Yasir Rehman

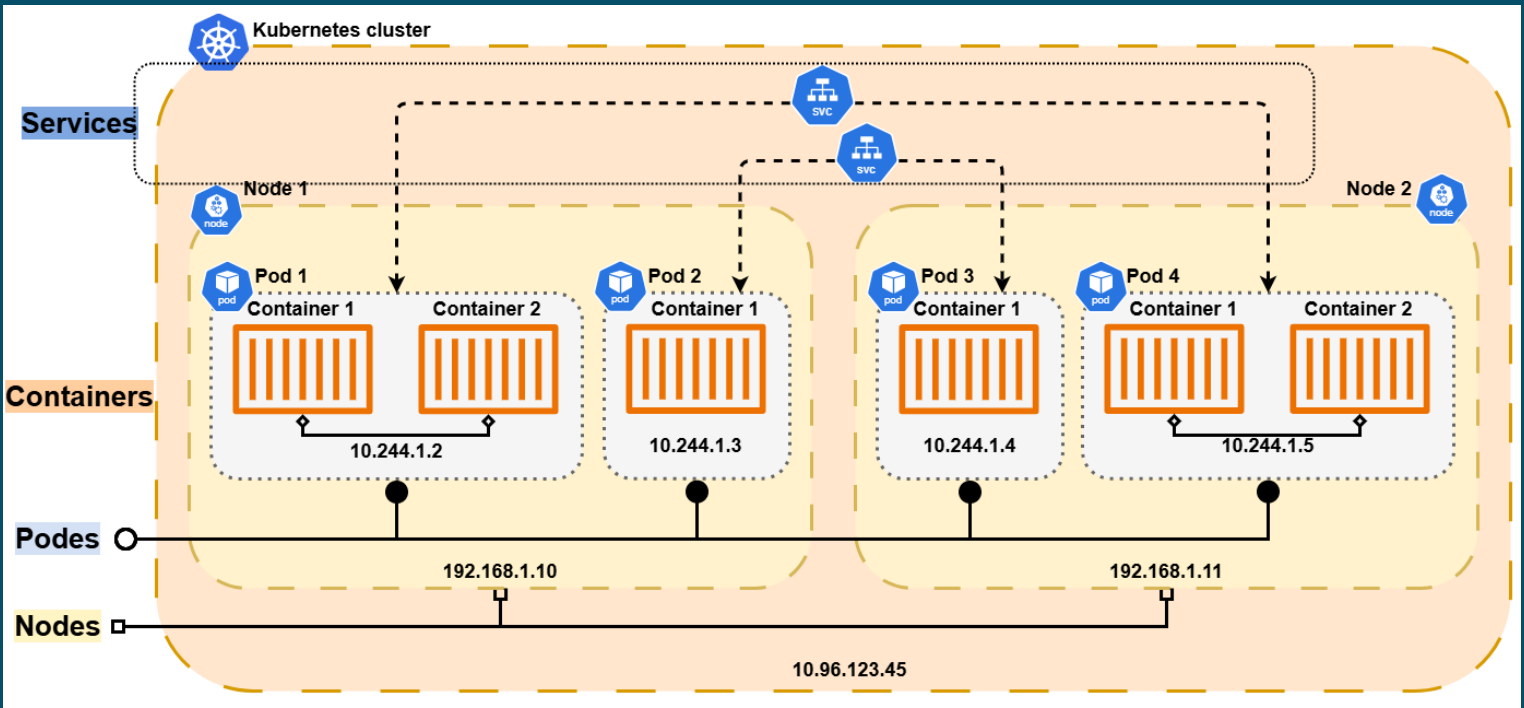
@ yasirrehman1

Exploring Kubernetes Networking

TECHNOLOGY: KUBERNETES

LEVEL: INTERMEDIATE

Kubernetes networking is built on a few core principles that ensure simplicity, scalability, and resilience.



