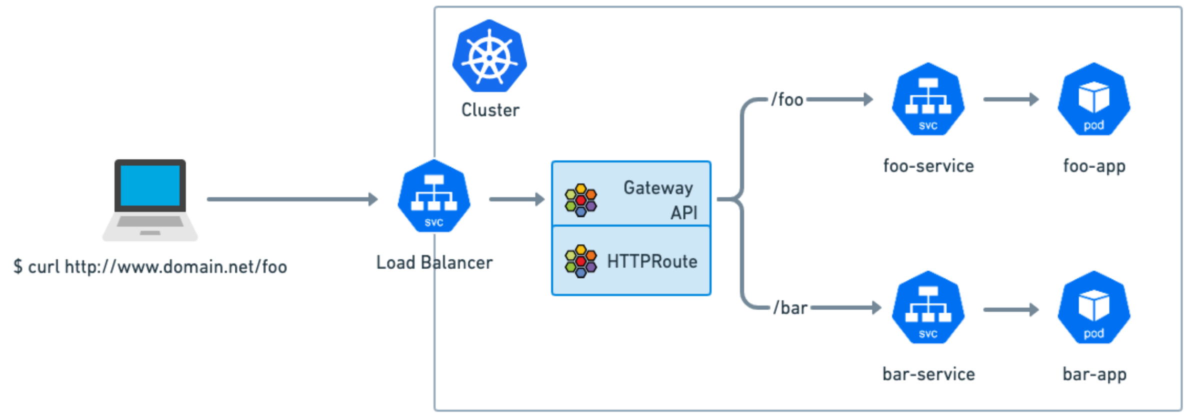
HTTP Routing

HTTP routing is the process of directing incoming HTTP requests to the appropriate backend service. With the Gateway API, users can define routes based on the incoming request's HTTP method, URL path, and headers. For example, you could route all GET requests to one backend service, while routing all POST requests to another.

To configure HTTP routing in Cilium, you first define a `Gateway` object, which represents the entry point for all incoming traffic. Then, you create a `HTTPRoute` object, which defines the routing rules for incoming HTTP requests. A `HTTPRoute` can include multiple route rules, each of which specifies a different backend service to send traffic to.

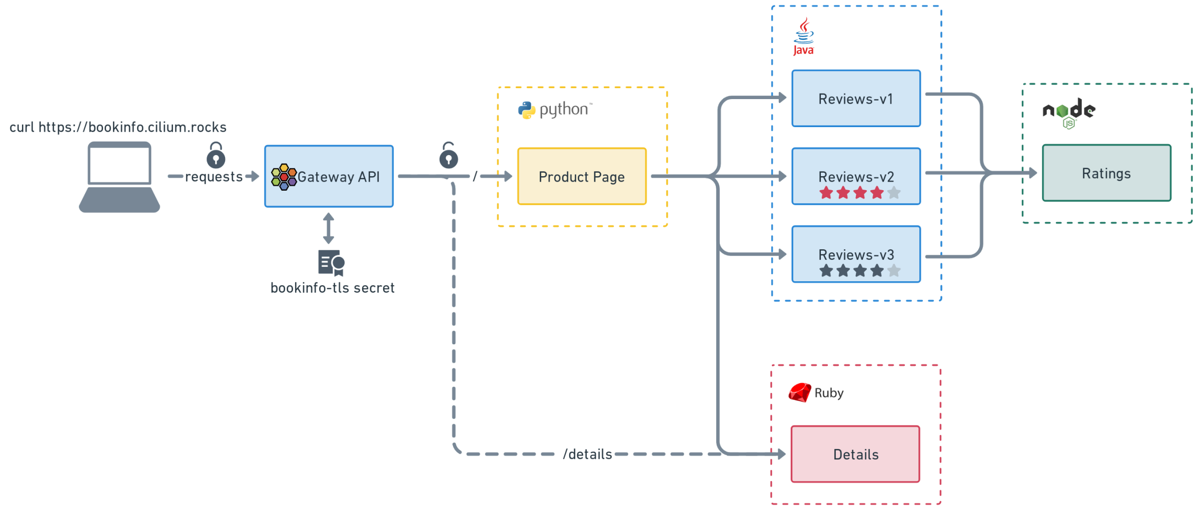


Here's a short video of how Cilium helps with HTTP routing into a Kubernetes cluster:

TLS Termination

TLS termination is the process of decrypting incoming TLS traffic at the gateway and forwarding it to the appropriate backend service over an unencrypted connection. The Cilium Gateway API includes built-in support for TLS termination, allowing users to easily secure incoming traffic into their Kubernetes clusters.

