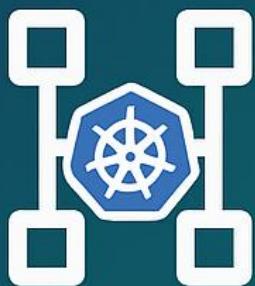




Master Kubernetes Networking

From Basics to Advanced
eBPF & Enterprise SDN



42 slides of pure
networking mastery

CNI Plugins • Service Types • OVN-K8s
eBPF • Production Tips

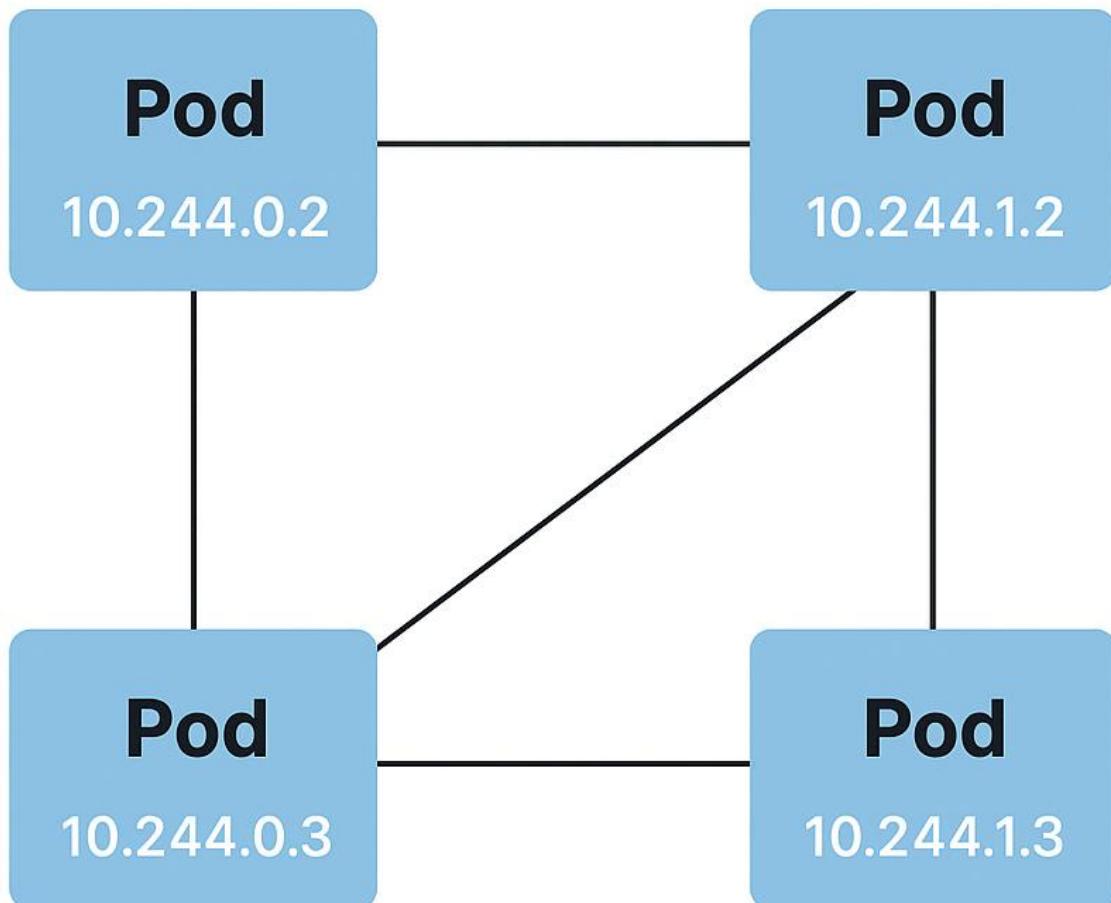
Swipe for complete networking guide →

What's Inside This Guide:

- Kubernetes networking fundamentals
- CNI plugins deep dive
- eBPF implementation secrets
- Real-world troubleshooting
- Production best practices
- Advanced load balancing

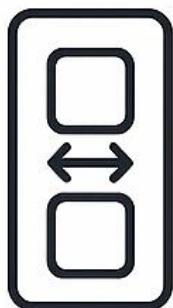
The K8s Networking Foundation

- Flat address space for all pods
- Each pod gets unique IP address
- No NAT required between pods
- Containers share pod IP via localhost

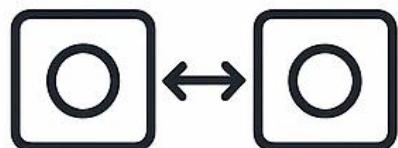


THE FOUR PILLARS OF K8s NETWORKING

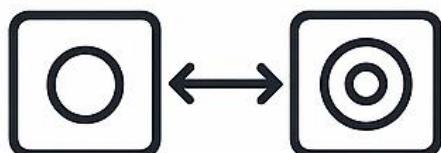
4 Core Communication Types:



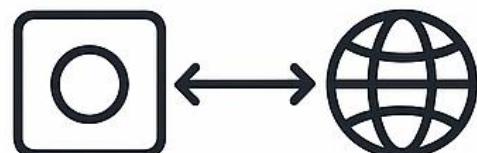
Intra-pod
(container \leftrightarrow container)



