

WHITEPAPER

CONSIDERATIONS FOR BUILDING A CONTROL PLANE FOR ENVOY PROXY

Manage Envoy Proxy at the edge, as
a gateway or in a service mesh





Envoy Proxy is an open source edge and service proxy designed for cloud-native applications. Created by Lyft, Envoy is hosted by the Cloud Native Computing Foundation (CNCF).

www.envoyproxy.io

Context

The rise of cloud native architectures have given rise to new networking components like Envoy Proxy as the universal data plane for distributed applications, favored for its speed and extensibility. Originally created internally at Lyft and then open sourced in September 2016, Envoy Proxy is considered the universal data plane API and being rapidly adopted by the community and ecosystem. However, the L7 application networking layer requires a control plane to manage and configure the behavior on the data plane. The options for control planes for organizations looking to adopt Envoy Proxy range from building their own or choosing from available open source or commercial software options.

This paper will cover key areas for consideration when building or evaluating a control plane for Envoy Proxy, including:

- 03** Consideration 1: Dynamic Configuration Updates
- 04** Consideration 2: Control Plane Components
- 05** Consideration 3: Use Case Configurations
- 06** Consideration 4: Extensibility and Pluggability
- 06** Consideration 5: Deployment Options
- 07** Getting Started: Build vs. Buy
- 07** Gloo as Control Plane



Consideration 1: Dynamic Configuration Updates

When operating a highly dynamic and ephemeral application environment like Kubernetes or Docker, the configurations for managing the data plane of these applications need to be served dynamically as well. How this translates into the level of dynamism that is most appropriate for your organization is a balance between the existing constraints and workflows of your environment and the desired state for the new applications.

The Envoy Proxy data plane xDS API allows the control plane to dynamically configure and update most of the runtime settings including; routing, discovery, and more, that organizations can select to be statically or dynamically updated.

The following runtime settings are available for dynamic configuration through xDS:

- [Listeners Discovery Service API](#): LDS to publish ports on which to listen for traffic
- [Endpoints Discovery Service API](#): EDS for service discovery
- [Routes Discovery Service API](#): RDS for traffic routing decisions
- [Clusters Discovery Service](#): CDS for backend services to which we can route traffic
- [Secrets Discovery Service](#): SDS for distributing secrets (certificates and keys)

Using the xDS API, the control plane makes the updates available to the data plane and each proxy is able to apply these updates independently. Proxies do not share information with each other and there is no expectation of atomic updates. Said another way, these configuration updates are eventually consistent. Workflows and processes can then be designed to both statically configure certain settings while xDS can dynamically discover and update the other settings at runtime. Since the configurations in these APIs are updated independent from one another, there is a possibility for race conditions (eg, RDS updates a route that references a cluster before CDS has updated the list of clusters) so a good option could be to use the “Aggregated Discovery Service” (ADS) API which orders the updates across all xDS APIs.

Consideration 2: Control Plane Components

Every organization has a different operating environment including infrastructure, systems, processes, applications and the control plane components needed will also vary to match their environment.

Depending on the implementation, different components are needed while some operational components apply across all implementations.

Architecturally, the control plane should be built as a set of loosely collaborating microservices to provide more flexibility in adding or removing functionality and to ease maintenance and upgrade cycles.

Depending on your environment, you may choose a static Envoy Proxy, fully dynamic xDS configuration, or some hybrid of the two. Control plane components to consider across this spectrum include:

Static Envoy Proxy File	Dynamic xDS	Operational Components for All Environments
<ul style="list-style-type: none">• Template engine• Data store or VCS for values in the templates• Service or application specific configurations• Orchestrator to put the pieces together• Delivery vehicle to Envoy Proxy• Triggers to reload or hot-restart configuration files	<ul style="list-style-type: none">• Service interface and implementation of xDS• Handler to register and deregister services into the service registry• A service registry• An abstract object model to describe the Envoy configuration• Configuration data store	<ul style="list-style-type: none">• Certificate or CA store• Statistics collection engine• Distributed tracing backend or engine• External authentication• Rate limiting services