To configure TLS termination in Cilium, you define a Gateway object with a TLS configuration. The TLS configuration includes the certificate and private key used to encrypt and decrypt the incoming traffic. You can then configure your `HTTPRoute` objects to use the TLS-enabled `Gateway` object to secure incoming traffic.

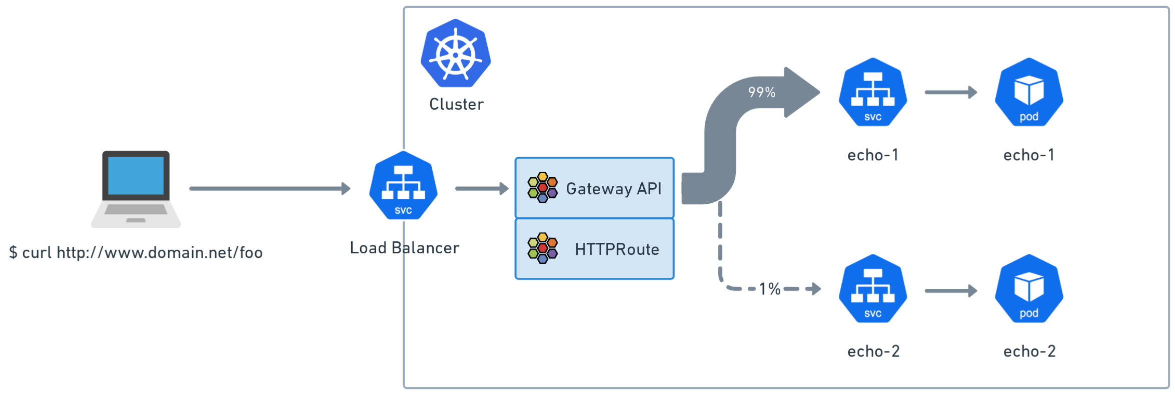


Here is a short demo of how Cilium implements TLS Termination for inbound traffic:

HTTP Traffic Splitting / Weighting

HTTP traffic splitting is the process of sending incoming traffic to multiple backend services, based on predefined weights or other criteria. The Cilium Gateway API includes built-in support for traffic splitting, allowing users to easily distribute incoming traffic across multiple backend services. This is very useful for canary testing or A/B scenarios.

To configure traffic splitting in Cilium, you define a `Gateway` object with multiple backend services, each with its own weight. You can then set up various weights in your `HTTPRoute` objects, specifying the percentage of traffic to send to each backend service.



Here's a short video of how Cilium provides HTTP Traffic Splitting:

HTTP Request Header Modification

HTTP header modification is the process of adding, removing, or modifying HTTP headers in incoming requests. The Cilium Gateway API lets users easily customize incoming traffic to meet their specific needs.

To configure HTTP header modification in Cilium, you define a `Gateway` object with one or more HTTP filters. Each filter specifies a specific modification to make to incoming requests, such as adding a custom header or modifying an existing header.

Here's a short video showing how Cilium can add a header to a HTTP packet:

HTTP Response Header Modification

Just like editing request headers can be useful, the same goes for response headers. For example, it allows teams to add/remove cookies for only a certain backend, which can help in identifying certain users that were redirected to that backend previously.

Another potential use case could be when you have a frontend that needs to know whether it's talking to a stable or a beta version of the backend server, in order to render different UI or adapt its response parsing accordingly.

Here's a short video showing how Cilium can add a header to a HTTP response:

