

# SERVICE MESH AT SCALE

Challenges and approaches to  
multi-cluster deployment patterns



# Table of Contents

- 4** What is Service Mesh?
- 5** Going Beyond a Single Cluster
- 6** Considerations for Multi-Cluster Service Mesh Patterns
- 7** Gloo Mesh for Service Mesh Management
- 8** How it works: Gloo Mesh Editions and Extensions
- 9** Gloo Mesh Gateway
- 10** Gloo Mesh Core
- 11** Gloo Mesh Extensions
- 12** Gloo Portal



# Introduction

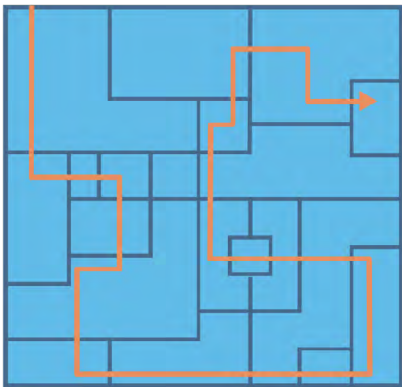
Application modernization initiatives typically include adoption of cloud, Kubernetes-orchestrated containers, and distributed microservices. These initiatives are driven by the need for greater agility in software innovation to delight customers and streamline business operations. Compared to monolithic applications, microservices allow organizations to make changes to a portion of the application quickly and without impacting the entire application. This architecture also allows developers greater freedom in building individual services in different languages, so they can use the right technology for the job without worry of compatibility issues.

With this transformation of the application architecture comes new challenges including how to:

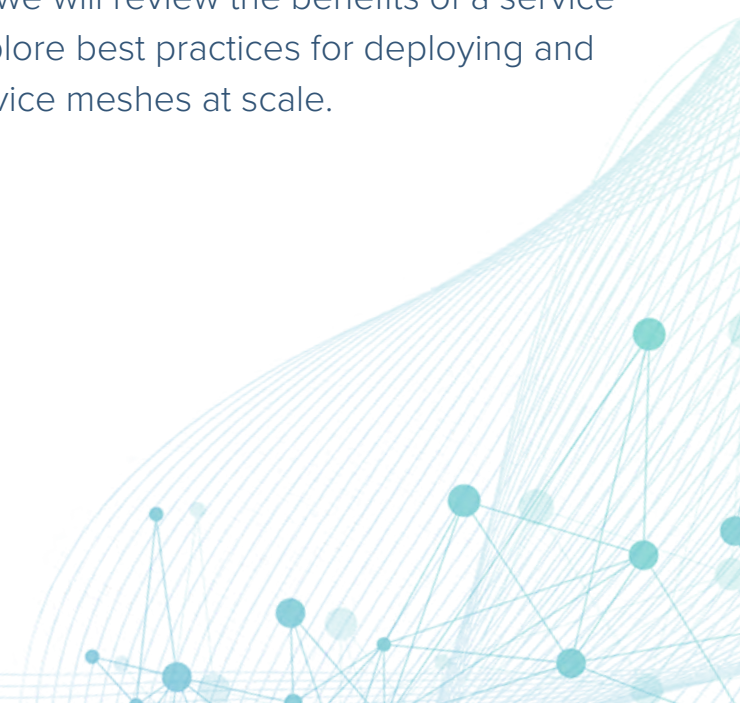
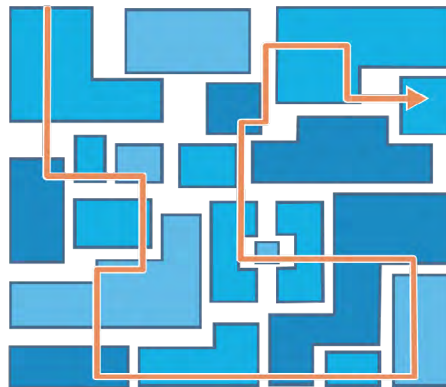
- Manage service-to-service communication
- Share and secure APIs
- Get observability into the application network

Then these need to be solved for multi-cluster and multi-platform deployment patterns across clouds and on-premises for scale and application reliability. In this eBook we will review the benefits of a service mesh and explore best practices for deploying and operating service meshes at scale.