**Pod-to-pod
communication**



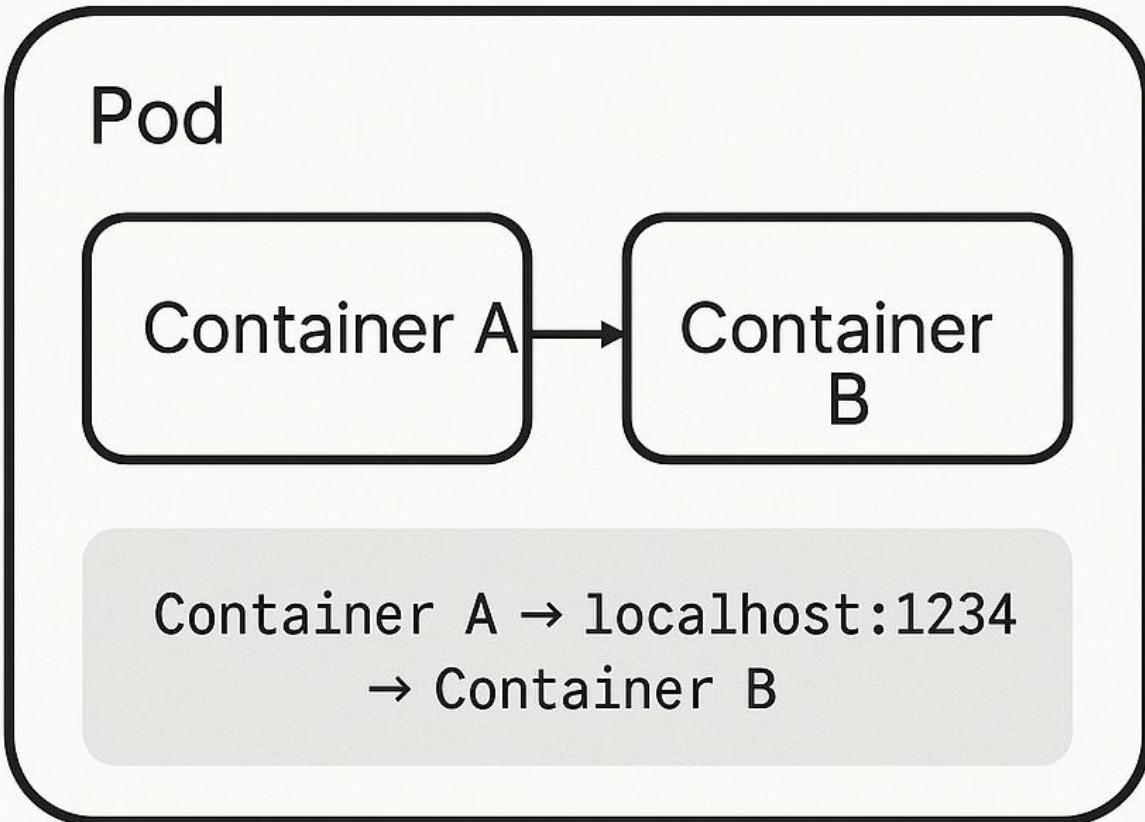
**Pod-to-service
communication**



**External access
patterns**

Master these = Master K8s networking

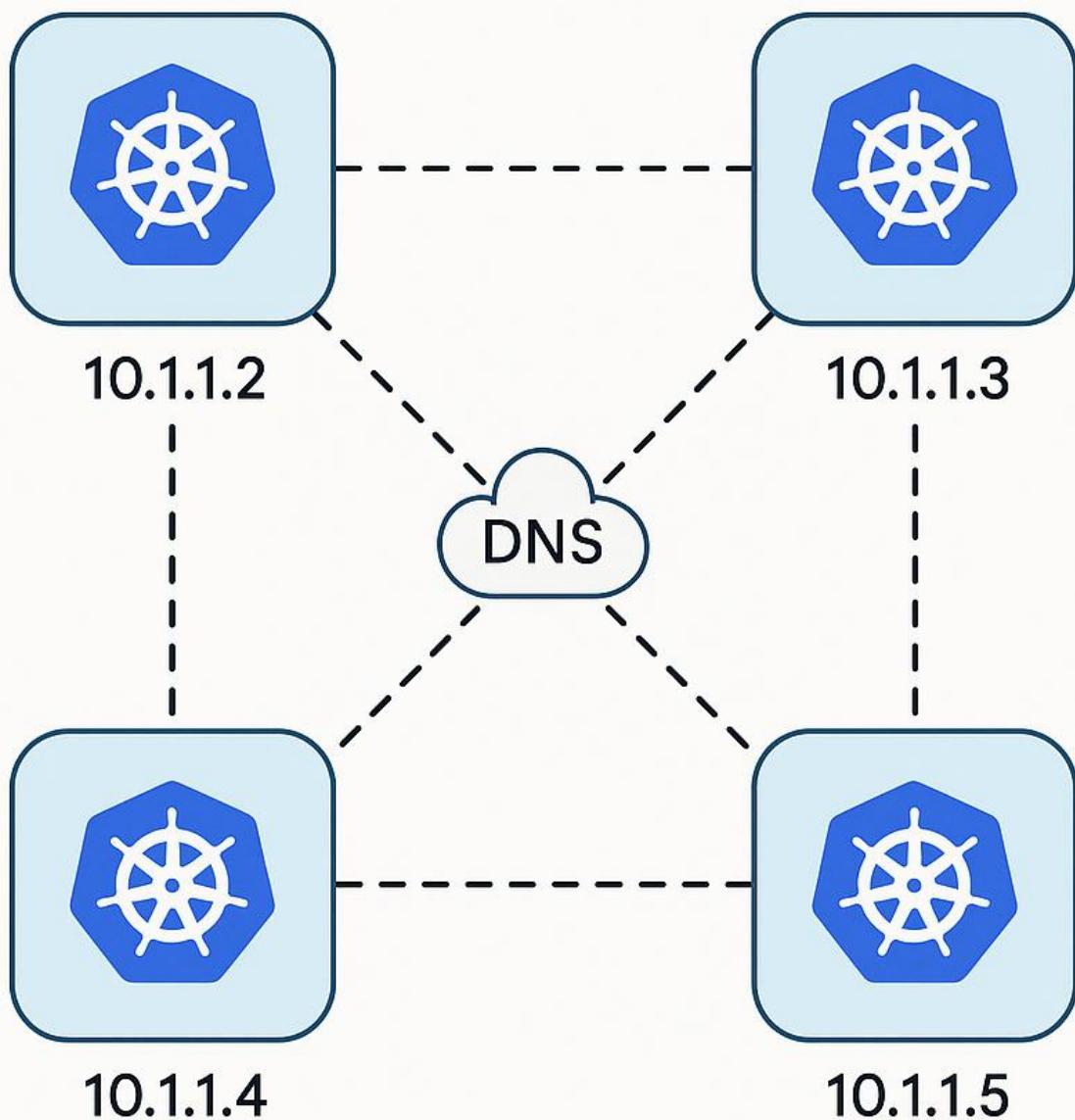
Inside the Pod Network



Containers share
network namespace

Pro Tip:
No port conflicts between pods!

Direct Pod Communication



- Network-visible IP addresses
- No NAT/tunnels/proxies needed
- Standard DNS works out of the box
- Well-known ports for easy config

