traefiklabs

The Buyer's Guide

for Modern API Gateways

Table of Contents

Introduction

In today's cloud-native world, APIs are everywhere. According to a SlashData survey, almost 90 percent of developers use APIs to power the applications they build—including both external APIs, which allow applications to connect to third-party services and resources, and internal APIs and microservices within a company's own IT estate.

Given the pervasive role that APIs play in modern operations, having a way to simultaneously manage and secure API transactions and microservices is a critical consideration for many organizations today.

That's where API gateways come in. API gateways route, secure, and manage API and microservices transactions. By acting as intermediaries in API and microservices requests, API gateways can help to centralize, monitor, route, and secure the tremendous volumes of transactions that power operations at a typical organization today.

All API gateways provide the core capabilities we just mentioned. However, different API gateway solutions work in different ways. Depending on your business's needs and priorities, one API gateway may be a better choice than another.

We've prepared this buyer's guide to help decision-makers navigate the complex landscape surrounding API gateways today. In the following pages, you'll learn what API gateways do, how they benefit modern businesses, and which key considerations to weigh when comparing API gateway solutions. We'll also compare some of the leading API gateways available on the market today and explain where each one excels.

The guide includes three main parts:

Part 1:
Understanding API Gateways explains what API gateways are and why they're essential for modern businesses.

Part 2:
Critical Trends in Application Development and API Gateway Requirements discusses the changing requirements of modern application infrastructures. It also explains what API gateways must do to meet these challenges.

Part 3:
Choosing an API Gateway offers guidance on selecting an API gateway from the leading solutions available today.

Part 1.

Understanding API Gateways

Even for seasoned engineers, understanding exactly what an API gateway does and why it's important can be challenging. After all, API gateways weren't an important part of conventional technology stacks, so many teams lack experience working with them. But as this section explains, API gateways play an absolutely vital role in enabling modern approaches to application deployment.

What is an API Gateway?

An API gateway sits in the critical path of a data flow to accept incoming requests from clients, route them to the appropriate services, and send responses back to the clients.

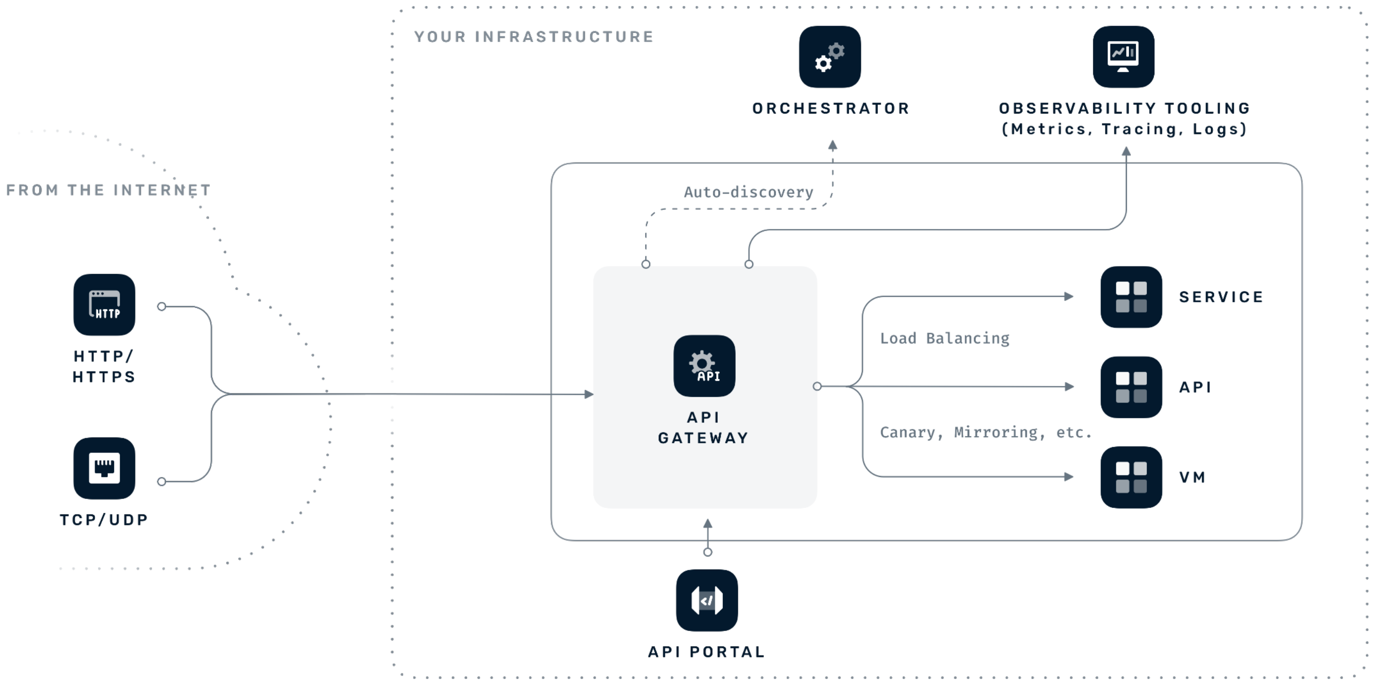
Importantly, the transactions that API gateways support don't necessarily need to take the form of formal API requests made using API protocols (like REST or gRPC). Despite the nomenclature, API gateways can manage and secure any request to an application service or microservice, even if it's not an API request.

