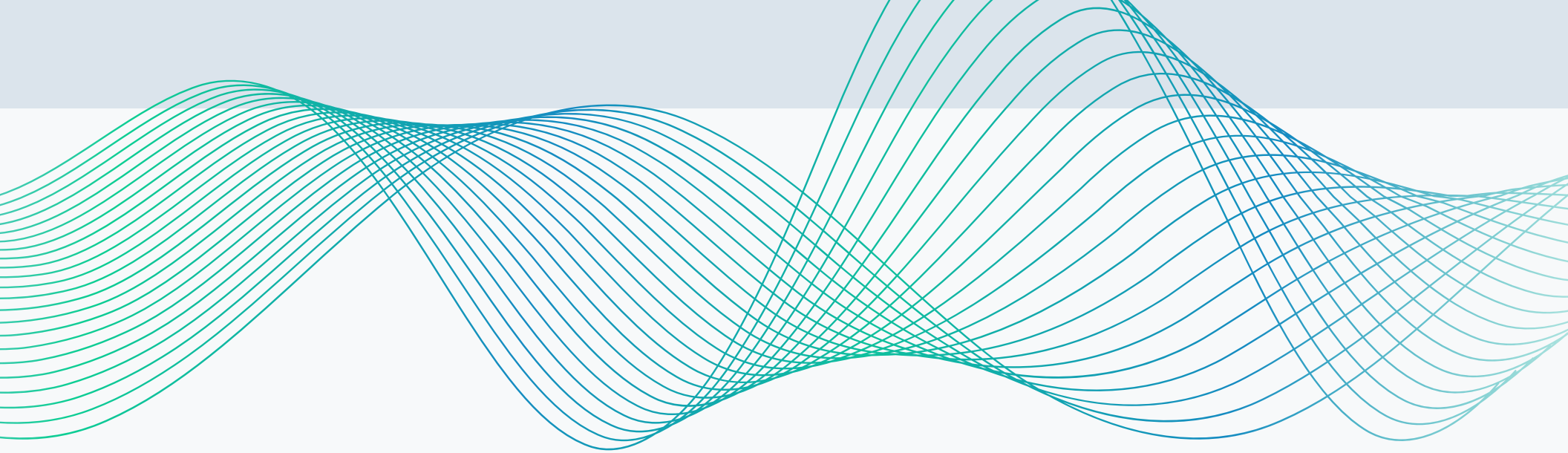
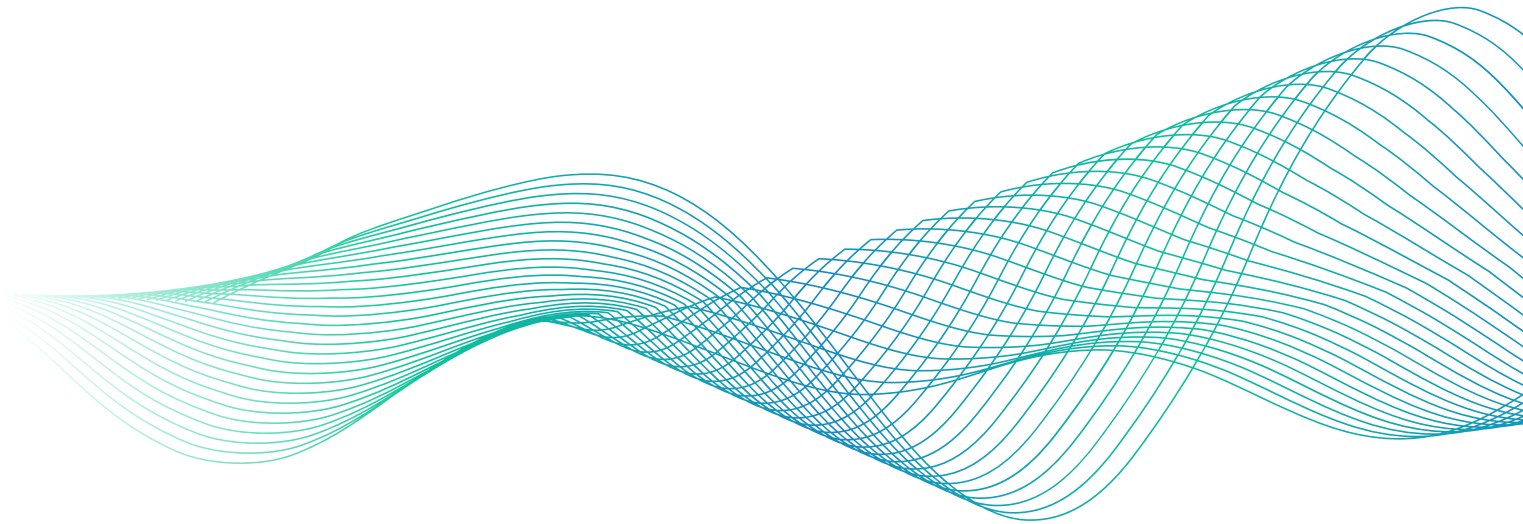


