


Managed service mesh as a distributed cloud service



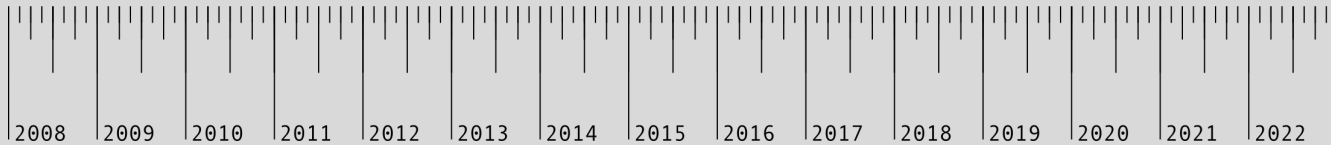
Gergő Huszty

Software / Network / Cloud infrastructure
Engineer @ IBM



 gergo_husztty

 libesz



Agenda

- Some context about distributed clouds
- Istio split control-plane model
- Challenges of implementing it
 - Istiod access to Mesh cluster K8S API
 - Remote cluster management with K8S operators
 - Exposing control-plane endpoints
 - Metrics collection
 - Fix istioctl commands with external control-plane
 - Multi-tenancy on the control-plane

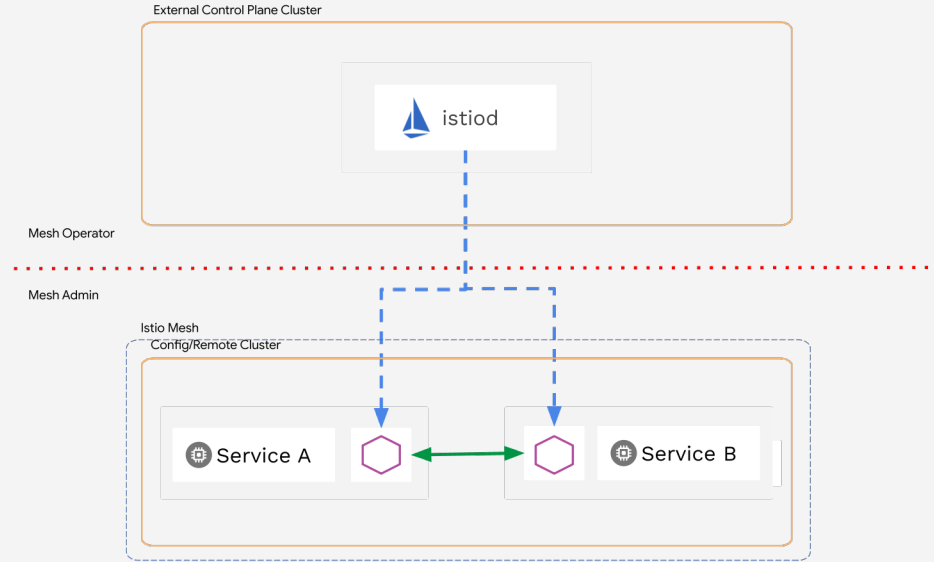
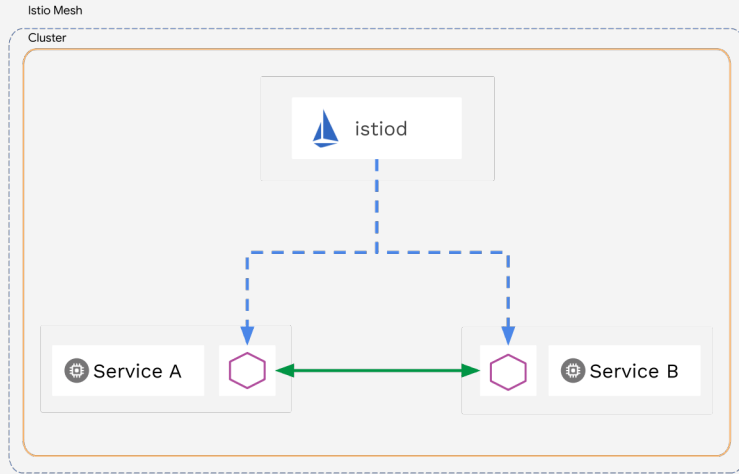
A few words about distributed cloud solutions

- Cloud providers tend to extend their offerings outside the cloud itself
- Customers can consume cloud services inside their own DC, or anywhere else (on edge, or in other clouds)
- Now it isn't someone else's computer anymore
 - You give the compute hosts that becomes managed by the provider
 - Compute hosts will either host your workload or become part of the provider's management infrastructure

The biased principle of any managed cloud service

- Run as less management entity on the user's side as possible, ideally 0.
 - User should not host (and pay for) stuff that is relevant for the service provider.
 - User will eventually break things that is accessible in the account. `_(`ツ)_`
 - Service provider has more tools to manage / scale / upgrade / monitor / heal the service control-plane if it runs on the provider side.

Istio deployment models



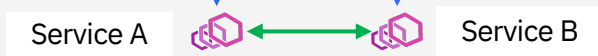
Basic mesh operation.