Because API gateways sit between clients and services, they provide a central, consolidated location to which clients direct all requests, and from which all responses are directed back to clients. They can monitor the status of requests and help teams detect problems like malformed API calls. They can also assist with authentication and monitoring. Finally, API gateways can track and help respond to security problems, such as malicious requests or attempts to overwhelm a service with a flood of traffic.

Strictly speaking, you don't need an API gateway to use APIs or microservices. You could allow clients to make API calls to services directly. However, that approach is inefficient by modern standards and considered a design anti-pattern. It forces teams to duplicate code across services. It also significantly complicates API management.

In addition, it defies the Do Not Repeat Yourself (DRY) principle, which calls for implementing common policies around security or observability that apply to a myriad of services. With an API gateway, teams can enforce security policies across all exposed services, minimizing the risk of oversights or omissions.

So, while you may be able to get by without an API gateway if you have just a handful of services, you'll likely want an API gateway that brings efficiency to security and observability, while freeing your teams to focus on creating true business value rather than tedious operations.



Core Capabilities to Expect from Any API Gateway

API gateways deliver six key types of capabilities:

Security: API gateways can help intercept and block malicious requests. They also support certificate management and can operate as Web Application Firewalls (WAFs), helping prevent malicious requests from reaching backend services.

Authentication and authorization: By using API gateways to authenticate and authorize requests based on standards like OIDC, JWT, and OAuth—rather than relying on individual services to handle those tasks—organizations gain a consistent, centralized approach to ensuring that clients are authorized to make requests before the requests are served. This also helps reduce security risks.

Traffic management: API gateways help to route traffic efficiently across a range of services based on load balancing, circuit breaking, rate limiting, and so on. They can also reduce the risk of performance issues due to malformed requests and deliver active health checks and rate limiting to keep traffic flowing smoothly. In addition, API Gateways can support A/B testing and canary releases, which also bring efficiency and lower risk to application deployment.

Certificate management: Issuing TLS certificates is typically a manual process that has to be performed for each service. An API gateway, however, can centralize and even automate this process so your services are never left without encryption.

Caching: Data caching within API gateways helps applications perform better and teams to move faster.

Collaboration: When you choose an API gateway that provides a developer portal with OpenAPI support, you help developers work together with each other and other stakeholders efficiently and continuously. You also benefit from the ability to integrate some API gateways with Git-based configuration management.