Seamless app migration

Kubernetes Services Explained

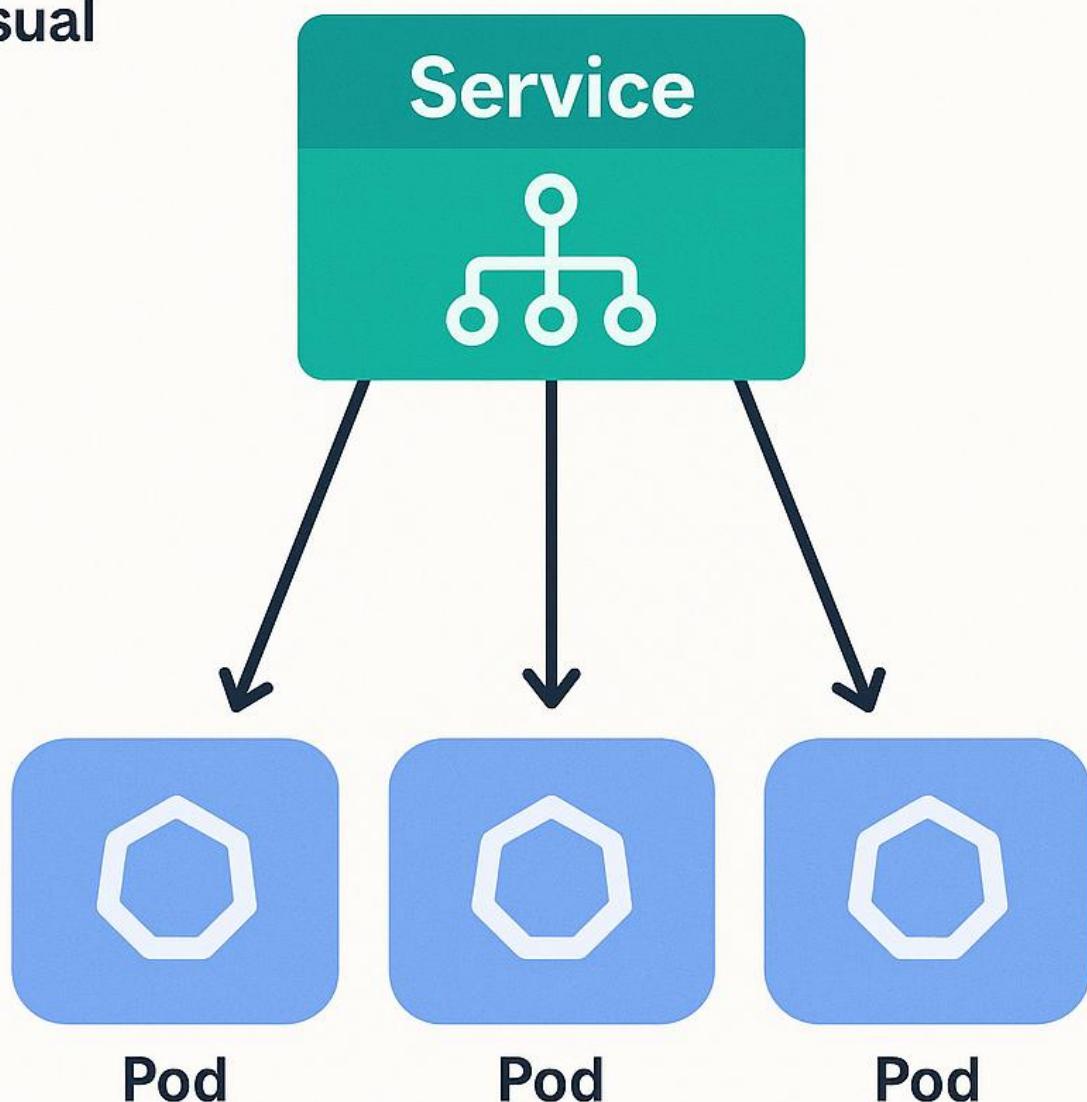
Problem: Pods are ephemeral, IPs change

Solution: Services provide stable endpoints

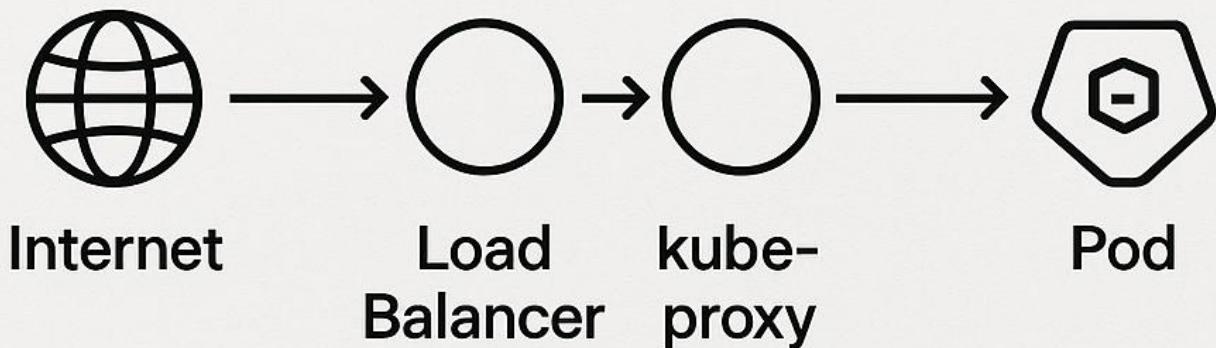
Benefits

- Automatic load balancing
- Service discovery
- Health checking

Visual



Reaching Your Apps

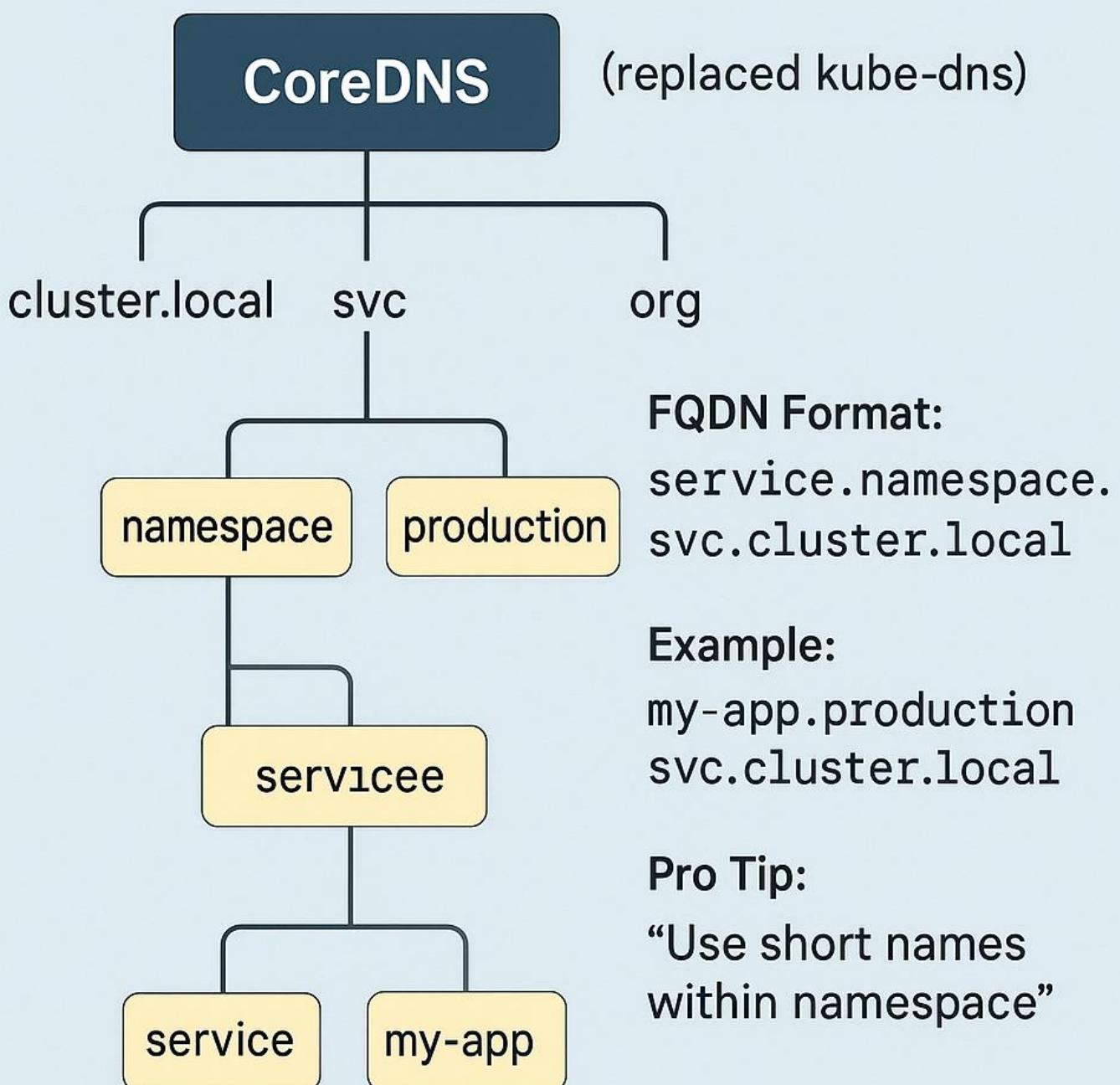


Challenge: Double-hop routing overhead

Solutions: NodePort,
LoadBalancer, Ingress

Performance tip: Consider service mesh for optimization

Name Resolution Made Simple



DNS Setup in Action

- Headless service creation

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  clusterIP: None
  ports: 80
  targetPort: 9376
```

- Pod hostname configuration

```
apiVersion: v1
kind: Pod
metadata: name-my-pod
spec:
  hostname: busybox
  containers:
    - name: busybox
      image: busybox
```

- FQDN resolution

