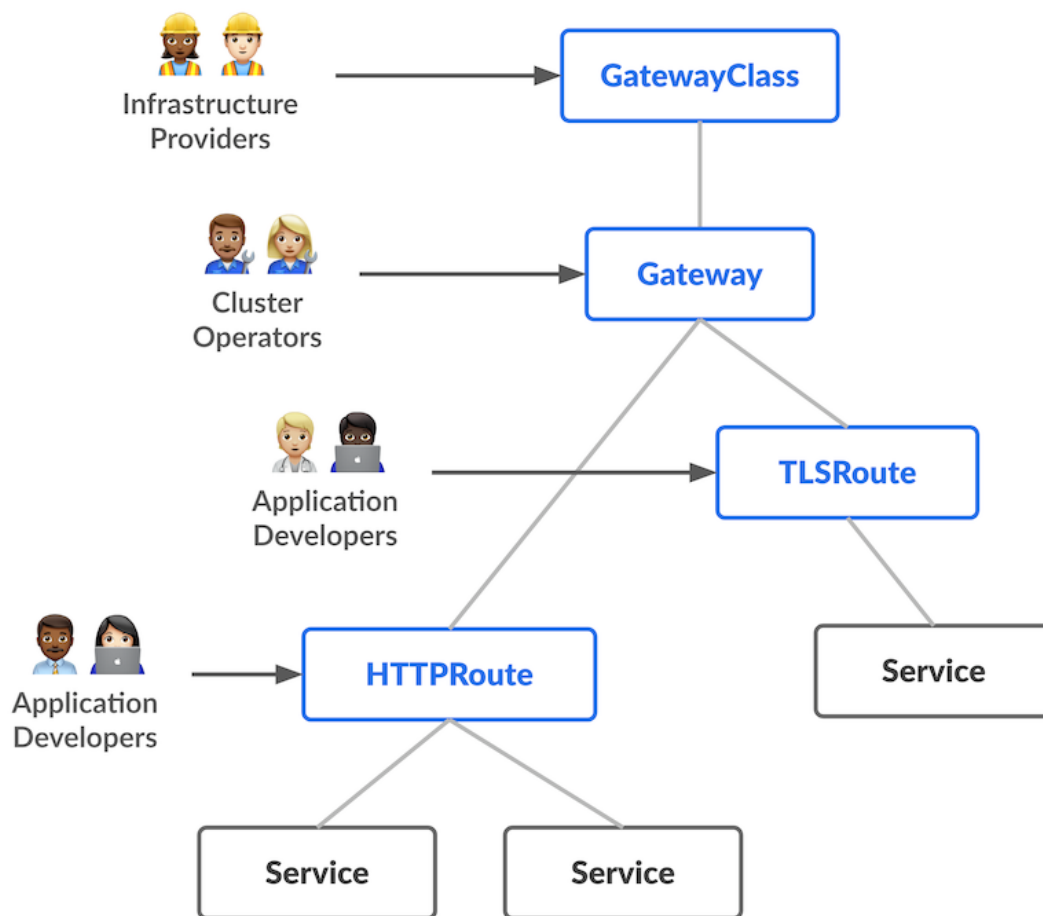


Introduction

Gateway API is an official Kubernetes project focused on L4 and L7 routing in Kubernetes. This project represents the next generation of Kubernetes Ingress, Load Balancing, and Service Mesh APIs. From the outset, it has been designed to be generic, expressive, and role-oriented.

The overall resource model focuses on 3 separate [personas](#) and corresponding resources that they are expected to manage:



Most of the configuration in this API is contained in the Routing layer. These protocol-specific resources ([HTTPRoute](#), [GRPCRoute](#), etc) enable advanced routing capabilities for both Ingress and Mesh.

The Gateway API Logo helps illustrate the dual purpose of this API, enabling routing for both

North-South (Ingress) and East-West (Mesh) traffic to share the same configuration.



gateway api

Gateway API for Ingress

✓ Standard Channel since v0.5.0 ✓

Gateway, GatewayClass, and HTTPRoute have been part of the Standard Channel of Gateway API since v0.5.0 and are considered stable APIs. For more information refer to our [versioning guide](#).

When using Gateway API to manage ingress traffic, the [Gateway](#) resource defines a point of access at which traffic can be routed across multiple contexts -- for example, from outside the cluster to inside the cluster ([north/south traffic](#)).

Each Gateway is associated with a [GatewayClass](#), which describes the actual kind of [gateway controller](#) that will handle traffic for the Gateway; individual routing resources (such as [HTTPRoute](#)) are then [associated with the Gateway resources](#). Separating these different concerns into distinct resources is a critical part of the role-oriented nature of Gateway API, as well as allowing for multiple kinds of gateway controllers (represented by GatewayClass resources), each with multiple instances (represented by Gateway resources), in the same cluster.

Gateway API for Service Mesh (the [GAMMA initiative](#))

✓ Standard Channel since v1.1.0 ✓

The [GAMMA initiative](#) work for supporting service mesh use cases has been part of the Standard Channel since v1.1.0 and is considered GA. For more information refer to our [versioning guide](#).