Core principles:

- Each Pod is assigned a unique IP address from the CIDR range of its network, eliminating the need for NAT and simplifying communication and debugging.

- Pods can communicate with each other across nodes, ensuring seamless operation and facilitating scalability as the cluster expands.

- Services abstract Pods, providing a stable Cluster IP that decouples applications, enables load balancing, and handles dynamic environments.

- Kubernetes' built-in DNS service allows Services to be discovered using human-readable DNS names, simplifying communication and eliminating the need for hardcoded IPs.

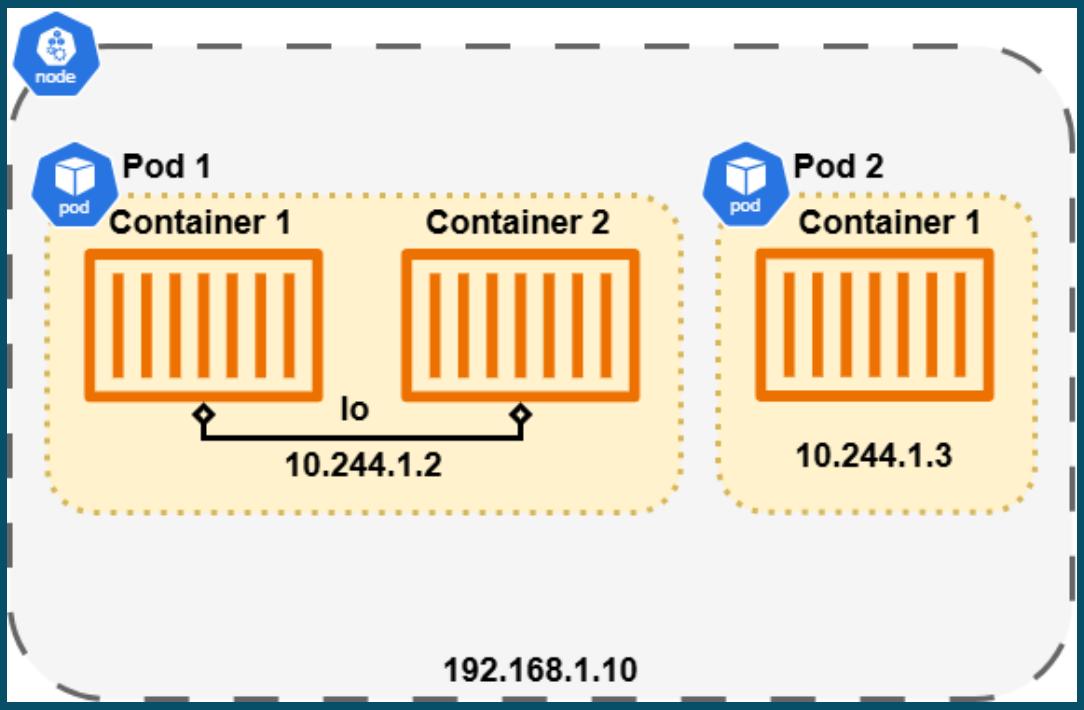
- Kubernetes offers several mechanisms—NodePort, LoadBalancer, and Ingress—to expose applications externally, making them accessible to users and systems outside the cluster.

- CNI plugins, such as Calico, Flannel, and Weave, manage Pod IP assignment, routing, and network policies, offering flexibility in choosing networking solutions tailored to specific needs.

Networking scopes

Container-level network

The container level network refers to the networking stack that is specific to an individual container.



Each container has its own network namespace, which includes:

- A loopback interface (lo) for internal communication.
- A virtual Ethernet interface (veth) that connects the container to the host's network stack.

How it works?