Moving to a modern API gateway

A technology comparison

solo.io

Table of Contents

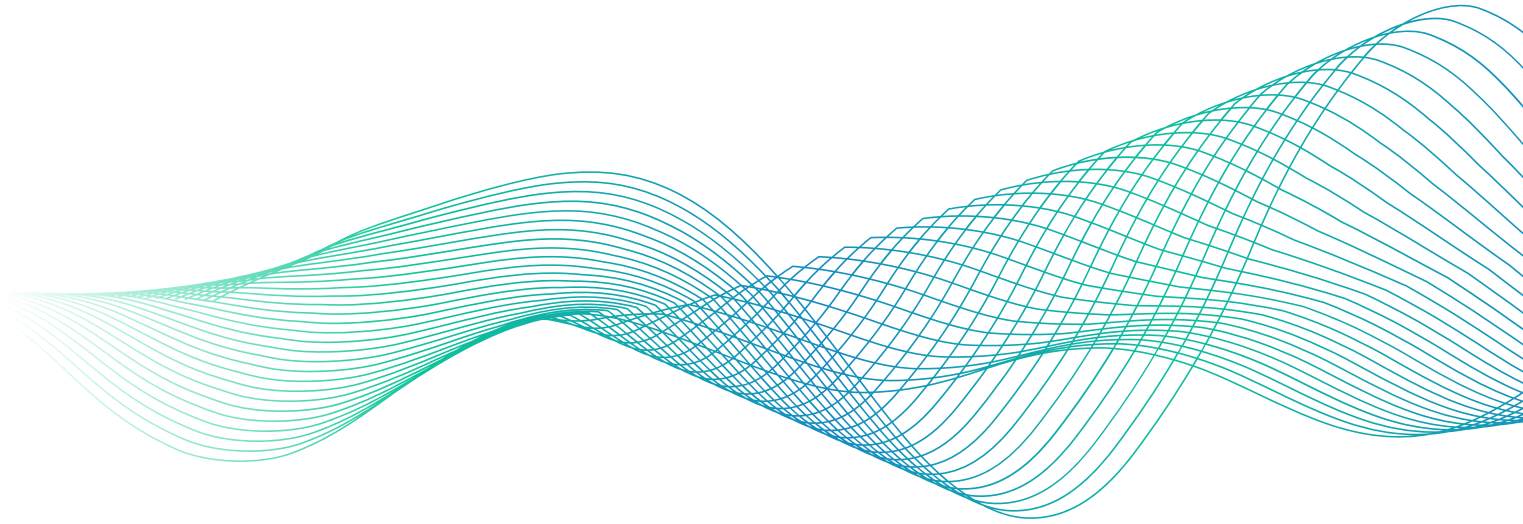


Introduction

API gateways have been critical components of applications for years, but legacy solutions are showing their age.

