Services, Load Balancing, and Networking

Concepts and resources behind networking in Kubernetes.

The Kubernetes network model

The Kubernetes network model is built out of several pieces:

- Each pod in a cluster gets its own unique cluster-wide IP address.
  - A pod has its own private network namespace which is shared by all of the containers within the pod. Processes running in different containers in the same pod can communicate with each other over `localhost`.
- The *pod network* (also called a cluster network) handles communication between pods. It ensures that (barring intentional network segmentation):
  - All pods can communicate with all other pods, whether they are on the same node or on different nodes. Pods can communicate with each other directly, without the use of proxies or address translation (NAT).

    On Windows, this rule does not apply to host-network pods.
  - Agents on a node (such as system daemons, or kubelet) can communicate with all pods on that node.
- The Service API lets you provide a stable (long lived) IP address or hostname for a service implemented by one or more backend pods, where the individual pods making up the service can change over time.
  - Kubernetes automatically manages EndpointSlice objects to provide information about the pods currently backing a Service.
  - A service proxy implementation monitors the set of Service and EndpointSlice objects, and programs the data plane to route service traffic to its backends, by using operating system or cloud provider APIs to intercept or rewrite packets.
- The Gateway API (or its predecessor, Ingress) allows you to make Services accessible to clients that are outside the cluster.
  - A simpler, but less-configurable, mechanism for cluster ingress is available via the Service API's `type: LoadBalancer`, when using a supported Cloud Provider.
- NetworkPolicy is a built-in Kubernetes API that allows you to control traffic between pods, or between pods and the outside world.

In older container systems, there was no automatic connectivity between containers on different hosts, and so it was often necessary to explicitly create links between containers, or to map container ports to host ports to make them reachable by containers on other hosts. This is not needed in Kubernetes; Kubernetes's model is that pods can be treated much like VMs or physical hosts from the perspectives of port allocation, naming, service discovery, load balancing, application configuration, and migration.

Only a few parts of this model are implemented by Kubernetes itself. For the other parts, Kubernetes defines the APIs, but the corresponding functionality is provided by external components, some of which are optional:

- Pod network namespace setup is handled by system-level software implementing the Container Runtime Interface.
- The pod network itself is managed by a pod network implementation. On Linux, most container runtimes use the Container Networking Interface (CNI) to interact with the pod network implementation, so these implementations are often called *CNI plugins*.
- Kubernetes provides a default implementation of service proxying, called kube-proxy, but some pod network implementations instead use their own service proxy that is more tightly integrated with the rest of the implementation.
- NetworkPolicy is generally also implemented by the pod network implementation. (Some simpler pod network implementations don't implement NetworkPolicy, or an administrator may choose to configure the pod network without NetworkPolicy support. In these cases, the API will still be present, but it will have no effect.)
- There are many implementations of the Gateway API, some of which are specific to particular cloud environments, some more focused on "bare metal" environments, and others more generic.

What's next

The Connecting Applications with Services tutorial lets you learn about Services and Kubernetes networking with a hands-on example.

Cluster Networking explains how to set up networking for your cluster, and also provides an overview of the technologies involved.

Service
Expose an application running in your cluster behind a single outward-facing endpoint, even when the workload is split across multiple backends.

Ingress
Make your HTTP (or HTTPS) network service available using a protocol-aware configuration mechanism, that understands web concepts like URIs, hostnames, paths, and more. The Ingress concept lets you map traffic to different backends based on rules you define via the Kubernetes API.

Ingress Controllers
In order for an Ingress to work in your cluster, there must be an *ingress controller* running. You need to select at least one ingress controller and make sure it is set up in your cluster. This page lists common ingress controllers that you can deploy.

Gateway API
Gateway API is a family of API kinds that provide dynamic infrastructure provisioning and advanced traffic routing.

EndpointSlices
The EndpointSlice API is the mechanism that Kubernetes uses to let your Service scale to handle large numbers of backends, and allows the cluster to update its list of healthy backends efficiently.

Network Policies
If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), NetworkPolicies allow you to specify rules for traffic flow within your cluster, and also between Pods and the outside world. Your cluster must use a network plugin that supports NetworkPolicy enforcement.

DNS for Services and Pods
Your workload can discover Services within your cluster using DNS; this page explains how that works.

IPv4/IPv6 dual-stack
Kubernetes lets you configure single-stack IPv4 networking, single-stack IPv6 networking, or dual stack networking with both network families active. This page explains how.

Topology Aware Routing
Topology Aware Routing provides a mechanism to help keep network traffic within the zone where it originated. Preferring same-zone traffic between Pods in your cluster can help with reliability, performance (network latency and throughput), or cost.

Networking on Windows

Service ClusterIP allocation

Service Internal Traffic Policy
If two Pods in your cluster want to communicate, and both Pods are actually running on the same node, use *Service Internal Traffic Policy* to keep network traffic within that node. Avoiding a round trip via the cluster network can help with reliability, performance (network latency and throughput), or cost.

Last modified November 05, 2025 at 11:25 AM PST: Fix broken CRI link in services-networking documentation (982a59a1f9)