Split deployment model

Control-plane cluster



Data-plane cluster

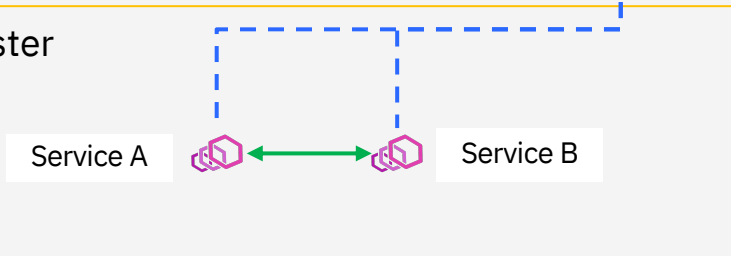


Proxies are clients on TCP level

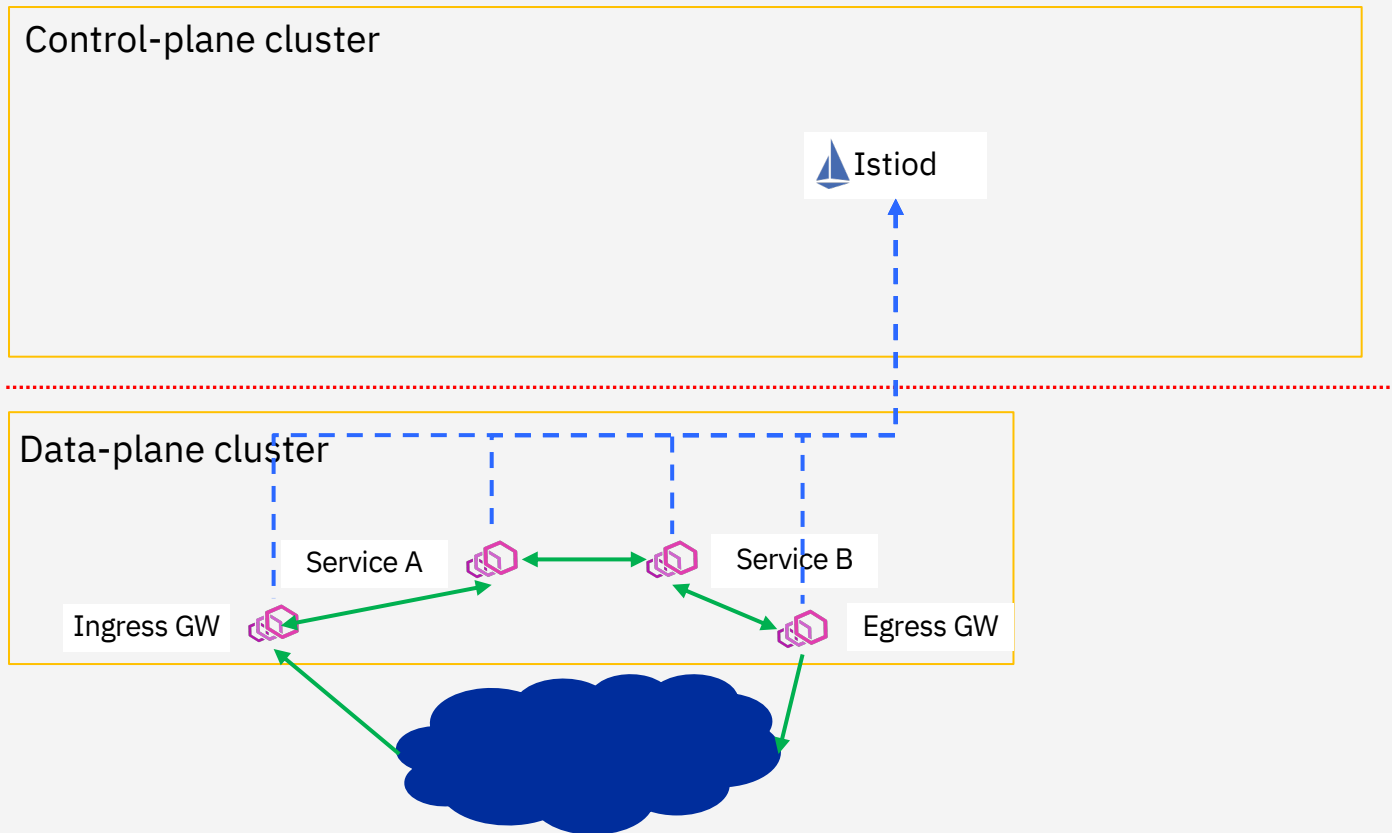
Control-plane cluster



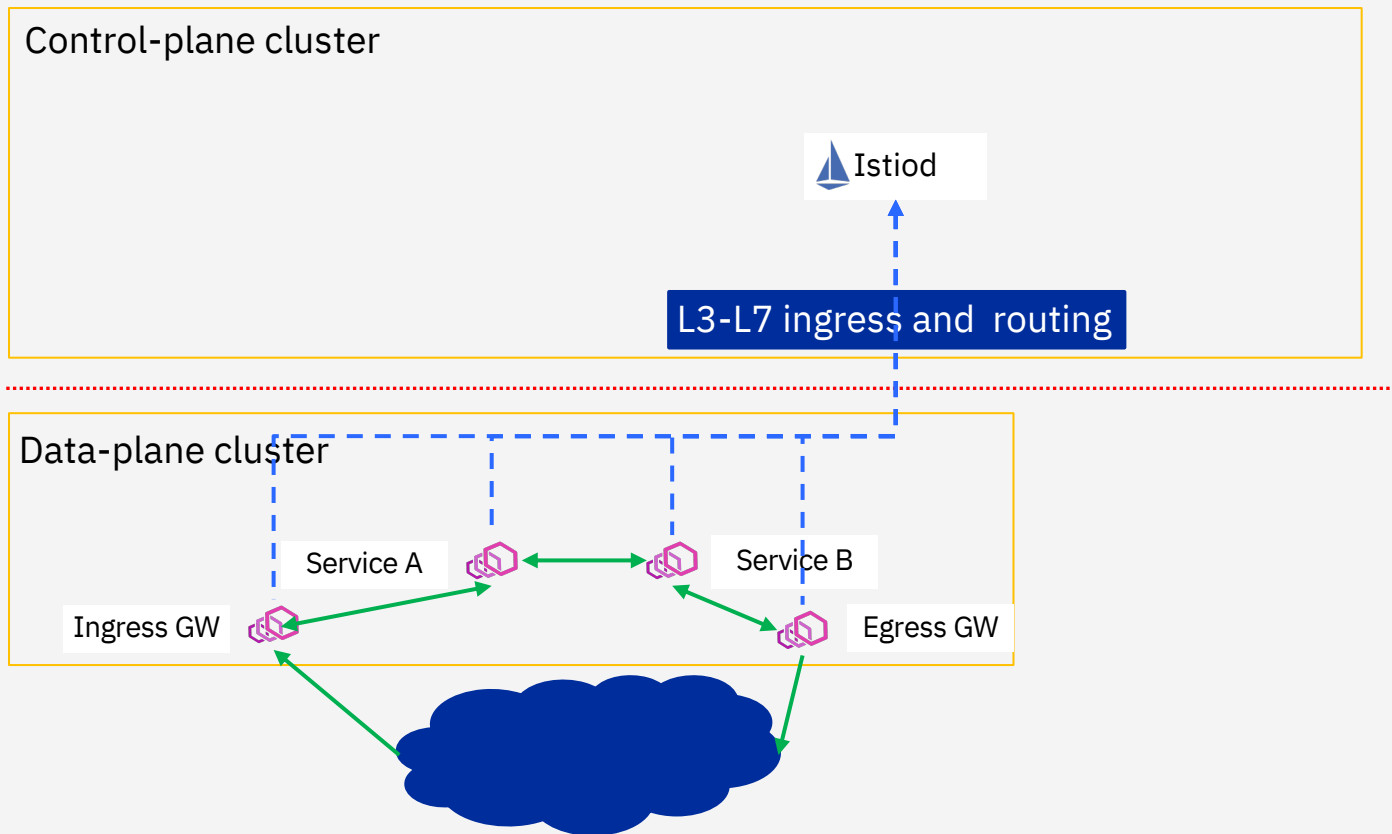
Data-plane cluster



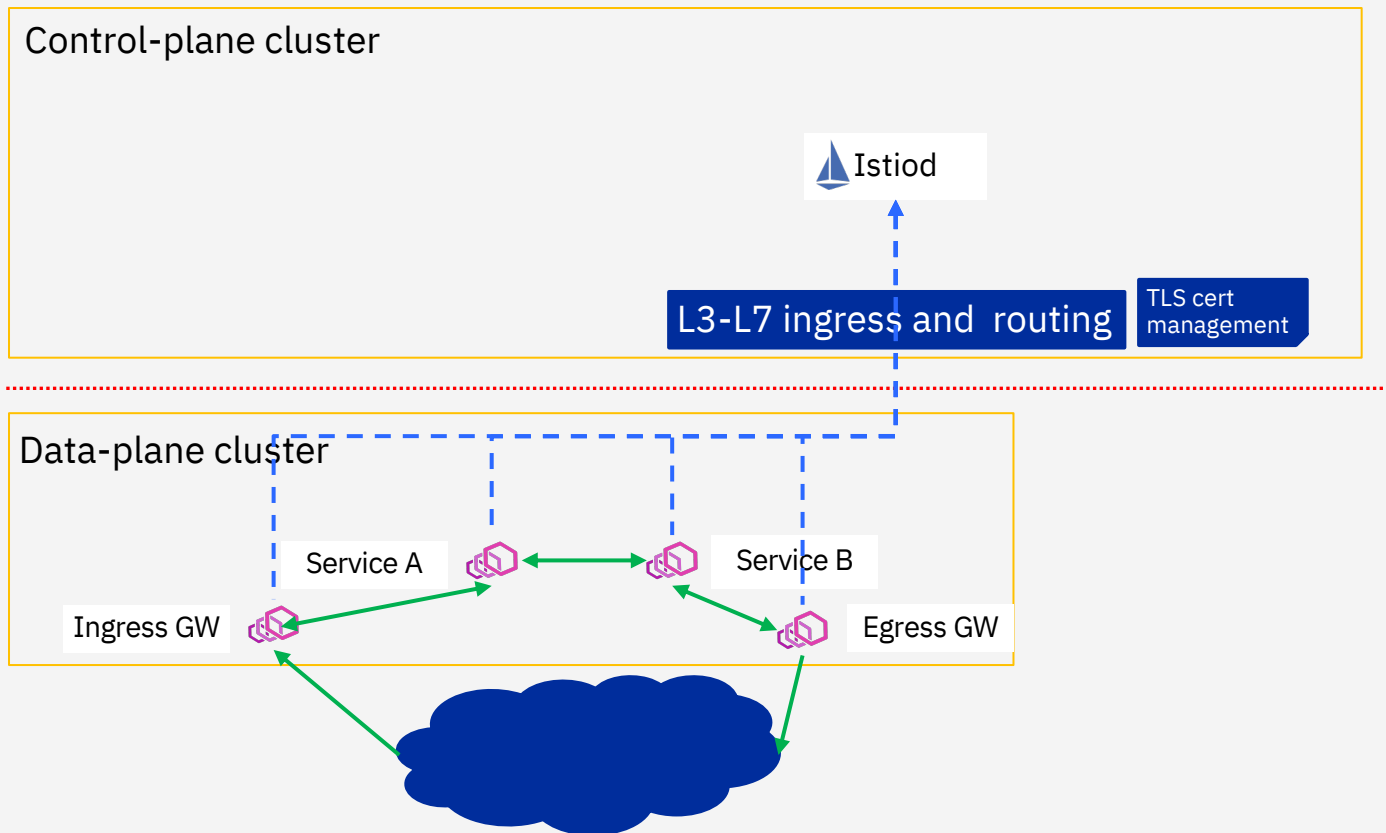
And we also have gateway proxies that handles edge traffic



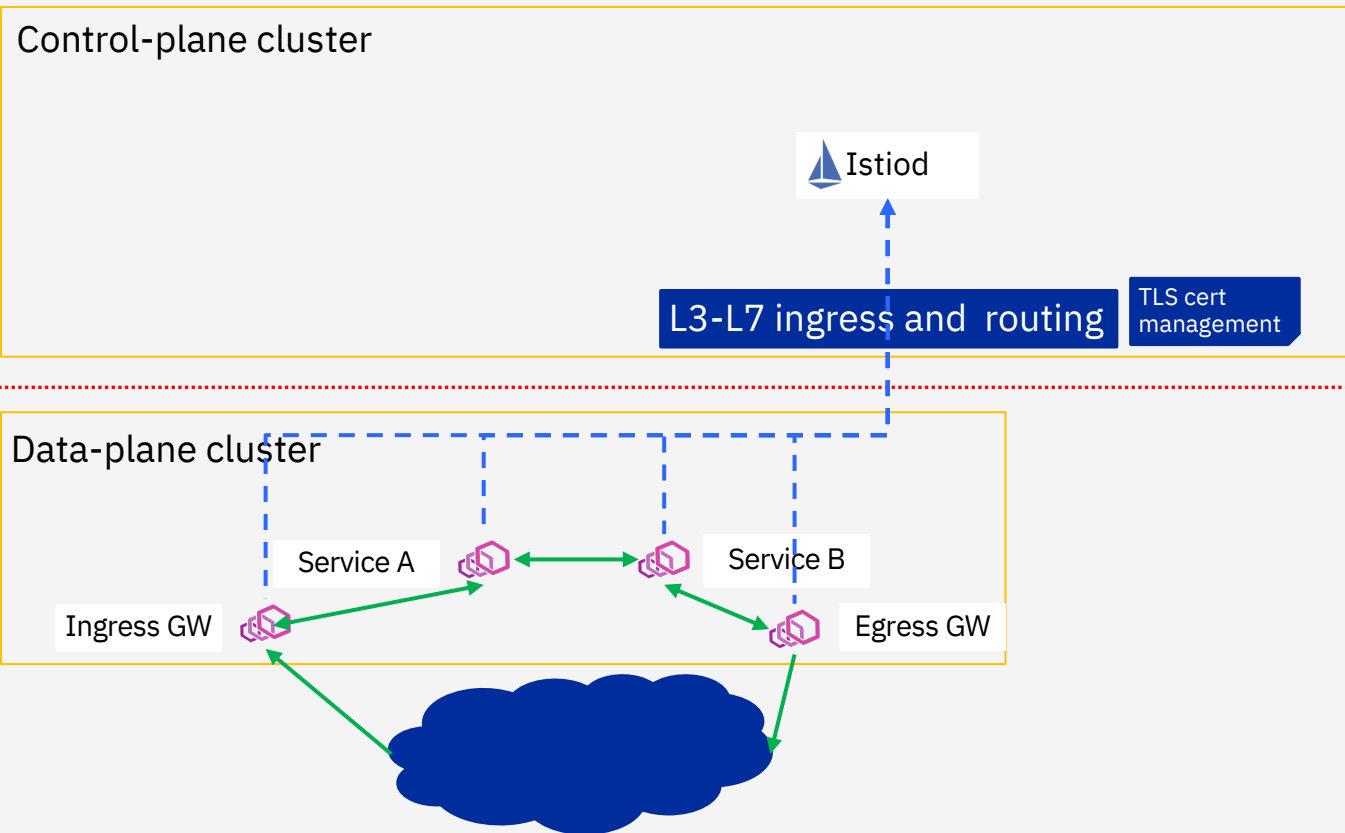
So we need proper and secure exposure for Istiod