As organizations increase their reliance on microservices and container-based architectures, they're finding that legacy API gateways fall short in regard to scalability, traffic management, security, and observability. To support the evolving needs of modern enterprises, API gateways need to evolve in areas such as Kubernetes support, microservices, multi-cloud deployments, and cloud native innovation.

This ebook explores the defining characteristics of a modern API gateway and uses that framework to compare and contrast the current supporting technologies.



API gateways

Essential capabilities for today's challenges

So, what are the defining features of a modern API gateway built for the rigors of today's microservices journeys? At Solo.io, we're trusted experts in helping enterprises adopt, secure, and operate innovative cloud native technologies. In working with our clients to deploy future-ready API infrastructure from the edge to service mesh, we've seen that a truly modern API gateway has several key characteristics:

- Built on Envoy Proxy
- Supports today's architectures, future-proofed for innovation
- Extensible across operating environments, customizable with any language
- Capable of zero trust security and advanced threat prevention
- Built-in internet scalability and high availability, with lower resource use
- Integrates seamlessly into DevOps/GitOps workflows

Let's take a closer look at each of these areas.

Modern

Built on Envoy Proxy

Built to solve internet-scale API challenges, Envoy Proxy is the foundation of next-generation API gateway architectures. Leveraging an open source community consisting of more than 300 contributing companies, Envoy has emerged as the de-facto data-plane standard for cloud native applications and APIs. Envoy abstracts the network, providing infrastructure-as-code flexibility, while delivering traffic management, security, and observability in a platform-agnostic manner. This foundation enables a modern API gateway that improves security, reliability, filtering, transformations, and routing. A modern API gateway must also provide higher-level services such as federation, high availability, load balancing, failover, zero trust security, tracing, and metrics gathering. Envoy Proxy delivers industry-leading capabilities in these areas as well.

Architecture

Supports today's architectures, future-proofed for innovation

Today's API gateways have to support both traditional architectures (including monolithic applications and VMs) as well as newer, cloud native services and containers, and serverless workloads. In addition, a modern API gateway has to accommodate the latest technologies, including RESTful APIs, gRPC, and GraphQL. Many API gateways can support legacy architectures, but a modern solution will have the flexibility to accommodate emerging innovations and standards.

Flexibility

Extensible across operating environments, customizable with any language

Modern API gateways allow organizations to "extend" the architecture with new capabilities in a language-independent manner via WebAssembly—with the ability to decode, interpret, and filter wire protocol formats. This helps to route traffic more efficiently and allows offloading of tasks from other services. Modern API gateways can also easily add DLP and WAF capabilities (usually at no extra cost), while incorporating external authentication servers, rate-limiting servers, and request/response transformation and translation. Legacy API gateways simply can't provide the same level of flexibility.

Security

Capable of zero-trust security and advanced threat prevention

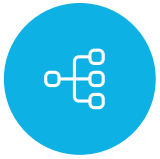
A zero trust security model differs from the traditional perimeter-trust approach by requiring strict identity verification at "call-time" for every user and service request, as well as encryption of traffic between services. Modern API gateways support a zero trust model, including invoking external authentications, applying network encryption (TLS/mTLS), and filtering requests with a web application firewall (WAF)— as well as combining features to support Forrester's ZTX and Gartner's CARTA strategies.

Scalability

Built-in internet scalability and high availability, with lower resource use

As the volume of API calls increases and infrastructure and microservice applications grow more complex, it becomes harder to connect, secure, and observe network traffic. Legacy API gateway architectures that use NGINX, Lua, Java, and other outdated technologies are too resource intensive, lack the native distributed architecture, and don't deliver the performance to support medium- and large-scale applications. Also, legacy API gateways often rely on supporting technologies (such as operational databases) that can become single points of failure and ultimately drive up costs and complexity.