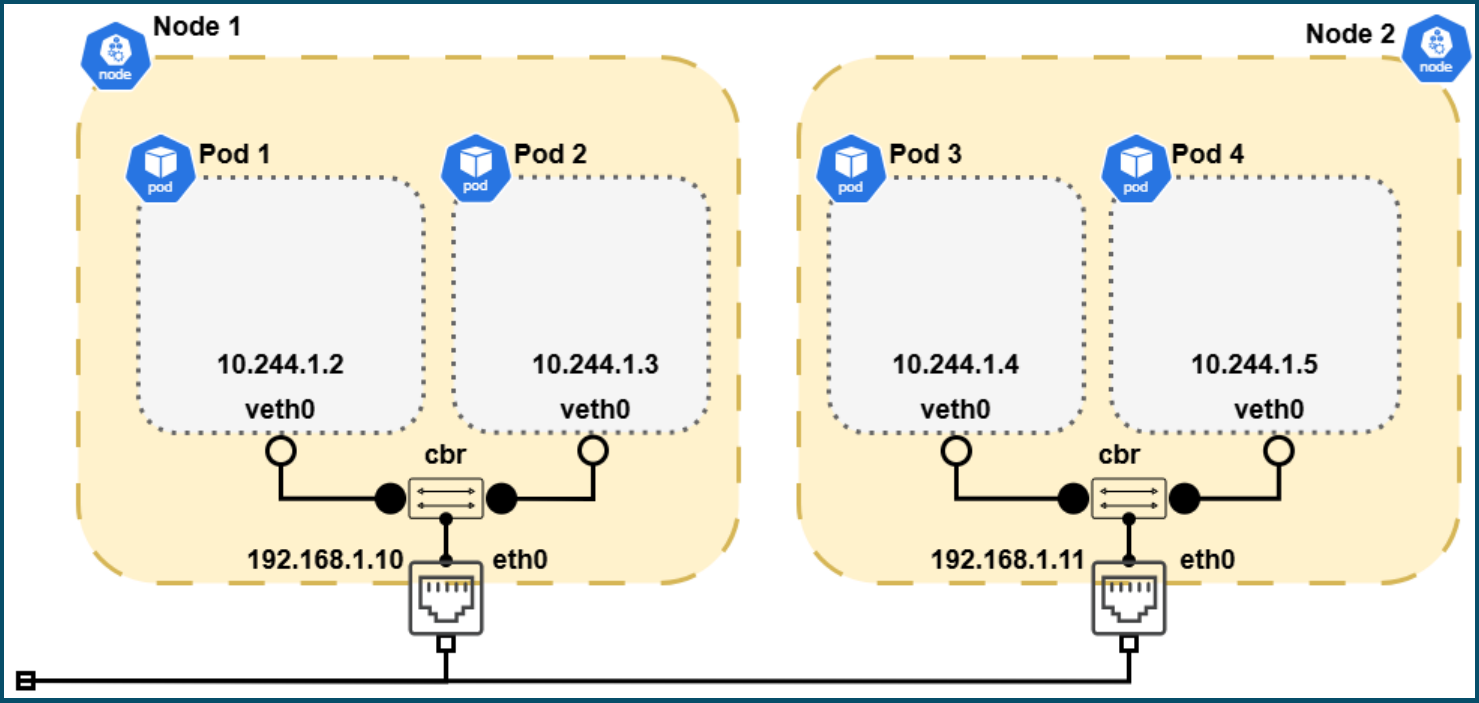
- When a container is created, it is assigned a virtual Ethernet interface (veth), which is paired with a corresponding interface on the host.
- The container’s network namespace is isolated from other containers, meaning it has its own IP address, routing table, and firewall rules.

Key characteristics:

- Each container has its own network namespace.
- Containers within the same Pod share the same network namespace.

Pod-level network

The Pod network is the network scope that enables communication between Pods in a Kubernetes cluster.



It is a flat network where:

- Every Pod gets a unique IP address.
- Pods can communicate with each other directly, regardless of which node they are running on.

How it works?

- When a Pod is created, it is assigned an IP address from the Pod network's CIDR (Classless Inter-Domain Routing) range.
- The CNI (Container Network Interface) plugins such as Calico, Flannel, and Weave are responsible for managing IP assignment, routing, and network policies for Pods.