```
$ host my-pod.my-service.default.svc.
cluster.local has address 10.1.0.120
```

Troubleshooting: Check /etc/resolv.conf in pods

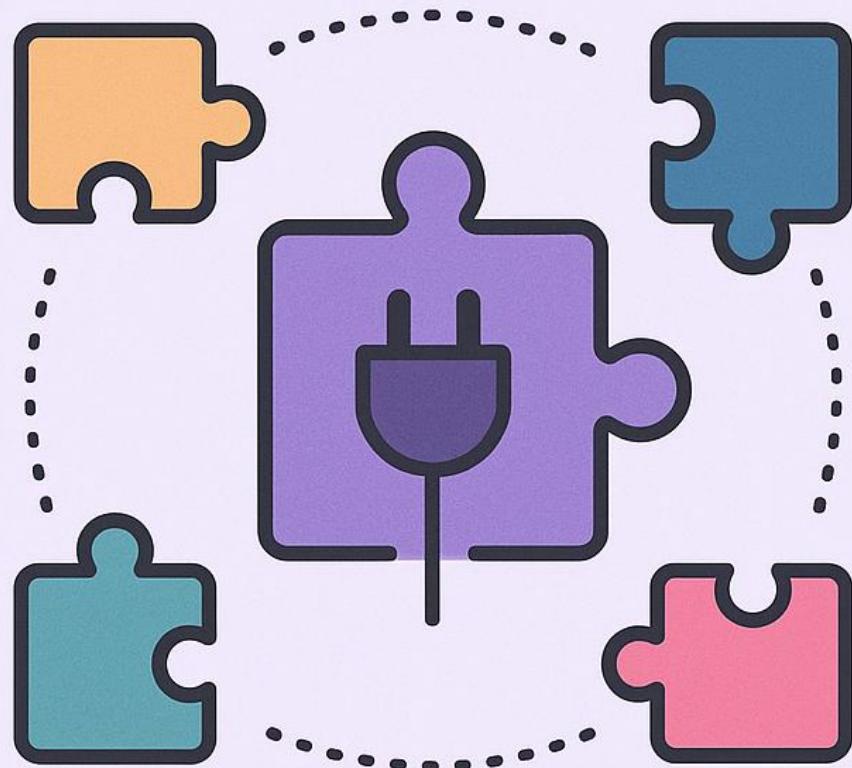
The Plugin Revolution

Why needed:

Diverse networking requirements

Two main types:

- Kubenet (basic, legacy)
- CNI (modern, extensible)



**Choose what fits
your needs**

Built on Linux Networking

- Network namespaces
- Virtual Ethernet (veth) pairs
- Bridges and routing
- IP addresses and ports

Understanding Linux =
Understanding K8s

veth: The Virtual Cable

Purpose

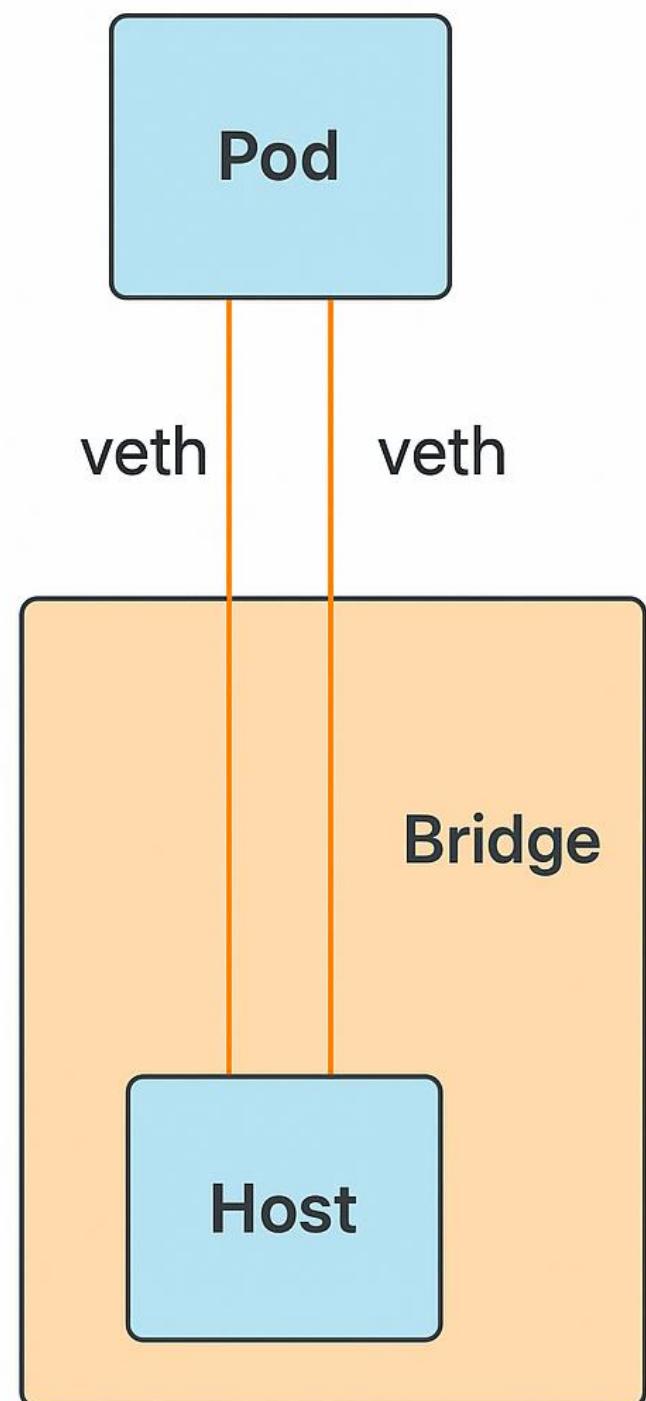
Connect containers to host network

How it works

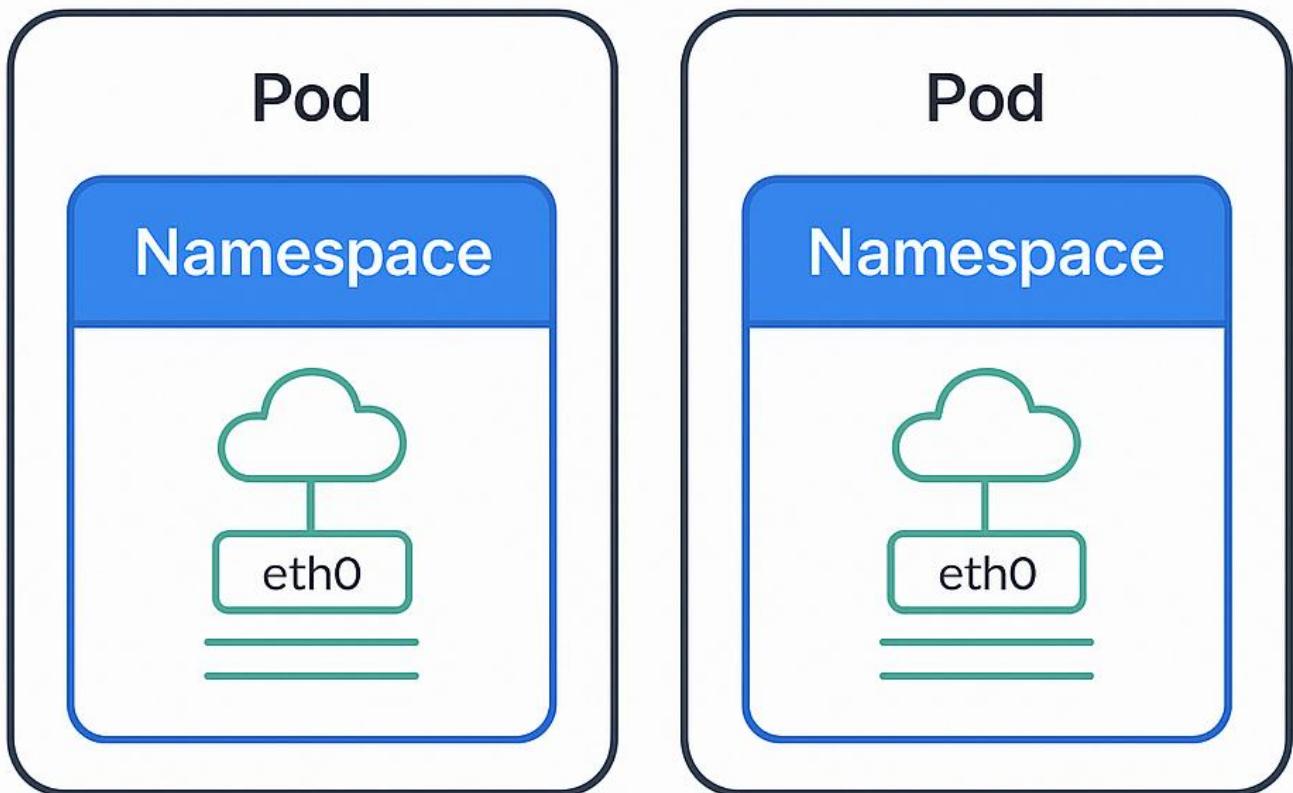
Paired virtual interfaces

Use case

Pod \leftrightarrow Host communication



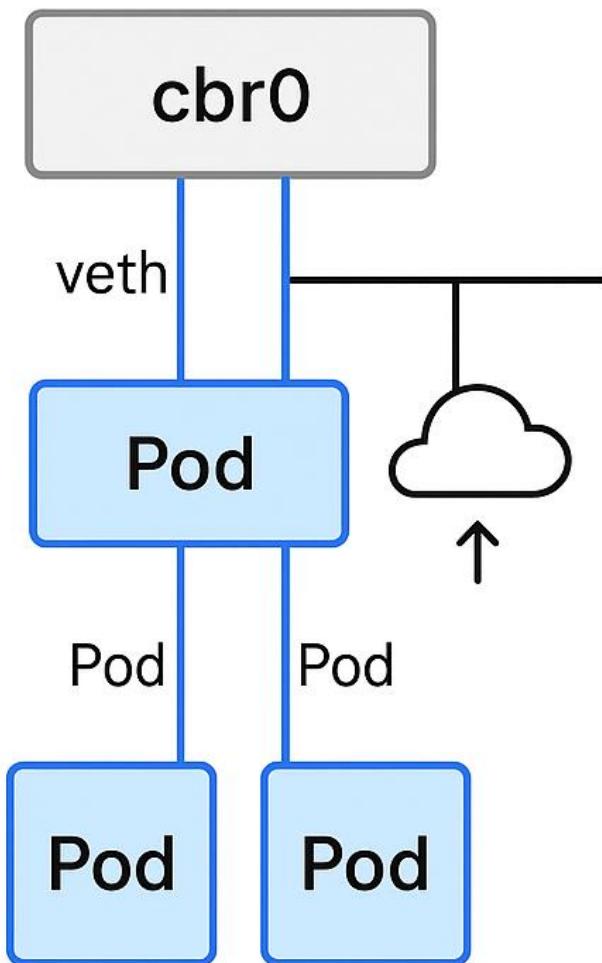
Isolation Through Namespaces



- Network isolation
- No IP conflicts
- Security boundaries

Each pod = own namespace

Kubenet: Keep It Simple



Creates cbr0 bridge
+ veth pairs

Use cases:

- Single-node environments
- Cloud provider routing
- Learning/development

Limitation:

“Basic functionality only”

CNI: Container Network Interface

Why it matters

Industry standard

Used by:

Kubernetes

OpenShift

Mesos

AWS EKS



Kubernetes OpenShift Mesos AWS EKS

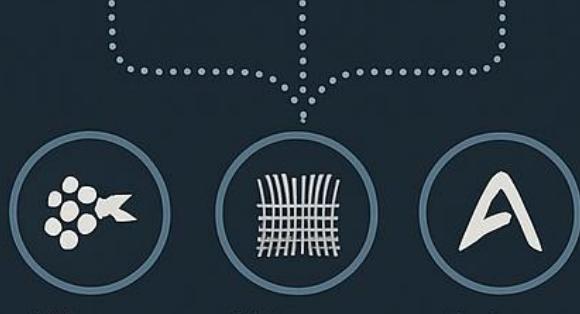
Benefits:

- Vendor flexibility
- Rich plugin ecosystem
- Standardized interface

CNI



Calico Cilium Flannel



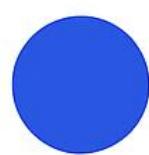
Weave Weave Antrea

CNI Plugin Ecosystem



Calico

Layer 3 virtual
networking



Flannel

Simple overlay
networking



Weave Net

Zero-config
networking

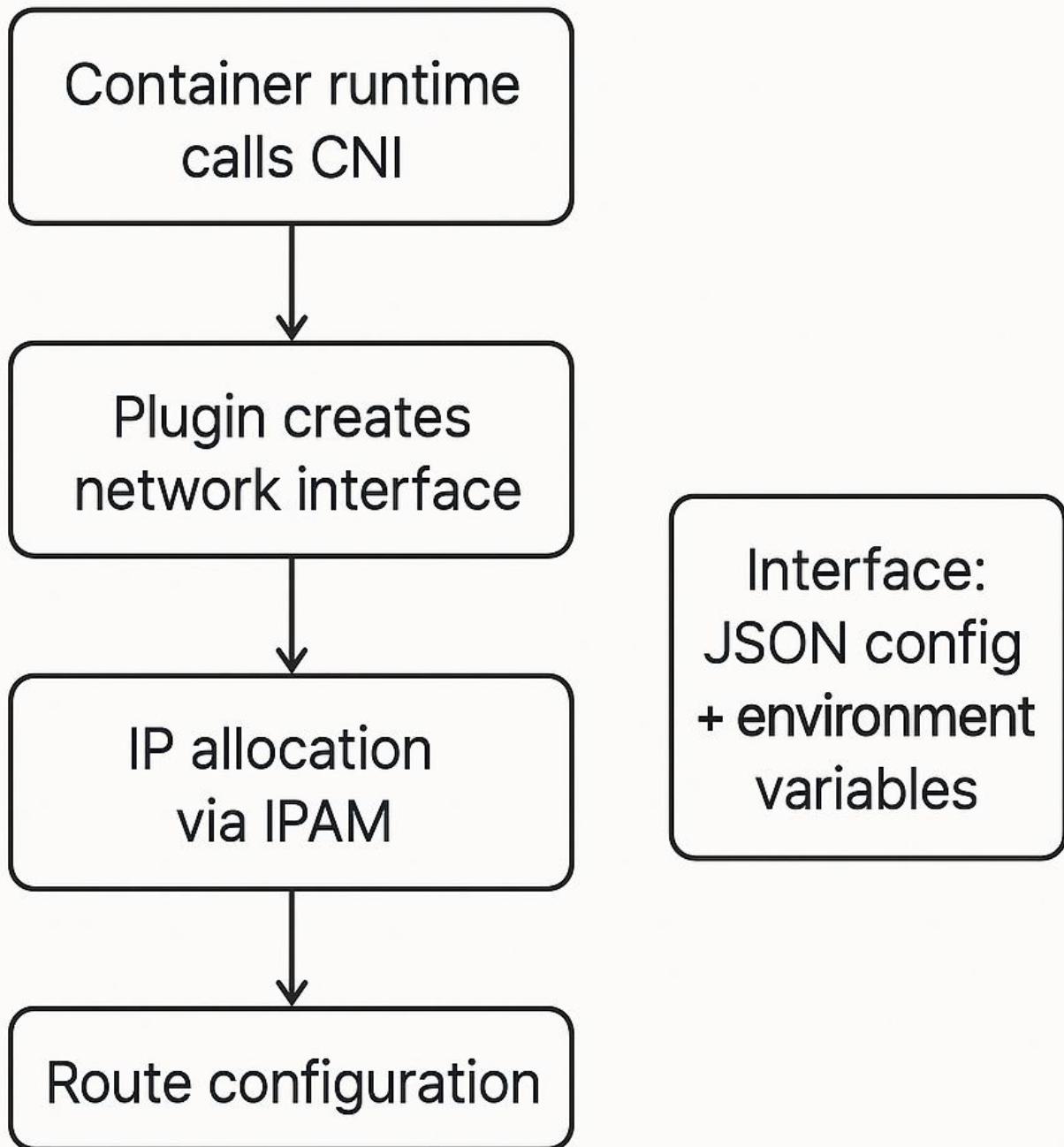


Cilium

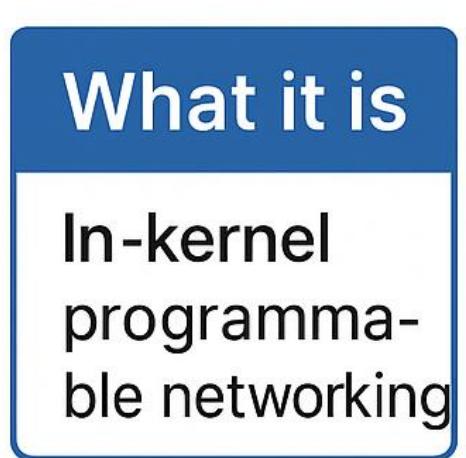
eBPF-powered
networking

Each with specific strengths and use cases

How CNI Plugins Work



eBPF: The Linux Superpower



Benefits:

- No kernel modules needed
- High performance
- Advanced observability

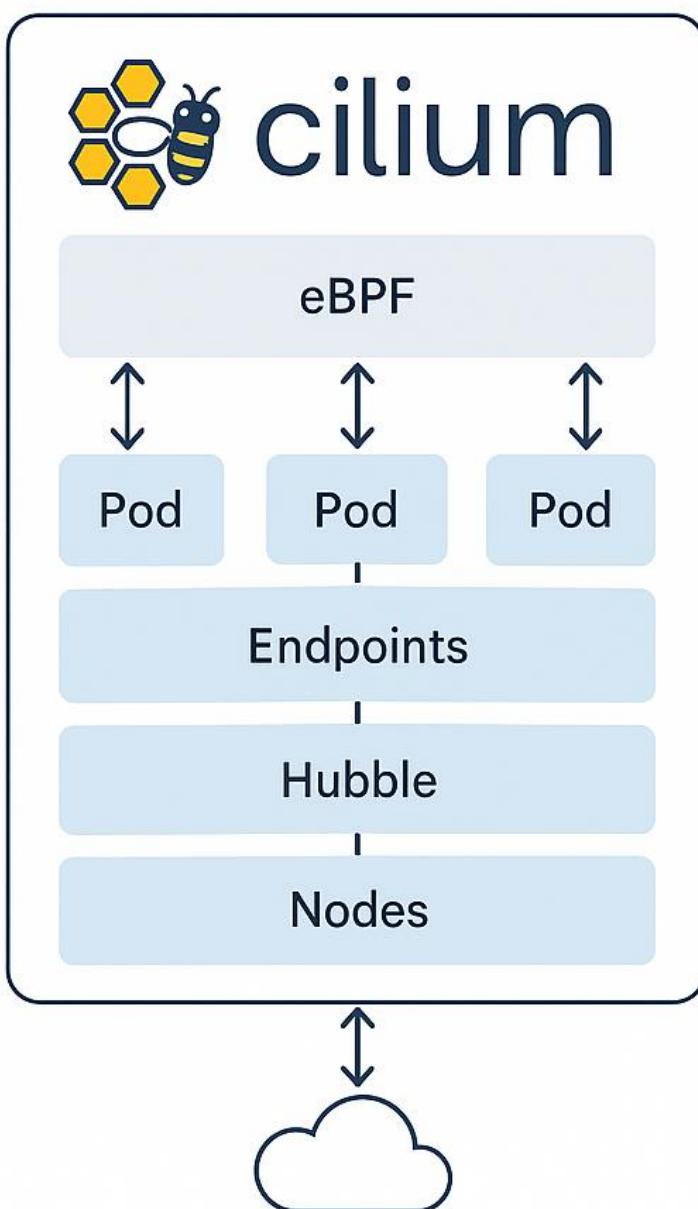
“The Linux superpower”
—Brendan Gregg

Why eBPF Matters

Traditional	eBPF
Networking	User-space → kernel transitions
Performance	In-kernel processing
Use cases	Load balancing, security, observability
Future	The new networking standard

Cilium: eBPF Networking Pioneer

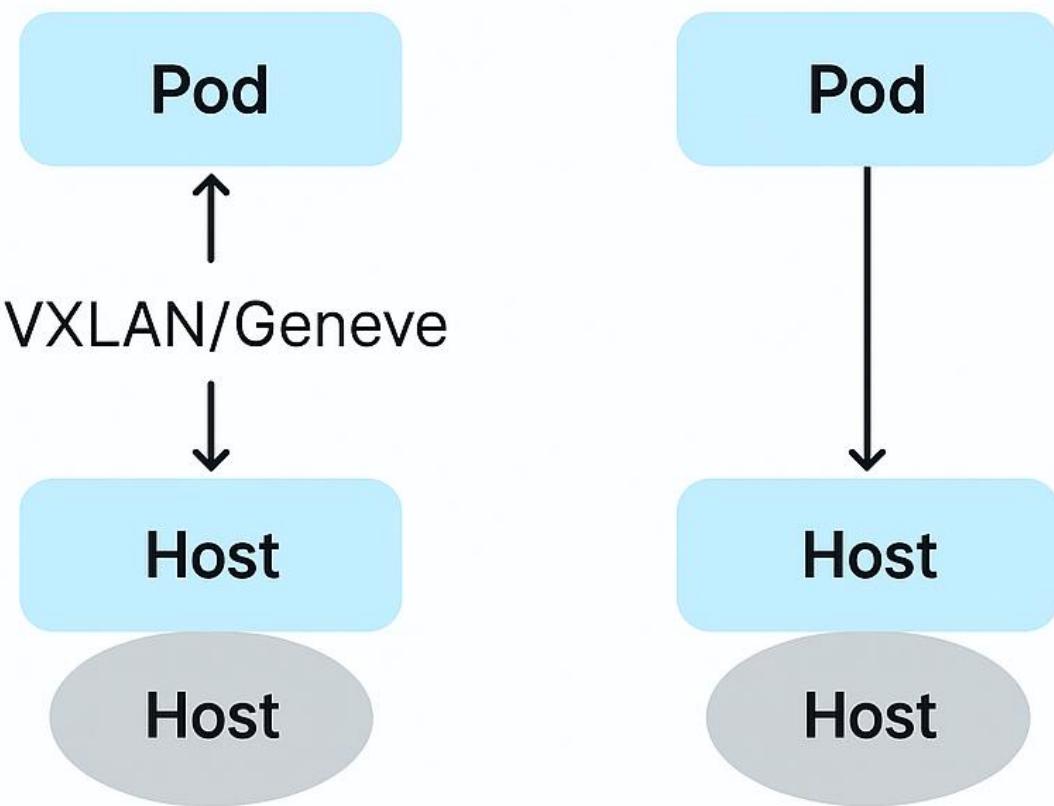
CNCF Project: Production-ready



- Identity-based security
- Load balancing without kube-proxy
- Advanced observability (Hubble)
- Multi-cluster networking

Cilium Flexibility

Overlay Mode	Native Routing
VXLAN/Geneve encapsulation	Direct host routing



- Choice depends on:
 - Infrastructure capabilities
- Performance: Native > Overlay
- Complexity: Overlay < Native

Load Balancing Reimagined

Traditional

kube-proxy

iptables

Cilium

eBPF
hashtables

- Better performance
- Lower CPU usage
- Advanced algorithms (Maglev)

Scale beyond limits

Zero-Trust Networking

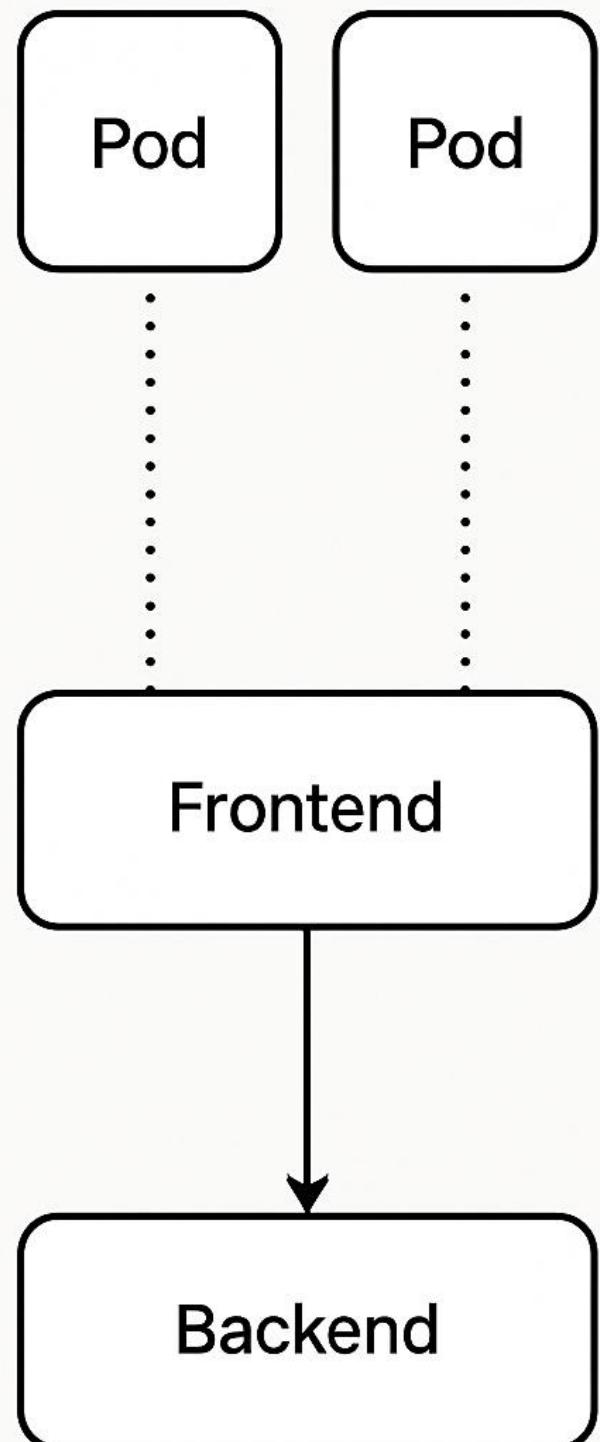
Default: All pods can communicate

Network Policies: Explicit allow rules

Implementation: Calico (reference)

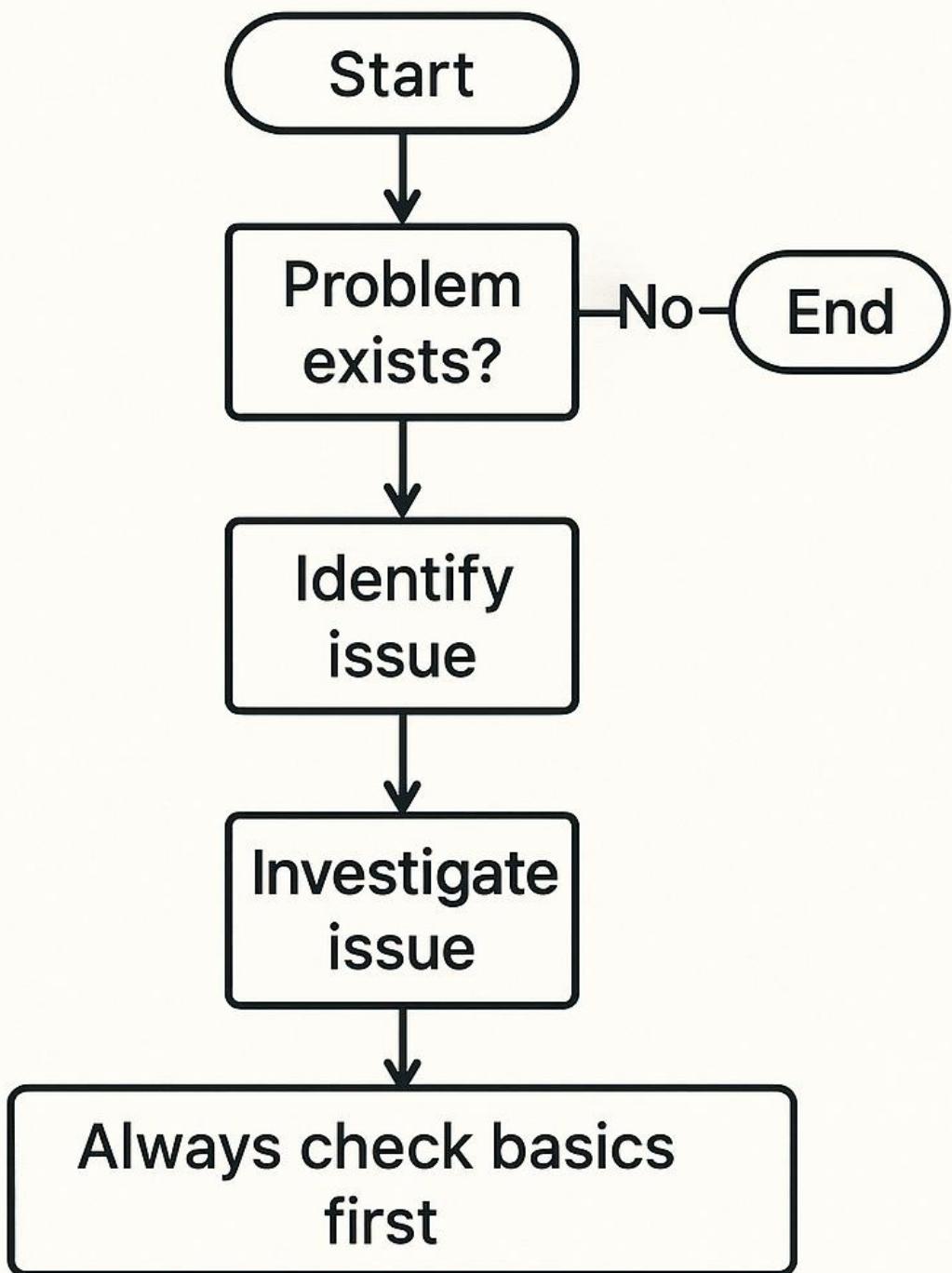
Best Practice: "Start with deny-all"

Example: Frontend → Backend only



Network Debugging Toolkit

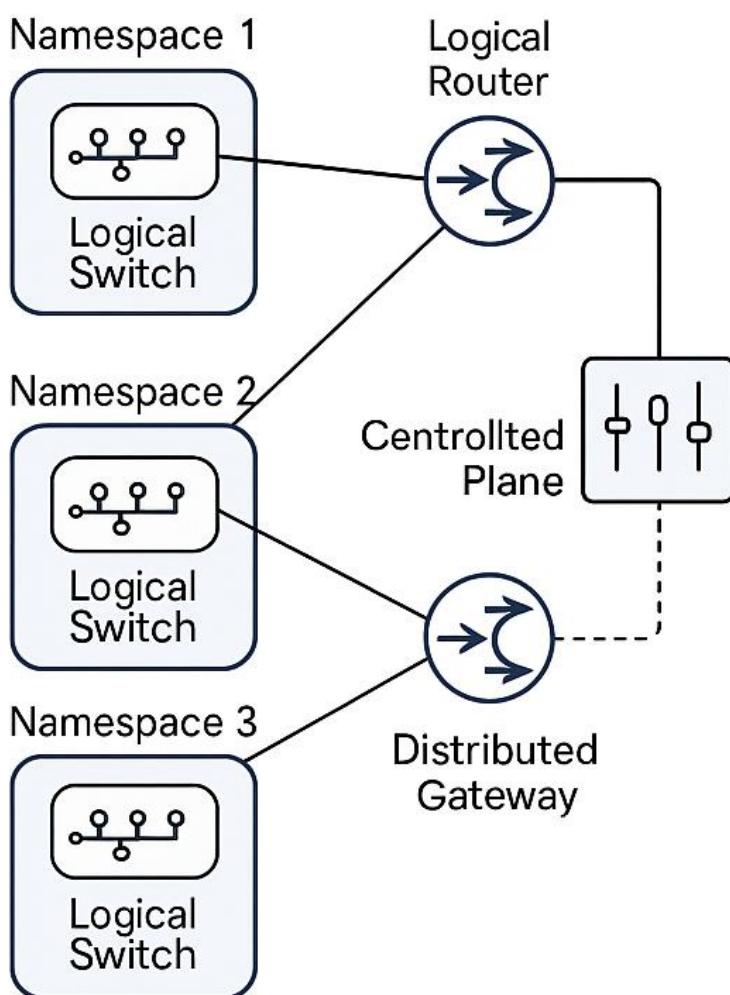
- kubectl exec for pod access
- dig/nslookup for DNS
- netstat/ss for connections
- tcpdump for packet capture



OVN-Kubernetes: Software-Defined Networking

What it is: CNI plugin based on Open Virtual Network (OVN)

Built on: Open vSwitch (OVS) data plane



Key Components

- Logical switches per namespace
- Logical routers for inter-namespace routing
- Distributed gateway for external access
- Network policies via CLs

Architecture Benefits

- True multi-tenancy
- Advanced networking features
- Centralized control plane
- Hardware offload support

Use Cases

Enterprise environments, multi-tenant clusters, complex networking requirements

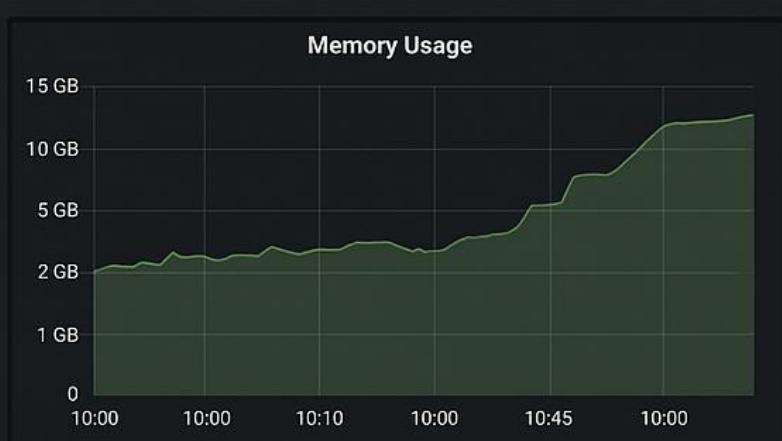
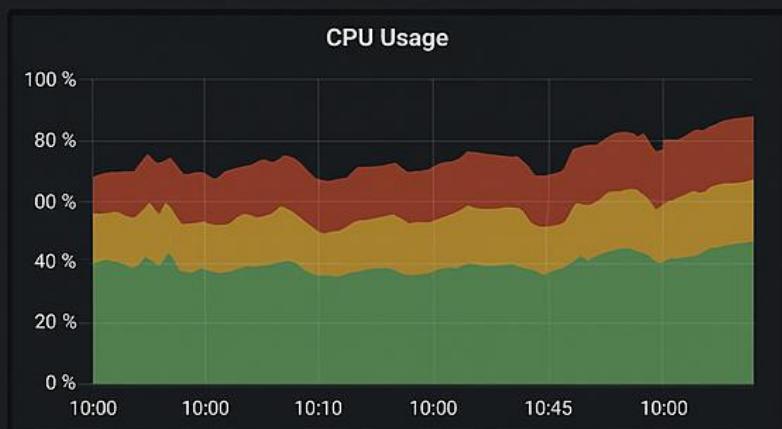
OVN-K8s: Advanced Networking Features

Enterprise Features	Fix These Common Issues	
<ul style="list-style-type: none">• Multiple external gateways• Network segmentation• Quality of Service (QoS)• IPv6 dual-stack support• Hardware acceleration• DPDK integration	DNS resolution failures	Check CoreDNS
<h3>Network Policies</h3> <ul style="list-style-type: none">• Default deny-all security• Granular traffic control• Stateful connection tracking	Pod communication issues	Verify CNI
	Service unreachable	Check endpoints
<h3>Performance</h3> <ul style="list-style-type: none">• Hardware offload capabilities• DPDK for high throughput• Smart NIC integration	Slow performance	Profile network path
	MTU problems	Check path MTU
Best for: Telecom, financial services, large enterprises with complex security requirements		

Optimize for Performance

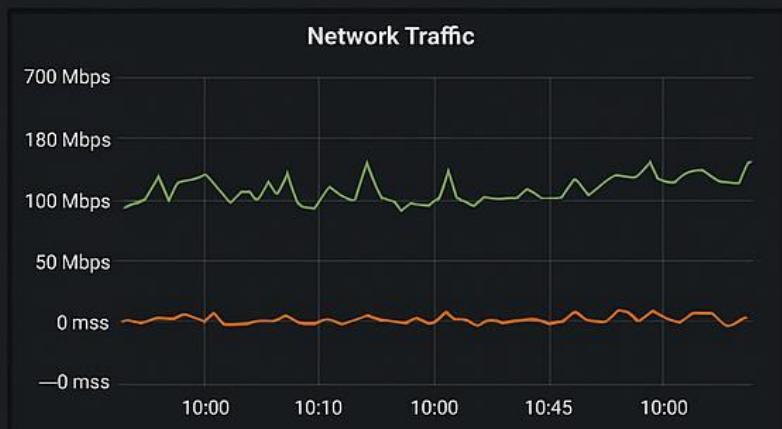
Strategies:

- Choose right CNI plugin
- Tune MTU settings
- Use native routing when possible
- Consider eBPF solutions



Monitoring:

Prometheus + Grafana



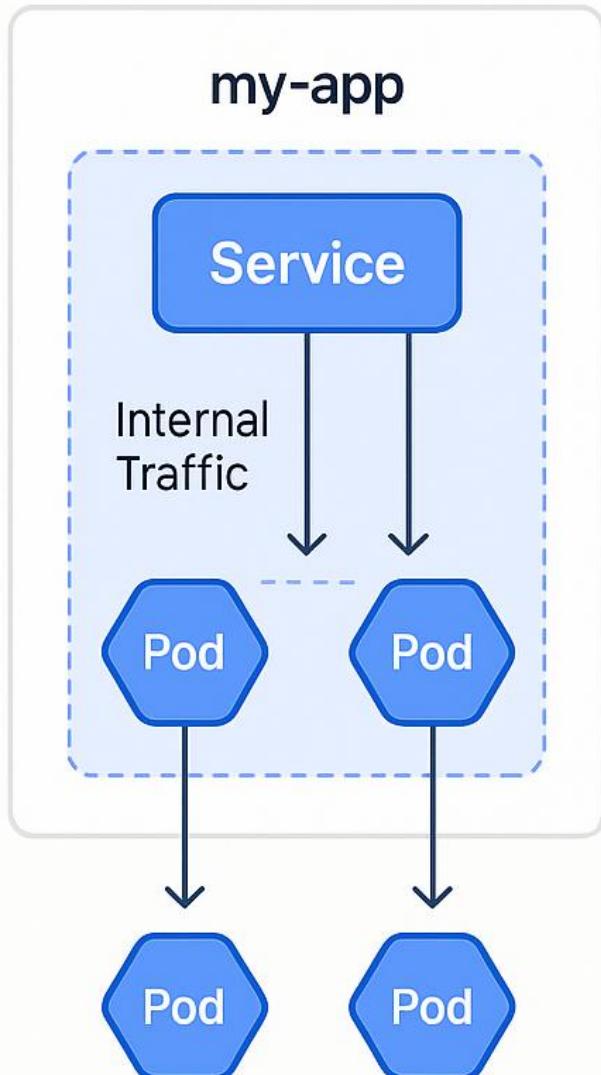
Secure Your Network

-  Implement network policies
-  Use encrypted communication
-  Segment sensitive workloads
-  Monitor network traffic
-  Regular security audits

Falco for runtime security

ClusterIP: Internal Communication Hub

Default service type for cluster-internal access



Key Features:

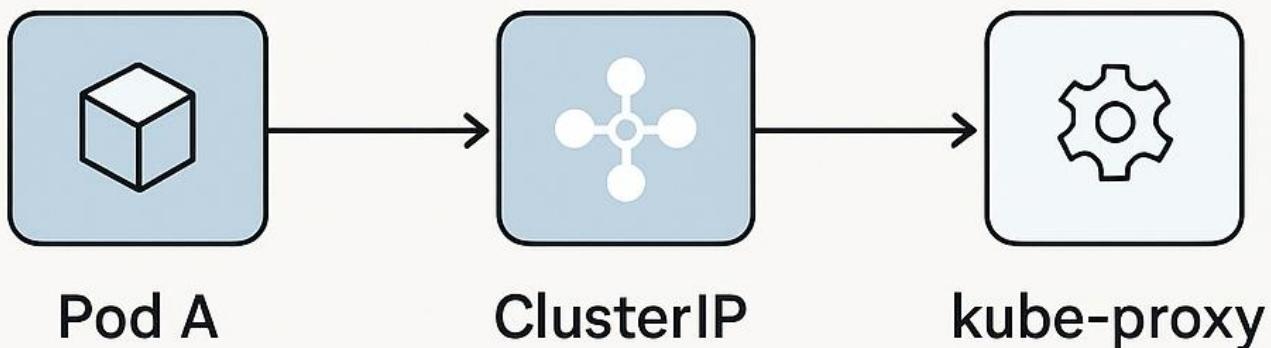
- Virtual IP accessible only within cluster
- Automatic load balancing to healthy pods
- DNS name: service-name.namespace svc.cluster.local
- Port mapping: Service port → Target port

YAML Example:

```
apiVersion: v1
kind: Service
metadata:
  name: my-app
spec:
  type: ClusterIP
  selector:
    app: my-app
  ports
  - port: 80
    targetPort:8080
```

Use Case: Backend services, databases, internal APIs

ClusterIP Traffic Flow



Behind the scenes

- kube-proxy creates iptables rules
- Service IP is virtual (not assigned to any interface)
- Load balancing algorithms: Round-robin, least connections

Troubleshooting

- Check endpoints: `kubectl get endpoints`
- Verify selector labels match
- Test from within cluster: `kubectl exec -it pod -- curl service-ip`

Pro Tip: Use service names, not IPs for better portability

NodePort: External Access Gateway

Exposes service on each node's IP at static port

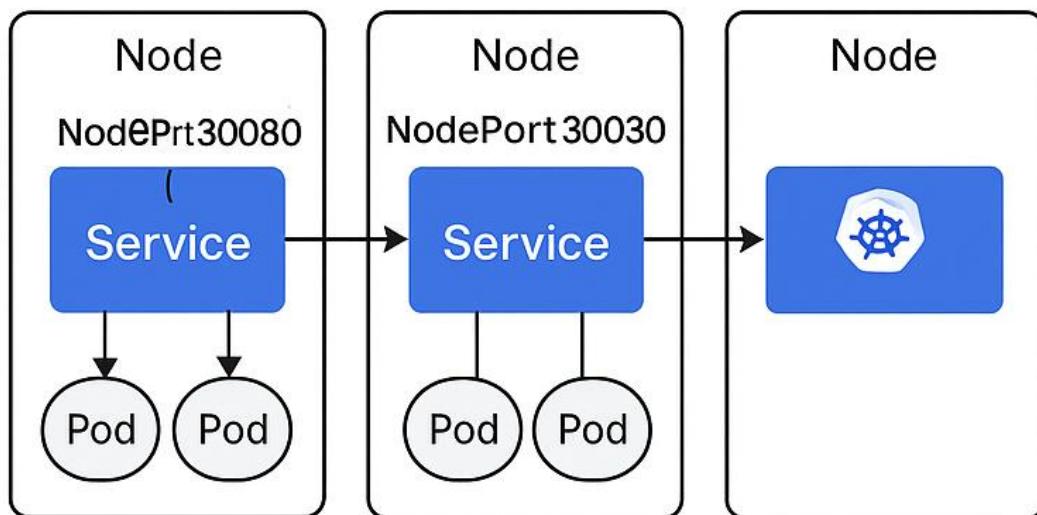
Port Range: 30000-32767 (configurable)

Access Pattern: NodeIP:NodePort → Service → Pocs

Key Features: External accessibility

High availability across nodes

Built on top of ClusterIP



- External accessibility
- High availability across nodes
- Built on top of ClusterIP

YAML Example:

```
apiVersion: v1
kind: Service
metadata:
  name: my-web-api
spec:
  type: NodePort
  selector:
    app: web
  ports:
    - port: 80
      targetPort: 80
      nodePort: 3001
```

NodePort Implementation Details

How it works:

- Creates ClusterIP service
- Allocates port on ALL nodes
- Routes traffic from any node to healthy pods

Traffic Flow



Advantages

- Simple external access
- No external load balancer needed

Limitations

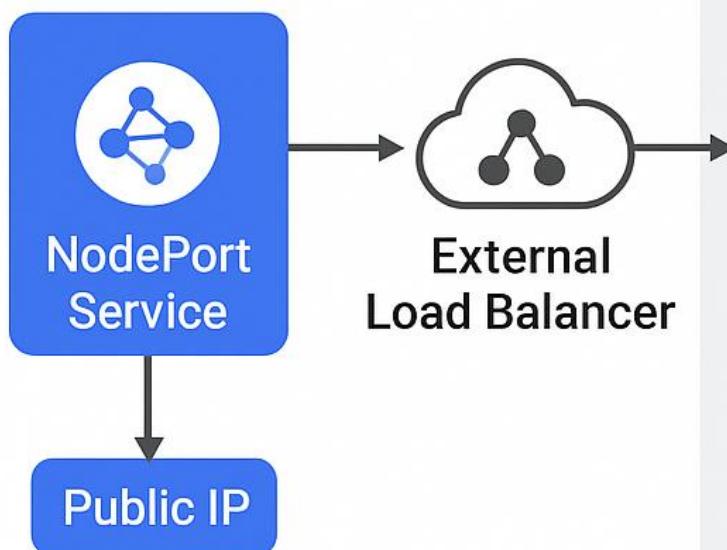
- Limited port range
- Manual port management
- No SSL termination

Best for:

Development, small deployments,
cost-conscious setups

LoadBalancer: Cloud-Native External Access

- Cloud provider's load balancer integration
- Creates: NodePort service (autab)
- External load balancer
- Public IP assignment



YAML Example

