Kubernetes Networking Concepts

1 Service

A Service in Kubernetes is an abstraction that defines a logical set of Pods and a policy by which to access them. Services enable network access to a set of Pods, providing:

- Stable IP address and DNS name
- Load balancing across Pods
- Decoupling from ephemeral Pod IP addresses

Services can be exposed in different ways: ClusterIP (internal), NodePort (static port on each node), LoadBalancer (cloud provider integration), or ExternalName (CNAME redirect).

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
  type: ClusterIP
```

2 Ingress

An Ingress is an API object that manages external access to HTTP/HTTPS services in a cluster. It provides:

- Layer 7 (HTTP/HTTPS) routing capabilities
- Host-based and path-based routing
- TLS termination
- A single entry point for multiple services

Unlike Services which work at layer 4, Ingress operates at the application layer and provides more advanced routing features.

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

3 Ingress Controllers

An Ingress Controller is responsible for fulfilling Ingress requests. It's a daemon that runs in a cluster and evaluates Ingress rules to manage traffic routing. Common implementations include:

- NGINX Ingress Controller
- HAProxy Ingress
- Traefik
- Cloud-provider specific controllers (e.g., AWS ALB Ingress Controller)

You must deploy an Ingress Controller for your Ingress resources to function.

4 Gateway API

The Gateway API is an evolution of the Ingress API that provides more expressive and extensible routing capabilities. It consists of several resources:

- GatewayClass: defines a class of Gateways
- Gateway: requests a point where traffic can be translated to Services
- HTTPRoute: specifies routing rules for HTTP traffic

This API supports more advanced routing scenarios and role-based access control.

```yaml
apiVersion: gateway.networking.k8s.io/v1beta1
kind: Gateway
metadata:
  name: my-gateway
spec:
  gatewayClassName: acme-lb
  listeners:
```

```
  - name: http
    port: 80
    protocol: HTTP
    allowedRoutes:
      namespaces:
        from: Same
```

5 EndpointSlices

EndpointSlices provide a scalable and extensible alternative to the Endpoints API for tracking network endpoints in a Kubernetes cluster. They:

- Split endpoints across multiple resources for better performance
- Support additional metadata and attributes
- Enable more efficient endpoint tracking in large clusters

Kubernetes automatically creates EndpointSlices when a Service selector is defined.

```
apiVersion: discovery.k8s.io/v1
kind: EndpointSlice
metadata:
  name: example-endpoints
  labels:
    kubernetes.io/service-name: example-service
addressType: IPv4
ports:
  - name: http
    protocol: TCP
    port: 80
endpoints:
  - addresses:
      - "10.1.2.3"
    conditions:
      ready: true
```

6 Network Policies

Network Policies are Kubernetes resources that control traffic flow between Pods. They provide:

- Pod-level firewall capabilities
- Ingress and egress rule definitions
- Selector-based targeting of Pods
- Protocol and port specifications

Network Policies require a compatible CNI (Container Network Interface) plugin that supports the NetworkPolicy API.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: test-network-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```

7 DNS for Services and Pods

Kubernetes includes a built-in DNS service that provides name resolution for:

- Services: `<service-name>.<namespace>.svc.cluster.local`
- Pods: `<pod-ip>.<namespace>.pod.cluster.local` (with dashes instead of dots in the IP)

CoreDNS is the default DNS server, replacing kube-dns in modern Kubernetes deployments.

8 IPv4/IPv6 dual-stack

Kubernetes supports dual-stack networking, allowing:

- Simultaneous assignment of IPv4 and IPv6 addresses to Pods and Services
- api-family selection (IPv4, IPv6, or both)
- Compatibility with dual-stack network environments

Dual-stack must be enabled at cluster creation time and requires CNI plugin support.

```
apiVersion: v1
kind: Service
metadata:
  name: dualstack-service
spec:
  ipFamilyPolicy: RequireDualStack
  selector:
    app: MyApp
  ports:
    - protocol: TCP
```

```
    port: 80
```

9 Topology Aware Routing

Topology Aware Routing directs traffic to endpoints within the same zone to:

- Reduce cross-zone network costs
- Improve network latency
- Increase reliability

This feature uses topology hints from EndpointSlices to make intelligent routing decisions.

```
apiVersion: v1
kind: Service
metadata:
  name: topology-service
  annotations:
    service.kubernetes.io/topology-aware-hints: "auto"
spec:
  selector:
    app: MyApp
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```