Pod-to-Service Networking

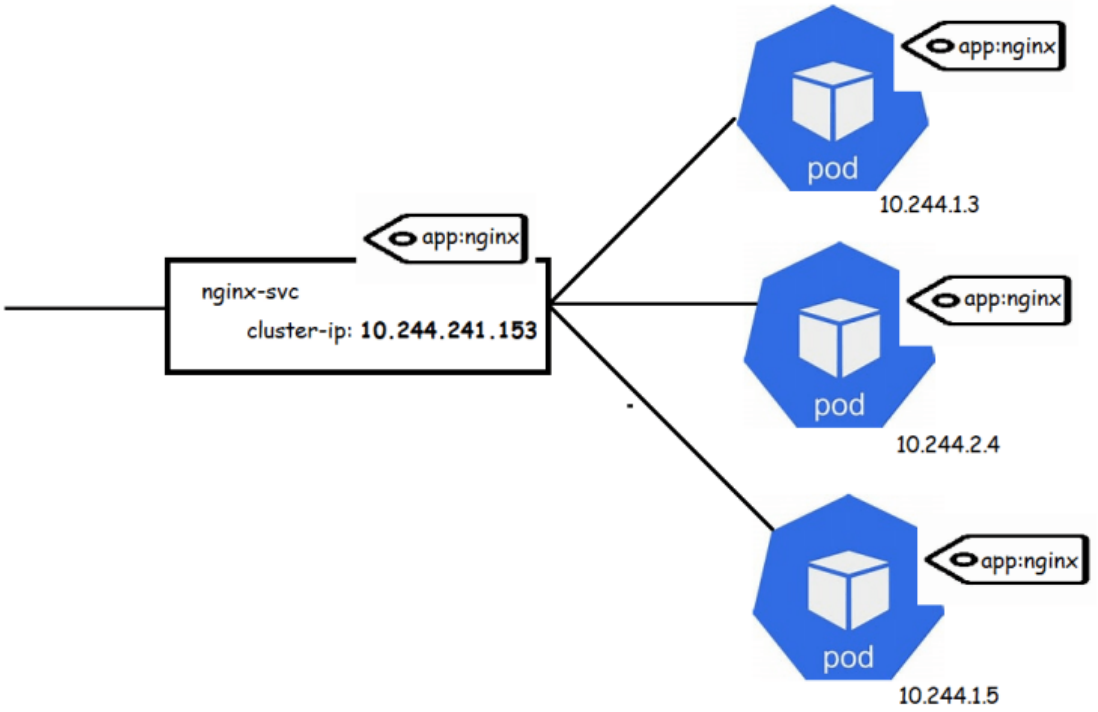
- Let's consider the following manifests.
- Deployment Manifest

```
apiVersion: apps/v1
kind: Deployment
metadata:
  # Unique key of the Deployment instance
  name: deployment-example
spec:
  # 3 Pods should exist at all times.
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        # Apply this label to pods and default
        # the Deployment label selector to this value
        app: nginx
    spec:
      containers:
      - name: nginx
        # Run this image
        image: nginx:1.14
```

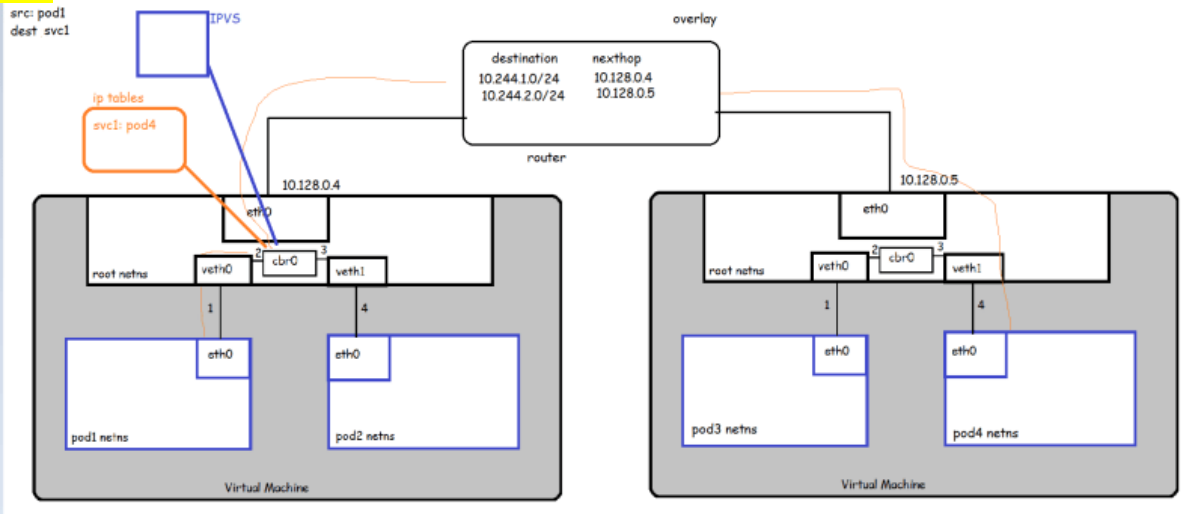
Service Manifest

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-svc
spec:
  selector:
    app: nginx
  ports:
    - port: 80
      targetPort: http
```

Above manifest will have pod and service communication as shown below.



- Each node will have a kube-proxy which is a key component. Its role is to load-balance the traffic that is desired for services (via cluster ip and node ports) to the correct backend pods.
- kube-proxy can run in one of three modes
  - user space
  - iptables
  - IPVS



Kubernetes networking uses **iptables** to control the network connections between pods (and between nodes), handling many of the networking and port forwarding rules. ... Also, port mapping is greatly simplified (and mostly eliminated) since each pod has its own IP address and its container can listen on its native port.

IPVS is incorporated into the LVS (Linux Virtual Server), where it runs on a host and acts as a load balancer in front of a cluster of real servers.

Userspace CNI is a Container Network Interface (CNI) Kubernetes* plugin that was designed to simplify the process of deployment of DPDK-based applications in Kubernetes pods. The plugin uses Kubernetes and Multus CNI CRDs to provide pods with a virtual DPDK-enabled Ethernet port.