MONOLITH



MICROSERVICES

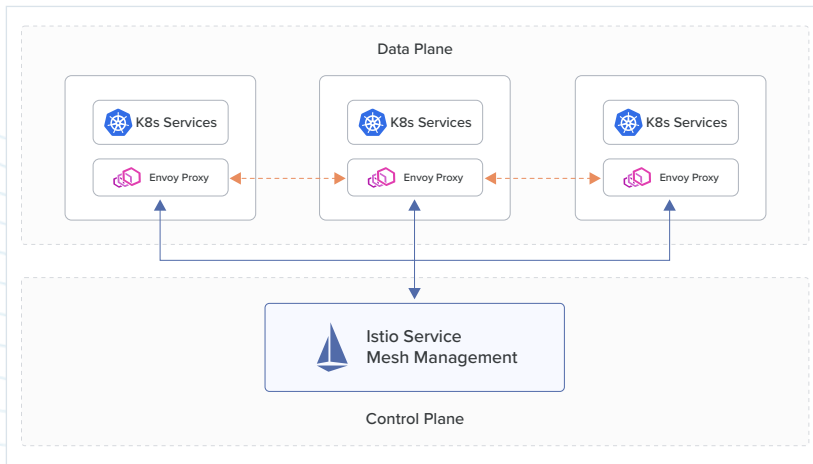


# What is Service Mesh?

A service mesh is an infrastructure layer that abstracts application networking from the business logic of the application to provide a configurable network layer to facilitate communication between services using their APIs. This architecture is facilitated by deploying a proxy as a sidecar alongside each application service. All communications between the application services are facilitated through the sidecar proxies (data plane) that are configured and managed through a control plane.

The shift to microservices architecture creates a new challenge in solving the service-to-service communication to form an application. What was once a monolithic code base is now potentially hundreds of loosely coupled services that are dynamic, ephemeral, and distributed, making the network between them a critical part of the application infrastructure.

A service mesh solves major challenges with API through L7 networking to gain more insight and control of distributed application behavior. Service meshes provide essential capabilities to application developers, including service discovery, client-side load balancing, timeouts, retries, and circuit breaking. These capabilities work regardless of their application framework or language. For operators, service mesh provides a set of L7 controls over traffic routing, policy enforcement, identity (authentication/ authorization), and security (encryption, mTLS, WAF, DLP.) A service mesh is also an extension point to enable new application functionality delivered through the network with custom filters.



# Going Beyond a Single Cluster

Building application environments for resiliency, scale, security, and compliance can require multiple instances of the application deployed together, in different clusters, or across different regions.

Businesses with customers in many different countries are learning of new data sovereignty regulations that dictate where and how their applications are served from and customer data stored.

Most organizations have many application environments geographically dispersed to serve applications globally. The architectural choices between having a few large clusters and many small clusters have implications on infrastructure configuration, operations, development workflows, and collaboration.

A best practice for containerized microservices running in Kubernetes is to deploy many, smaller clusters to form more explicit boundaries than multi-tenancy can provide. While beneficial, multi-cluster Kubernetes and service mesh environments introduce new questions and challenges for organizations to consider when designing their architecture.

## REASONS FOR MULTI-CLUSTER

- |   |  |
|---|--|
| <ul style="list-style-type: none"><li>• Strong network isolation</li><li>• Regulatory compliance</li><li>• High availability</li><li>• Scalability</li><li>• Data Locality and Cost</li></ul> | <ul style="list-style-type: none"><li>• Configuration</li><li>• Service discovery</li><li>• Identity federation</li><li>• Security</li><li>• Shared / Separate / Federated</li></ul> |
|---|--|

# Considerations for Multi-Cluster Service Mesh Patterns

Multi-cluster service mesh environments raise new questions on how these distributed clusters of service will discover, route, and operate as a unified application delivering a seamless customer experience.



## CONFIGURATION AND ADMINISTRATION

Will the clusters be completely uniform from the services to network policies, completely different or somewhere in between? Who owns which aspects of the cluster, service mesh, and services?