Key characteristics:

- All Pods can communicate with each other without NAT.
- Pods on different nodes can communicate seamlessly.
- Each Pod has a unique IP address that is routable within the cluster.

Node-level network

The node network refers to the physical or virtual network that connects the nodes (machines) in a Kubernetes cluster.

This network is responsible for:

- Communication between the control plane (e.g., API server, etcd) and worker nodes.
- Communication between worker nodes for Pod-to-Pod traffic across nodes.

How it works?

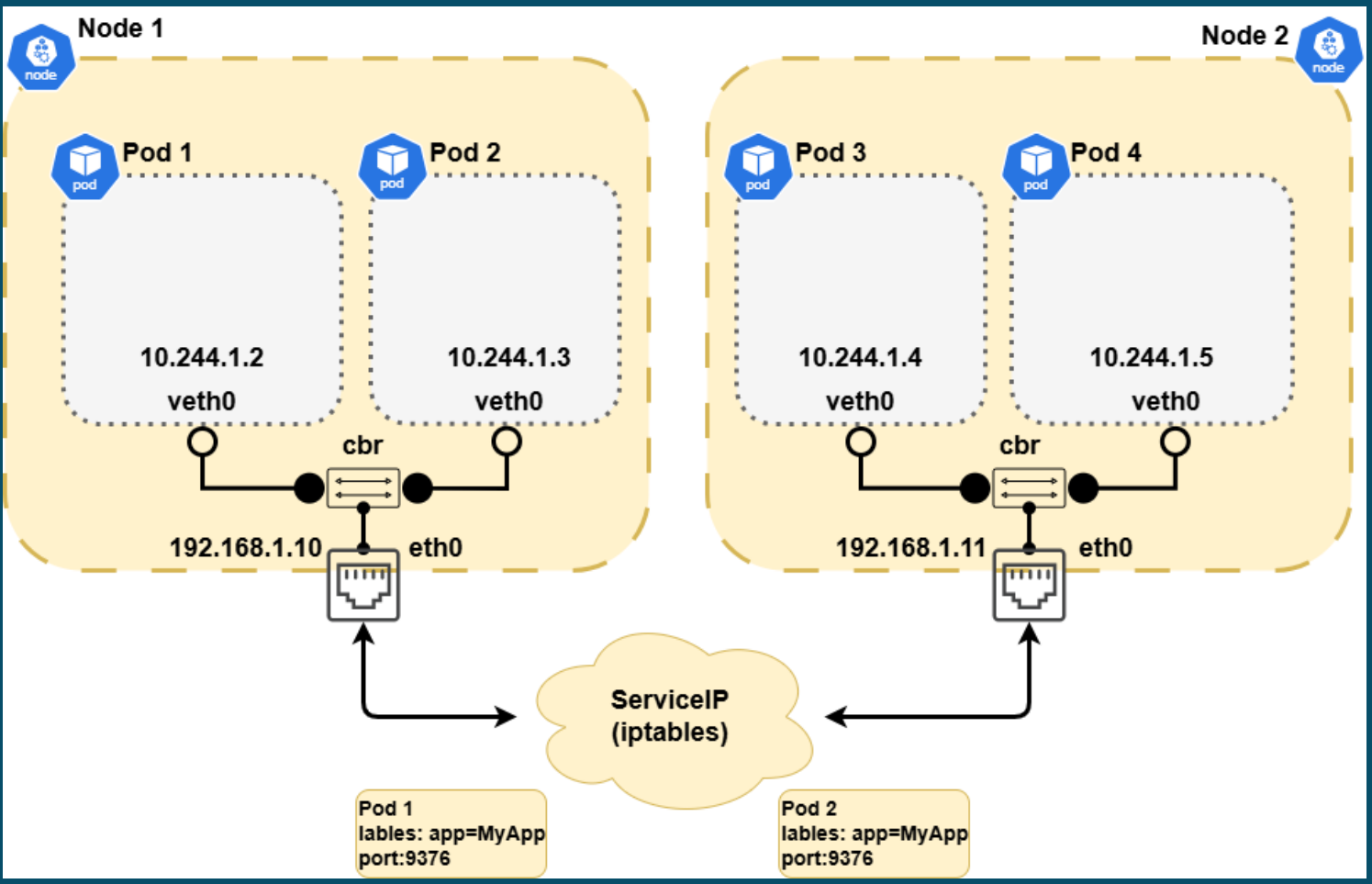
- Each node has a physical or virtual network interface (e.g., eth0) with an IP address assigned from the node network's CIDR range.
- The CNI plugin ensures that traffic between Pods on different nodes is routed correctly through the node network.