Consideration 3: Use Case Configurations

Consider creating a domain-specific configuration based on your use cases. Envoy Proxy configuration can be complicated and it is best to simplify it for your users with a custom configuration API. Defining who your end users are and how they will interact with the control plane will influence the configuration objects used to configure Envoy Proxy. Consider the use cases for north/south and east/west traffic through these examples:

Possible Roles for Envoy Proxy	Example of Istio Service Mesh	Example of VMware Heptio Contour
<ul style="list-style-type: none">• API Gateway• Kubernetes Edge Load Balancer• Kubernetes Reverse Proxy• Kubernetes Ingress Control• Shared Services Proxy• Per Service Sidecar	<ul style="list-style-type: none">• Gateway: Define a shared proxy component to specify protocol, TLS, port, and host authority to load balance and route traffic• Virtual Service: How to interact with a specific service for route matching behavior, timeouts, retries, etc.• Destination Rule: How to interact with a service for circuit breaking, load balancing, mTLS policy, subsets definitions of a service, etc.• Service Entry: Adds a service to Istio registry	<ul style="list-style-type: none">• Ingress Route: A Kubernetes CRD that provides a single location to specify configuration for the Contour proxy• Ingress Resource Support: Allows you to specify annotations on your Kubernetes Ingress resource

Establishing domain-specific configuration objects and API customizes the control plane for the end users and use cases and improves the workflow for operating Envoy Proxy in your organization.

Consideration 4: Extensibility and Pluggability

Depending on the environment, there are different ways in which the Envoy Proxy and control plane architecture can be extended to have additional functionality added.

The control plane can be designed to be as simple or as complicated as you need. Focus on a simple core to the control plane and extend through plugins and microservices controllers to quickly add or remove features or support additional environments as needed. The control plane engine can be built to be pluggable to add new Envoy Proxy features or extend end user facing domain specific configuration objects to take advantage of these new capabilities. Following the principles of loosely coupled microservices, augment existing controllers to do this or add new ones.

Consideration 5: Deployment Options

How the control plane and its supporting components are deployed is just as important as how it is designed in consideration of the security, scalability and usability needs of your specific environment. Although the control plane and data plane components can be co-deployed, completely separated or partially co-located, the recommendation is to keep them separate for these reasons:

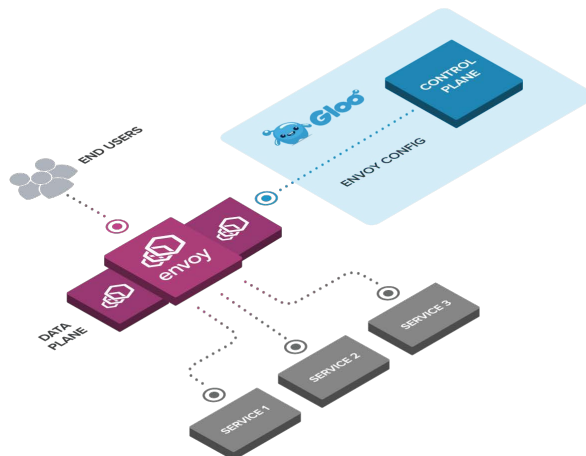
- **Security:** Reduce the risk to the control plane if the data plane is compromised by keeping the deployment environments separate. Additionally, it is a best practice to keep a control plane that deals with the distribution of keys, certificates, or other secrets be separate from the data plane.
- **Scaling:** The data plane and control plane will likely scale differently and keeping them separate eliminates any potential choke points and dependencies.
- **Grouping:** The data plane consists of different roles and responsibilities so keeping the control plane separate makes it easier to keep data and configuration separate.
- **Resource Usage:** Separation of data and control plane allows for the ability to assign or throttle resource usage of the components with fine-grained resource pool options and avoids potentially hard to diagnose latencies.
- **Deployment and Lifecycle:** Allow for independent patch and upgrade cycles of the control plane and data plane.
- **Storage:** Ability to configure storage independent of the data plane or control plane components.