*Federation provides flexibility in having some aspects of the configuration consistent and shared across all clusters while leaving other aspects of the configuration available for each team to customize as needed. Federation and delegation provide flexibility to separate out administrative domains by team or network.*

[Learn More](#)

## SERVICE DISCOVERY

Will the clusters be completely uniform from the services to network policies, completely different or somewhere in between? Who owns which aspects of the cluster, service mesh, and services?

*Service mesh clusters can be registered to an independent control plane for service discovery and master index of the services, clusters, networks, and make that information available without having to replicate the services across all clusters.*

[Learn More](#)

## IDENTITY FEDERATION

How will services in one network know which services from another network are allowed to communicate with each other?

*An independent control plane can unify the root identity between multiple service mesh clusters with a shared trust model for the registered clusters and the services that reside across them.*

[Learn More](#)

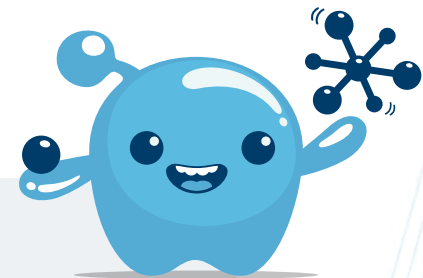
# Gloo Mesh for Service Mesh Management

Gloo Mesh is a service mesh management plane for configuring, routing, and operating single to multi-cluster and multi-platform environments with production and long-term support for upstream Istio software.

Gloo Mesh creates a “virtual mesh” of target service mesh clusters to establish shared-trust, cross-cluster service discovery, and identity federation to enable traffic and access control policies across the multi-cluster environment. Global policies for failover or locality-aware routing can also be configured to provide high availability and compliant services. The unique role-based API allows for the delegation of service mesh access and configuration ownership by different personas and teams enabling management efficiency at scale.

## GLOO MESH TUTORIALS

- [Compare Gloo Mesh and open source editions](#)
- [The operational overhead of Istio's external control plane](#)
- [Upgrading Istio without downtime](#)
- [How to configure an Istio service mesh with IPv6](#)