Things are a bit different when using Gateway API to manage a [service mesh](#). Since there will usually only be one mesh active in the cluster, the [Gateway](#) and [GatewayClass](#) resources are not

used; instead, individual route resources (such as [HTTPRoute](#)) are [associated directly with Service resources](#), permitting the mesh to manage traffic from any traffic directed to that Service while preserving the role-oriented nature of Gateway API.

To date, [GAMMA](#) has been able to support mesh functionality with fairly minimal changes to Gateway API. One particular area that has rapidly become critical for GAMMA, though, is the definition of the different [facets of the Service resource](#).

Getting started

Whether you are a user interested in using Gateway API or an implementer interested in conforming to the API, the following resources will help give you the necessary background:

- [API overview](#)
- [User guides](#)
- [Implementations](#)
- [API reference spec](#)
- [Community links](#) and [developer guide](#)

Gateway API concepts

The following design goals drive the concepts of Gateway API. These demonstrate how Gateway aims to improve upon current standards like Ingress.

- **Role-oriented** - Gateway is composed of API resources which model organizational roles that use and configure Kubernetes service networking.
- **Portable** - This isn't an improvement but rather something that should stay the same. Just as Ingress is a universal specification with [numerous implementations](#), Gateway API is designed to be a portable specification supported by many implementations.
- **Expressive** - Gateway API resources support core functionality for things like header-based matching, traffic weighting, and other capabilities that were only possible in Ingress through custom annotations.
- **Extensible** - Gateway API allows for custom resources to be linked at various layers of the API. This makes granular customization possible at the appropriate places within the API structure.

Some other notable capabilities include:

- **GatewayClasses** - GatewayClasses formalize types of load balancing implementations. These

classes make it easy and explicit for users to understand what kind of capabilities are available via the Kubernetes resource model.

- **Shared Gateways and cross-Namespace support** - They allow the sharing of load balancers and VIPs by permitting independent Route resources to attach to the same Gateway. This allows teams (even across Namespaces) to share infrastructure safely without direct coordination.
- **Typed Routes and typed backends** - Gateway API supports typed Route resources and also different types of backends. This allows the API to be flexible in supporting various protocols (like HTTP and gRPC) and various backend targets (like Kubernetes Services, storage buckets, or functions).
- Experimental **Service mesh support** with the [GAMMA initiative](#) - Gateway API supports associating routing resources with Service resources, to configure service meshes as well as ingress controllers.

Why does a role-oriented API matter?

Whether it's roads, power, data centers, or Kubernetes clusters, infrastructure is built to be shared. However, shared infrastructure raises a common challenge - how to provide flexibility to users of the infrastructure while maintaining control by owners of the infrastructure?

Gateway API accomplishes this through a role-oriented design for Kubernetes service networking that strikes a balance between distributed flexibility and centralized control. It allows shared network infrastructure (hardware load balancers, cloud networking, cluster-hosted proxies etc) to be used by many different and non-coordinating teams, all bound by the policies and constraints set by cluster operators.

The roles used for Gateway API's design are defined by three personas:

Personas

- **Ian** (he/him) is an *infrastructure provider*. His role is the care and feeding of a set of infrastructure that permits multiple isolated clusters to serve multiple tenants. He is not beholden to any single tenant; rather, he worries about all of them collectively.
- **Chihiro** (they/them) is a *cluster operator*. Their role is to manage a single cluster, ensuring that it meets the needs of its several users. Again, Chihiro is beholden to no single user of their cluster; they need to make sure that the cluster serves all of them as needed.
- **Ana** (she/her) is an *application developer*. Ana is in a unique position among the Gateway API roles: her focus is on the business needs her application is meant to serve, *not* Kubernetes or

Gateway API. In fact, Ana is likely to view Gateway API and Kubernetes as pure friction getting in her way to get things done.

(These three are discussed in more detail in [Roles and Personas](#).)

It should be clear that while Ana, Chihiro, and Ian do not necessarily see eye-to-eye about everything, they need to work together to keep things running smoothly. This is the core challenge of Gateway API in a nutshell.

Use Cases

The [example use cases](#) show this role-oriented model at work. Its flexibility allows the API to adapt to vastly different organizational models and implementations while remaining a portable and standard API.

The use cases presented are deliberately cast in terms of the roles presented above. Ultimately Gateway API is meant for use by humans, which means that it must fit the uses to which each of Ana, Chihiro, and Ian will put it.

What's the difference between Gateway API and an API Gateway?

An [API gateway](#) is a tool that aggregates unique application APIs, making them all available in one place. It allows organizations to move key functions, such as authentication and authorization or limiting the number of requests between applications, to a centrally managed location. An API gateway functions as a common interface to (often external) API consumers.

Gateway API is an interface, defined as a set of Kubernetes resources, that models service networking in Kubernetes. One of the main resources is a `Gateway`, which declares the Gateway type (or class) to instantiate and its configuration. As a Gateway provider, you can implement Gateway API to model Kubernetes service networking in an expressive, extensible, and role-oriented way.

Some API gateways can be programmed using the Gateway API.

Who is working on Gateway API?

Gateway API is a [SIG-Network](#) project being built to improve and standardize service networking in Kubernetes. Check out the [implementations reference](#) to see the latest projects & products that support Gateway. If you are interested in contributing to or building an implementation using Gateway API then don't hesitate to [get involved](#)!