Key characteristics:

- The node network is typically provided by the underlying infrastructure (e.g., cloud provider, on-premises network).
- Each node has a unique IP address that is used for communication between nodes.

Service-level network

The service network is the layer in Kubernetes that provides stable, abstracted endpoints for accessing Pods.



Services act as an intermediary between Pods and internal/external systems, providing load balancing, service discovery, and external access.

How it works?

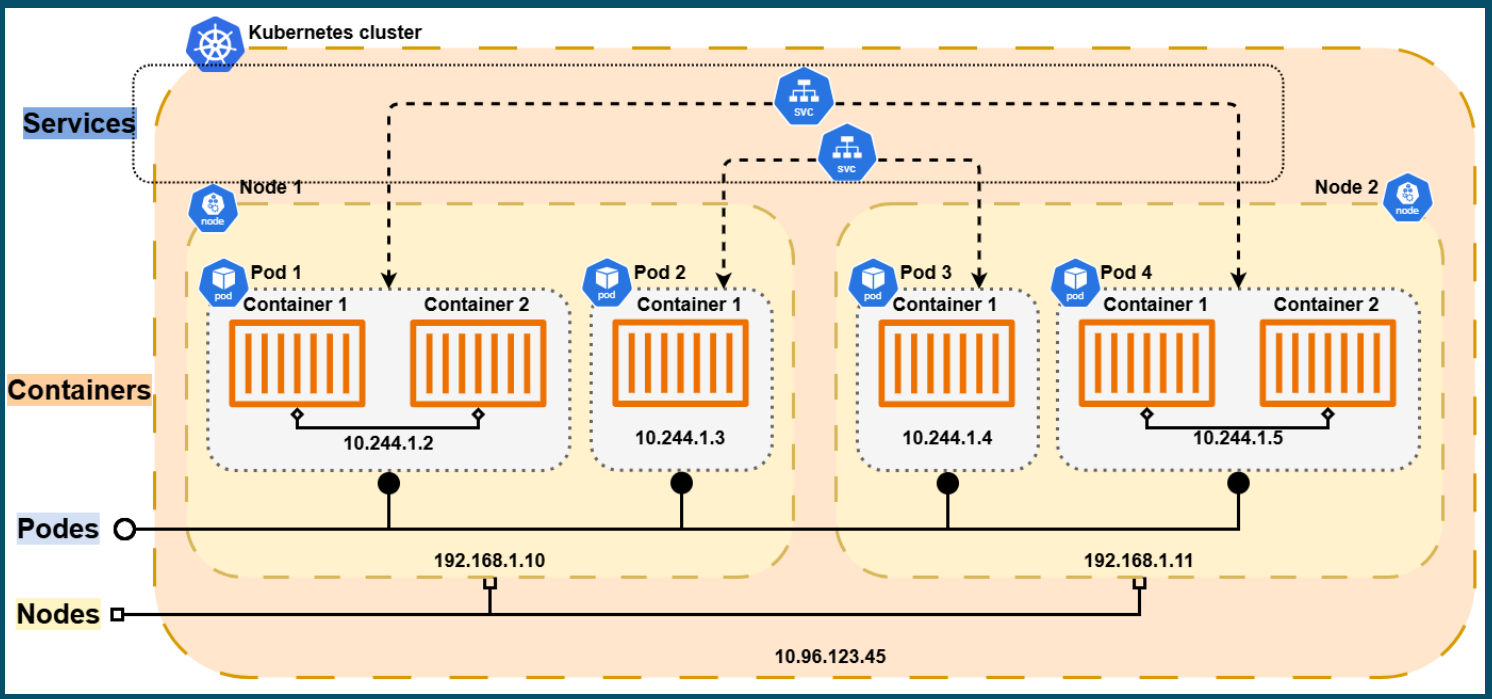
- A Service is created with a selector that matches the labels of the Pods it should route traffic to.
- The Service is assigned a Cluster IP, which is a virtual IP address that remains stable.
- The Service dynamically updates its Endpoints based on the Pods that match its selector.