So we need proper and secure exposure for Istiod

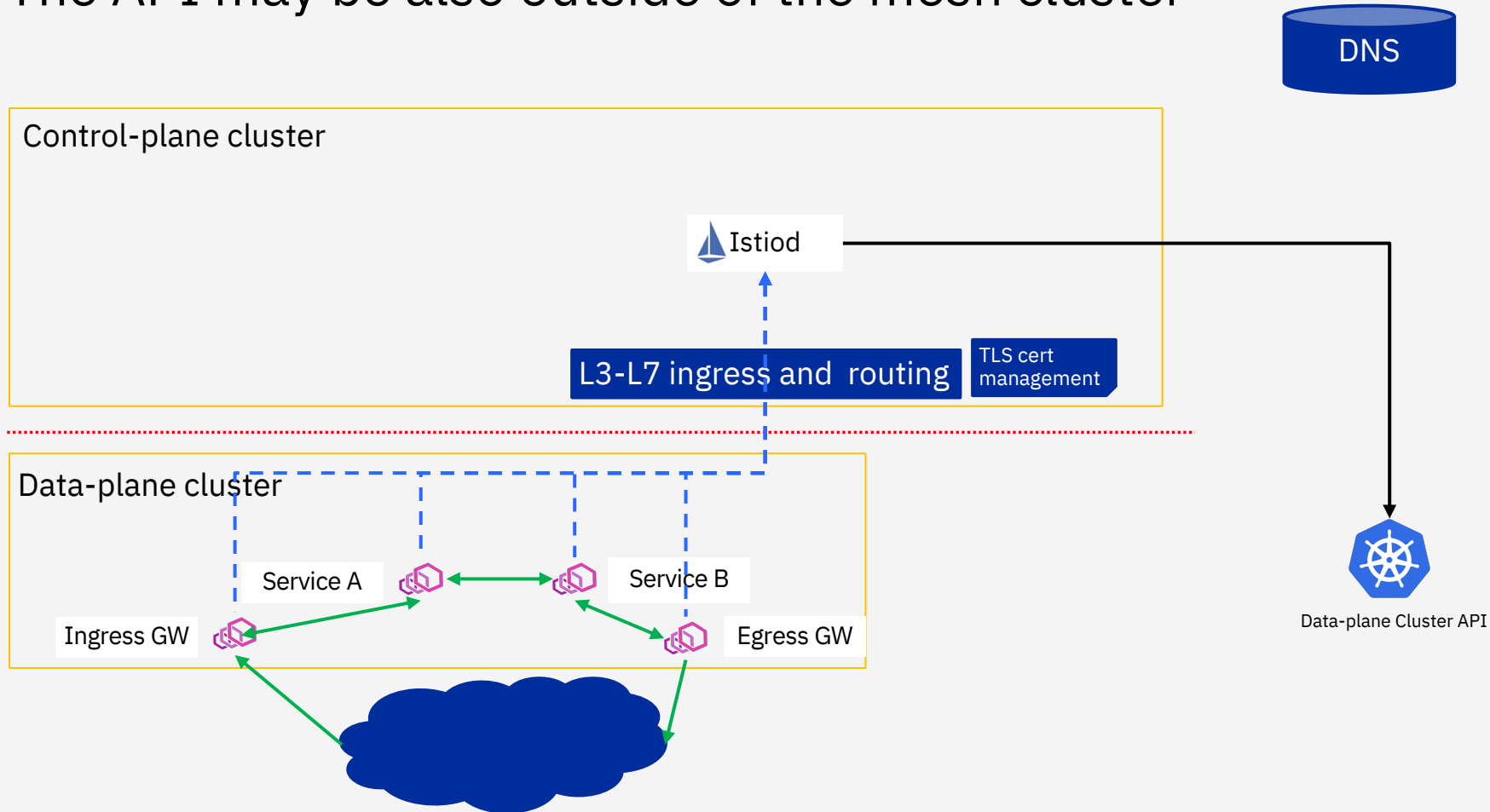


Istiod needs access to the mesh cluster API, but where is that API exactly?

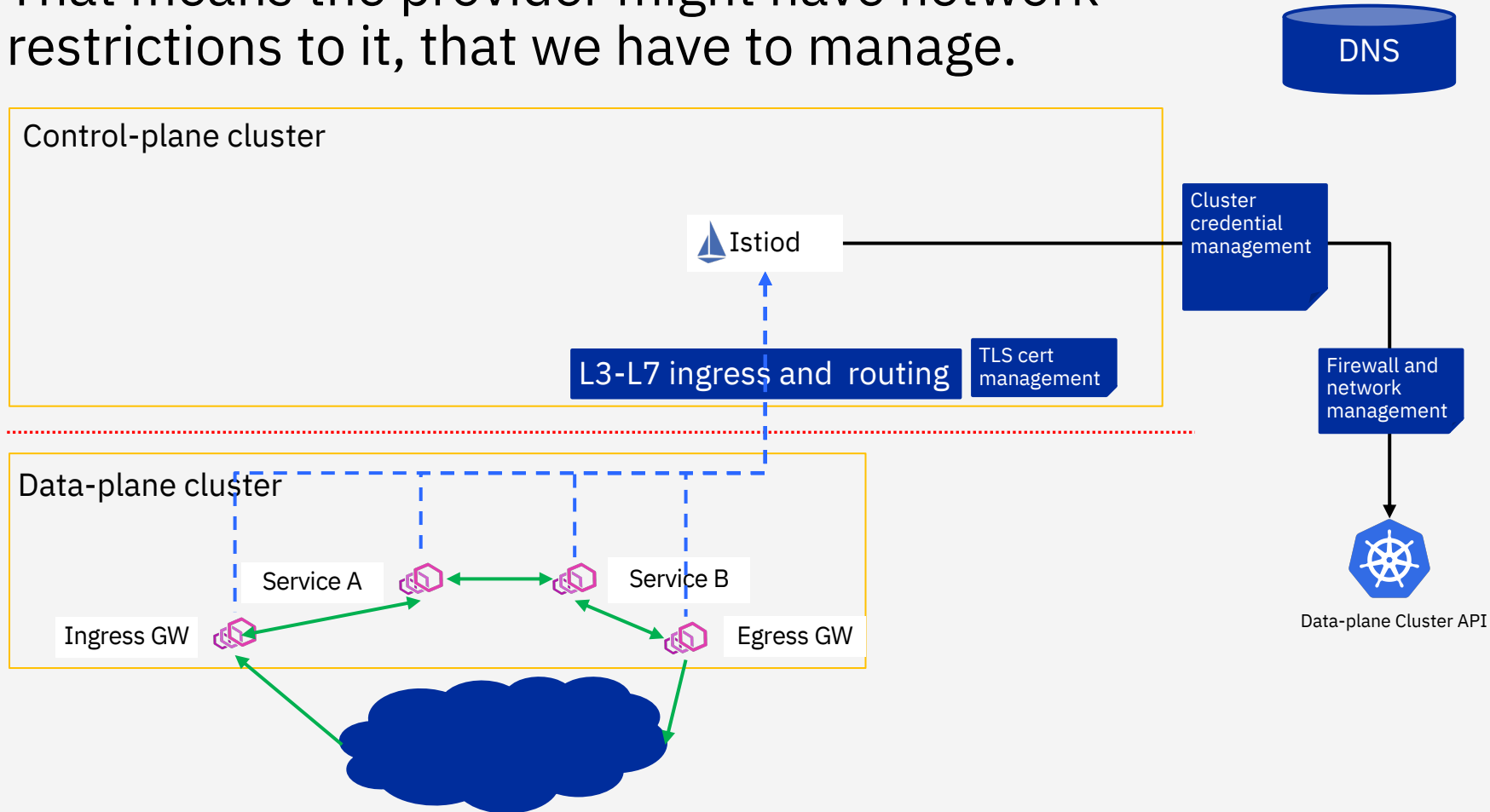


Data-plane Cluster API

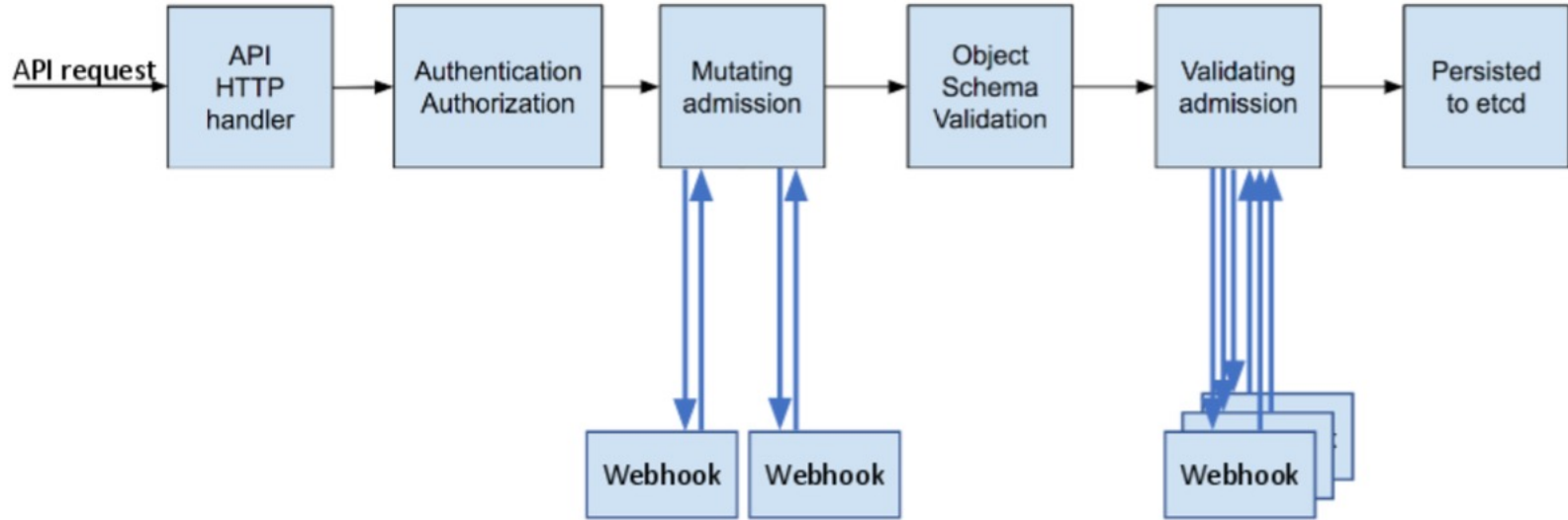
The API may be also outside of the mesh cluster