# HOW IT WORKS

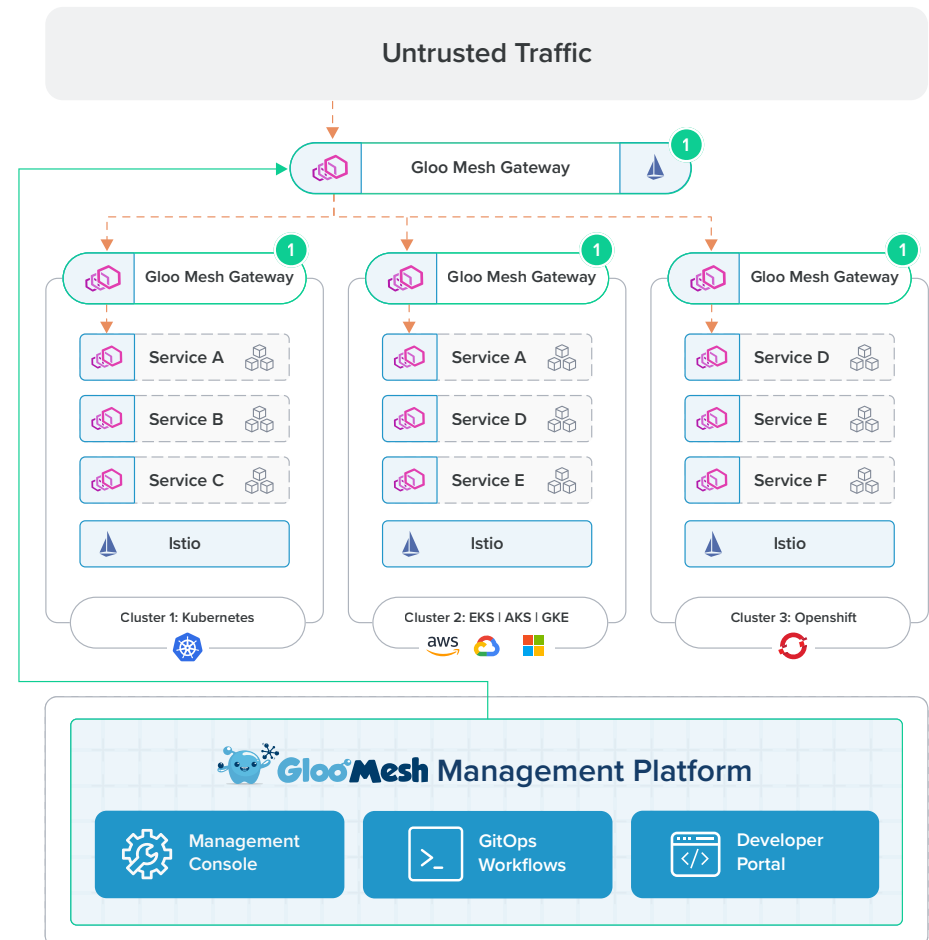
Gloo Mesh Editions and Extensions



# 1 Gloo Mesh Gateway

Istio-based north-south API gateway to govern and manage requests for services

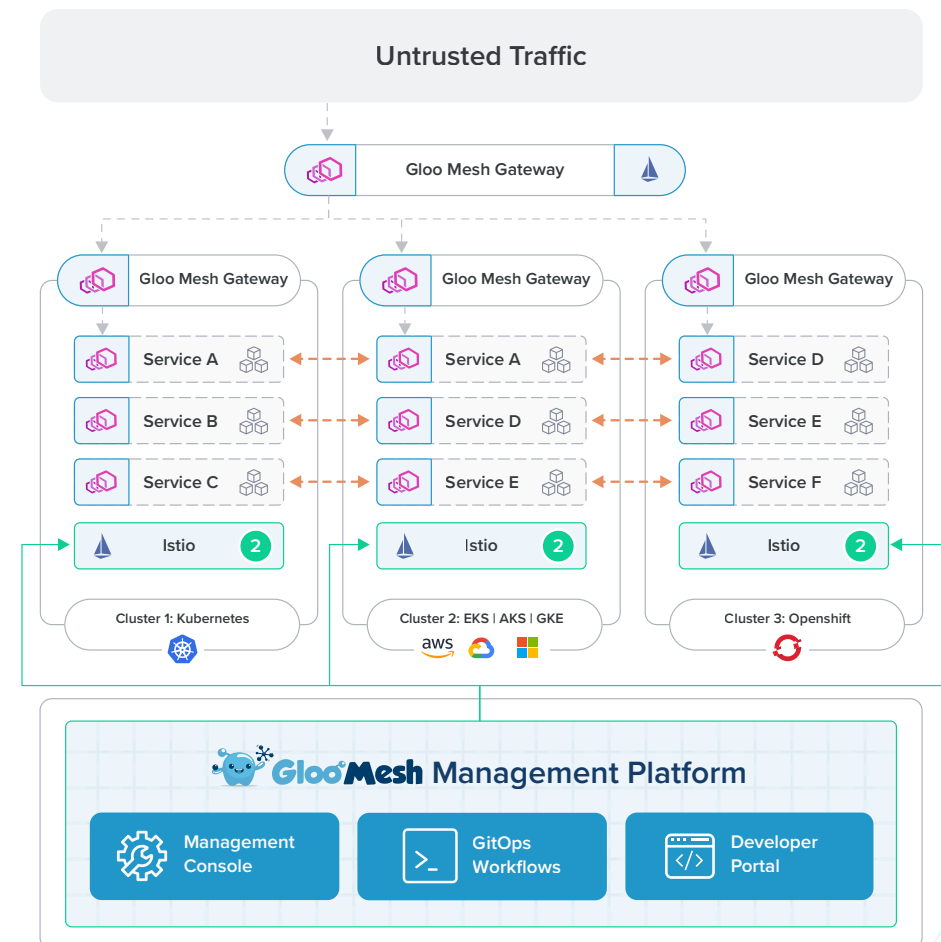
- Certificate management and rotation
- Integrate with Identity & Access Management systems to leverage existing security policies
- Enforce authentication, authorization, and encryption including mTLS
- Manage request routing, rate-limiting, load balancing, circuit breaking and failover traffic based on locality and affinity rules
- Protect against attacks with a built-in web application firewall (WAF)
- Guard against sensitive info breaches with data loss prevention (DLP)
- Collect metrics for observability, troubleshooting, and auditing with Prometheus and Grafana
- Transformations filter / SOAP