```
apiVersion: v1
kind: Service
metadata:
  name: web-service
spec:
  type: LoadBalancer
  selector:
    app: web
  ports:
    port: 80
    targetPort: 8080
```

Cloud Support: AWS ELB/ALB,
GCP Load Balancer, Azure LoadBalancer

Benefits: Production-ready, scalable, managed

ExternalName: DNS Alias Service

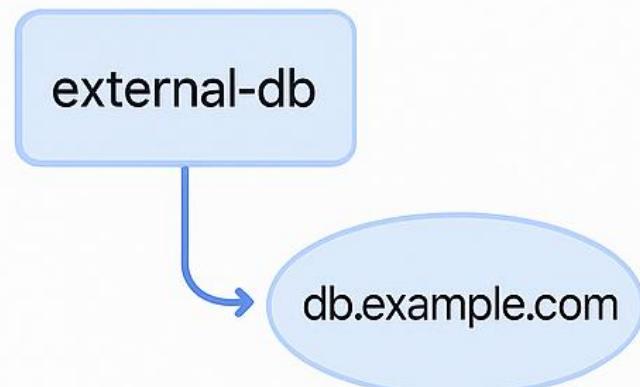
Purpose: Map service to external DNS name

No selectors or endpoints needed

Use cases:

- External database connections
- Third-party API integration
- Service migration scenarios

```
apiVersion: v1
kind: Service
metadata:
  name external-db
spec:
  type:
    ExternalName
  db.example.com
```