Key characteristics:

- Services are assigned DNS names in the format <service-name>.<namespace>.svc.cluster.local.
- kube-proxy uses iptables or IPVS to implement load balancing across the Pods.
- Traffic is distributed evenly across backend Pods, ensuring high availability and scalability.

Cluster-level network

The cluster network is the overarching network that encompasses all the networks within a Kubernetes cluster.



It includes:

- The pod network (for pod-to-pod communication).
- The node network (for node-to-node communication).
- The service network (for Service IPs).

How it works:

- The cluster network is managed by Kubernetes components like kube-proxy and the CNI plugin.
- kube-proxy ensures that Services are reachable, and that traffic is routed correctly to Pods.
- The CNI plugin manages IP assignment, routing, and network policies for Pods.

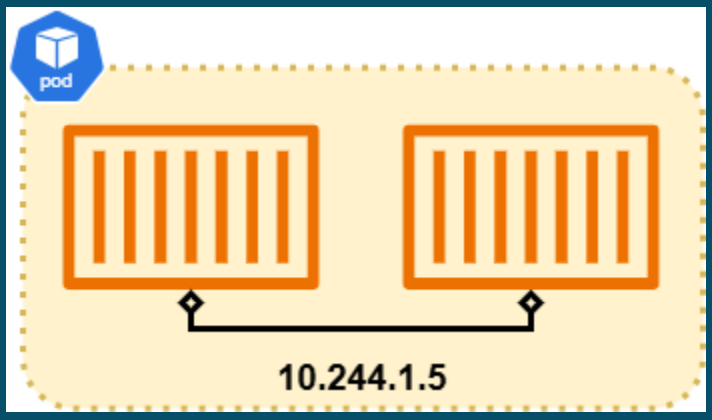
Key characteristics:

- Services are assigned IPs from a separate CIDR range (e.g., 10.96.0.0/12).
- Kubernetes provides DNS-based service discovery within the cluster network.
- Network policies can be applied to control traffic flow within the cluster network.

Communication scenarios

Intra-Pod (container-to-container)

A Pod contains multiple containers that need to communicate with each other.



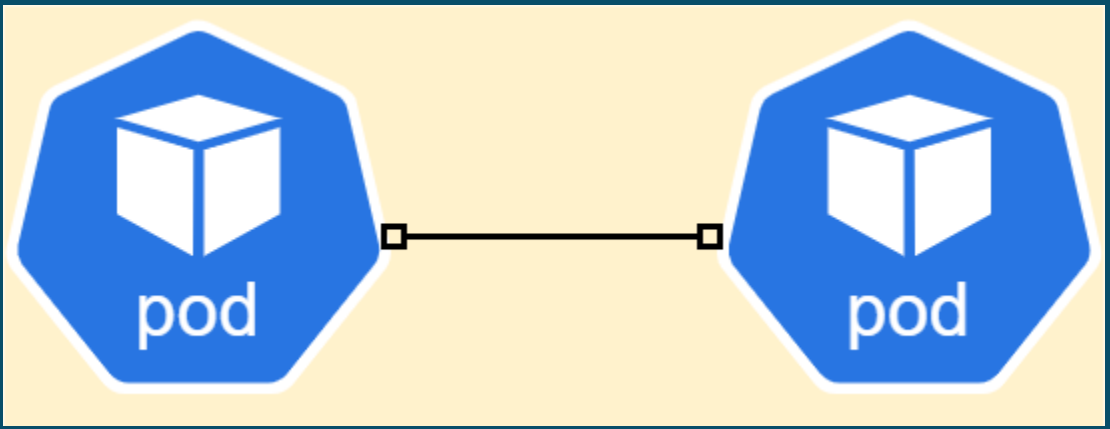
Containers within the same Pod share the same network namespace and can communicate using localhost(127.0.0.1).