## 2 Gloo Mesh Core

Manage cluster ingress (and egress) for Kubernetes clusters, VMs, and legacy applications:

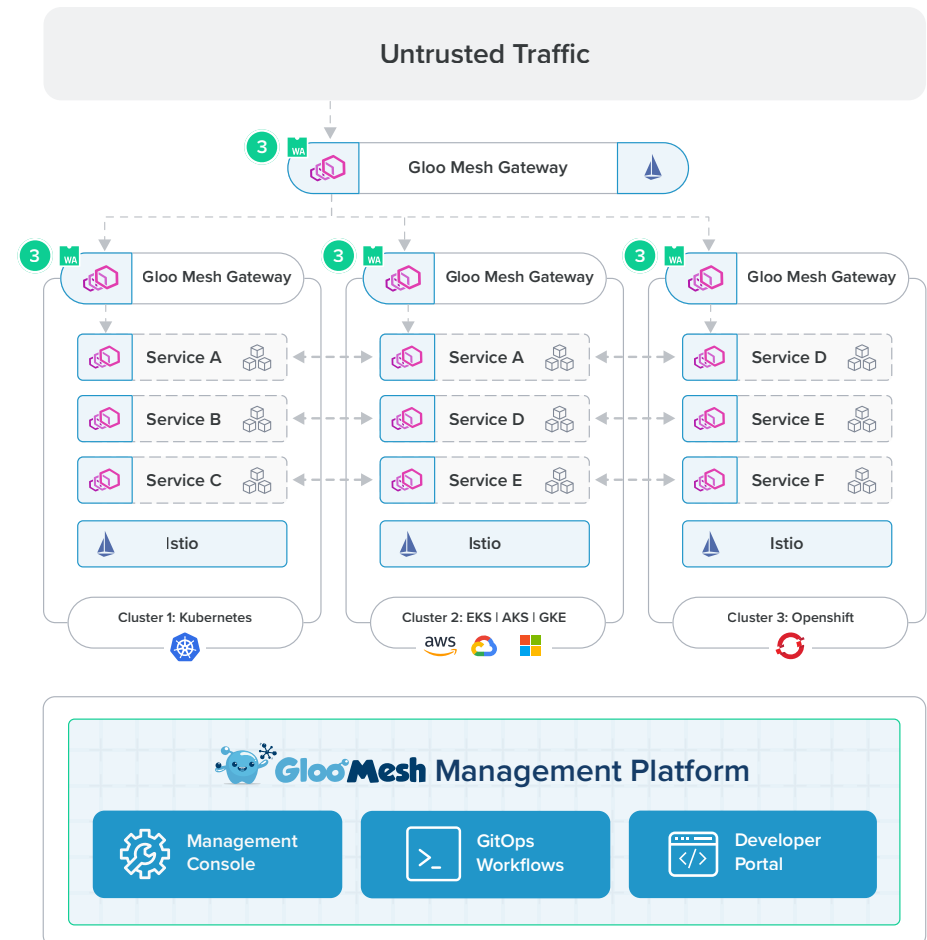
- FIPS-ready Istio-based service mesh
- Automated Service and API Discovery
- Enforce zero-trust security with authentication, authorization, and encryption
- Apply custom policies to route, filter, and transform L4 and L7 traffic
- Manage retries, timeouts, and circuit breakers
- Load balance and failover traffic based on locality and affinity rules
- Guard against sensitive info breaches with data loss prevention
- Collect metrics for observability, troubleshooting, and auditing with Prometheus and Grafana



# 3 Gloo Mesh Extensions

Extend and customize your API infrastructure with tooling for WebAssembly, plugins, and operators