Cloud native Ops

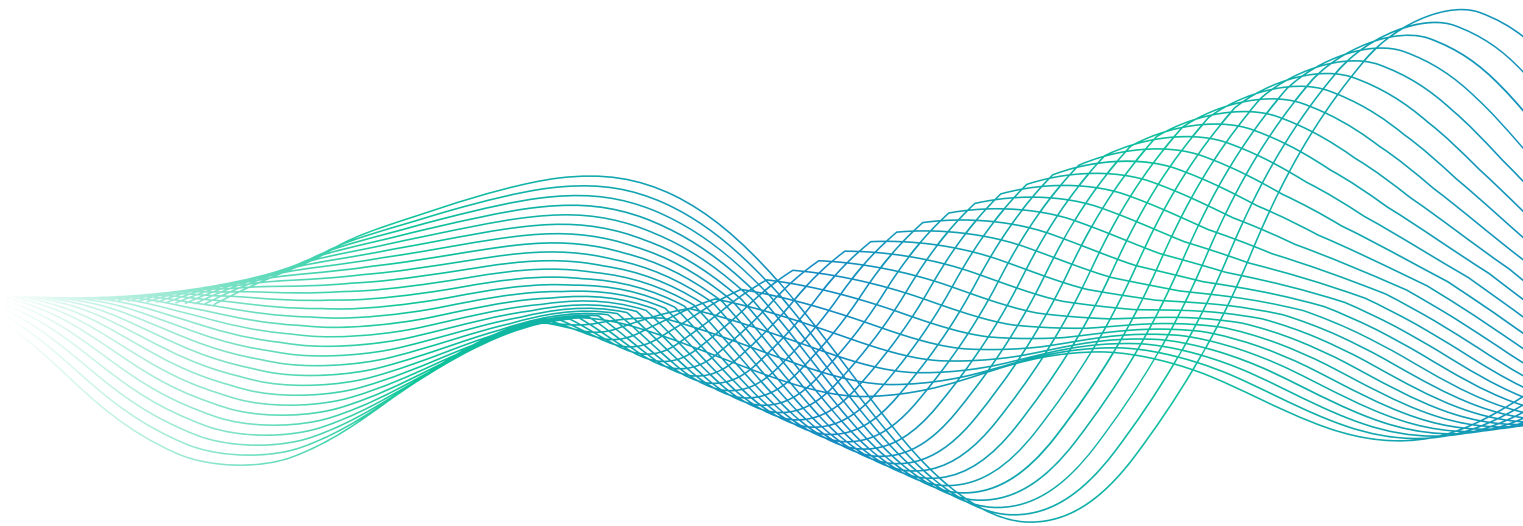


Integrates seamlessly into DevOps/GitOps workflows

Modern API gateways allow developers and operators to use declarative CRDs, usually as part of a DevOps/GitOps process, to manage traffic, enforce security policy, and configure observability. Dynamic updates without restarts, canary deployments, and federated configurations should be native components of a cloud-ready solution. DevOps teams need to deploy and manage an API gateway, often in concert with a service mesh, to programmatically manage application networking— eliminating the need to separately access and manage individual resources and services.

Comparing today's most prevalent API gateway solutions

Given these criteria for a modern, future-proofed API gateway solution, how do the most commonly deployed technologies stack up? What are their strengths and shortcomings in the face of today's networking challenges? Here's a closer look.



Hardware-based load balancers

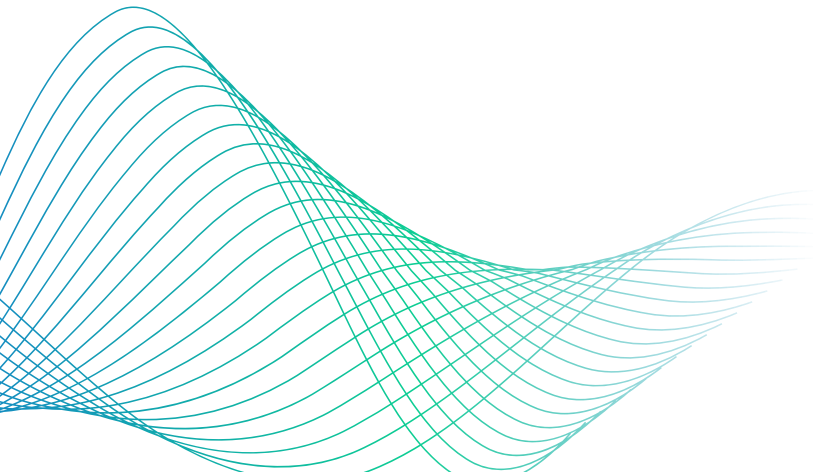
Examples: F5 BIG-IP, Citrix ADC

Hardware load balancers—or application delivery controllers (ADCs)—are legacy data center technologies for traffic management at the perimeter.