API Gateways vs. Proxies, Load Balancers, and Ingress Controllers

API gateways provide capabilities that overlap in some ways with certain other types of solutions; however, there are fundamental differences between these tools:

- Reverse proxies also route incoming requests to backend services; in fact, API gateways are essentially one form of reverse proxy—but they specialize in serving API and microservices requests, whereas other reverse proxies are general-purpose solutions that don't address the unique management and security challenges of API and microservices requests. Additionally, reverse proxies usually only support traffic to Web applications, as opposed to managing requests for a wide variety of services and protocols, as API gateways can do.
- A load balancer's main job is to accept incoming traffic and route it to multiple instances of the same backend service, with the goal of ensuring that requests are balanced according to your load balancing strategy. API gateways can typically do this as well; however, API gateways provide many additional capabilities—such as security monitoring and observability—that load balancers lack.
- Like API gateways, ingress controllers can accept incoming requests and route them to services. However, ingress controllers typically only support HTTP requests that are formatted in specific ways. API gateways are more flexible; they support multiple protocols, including but not limited to HTTP, and they can handle requests formatted in virtually any way.

For these reasons, it would be a mistake to think of API gateways as a mere alternative to or substitute for other types of solutions that can manage and route requests on a network. API gateways provide capabilities on top of those provided by the other solutions.

If your goal is only to manage HTTP requests, an ingress controller, reverse proxy, or load balancer might meet your needs. But if you require support for a range of protocols, and if you want extended features (security, distributed rate limiting, etc.), an API gateway is your best option.

Benefits of API Gateways

API gateways provide a range of benefits. Let's break them down based on two key categories: business benefits and operational benefits.

Business Benefits

From a business perspective, API gateways enable:

- **Agility:** A centralized point for managing API and microservice routing and security configuration makes it easier to adapt to market changes, introduce new features, and stay ahead of competitors.
- **Digital transformation:** Because APIs and microservices have become a fundamental component of digital strategies as businesses shift from monolithic, VM-centric solutions into distributed, cloud-native alternatives, managing requests efficiently and securely is critical for ongoing digital transformation success.
- **Revenue generation:** Exposing services and data to external developers or partners with controlled access via API gateways can open new opportunities for generating revenue or selling services.
- **Cost efficiency:** Features like caching and rate limiting via a centralized controller make it easier to protect the financial interests of the business.
- **Security and compliance:** API gateways mitigate security and compliance risks by providing authentication, authorization, and encryption mechanisms.
- **Enhanced customer experience:** With a consistent, reliable, and high-performance experience, customers are more likely to keep coming back to your business.
- **Innovation and partnerships:** API gateways make it easy to adopt new technology and integrate with external services quickly with help from APIs.
- **Scalability and growth:** The ability to handle increased traffic efficiently with API gateways helps ensure that your business can keep growing steadily.
- **Competitive advantage:** Stay ahead in technology adoption by differentiating offerings and capturing market share through a more flexible, scalable, and modern strategy.

Operational Benefits

For DevOps, DevSecOps, and IT teams, API gateways provide a range of operational benefits:

- **Centralized management:** All services can be managed and secured using a central tool, even in complex architectures, saving IT teams enormous time and effort.
- **Simplified client access:** A consistent API interface makes it easy for IT departments to expose services to clients, without requiring a bespoke implementation for each client or service.
- **Security enhancements:** Centralized security monitoring and policy enforcement make it more efficient for IT and DevSecOps teams to manage security risks.
- **Request and response transformation:** API gateways can automatically transform requests and responses to speed up the time to market.
- **Observability:** API gateways provide a central vantage point for collecting logs and metrics, which engineers can then analyze to maintain service quality and security. By centralizing the management of observability, teams simplify what can otherwise be a complex operation.
- **Canary releases and blue/green deployment:** By helping to enable canary releases and blue/green deployments—which reduce the risk of problems during application updates—API gateways support smooth rollouts and minimize disruptions that impact end-users.
- **Scalability:** No matter how many services you have to support or how many requests they receive, most API gateways can scale automatically to manage them effectively. As a result, applications can scale seamlessly, without relying on engineers to add new infrastructure or service instances manually.
- **Third-party integrations**: Rather than having to set up custom integrations with each third-party service, IT teams can use API gateways to provide a central integration hub that saves time when rolling out application stacks.

Part 1 Wrap Up

API gateways are much more than management interfaces. They can help to support multiple aspects of application deployment, security, scaling, observability, and more. They also simplify operations by providing centralized configuration and enforcing strong boundaries between development and operations work.