No additional network configurations are needed for intra-pod communication.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: multi-container-pod
spec:
  containers:
  - name: app
    image: nginx
  - name: sidecar
    image: busybox
    command: ["sleep", "3600"]
```

Inter-Pod (pod-to-pod)

Pods on different nodes need to communicate with each other.



Pods are assigned unique IP addresses from the Pod network, and the CNI plugin ensures that traffic is routed correctly between nodes.

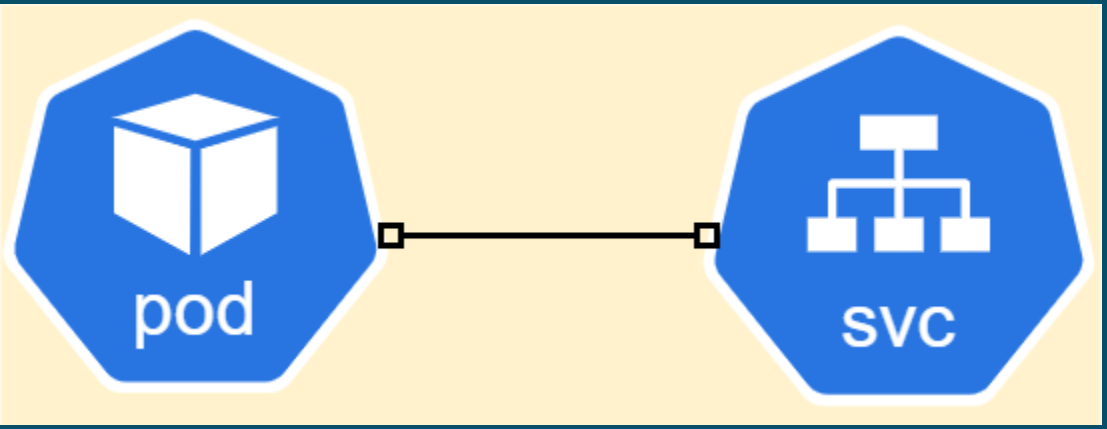
Communication happens without Network Address Translation (NAT).

Example: Direct pod communication