These legacy products are expensive, don't accommodate cloud native architectures, and can introduce single points of failure.

In addition, their configuration is typically managed by a separate network team (in other words, they aren't DevOps-friendly). If you are migrating workloads to the cloud, you may want to transition from this decades-old technology.

HARDWARE-BASED LOAD BALANCERS	
Built on Envoy Proxy and other current technologies	✗ **No.**
Built to support today's architectures	✗ **No.** No understanding of microservices and Kubernetes.
Extensible across different architectures	✗ **No.**
Capable of zero trust security	✗ **No.** Not designed for zero trust architectures.
Built-in internet scalability and high-availability	✗ **No.** Not scalable on-demand and introduces single points of failure.
DevOps and GitOps ready	✗ **No.** Configuration is typically managed by a separate network team (not DevOps-friendly).



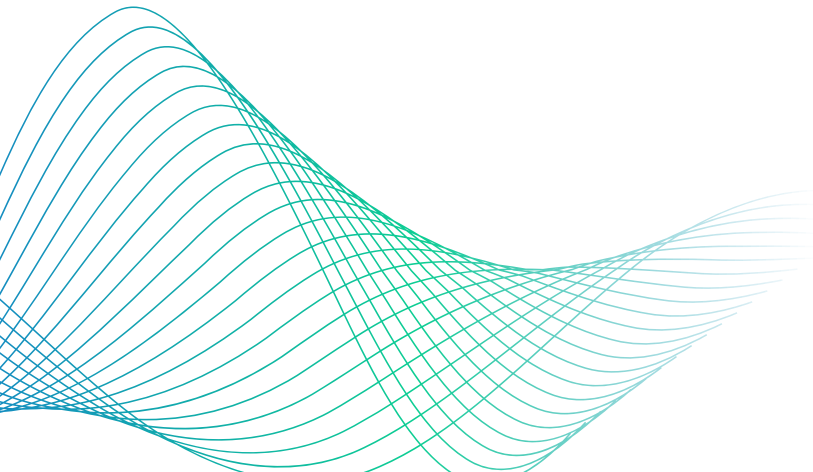
Web server-based load balancers

Examples: NGINX

NGINX is the most commonly used technology in this category. It acts as a load balancer and reverse proxy for HTTP and other protocols.

These technologies can be reliable for static content and ingress and egress, but this is another aging solution that isn't built for highly dynamic environments (requiring hot restarts, periodically dropping connections, and other issues).

If you are deploying a dynamic cloud infrastructure, implementing a zero-trust security model, want a declarative CRD-based configuration model, or require advanced capabilities such as DLP and WAF as part of the core product (and not an expensive add-on), web server-based load balancers won't meet your needs.



WEB SERVER-BASED LOAD BALANCERS	
Built on Envoy Proxy and other current technologies	✗ **No.**
Built to support today's architectures	? **Limited.**
Extensible across different architectures	? **Limited.** NGINX alone isn't a suitable API gateway and requires add-ons such as NGINX+, NGINX Controller, NGINX App Protect, and NGINX Amplify to support core capabilities for API management.
Capable of zero trust security	? **Limited.** Not designed for zero trust architectures.
Built-In Internet scalability and High-Availability	✓ **Yes.**
DevOps and GitOps ready	? **Limited.** Often requires additional products.