Ultimately, they free engineers to focus on creating business value instead of managing tedious, manual tasks, which in turn leads to faster time to market.

Part 2.

Critical Trends in Application Development and API Gateway Requirements

We just learned all about what API gateways are and why they're important. But we didn't delve into nuances surrounding how new trends in application development are leading to new requirements. Let's explore that topic by looking at what API gateways must do to deliver optimal value in the face of ever-evolving development trends and strategies.

The Cloud Native Revolution

The concept of API gateways emerged well over a decade ago, before cloud native computing became widespread. As a result, many API gateways lack the ability to keep pace with later-born cloud native environments that may include hundreds of servers, services, and deployments per day. While there are exceptions, notably Traefik Labs' API gateway which was built as a cloud native solution, many still reside on bare-metal or virtual servers rather than running in containers, and they can't scale or adapt efficiently by cloud-native standards.

Most applications are now migrating to the cloud, at least in part, but this is a long process. In many cases businesses are not yet fully in the cloud but remain dependent on hybrid models that include a mix of legacy and cloud native technology.

One major benefit of the cloud native era is that solutions are built on standards that prioritize interoperability. As a result, teams benefit from a large array of choices and can reduce their risk of vendor lock-in.

So to thrive in this modern world, API gateways must support applications deployed in any environment, using any cloud architecture alongside other mission critical tools. They must be able to route requests to microservices hosted in a Kubernetes cluster in a public cloud, as well as requests to monolithic applications hosted on-prem. And to deliver maximum benefit, API gateways should enforce the same security controls and deliver the same governance capabilities across all of the applications they help manage—a critical benefit for organizations seeking to tame the complexity of hybrid cloud and multi cloud deployment models.

The DevOps and GitOps Revolutions

Likewise, API gateways must keep pace with innovations in application development and delivery, such as the adoption of DevOps and Continuous Integration/Continuous Delivery (CI/CD).

For example, using GitOps, the CI/CD strategy that many DevOps engineers use to develop, test, build, and deploy code, can connect seamlessly to API gateways. This approach is not viable in every case, but when it is used, it enables teams to control and even automate their routing, load balancing, security policies, and other configurations (including any custom objects) via a declarative approach. Teams then manage the entire lifecycle of the system within reviewable change requests, avoiding human errors and allowing for quick rollbacks.

But as stated earlier, many API gateways were designed before the cloud-native era. This means they don't take full advantage of these declarative approaches to configuration management.

GitOps is revolutionizing operations, making complex deployments and configurations auditable, repeatable, and scalable. For DevOps and platform teams to capitalize on these benefits, their API gateways must be fully declarative, ideally without requiring vendor-specific annotations.

Modern API Gateway Case Studies

Now that we've covered the role of API gateways and the trends they must adapt to, let's look at some examples of how real companies have used and benefited from modern API gateways, in these cases Traefik Labs' API Gateway.

Simplifying Network Management at IoT Scale

By design, networking is complex. But with the Internet of Things (IoT), network complexity—and, by extension, the complexity of network management solutions—is truly profound. With hundreds of thousands of endpoints and tens of thousands of customers, ABAX had to centralize and automate operations.

To simplify operations and consolidate the company's networking tools, the business adopted an API gateway model. The move boosted performance and security through capabilities like token-based authentication and data caching.

Lowering Costs for High-Volume Retail Traffic

Adeo's Leroy Merlin, an international home improvement retailer, was struggling to manage network traffic cost-effectively, due especially to the very high volume of requests that it had to support—as many as 100,000 per second. In addition, to deliver a great customer experience, Adeo targeted latency rates of just 20 milliseconds.

Implementing a modern API gateway helped the company not only to meet, but to exceed, these goals. The efficiency of a centralized hub for managing service requests enables the business to support up to 200,000 requests per second while also achieving uptime rates of 99.99 %. What's more, Adeo integrated its API gateway with monitoring and observability software to help detect performance issues, as well as identify and manage sensitive data.

Managing Hybrid Infrastructure In the Telco Industry

For Axione, a major French telco provider, streamlining connectivity across a complex infrastructure that included multiple public clouds, private data centers, and cloud native and legacy workloads running alongside one another was a key challenge.

