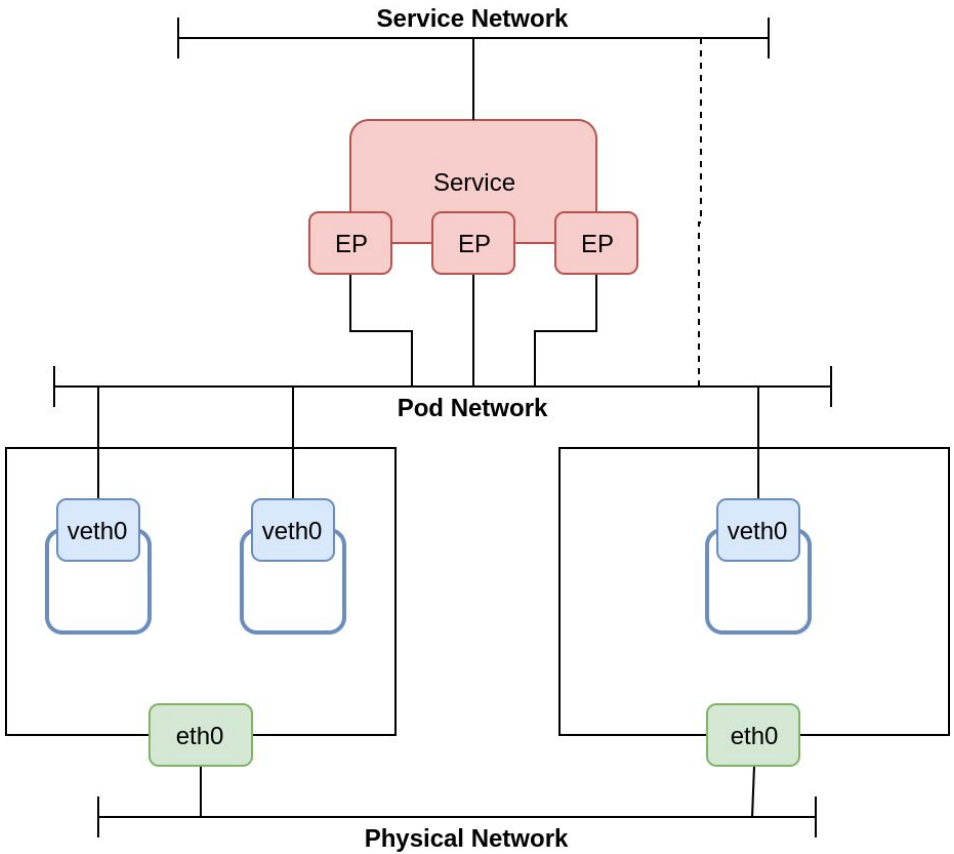
```
root@container-lab-node-1:/home/ubuntu# iptables -S -t nat
# Packet is trying to reach the Service
-A KUBE-SERVICES -d 10.101.186.205/32 -p tcp -m comment --comment "default/nginx: cluster IP" -m tcp --dport
80 -j KUBE-SVC-4N57TFCL4MD7ZTD

# Pick a backend at random
-A KUBE-SVC-4N57TFCL4MD7ZTDA -m statistic --mode random --probability 0.33332999982 -j KUBE-SEP-
G3LILAGUUTJ5TSQC
-A KUBE-SVC-4N57TFCL4MD7ZTDA -m statistic --mode random --probability 0.50000000000 -j KUBE-SEP-
R7LPE5NHODJD6NGK
-A KUBE-SVC-4N57TFCL4MD7ZTDA -j KUBE-SEP-LVBQQ74AIHZ2L74S

# Route to the Pod
-A KUBE-SEP-G3LILAGUUTJ5TSQC -p tcp -m tcp -j DNAT --to-destination 10.233.1.4:80
-A KUBE-SEP-R7LPE5NHODJD6NGK -p tcp -m tcp -j DNAT --to-destination 10.233.2.3:80
-A KUBE-SEP-LVBQQ74AIHZ2L74S -p tcp -m tcp -j DNAT --to-destination 10.233.3.2:80
```

```
$ kubectl describe svc -n kube-system kube-dns
Name:              kube-dns
Namespace:         kube-system
Labels:            k8s-app=kube-dns
                   kubernetes.io/cluster-service=true
                   kubernetes.io/name=KubeDNS
Annotations:       prometheus.io/port: 9153
                   prometheus.io/scrape: true
Selector:          k8s-app=kube-dns
Type:              ClusterIP
IP:                10.96.0.10
Port:              dns  53/UDP
TargetPort:        53/UDP
Endpoints:         10.233.0.2:53,10.233.0.3:53
Port:              dns-tcp  53/TCP
TargetPort:        53/TCP
Endpoints:         10.233.0.2:53,10.233.0.3:53
Session Affinity:  None
Events:            <none>
```

Summary

Three layers of networking:

1. **Physical network**: connecting nodes
2. **Pod network**: flat vs. overlay
3. **Service network**: providing static IPs + DNS A for pods



Thank you!


Pinned Tweet
Miek Gieben @miekg · 26 Jun 2018
Kubernetes networking
I SEE A SMALL cluster
FORTUNE TELLING
I SEE YOU... ...ALONE...
AND A LOT OF iptables
JESUS THAT'S A LOT OF iptables
15
797
2.0K