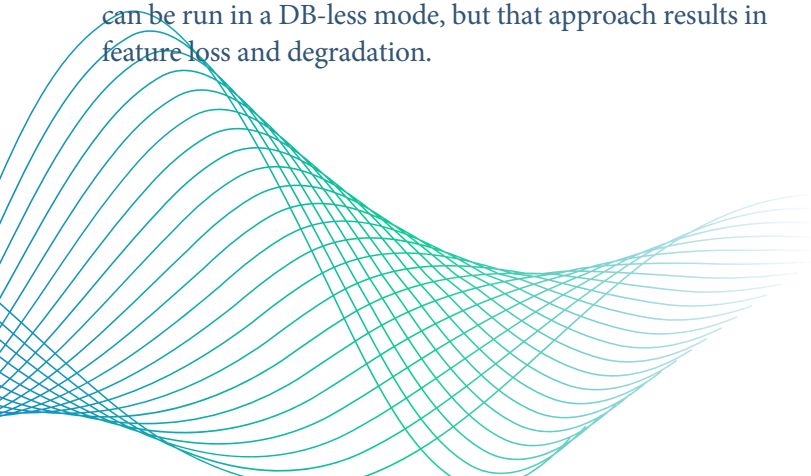
NGINX-based API gateways

Examples: Kong Gateway

While NGINX can provide the proxy foundation for an API gateway, it requires significant additional functionality. Several API gateway providers have taken this architectural approach.

One example is Kong Gateway, an API gateway that leverages NGINX, Lua (LuaJIT and LuaEngine), and a persistent data store. The primary issue with this approach is that it requires Lua expertise to implement and customize Kong Gateway.

In addition, implementing API gateways with outdated scripting languages has significant drawbacks with tail latency, debugging, scaling, and highly dynamic environments. Kong Gateway also incorporates a persistent data store using PostgresSQL or Cassandra, which increases operational complexity and expense. It can be run in a DB-less mode, but that approach results in feature loss and degradation.



NGINX-BASED API GATEWAYS	
Built on Envoy Proxy and other current technologies	✕ **No.** Built on NGINX.
Built to support today's architectures	? **Limited.** Mostly supports older architectures.
Extensible across different architectures	? **Possible** with plugins and modules, but with limitations. Kong Gateway is a Lua application running in NGINX and is distributed with OpenResty which is a module bundle that extends the Lua-NGINX module. Lua and LuaJIT are outdated technologies, with the most recent release of LuaJIT in May 2017.
Capable of zero trust security	? **Limited.** Note: Kong Gateway and Kong Mesh (Kuma) utilize different architectures and data planes.
Built-in internet scalability and high-availability	! **Not without complexity.** As Kong scales, it scales the ingress controller separately from the data plane. For each new gateway, Kong also gets a new ingress controller.
DevOps and GitOps ready	! **Not without Complexity.** Not designed for DevOps/ GitOps. Requires a persistent data store (such as PostgresSQL or Cassandra) that increases operational complexity and cost. New DB-less configuration is available, but doesn't support all features resulting in feature loss and degradation.

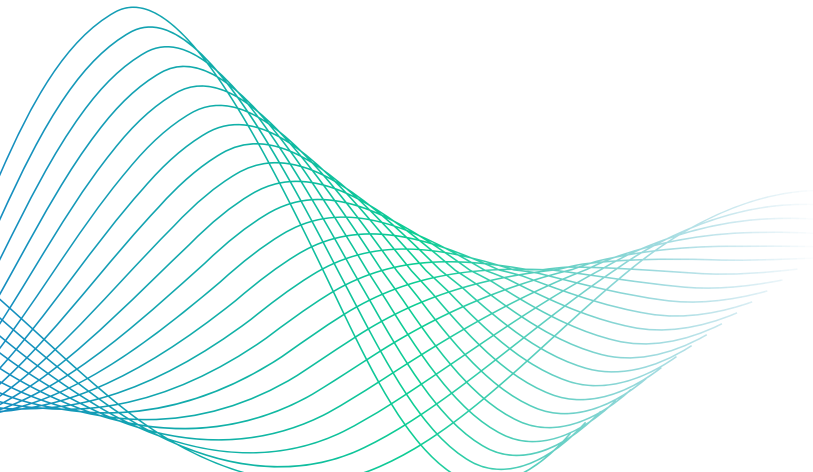
Full life-cycle API management products

Examples: Apigee

Full life-cycle API management products such as Apigee emerged as the need to share APIs across organizations grew along with the need to better manage, document, and monetize API traffic.