The ability to automate configuration management with help from a GitOps-driven API gateway solution helped the company rise to this challenge. Now, Axione benefits from simplified service deployment, automated service discovery, single sign-on functionality, end-to-end encryption of network data across its disparate environments, and a 20 % reduction in network latency—a key accomplishment in a vertical where delivering fast, reliable connectivity is paramount.

Part 2 Wrap Up

Cloud native solutions provide many opportunities, but they also create technical and organizational challenges. As a result, businesses need API gateways designed to make the most of cloud native environments and standards, while also offering compatibility with the legacy and hybrid technologies that remain important in many organizations.

Part 3.

Choosing Your API Gateway

Now that we've covered the basics of API gateways and the challenges they need to address, let's explore the key question that you need to answer: "How do you choose the right solution?"

This part of the guide highlights the main API gateway solutions available today and what you need to consider when evaluating them.

Vendor Comparison

First, let's take a detailed look at major API gateway solutions and how they compare.

Features and capabilities	Traefik Labs API Gateway	Kong Konnect	Gravitee API Gateway	Ambassador Edge Stack	NGINX Plus API Gateway	AWS API Gateway
Overview	Full-fledged modern API gateway based on a wildly popular cloud-native open-source proxy	A modern API gateway based on a hosted control plane deployment model	Modern API gateway with a special focus on treating asynchronous and synchronous APIs equally	A Kubernetes-based API gateway designed especially for Kubernetes ingress control	A solution mostly designed to function as a reverse proxy but with some API gateway features	A fully managed API gateway service, tightly integrated into the AWS ecosystem
Ease of Use	**High** Fully declarative, human-readable configuration with simple defaults and linter assistance to get started quickly and catch errors early. Web UI and pre-built Grafana dashboards are also included.	**Medium** Although a declarative configuration approach is available, its capabilities are limited, it is complex to scale to many APIs, and it is tailored primarily toward UI-based operations.	**Low** The declarative configuration is limited with little documentation, and the web UI is complex. It includes many moving parts that require administration and is taxing on resources.	**Low** There isn't a web UI to check or modify the current state, which can be challenging for novice users. Resource consumption can also be an issue since it's Envoy-based.	**Low** It doesn't have a web UI but can be added by third-party tools not considered enterprise-ready. Configuration changes need proxy reloads. Extensions are LUA-based and sometimes require custom compilation from source code.	**High** It has a web UI, and auxiliary tools allow for a declarative approach. Extensive documentation, tutorials, and training are available.
Breadth of API Gateway Features	**High** Provides routing, security, observability, and more	**High** Provides routing, security, observability, and more	**High** Provides routing, security, observability, and more	**Medium** Provides all core API gateway capabilities, but mostly only within Kubernetes environments	**Medium** Provides some capabilities, but advanced features (like security policy enforcement) are not available	**Medium** Provides all core API gateway capabilities, but customization and extensibility options are limited

Table continued on the next page →

Features and capabilities	Traefik Labs API Gateway	Kong Konnect	Gravitee API Gateway	Ambassador Edge Stack	NGINX Plus API Gateway	AWS API Gateway
Orchestrator Compatibility	**High** Designed to be cloud native from the start and supports all major enterprise tools including HashiCorp Vault, Azure Service Fabric, Docker Swarm, Amazon ECS, and all Kubernetes distributions	**Medium** Not originally designed to support cloud-native computing stacks but can run inside Kubernetes	**Medium** Not originally designed to support cloud-native computing stacks but can run inside Kubernetes	**Medium** Not originally designed to support cloud-native computing stacks but can run inside Kubernetes	**Medium** Not originally designed to support cloud-native computing stacks but can run inside Kubernetes	**Low** It doesn’t provide any integration with other orchestrators and only works inside AWS.
GitOps compatibility and API management features	**High** Enables GitOps-based approach to end-to-end API management	**Medium** GitOps-based API management is possible but requires additional tools	**Low** GitOps compatibility is not a major focus	**High** Enables GitOps-based approach to end-to-end API management	**Low** No focus on API management	**Medium** GitOps-based API management is possible but requires additional tools
Deployment Flexibility	**High** Supports any common cloud or on-prem architecture	**Medium** Supports any environment, but the control plane is available only as a hosted service	**High** Supports any common cloud or on-prem architecture	**High** Supports any common cloud or on-prem architecture	**High** Supports any common cloud or on-prem architecture	**Low** Only available within AWS
Pricing Model	Node-based - Flexible to accommodate other models per customer choice	Mostly request-based - Some features and plugins cost extra	Pricing details for most scenarios not publicly available	Request-based - Free for up to 10,000 requests per month; pricing for higher-volume use cases not publicly available	Pricing details vary widely depending on the deployment type	Request-based pricing but also charges for the data transferred out and the used cache size
Licensing	Open source core	Open source core	Open source core	Open source core	Open source core	Proprietary