```
external-db.namespace.svc.cluster.local →
db.example.com
```

Pro Tip: Great for abstracting external dependencies

Headless Services: Direct Pod Access

Service with clusterIP: None

Behavior:

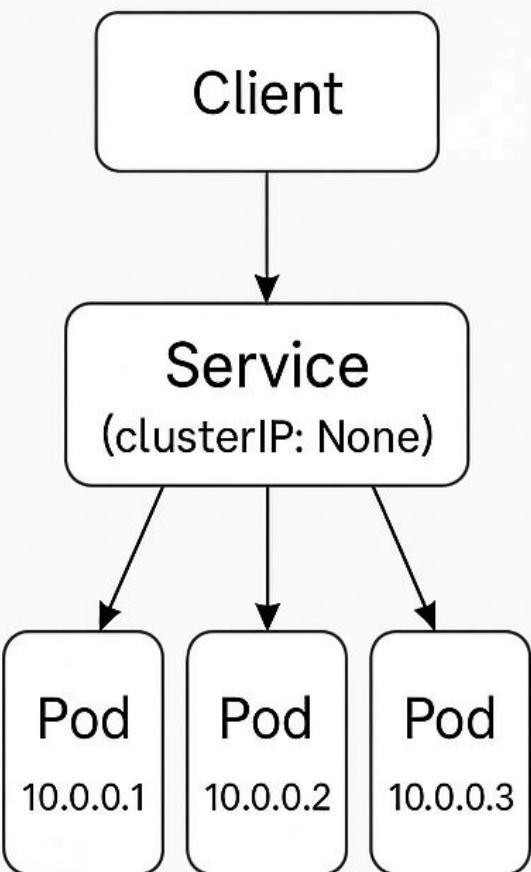
- No virtual IP assigned
- DNS returns pod IPs directly
- Client-side load balancing

Use Cases:

- StatefulSets with stable network identities
- Custom load balancing logic
- Service discovery scenarios

YAML Example:

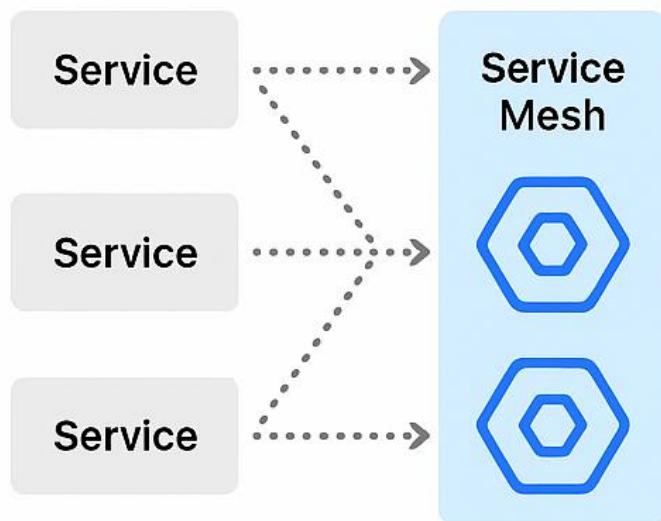
```
apiVersion: v1
kind: Service
metadata:
  name: headless-svc
spec:
  clusterIP: None
  selector:
    app: stateful-app
  ports
  - port: 80
```



DNS Result:

Multiple A records
for each pod

Services + Service Mesh = 🚀



What service mesh adds:

- Advanced traffic management
- Security (mTLS, policies)
- Observability and tracing
- Circuit breaking and retries

Popular solutions:

- Istio (most feature-rich)
- Linkerd (lightweight)
- Consul Connect

Benefits over basic services:

- Zero-trust security
- Traffic splitting
- Traffic splitting/canary deployments
- Distributed tracing

When to use: “Complex microservices architectures”

Optimize Service Performance

Key metrics to monitor

- Connection latency
- Throughput (request/sec)
- Error rates
- Pod health and readiness

Throughput



Connection Latency



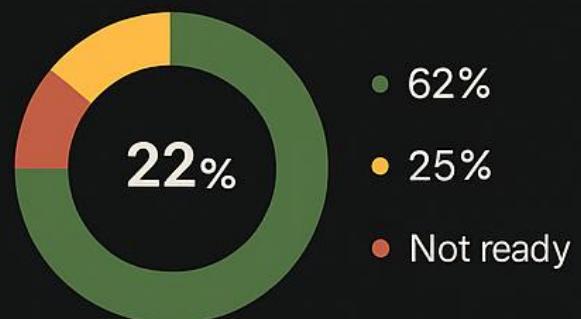
Optimization strategies

- Tune session affinity
- Optimize health checks
- Use appropriate service
- Consider connection po-

Tools

- Prometheus for metrics
- Jaeger for tracing
- Load testing with k6

Pod Health and Readiness



Goal: Sub-100ms service response times

Optimize Service Performance

Key metrics to monitor

- Connection latency
- Throughput (requests/sec)
- Error rates
- Pod health and readiness

Optimization strategies

- Tune session affinity
- Optimize health checks
- Use appropriate service type

Tools

- Prometheus for metrics
- Jaeger for tracing
- Load testing with k6

Connection latency

51 ms



Throughput

1,365 requests/sec

Error rates

1.0%

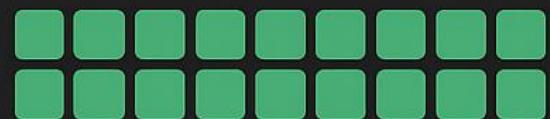
0.50%

0.0%

18:00 16:00 18:00 21:00 17:00

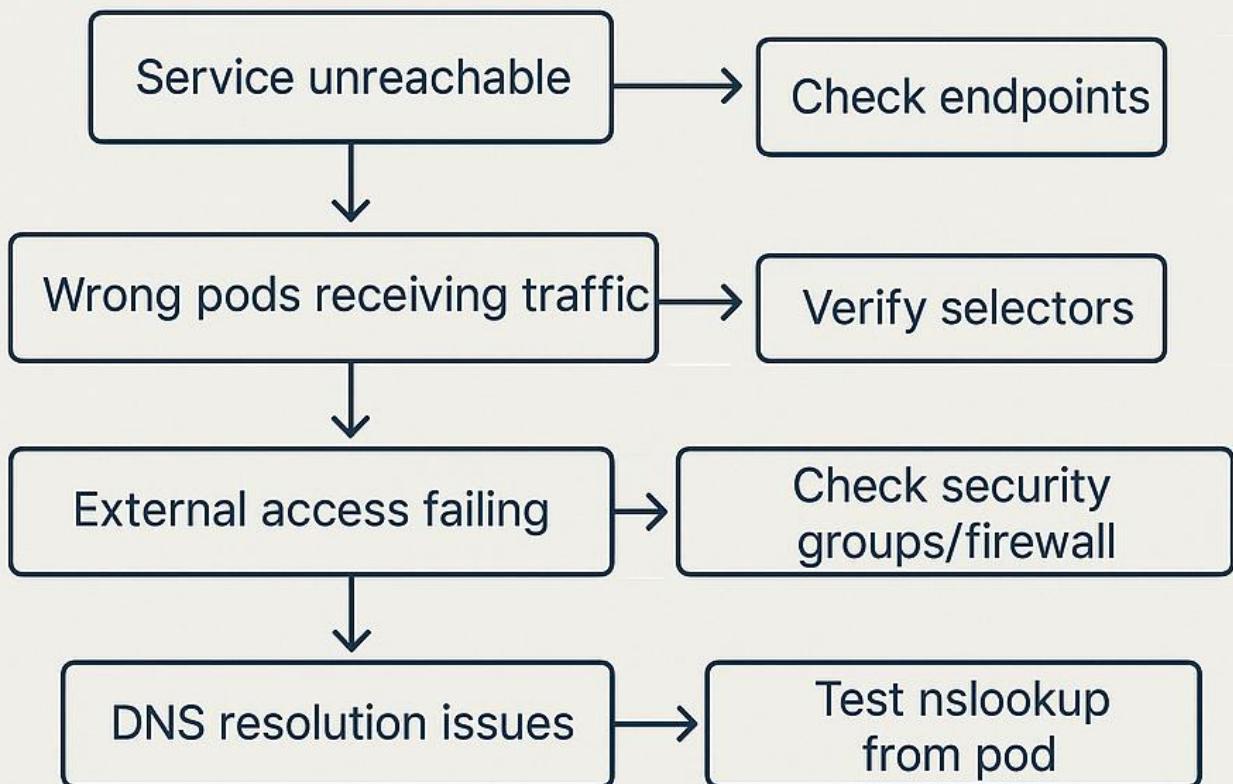
18:00 16:00 18:00 21:00 17:00

Pod health and readiness



Sub-100ms service response times

Debug Service Issues Like a Pro



Debug commands:

```
bash kubectl get svc, endpoints  
kubectl describe service my-service  
kubectl logs -l app=my-app  
nslookup my-service.default.svc.cluster
```

Pro tip:

Always verify pod labels match service selectors

Production Readiness

- CNI plugin selected and tested
- Network policies implemented
- Service types chosen appropriately
- Load balancing strategy defined
- Monitoring and alerting setup
- Disaster recovery tested
- Performance baselines established
- Security scanning automated
- Service mesh evaluated (if needed)

Ready to Go Deeper?



This was just the beginning of your K8s networking journey!



[Read my complete guide on Medium](https://medium.com/@salwan.mohamed/the-complete-kubernetes-networking-mastery-guide-from-basics-to-advanced-ebof-implementation-ebfddab65b2b)

<https://medium.com/@salwan.mohamed/the-complete-kubernetes-networking-mastery-guide-from-basics-to-advanced-ebof-implementation-ebfddab65b2b>*

What you'll get in the full guide:

- Step-by-step CNI plugin configurations
- Advanced eBPF implementation examples
- Production troubleshooting playbooks
- OVN-Kubernetes deployment guides
- Real-world case studies

Join 10,000+ engineers mastering K8s networking

👉 Like if this helped you

🔁 Repost to help your network

➕ Follow @salwan.mohamed for more K8s content

💬 Share your networking challenges below

Questions about your setup? Drop them in comments! ⤵

Save this post for your next K8s project 📒