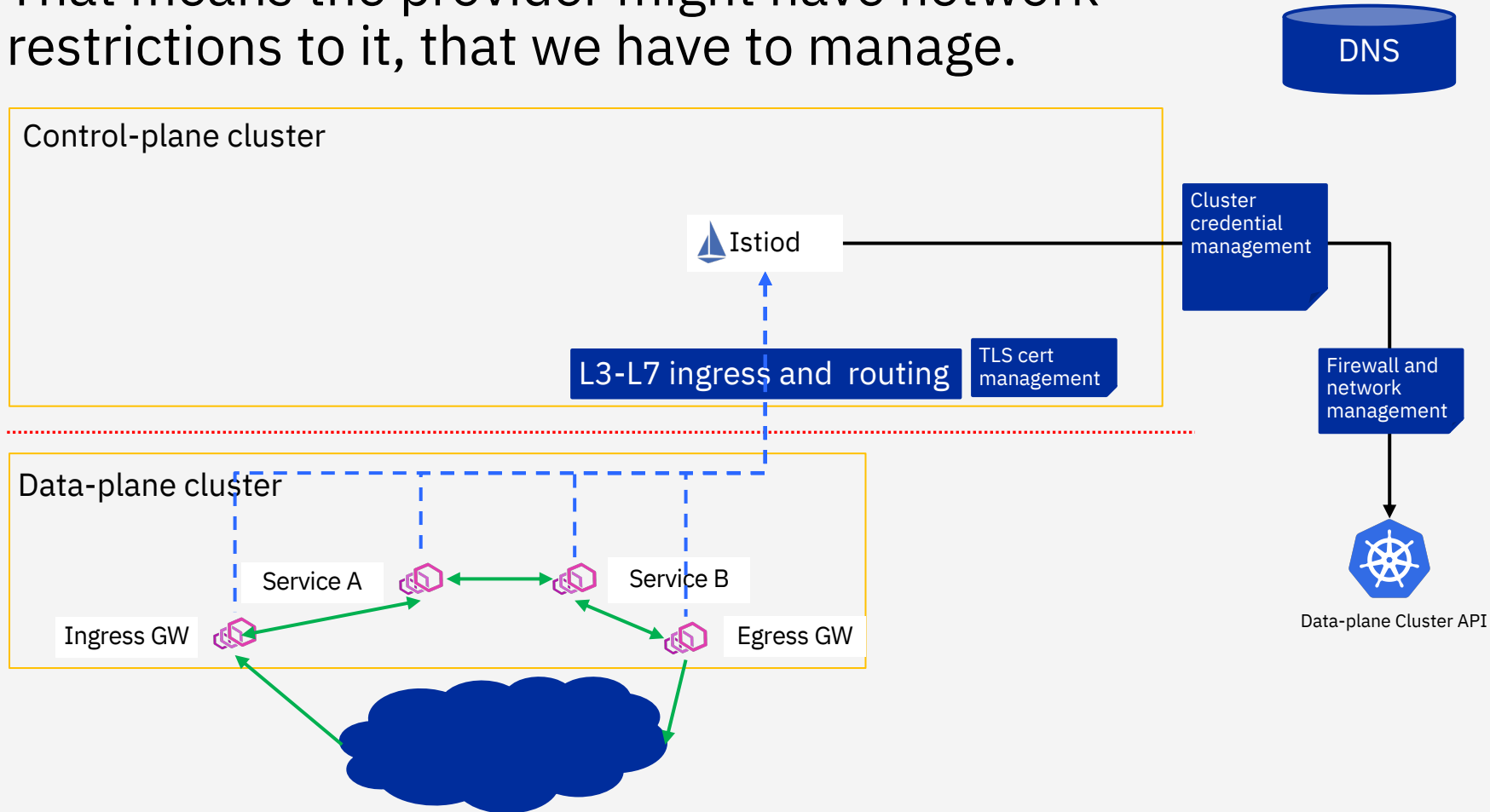
That means the provider might have network restrictions to it, that we have to manage.



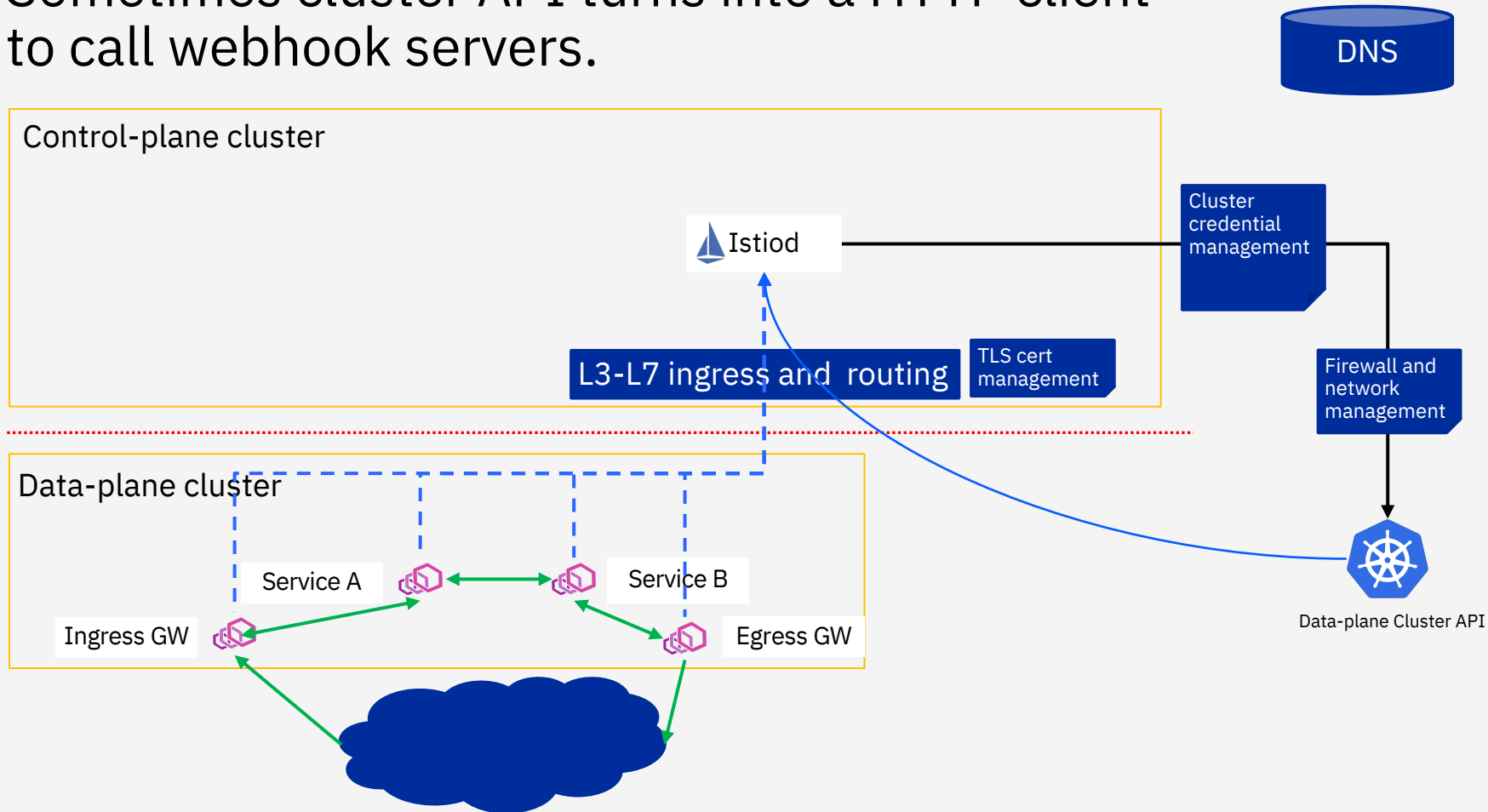
Life of an API request



That means the provider might have network restrictions to it, that we have to manage.

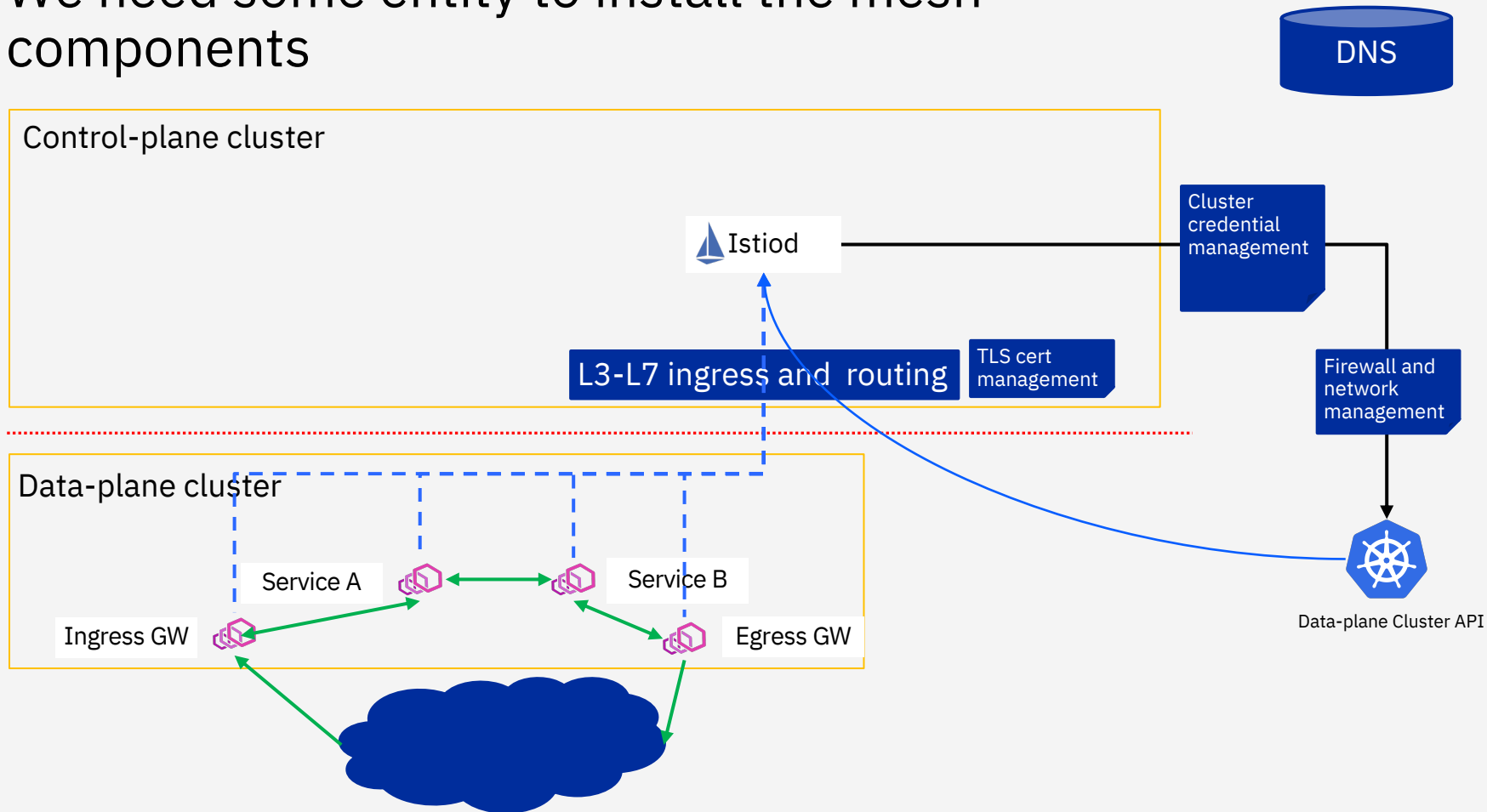


Sometimes cluster API turns into a HTTP client to call webhook servers.