```
kubectl run pod1 --image=busybox --restart=Never --command -- sleep 3600
kubectl run pod2 --image=busybox --restart=Never --command -- sleep 3600
kubectl exec -it pod1 -- ping pod2-ip-address
```

Pod-to-Service

A Pod needs to communicate with a Service.



The Service provides a stable Cluster IP, and kube-proxy ensures that traffic is routed to one of the backend Pods.

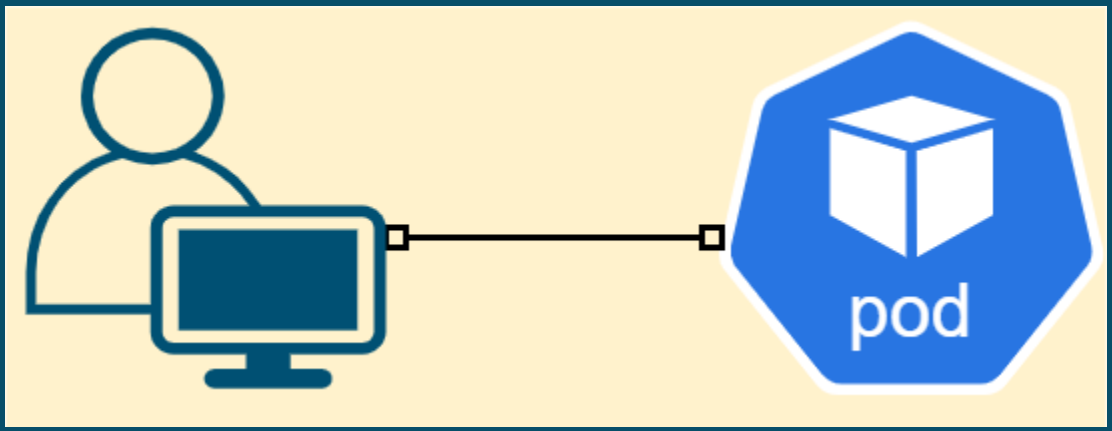
Since pod IPs are ephemeral, Services provide stable endpoints.

Example: Exposing a Deployment with a Service

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

External access (external-to-pod)

An application needs to be accessible from outside the cluster.



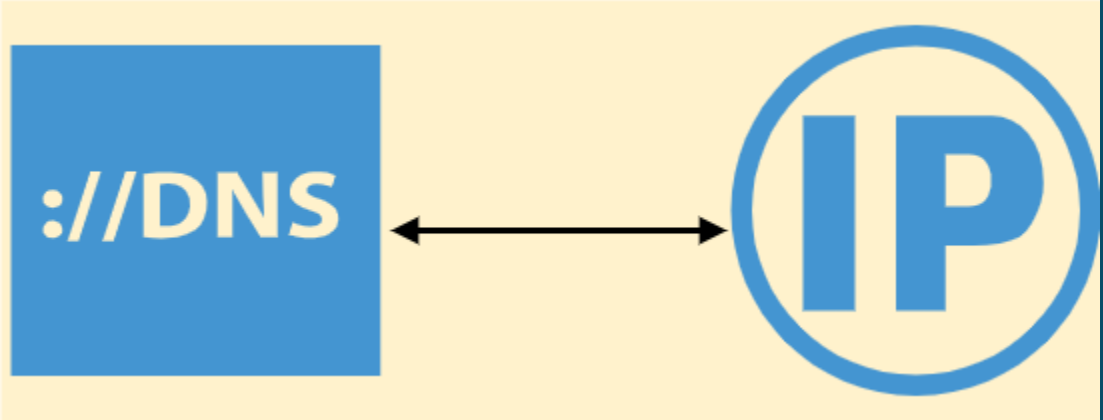
External traffic can reach the cluster through a LoadBalancer or Ingress, which routes the traffic to the appropriate Pods via the Service.

Example: Ingress resource

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
spec:
  rules:
  - host: myapp.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: my-service
            port:
              number: 80
```

DNS and service discovery

Services need to be discovered using DNS names.



Kubernetes provides DNS-based service discovery, allowing Pods to resolve Services using their DNS names.

Kubernetes provides CoreDNS for internal name resolution.

Services are accessible via *service-name.namespace.svc.cluster.local*.

Example: Resolving a service name

```
kubectl exec -it pod1 -- nslookup my-service.default.svc.cluster.local
```

Network policies

Network policies allow you to control traffic flow between Pods.

Network policies act as a firewall, allowing or denying traffic based on labels and namespaces.

By default, all pods can communicate with each other.

Example: Denying all traffic except from a specific app

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: restrict-access
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
    - Ingress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: frontend
```

Yasir Rehman

@ yasirrehman1

Find this insightful?

Repost! 🔁

I am a Senior DevOps Engineer and Team Lead.

I post about DevOps, cloud-native technologies, and compassionate leadership.