Guide to the Kubernetes Networking Model (Deep-Dive Summary)

For easier learning, interview preparation, and architecture understanding

---

1. Kubernetes Basics

1.1 API Server

- Central control plane component.
- Exposes REST API.
- Backed by **etcd** (cluster state store).
- Everything in Kubernetes = API call to the API server.

1.2 Controllers

- Watch the API (desired state) → compare with actual state → reconcile.
- Example:
  - Scheduler assigns Pods to nodes.
  - Kubelet on each node configures container runtime & networking.

1.3 Pods

- Smallest deployable unit.
- One or more containers.
- Share:
  - **Network namespace** (same IP, same localhost)
  - **Volumes**

1.4 Nodes

- Worker machines (VMs or bare metal).
- Run:

  - kubelet
  - kube-proxy
  - runtime (containerd, CRI-O)

---

2. The Kubernetes Networking Model (Design Principles)

Kubernetes imposes 3 hard rules:

All Pods can talk to all other Pods WITHOUT NAT.
All Nodes can talk to all Pods WITHOUT NAT.
A Pod's IP is the same inside and outside the Pod.

Because of this, Kubernetes must solve:

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

---

3. Container-to-Container Networking (Inside a Pod)

Linux provides **network namespaces**, giving each container an isolated network stack.

Inside a Pod:

- Containers share **the same network namespace**.
- They share:
  - IP
  - Ports
  - loopback
- Containers communicate using **localhost**.

---

4. Pod-to-Pod Networking

Pods get **their own network namespaces**, so Kubernetes must interconnect them.

How Pods are connected inside a node

Each Pod's namespace uses a **veth pair**:

```
Pod eth0 ←→ vethXXX ←→ root namespace
```

Inside the node:

- All veth interfaces connect to a **Linux bridge** (like `cbr0`).

Traffic (same node)

1. Pod sends traffic → its eth0
2. Goes to veth → bridge
3. Bridge uses ARP to find destination Pod → forwards to its veth peer.

Traffic (different nodes)

Key idea:
Each node gets a **Pod CIDR range**, e.g.,

```
Node1: 10.0.1.0/24
Node2: 10.0.2.0/24
```

Flow:

1. Pod → veth → bridge
2. Bridge cannot ARP → routes to node's default interface (eth0)
3. Node sends packet over the network to the destination node
4. Destination node receives → routes to Pod via its veth → namespace

How does the network know which node owns which Pod IP range?

This is CNI-specific.

Example: **AWS VPC CNI**

- Assigns real VPC IPs to each Pod (via ENIs)
- No overlays
- Pods behave as first-class VPC citizens.

Other CNIs:

- Calico
- Flannel
- Weave
  … use overlays or BGP.

---

5. Pod-to-Service Networking (ClusterIP)

Pods come & go → IPs change frequently.

Service fixes this by providing:

- A stable **virtual IP (ClusterIP)**
- A load balancer inside the cluster

How Service traffic is routed?

Two implementations:

A. iptables (classic)

kube-proxy writes iptables rules:

- Watch for packets to ServiceIP
- Randomly pick a backend Pod
- DNAT (Destination NAT) to selected Pod

Return traffic is SNAT'ed back to Service IP (via conntrack).

B. IPVS (Kubernetes modern default)

- Uses kernel-level load balancing
- More scalable
- Creates a **dummy interface** and binds the Service IP
- Faster + better for large clusters

DNS (CoreDNS)

Kubernetes automatically creates DNS records for every Service:

```
myservice.mynamespace.svc.cluster.local
```

CoreDNS runs as a Pod + Service inside cluster.

---

6. Internet-to-Service Networking

Two directions:

A. Egress (Pod → Internet)

Problem:
Internet Gateway knows only **Node IPs**, not Pod IPs.

Solution:
iptables SNAT every Pod→Internet packet:

```
Pod IP → Node IP
```

Then AWS internet gateway NATs Node private IP → public IP.

B. Ingress (Internet → Pods)

Two methods:

---

1. Service Type = LoadBalancer (Layer 4)

- Cloud provider creates:
  - AWS ELB
  - GCP LB
  - Azure LB
- LB forwards traffic → NodePort → Pods

Good for simple TCP/UDP.

---

2. Kubernetes Ingress Controller (Layer 7)

Examples:

- NGINX
- AWS ALB Ingress

- Traefik
- Istio Gateway

Provides:

- Host-based routing
- Path-based routing
- SSL termination

Ingress → points to Services → Pods.

In Short – Kubernetes Networking Explained in One Page

Layer	What’s happening?
Container-to-Container	Share same network namespace → localhost
Pod-to-Pod (same node)	veth pair → bridge → veth
Pod-to-Pod (cross node)	Routed through network based on Pod CIDR
Pod-to-Service	iptables/IPVS DNAT
DNS	CoreDNS resolves service names
Egress	SNAT Pod → Node → Internet
Ingress	LoadBalancer or Ingress Controller