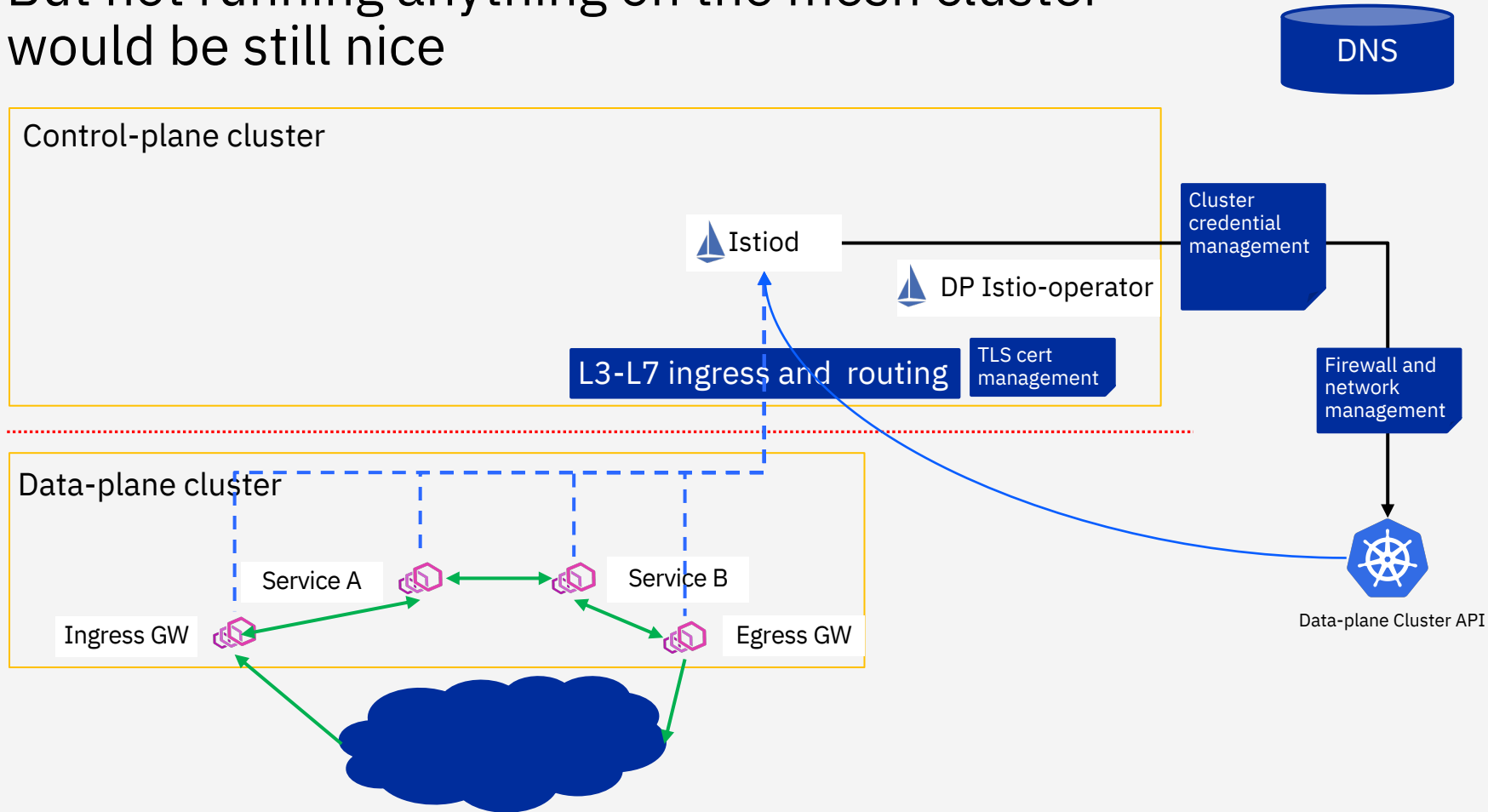


Mesh lifecycle management.

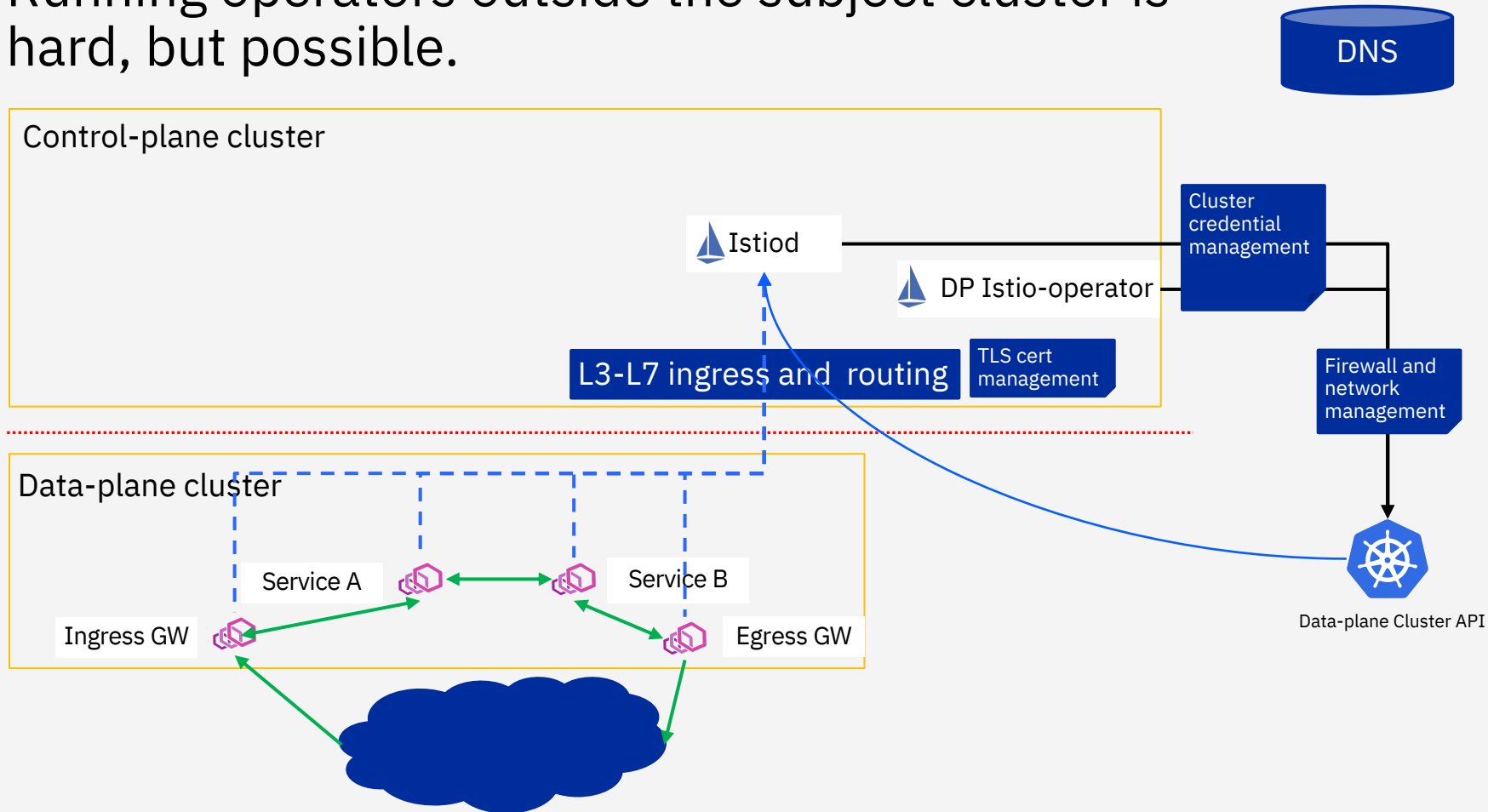
We need some entity to install the mesh components