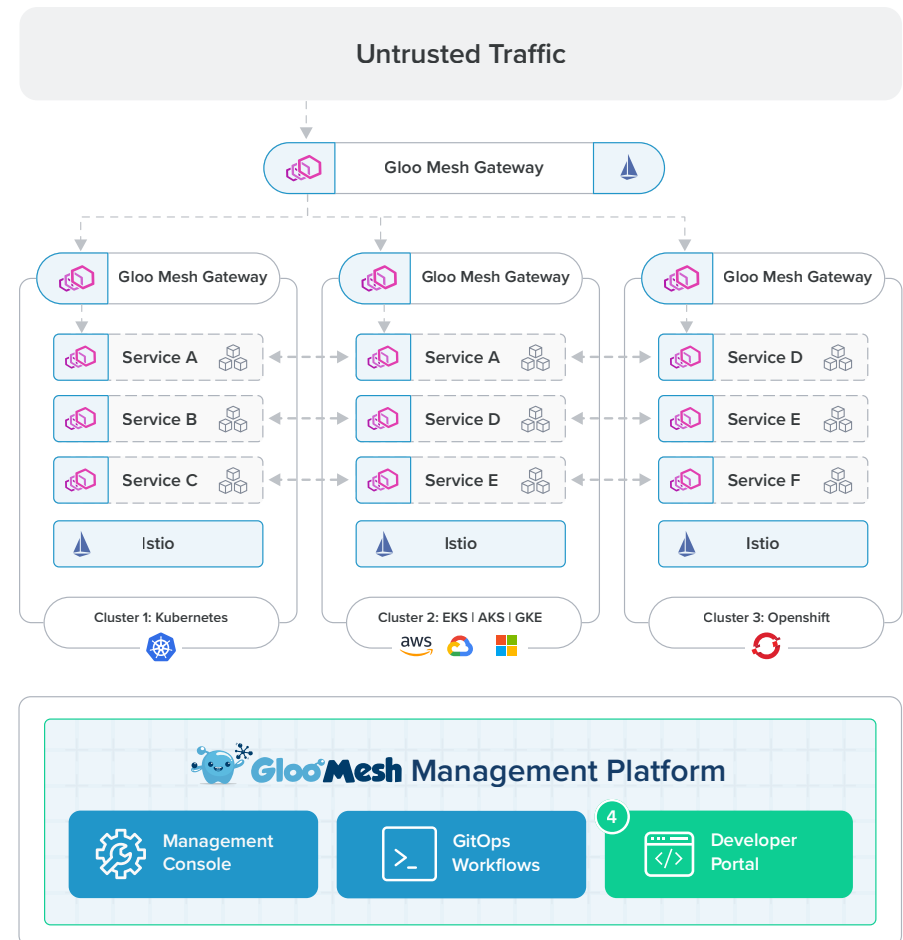
- Delegate authentication using OpenID Connect
- Pre-built support for external authentication (OIDC/OAuth), API Key, LDAP, and OPA Auth
- Extend Envoy Proxy capabilities with pre-built extensions including:
  - » Web Application Firewall (WAF)
  - » Data Lost Prevention (DLP)
  - » AWS Lambda
  - » Request and Response Transition
  - » SOAP
- Create custom Envoy Proxy filters with Web Assembly (Wasm)




# 4 Gloo Portal

Catalog, publish, and securely share APIs via a self-service developer portal

- CRD driven and works flawlessly with existing GitOps and CI/CD processes
- Accelerate developer onboarding with self-service documentation, and self-service sign up
- Manage gRPC APIs and REST APIs in the same developer portal
- Upload existing OpenAPI and proto documents to build the catalog
- Communicate authentication/authorization, usage plans and policies
- Showback, chargeback, and usage tracking of APIs
- Supports REST and gRPC APIs
- No database required





Solo.io, the modern service connectivity company, delivers application ng interface (API) infrastructure software that makes it easy for your architects and engineers to manage application traffic. As you move to cloud, microservices, Kubernetes containers, and serverless functions, you need a secure and reliable approach to application networking, with unified observability and control. Solo builds on open source Envoy Proxy and Istio to give you comprehensive API gateways and service meshes that work everywhere, at any scale. Founded in 2017 in Cambridge, MA, Solo is backed by Redpoint Ventures and True Ventures.

---

## Learn More

- Visit our website at [www.solo.io](http://www.solo.io)
- Request a personalized demo [solo.io/demo](http://solo.io/demo)
- Email us at [contact@solo.io](mailto:contact@solo.io)