Getting Started: Build vs. Buy

Building and maintaining a custom-built control plane is possible with Envoy Proxy but does require a thorough understanding of what is needed, the workflows, and deployment. Early on the only choice available was to deploy a custom homegrown solution, but with the rising popularity of Envoy Proxy, many viable control plane options have emerged in both the open source and commercial ecosystem. Organizations can direct their effort to evaluating the technical capabilities and software support options available to address the needs of their application environment.

Gloo as Control Plane

Gloo, by Solo.io, is a next generation API gateway, Kubernetes Ingress Controller and control plane for Envoy Proxy. Gloo connects, secures and controls application traffic through APIs for legacy monoliths, microservices and serverless functions on any infrastructure.



Control Plane Considerations with Gloo

1. Dynamic Configuration Updates in Gloo

Gloo implements the xDS APIs to serve the dynamic configuration of Envoy Proxy. The control plane API leverages gRPC streaming calls and stubs out the API to fill it with an implementation.

Using gRPC streaming API for the dynamic configurations is ideal because:

- Event-driven configuration updates pushes to Envoy Proxy when it becomes available
- Eliminate the need to poll for changes
- Eliminate the need to hot-reload Envoy Proxy
- Eliminate disruption to traffic

2. Gloo Control Plane Components

Gloo implements powerful discovery capabilities and a semantic understanding of a function to serve the Envoy Proxy configuration as a set of loosely coordinated components. When deployed with Kubernetes, Gloo configurations are represented by [Custom Resource Definitions \(CRDs\)](#). All user facing configurations and core configurations drive the xDS endpoints are CRDs. When not using Kubernetes, Gloo uses HashiCorp Consul and Vault to store configurations and secrets. Gloo has the following components for the control plane:

- **Gloo** is an event driven component responsible for generating configuration for and serving the core xDS services and configuration of custom Envoy filters
- **Discovery** works with service discovery services (Consul, Kubernetes, etc) to discover and advertise upstream clusters, endpoints and also discovers REST endpoints (using swagger), gRPC functions (based on gRPC reflection), and cloud functions (AWS Lambda, Azure, Google Cloud).
- **Gateway** allows users to use a more comfortable object model to configure an Envoy Proxy based on its role (edge gateway, shared proxy, Knative cluster ingress, etc) and generates the configuration that the Gloo component uses to generate Envoy Proxy configuration through xDS.

3. Use Case Configurations

The Gloo configuration objects are split into two levels of configuration to allow for extending the control plane capabilities while keeping the abstraction simple. The two levels include user facing configuration for best ergonomics of use cases that leave options for extensibility and lower level configuration that abstracts Envoy Proxy but is not expressly intended for direct user manipulation.

Gloo is designed for teams owning their routing configurations since the semantics of the routing are heavily influenced by the developers of APIs and microservices. The user facing API objects in Gloo drive the lower level objects used to derive the Envoy Proxy xDS configurations.

User Facing API Objects	Lower Level Core API Objects
<p>Gateway specify routes and API endpoints available at a specific listener port and what security accompanies each API</p> <p>VirtualService groups API routes into a set of “virtual APIs” that route to backed functions (gRPC, http/1, http/2, lambda, etc) and gives developers control over how a route proceeds with different transformations to decouple the front end API from the backend</p>	<p>Upstream captures the details about backend clusters, exposed and can understand the actual service functions available at a specific endpoint</p> <p>Proxy abstracts all of the configurations applied to Envoy Proxy-including listeners, virtual hosts, routes, and upstreams</p>

4. Gloo Extensibility and Pluggability

Gloo takes advantage of the versatility and innovation rate of Envoy Proxy to make an extensible control plane. The focus is on a simple control plane core and then extend it through plugins and microservices controllers through composability.