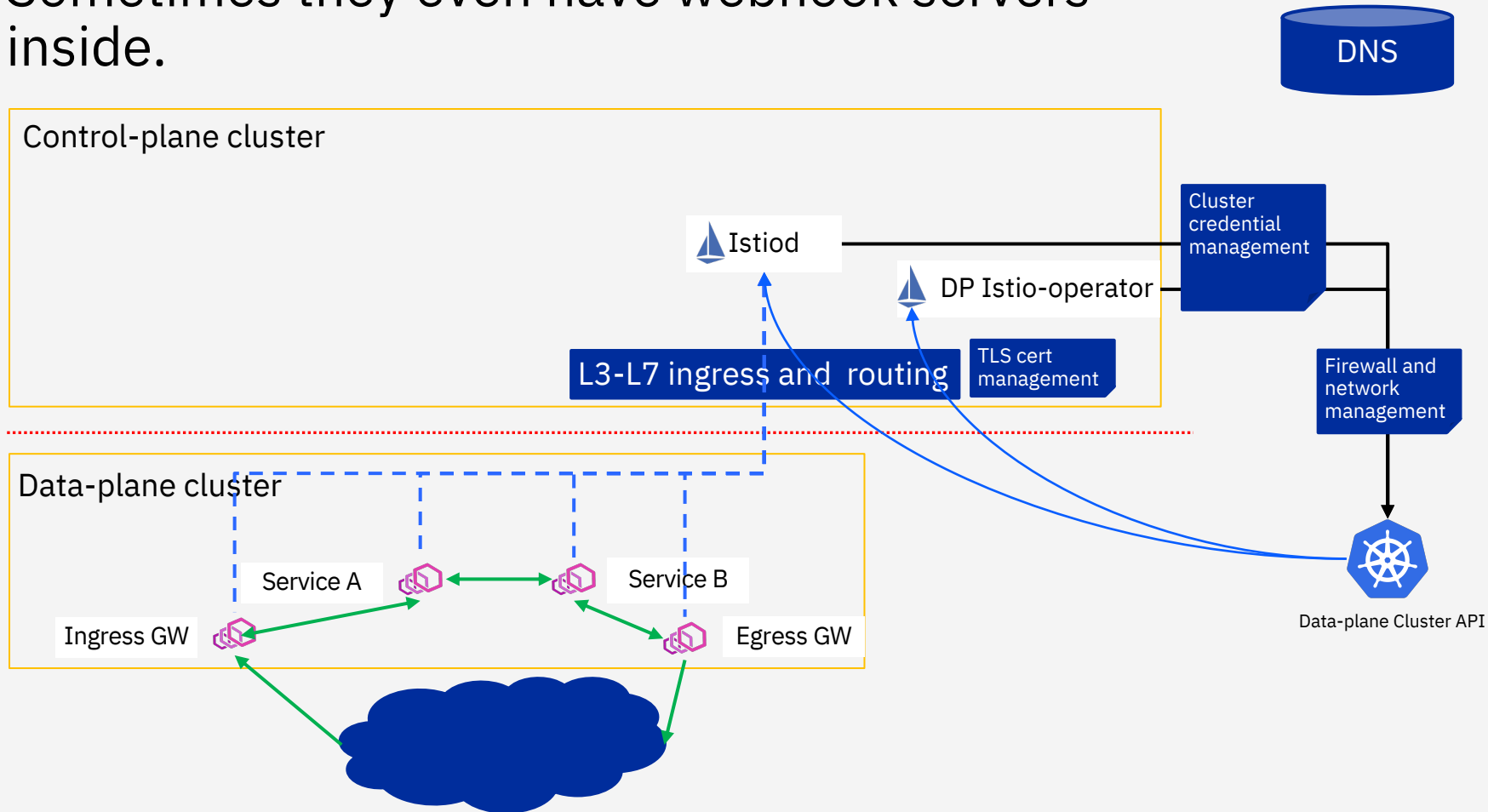
But not running anything on the mesh cluster would be still nice



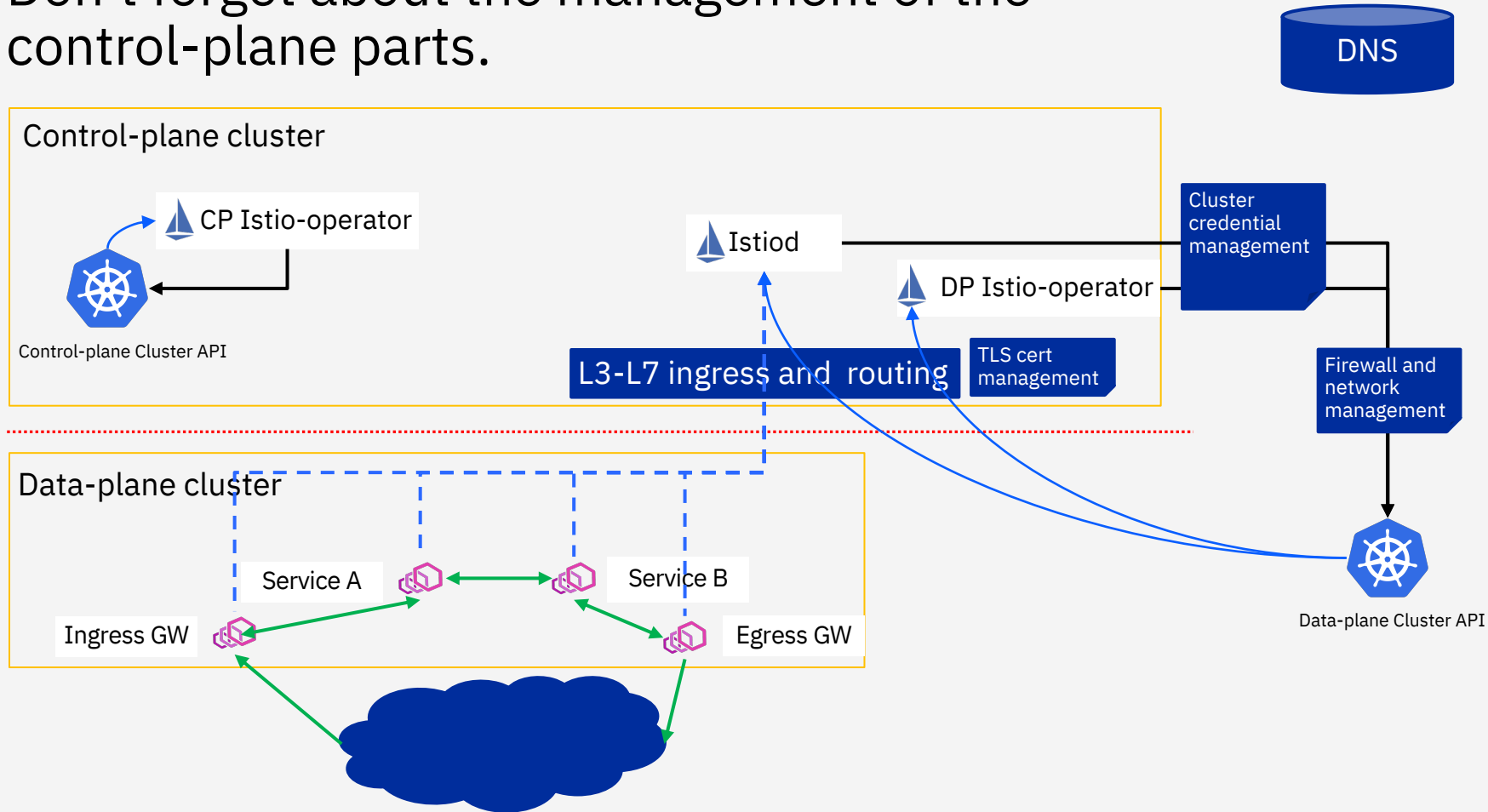
Running operators outside the subject cluster is hard, but possible.



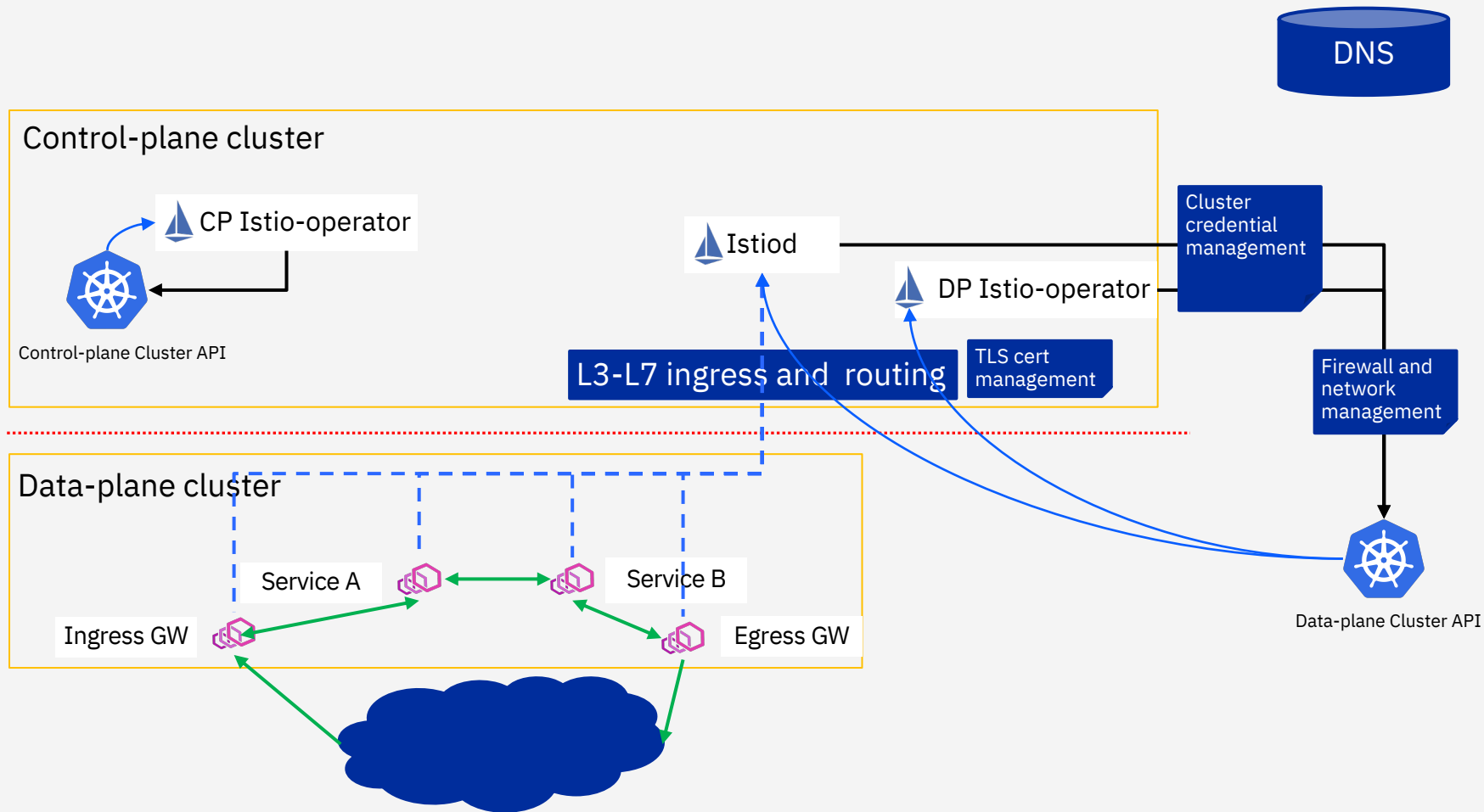
Sometimes they even have webhook servers inside.



Don't forget about the management of the control-plane parts.



**Adding some
cool extras.**





1.23.0-dev-9141f2

Search docs

About the documentation

Introduction

Getting Started

Configuration reference

Operations and administration

Extending Envoy for custom use cases

API

Supported API versions

v3 API reference

Bootstrap

Bootstrap

Stats

Metrics service

Overload Manager

Rate limit service

Internal Listener

Vcl Socket Interface configuration

» [API](#) » [v3 API reference](#) » [Bootstrap](#) » Metrics service

Metrics service

This documentation is for the Envoy v3 API.

As of Envoy v1.18 the v2 API has been removed and is no longer supported.