Most products in this category were developed during the era of monolithic application architectures and VMs and have been slow to adapt to cloud architectures.

In addition, they tend to suffer from performance and latency issues due to their reliance on Java-based architectures and other dated platform technologies. Finally, they have limited ability to integrate into DevOps/GitOps workflows and come with high operational costs.



FULL LIFE-CYCLE API MANAGEMENT PRODUCTS	
Built on Envoy Proxy and other current technologies	✗ **No.** Gateway is built on NGINX with Java.
Built to support today's architectures	? **Limited.** Mostly supports older architectures.
Extensible across different architectures	? **Very limited.** Architected for public APIs. Other scenarios complicated to support.
Capable of zero trust security	✗ **No.**
Built-in internet scalability and high-availability	! **Not without Complexity.** Monolithic application architecture with limited innovation on core platform. Requires Cassandra for persistent data store. Performance and latency issues due to the use of outdated technologies.
DevOps and GitOps ready	✗ **No.**

Solo Gloo Gateway

The leading next-generation, cloud native API gateway

Solo Gloo Gateway is a Kubernetes-native, next-generation API gateway built on Envoy Proxy to manage, secure, and observe traffic at the edge. Gloo Gateway configures the behavior of the Envoy Proxy data plane to ensure secure application networking and policy-based traffic management, while gathering metrics to improve observability.

Representative Products		Gloo Gateway	Apigee	Kong Gateway	NGINX+	F5
Modern	Built on Envoy Proxy and other current technologies	Envoy-based	NOT Envoy-based			It's just time to move from this rigid, legacy technology
Architecture	Built to support today's architectures	First class support of gRPC and RESTful APIs. Built-in GraphQL server	Limited support for RESTful APIs, gRPC, and GraphQL			
Flexibility	Extensible across operating environments	Built in support for DLP and WAF. Extend capabilities in language-independent manner via WebAssembly	Limited DLP and WAF. Can't extend capabilities in language-independent manner via WebAssembly			
Security	Capable of zero trust security	Designed for zero trust architectures	Not designed for zero trust	Limited	Not designed for zero trust	
Scalability	Built-in internet scalability and high-availability	Highly scalable	Performance and latency issues due to the use of outdated technologies	Does not scale ingress controller separately from the data plane leading to resource issues	Highly scalable	
Cloud native ops	DevOps and GitOps ready	Designed for DevOps and GitOps. Configured via CRDs	Not designed for DevOps/GitOps. Requires a persistent data store for fully functional product		Not designed for DevOps/GitOps	
Service mesh integration	Seamless integration with Istio Service Mesh	Shares the same control plane with Istio, Gloo gateway is part of the Service mesh	Provide limited integration with Istio service mesh.			

Not all API gateways are created equal.

Ideally, the API gateway sits in the data plane and manages "north-south" traffic through services such as security, reliability, filtering, transformations, and routing. Working collectively, API gateways can provide higher-level services such as high availability, load balancing, failover, zero trust security, tracing, and metrics gathering.

Next-generation API gateways are also purpose-built for highly dynamic, ephemeral environments such as Kubernetes and incorporate the design principles of declarative configuration, decentralized ownership, and self-service collaboration. In addition, next-gen gateways use declarative CRDs, enabling seamless integration into GitOps workflow. Finally, next-gen gateways secure tomorrow, providing zero trust security and seamless service mesh integrations.

Ready to learn more about Solo Gloo Gateway and why your next API gateway needs to be Envoy-based?

Find more information and further analysis on the different API gateways. **www.solo.io**.

solo.io