Gloo implements this on the following levels:

- Opinionated domain-specific configuration objects on top of a core GI configuration object
- Control plane *plugins* to augment the existing behavior of control plane
- Tools that expedite the previous two points

Additionally, the following Envoy Proxy filters are part of the Gloo;

- Authentication (integrated and as plugin)
- Rate Limiting
- [Squash](#) debugger
- Caching
- Request/Response Transformation
- NATS streaming
- AWS Lambda
- Google Cloud Functions
- Azure functions
- Web Application Firewall (a custom filter for Envoy Proxy)
- WebAssembly for building custom extensions

The extensibility of Gloo does not begin and end with the API gateway use case. Microservices has given rise to new application networking architectures, namely service mesh, to better handle the service to service communications for dynamic distributed applications. Service meshes use the same proxy as sidecars to every service in the application. Gloo provides seamless integration from the edge proxy as gateway to the sidecar in service mesh for enhanced traffic control, security and application observability.

5. Deployment Options

For Kubernetes environments, the Gloo control plane is deployed as [Custom Resource Definitions \(CRDs\)](#) and requires no additional database or storage prerequisites. Gloo also supports non-Kubernetes environments using HashiCorp Consul as the configuration store. Gloo can be deployed onto any infrastructure including on-premises or in the cloud via self hosted or managed Kubernetes / HashiCorp services.

Gloo deploys its control plane separately from the data plane to allow for independent considerations for scaling, security, an allocation. Because of this separation, Gloo can scale to very large and dense clusters as well as can be updated independently from the Envoy Proxies in the data plane.




Gloo Open Source and Enterprise

Gloo is available in Open Source and Enterprise editions to connect, secure and control incoming application traffic for all applications on any infrastructure and provides a stepping stone to service mesh. Gloo Enterprise provides expanded security and management features out-of-the-box, and enterprise support for mission critical application environments.



Learn more about Gloo:





- Visit the Gloo website solo.io/gloo
- Try the Gloo tutorial katacoda.com/solo-io
- Download Gloo open source docs.solo.io/gloo/latest/
- Start a free 30 day Enterprise trial solo.io/gloo-trial

CONNECT 	SECURE 	CONTROL 
<ul style="list-style-type: none"> All Workload Types Auto Service Discovery HTTP Routing TCP Proxy gRPC Web CORS Kubernetes Services Consul Services Serverless Functions Configuration Validation Request / Response Transformation Service Mesh integration 	<ul style="list-style-type: none"> TLS Hashicorp Vault Secrets Let's Encrypt Custom Authentication (DIY) 	<ul style="list-style-type: none"> Admin Dashboard (Read Only) Role Delegation Access Logging & Usage Stats Prometheus and Grafana Tracing Circuit Breaking Retries Timeouts Traffic Shifting Traffic Shadowing Rate Limiting (DIY)
	ENTERPRISE <ul style="list-style-type: none"> Data Loss Prevention Web App Firewall (WAF) Basic Authentication API Key JSON Web Token (JWT) LDAP Support OAuth / OIDC Open Policy Agent 	ENTERPRISE <ul style="list-style-type: none"> Admin Dashboard (Full Access) Advanced Rate Limiting

About Solo.io

Solo.io connects the world's applications with APIs and service mesh across any infrastructure. Our mission is to deliver innovative products to simplify the journey to a cloud native future with the flexibility and control to digitally transform at the pace of your business without disruption.



 solo.io
 @soloio_inc
 slack.solo.io
 contact@solo.io

About Solo.io

Solo.io connects the world's applications with APIs and service mesh across any infrastructure. Our mission is to deliver innovative products to simplify the journey to a cloud native future with the flexibility and control to digitally transform at the pace of your business without disruption.