If you are upgrading from v2 API config you may wish to view the v2 API documentation

[config/metrics/v2/metrics_service.proto](#)

config.metrics.v3.MetricsServiceConfig

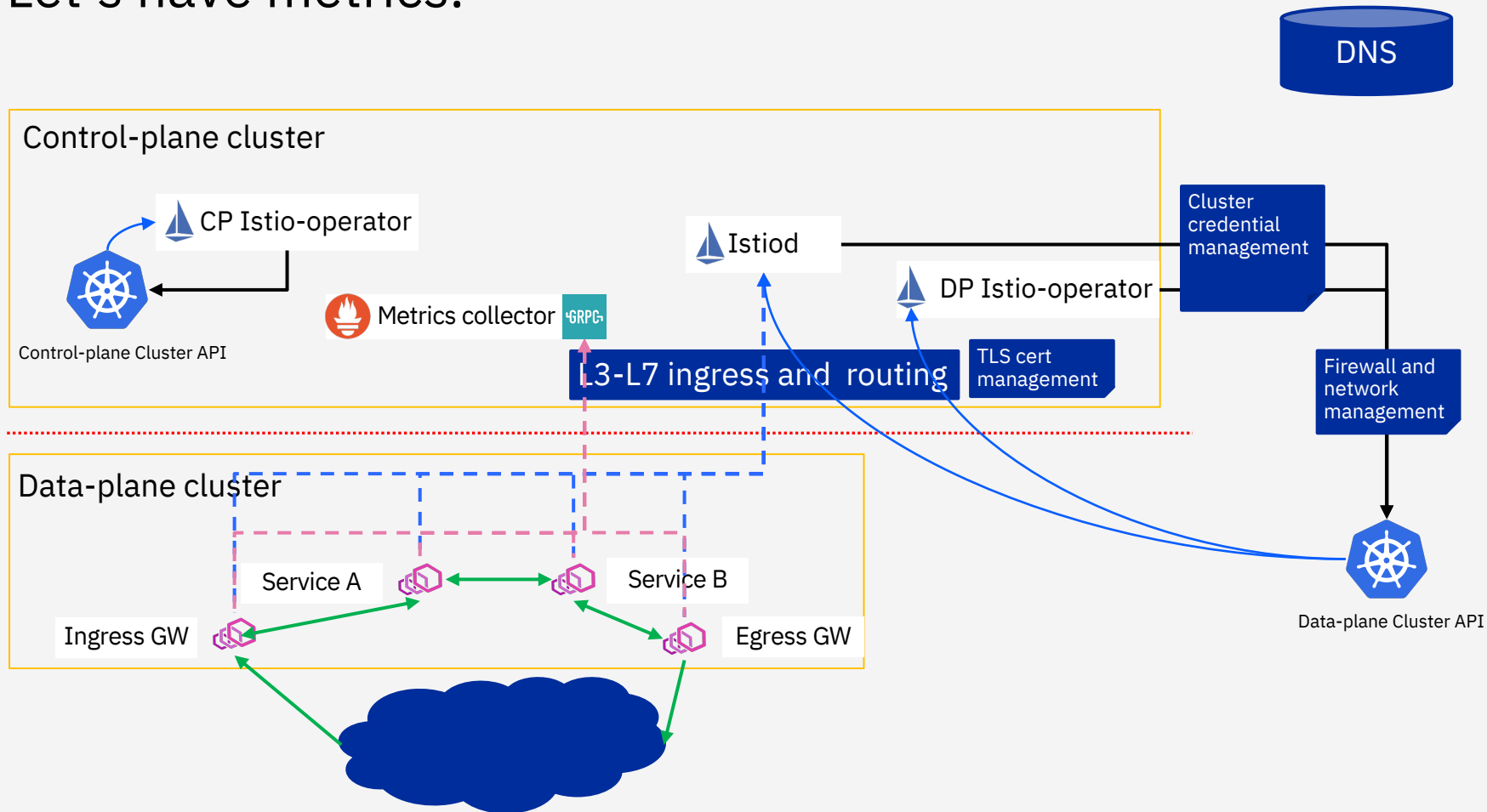
[\[config.metrics.v3.MetricsServiceConfig proto\]](#)

Metrics Service is configured as a built-in `envoy.stat_sinks.metrics_service` [StatsSink](#). This create Metrics Service.

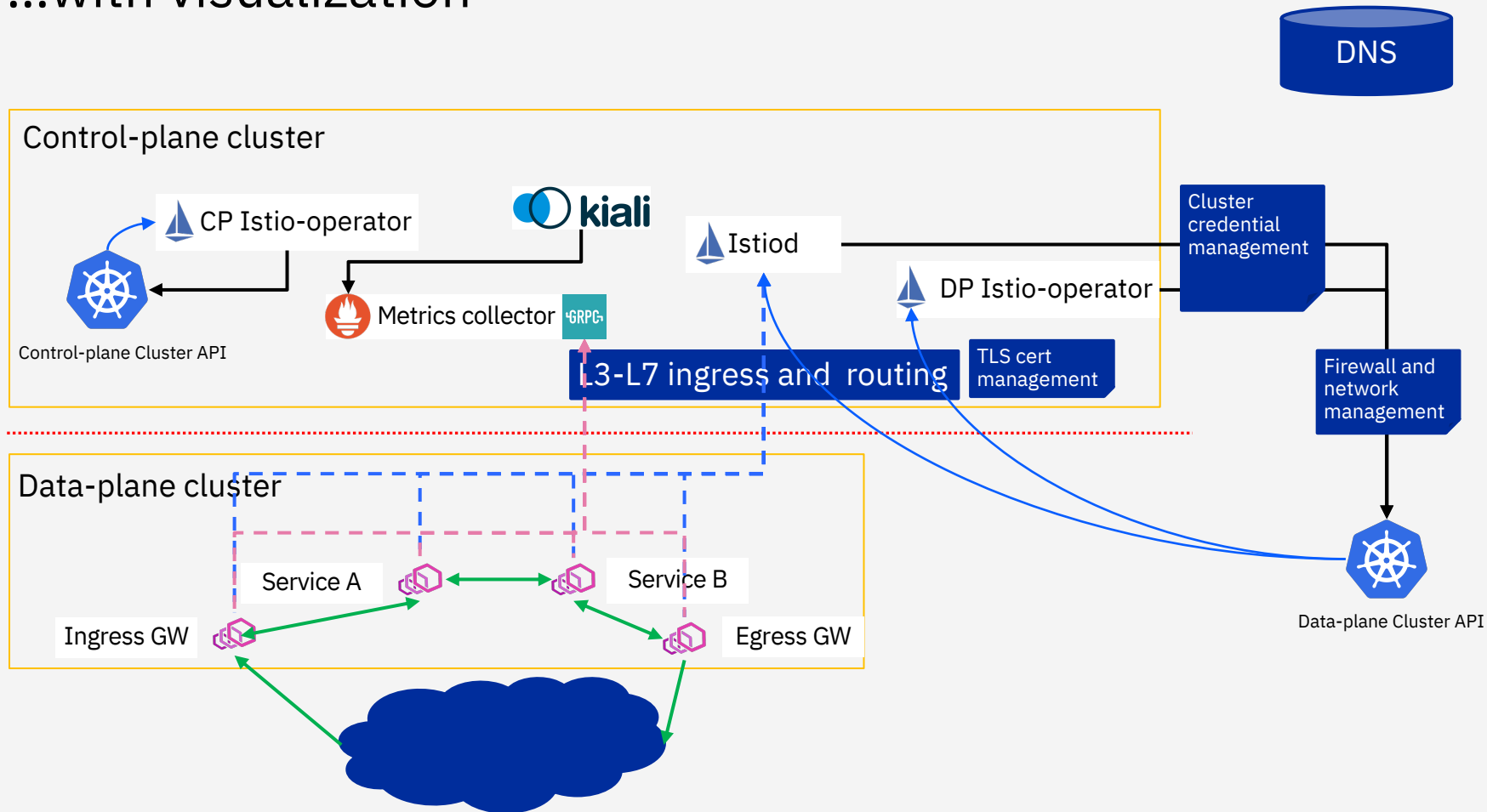
Example:

```
stats_sinks:
- name: envoy.stat_sinks.metrics_service
  typed_config:
    "@type": type.googleapis.com/envoy.config.metrics.v3.MetricsServiceConfig
    transport_api_version: V3
```

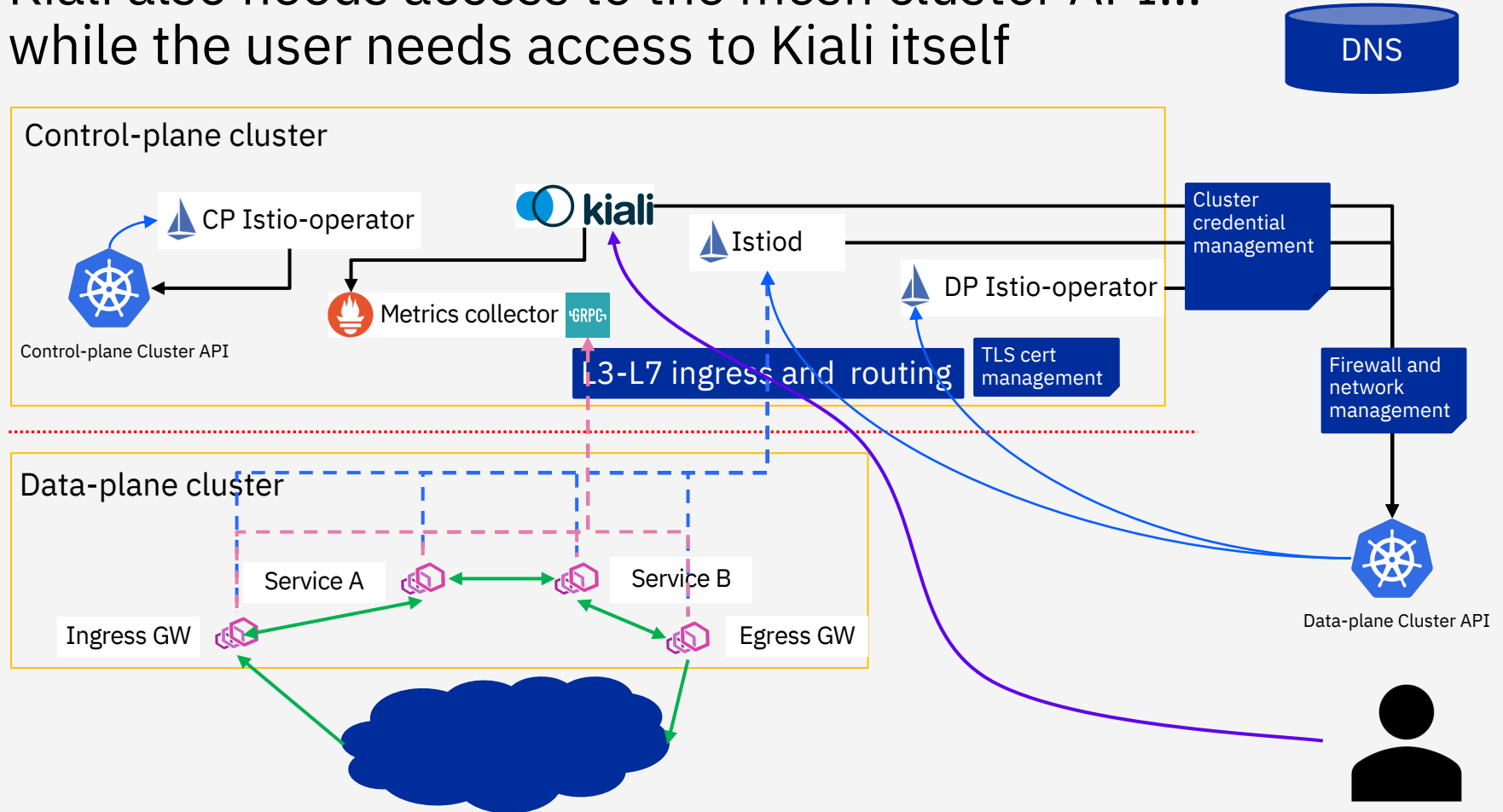
Let's have metrics.