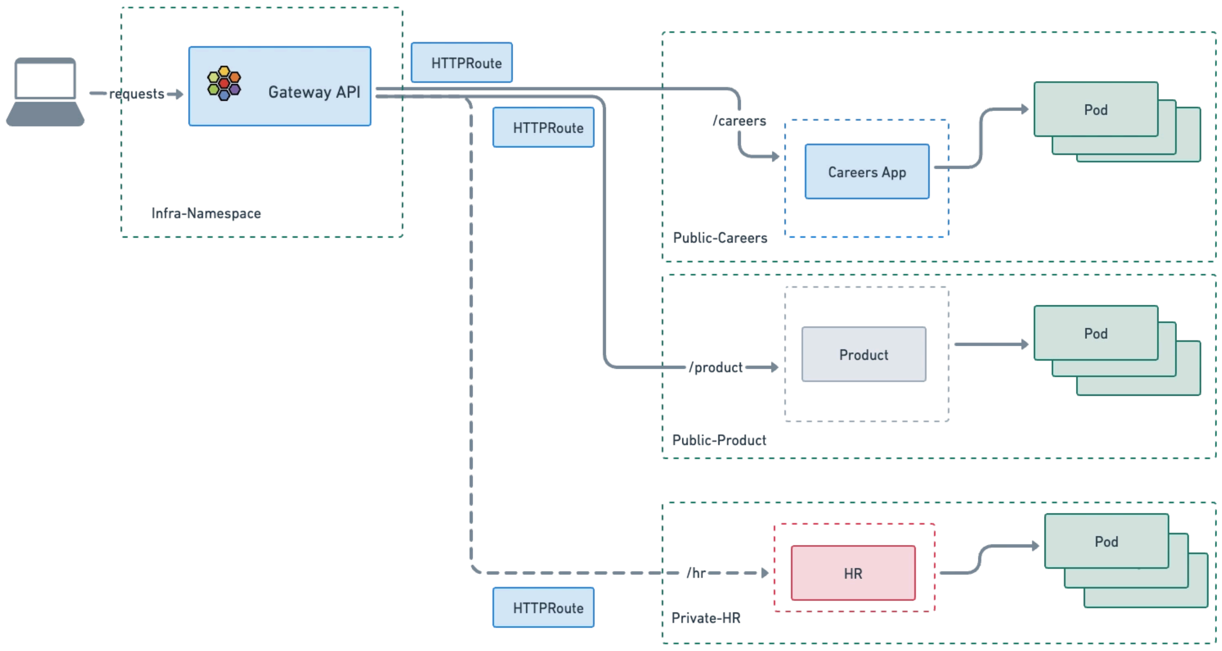
Cross Namespace Routing Support

The Gateway API has core support for cross Namespace routing. This is useful when more than one user or team is sharing the underlying networking infrastructure, yet control and configuration must be segmented to minimize access and fault domains.

`Gateways` and `Routes` can be deployed into different Namespaces and `Routes` can attach to `Gateways` across Namespace boundaries. `Gateway` and `Route` attachment is bidirectional – attachment can only succeed if the `Gateway` owner and `Route` owner owner both agree to the relationship.

This effectively creates a handshake between the infra owners and application owners that enables them to independently define how applications are exposed through Gateways.

It results in creating a policy that reduces administrative overhead. App owners can specify which `Gateways` their apps should use and infra owners can constrain the Namespaces and types of `Routes` that a Gateway accepts. An example cross-namespace architecture using a Shared Gateway API is represented below (you can test this specific scenario in the Advanced Gateway API lab).



Here is a short video that walks through the benefits of Cross-Namespace Routing with the Gateway API, followed by a demo:

Migrating to the Gateway API

For users interested in migrating from Ingress API to Gateway API, we have tested the experimental Ingress2Gateway tool that helps users migrate their Ingress configuration to Gateway API configuration. While still a prototype, the tool accurately converted simple Ingress Resources to Gateway API Resources.

Watch this brief video to learn more:

Getting Started

You have several options at your disposal to get started with the Cilium Gateway API:

Official Docs

Read the official docs on Gateway API.

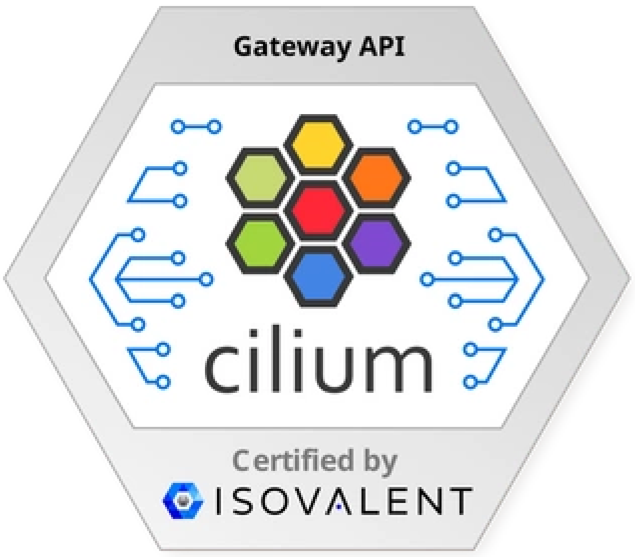
Tutorial

Read our walkthrough of Gateway API.

Labs

The Cilium Gateway API lab is a hands-on experience for users to learn how to deploy and use Cilium Gateway API. The lab covers topics such as deploying Cilium, deploying the Gateway API, and configuring HTTP routing and TLS termination.

On completion, you will receive a Silver Badge – feel free to share your achievement with your peers on social media!



Once you've completed this first lab, you now can take a follow-up lab: the Cilium Advanced Gateway API Use Cases lets you explore HTTP Header Request and Response modifications, Traffic Splitting and Cross-Namespace Routing. On completion of this lab, you will receive a Gold Badge:

The Future of the Gateway API