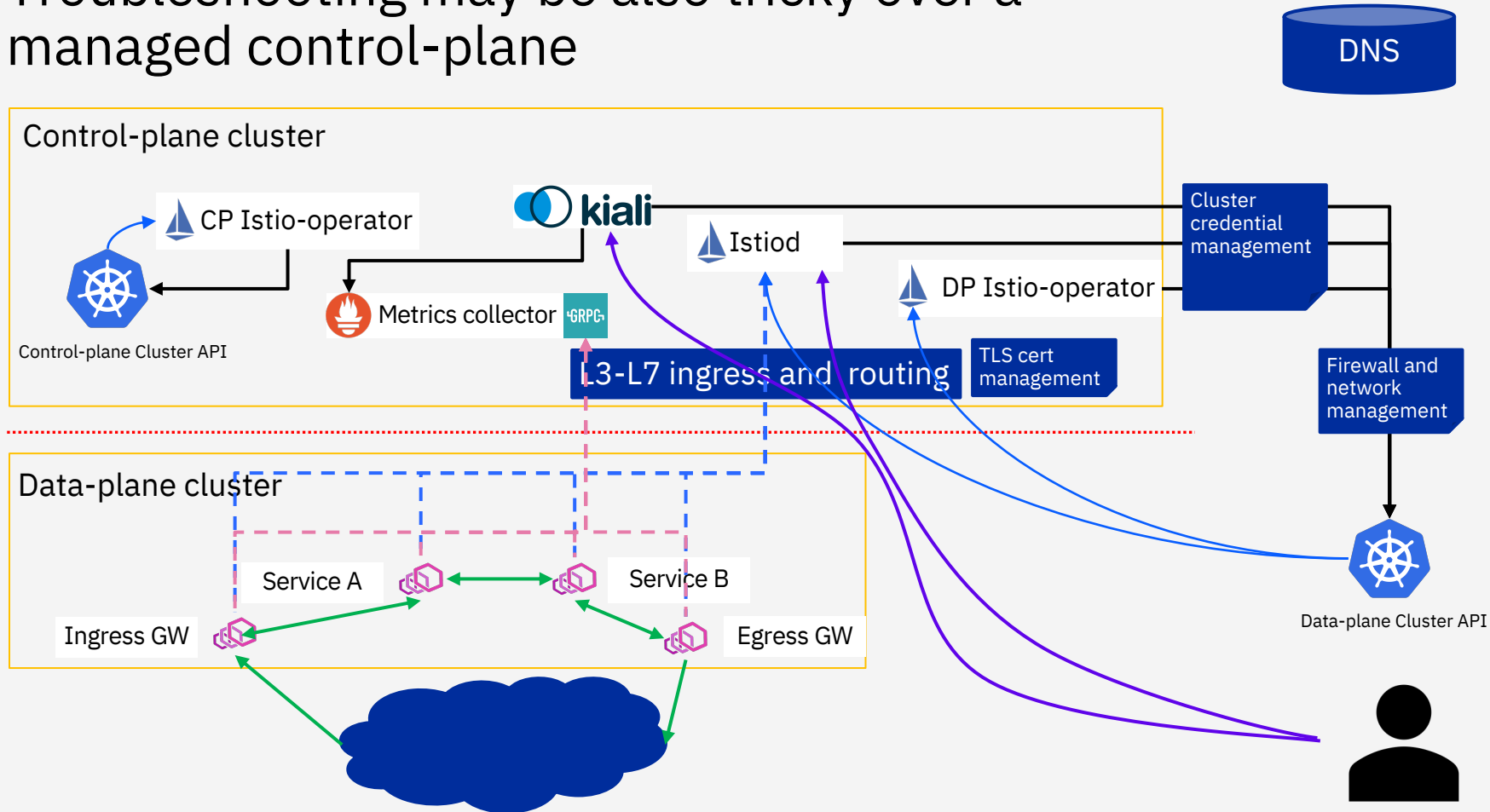
...with visualization



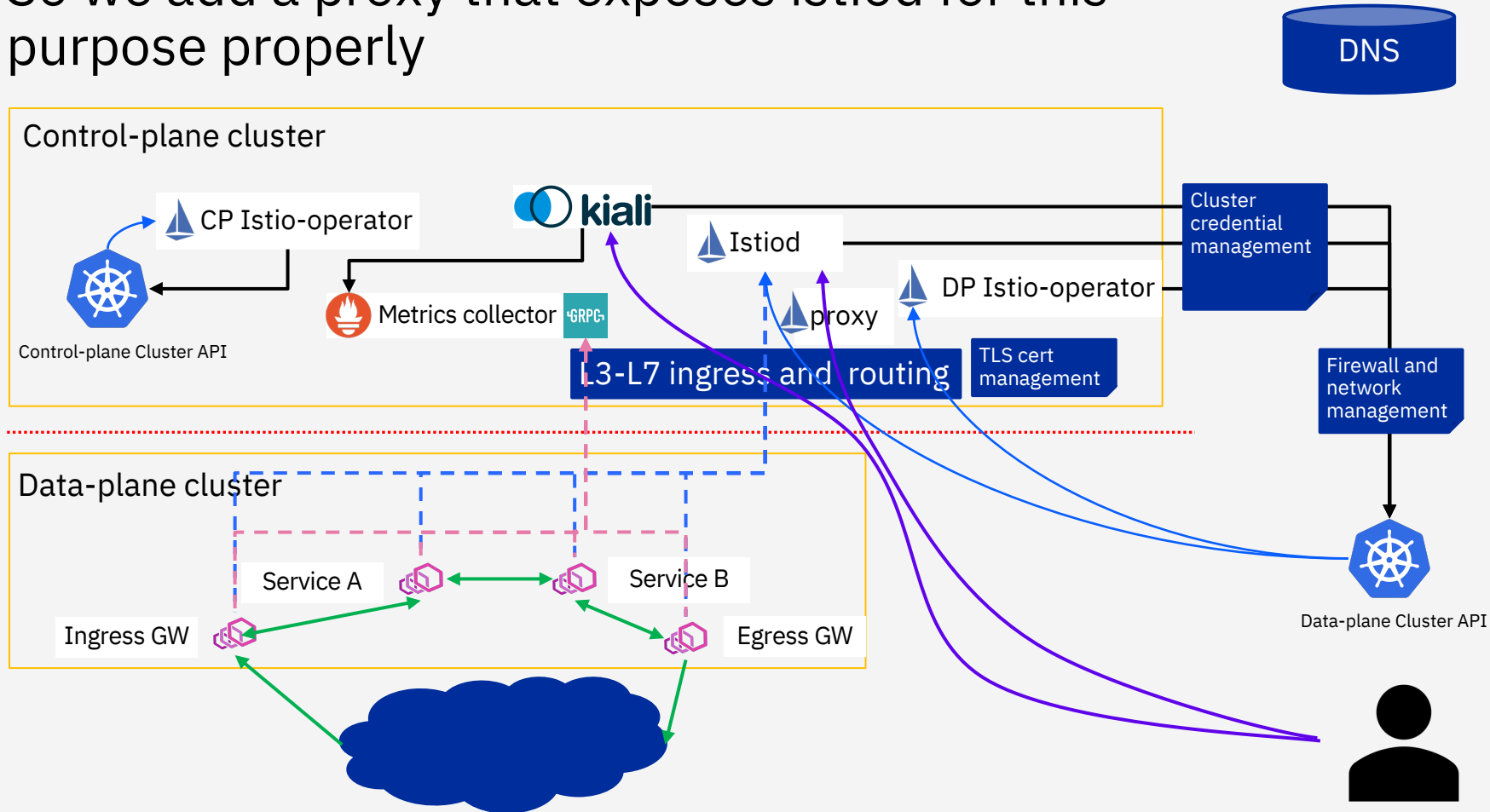
Kiali also needs access to the mesh cluster API...
while the user needs access to Kiali itself



Troubleshooting may be also tricky over a managed control-plane



So we add a proxy that exposes istiod for this purpose properly



...and then...

...you just need to implement a master controller to orchestrate everything on-demand.

Kiali challenges in Istio external control plane

The issue:

Since Kiali instance can only be configured to access one Istiod instance and also use the credential to watch workload namespaces, when deals with Istiod running in one cluster, and workload running in another, Kiali needs a way to have access to more than one cluster, currently it can only take one kubeConfig.

The workaround:

A not so nice workaround is to have two instances of Kiali, one should be configured to manage Istiod, one should be configured to watch the workloads, when more remote clusters added into the mix, to be able to see workloads in the remote clusters will require more Kiali instances. This leaves the Kiali instance points to config or remote cluster without Istiod access.

The wish:

Design Kiali so that all the workload info coming from metrics. So that metrics aggregated by Prometheus to draw various charts. Kiali can stay working with just one kubeConfig which points to a cluster in which Istiod will be running