Understanding API Gateway Pricing Models

In addition to the differences in features laid out above, API gateways also vary in terms of how they are priced. Most API gateway vendors use one of the following pricing models:

CPU Core-Based Pricing: The traditional pricing model for gateways is to charge based on the number of CPUs used to run the gateway. This makes sense if your gateway runs on bare-metal or on-prem hardware. But in the modern world of virtual CPUs and infrastructure that is constantly scaling up and down, CPU-based pricing makes it difficult to predict how much you'll actually pay. As you can see in the comparison chart, this has become less common.

Request-Based Pricing: Many vendors charge based on the number of requests that pass through a gateway. While this may seem simple enough, a major downside is that spikes in demand can trigger overage charges, which may come at a premium. This makes request-based pricing suboptimal if your traffic volume fluctuates or you experience period surges in demand (as you likely do if, for example, you're a retailer who sees spikes during events like Black Friday).

API-based pricing: In this model, vendors charge based on the total number of APIs available in your system—e.g., if you expose 10 APIs, you are billed for 10 APIs. While this ensures predictable pricing, the API-based model constrains your team's architectural decision-making and can force them to create artificial grouping for the sake of saving money.

Node-Based pricing: Since nodes (meaning the servers that host cloud native apps) are a fundamental building-block of modern infrastructure, vendors may price API gateways based on the total nodes used. This model aims to provide predictable pricing that scales up and down with your infrastructure, while allowing you to avoid having to pay for unused gateway capacity, or for overcharge charges in the event your gateway serves more requests than you expected.

Checklist: Considerations When Choosing a Modern API Gateway

Actually selecting an API gateway is a multi-step process that can feel overwhelming—but when you break it down step-by-step, it becomes manageable. Here are the major items to consider when evaluating API gateway solutions, divided into four key stages: Planning, Day 0 (deployment), Day 1 (initial setup) and Day 2 (ongoing management).

Planning Considerations

During the planning phase, consider:

Vendor reliability and support: How long has the API gateway vendor been around, which support options are available, and how much do they cost? Does the vendor offer Service Level Agreements (SLAs) guaranteeing certain levels of performance and reliability? Can they customize their product to meet your bespoke business requirements, or do they offer only a generic gateway solution?

Pricing model and cost: Which pricing model does the vendor use, and how predictable will it make your bills? Will you run the risk of wasting money on unused capacity or paying overcharge fees if you exceed a preset quota? How does your projected total cost of ownership compare across vendors? Be sure to consider, too, the upfront acquisition cost of the tool (in other words, your CAPEX costs), as well as ongoing operational (OPEX) costs.

Criteria and features: Does the API gateway provide all of the features you need? As we've noted, all modern gateways provide core request management and security features, but will the solution integrate with your preferred tool set or enable advanced practices like GitOps?

Day 0 Considerations

When evaluating what to expect from the solution on Day 0, consider:

The deployment model: Where can you deploy the solution? Does it only work with a certain type of infrastructure (such as only on-prem or only on bare metal), or can you run it anywhere? Will it operate across multi or hybrid cloud environments, or only within a simpler architecture?

Flexibility: How flexible is the solution with regard to the protocols it supports, the tools it integrates with, and the types of complementary software (such as orchestrators) you'll leverage to help manage the gateway?

Adaptability to business requirements: How easily can the solution change as your business requirements change? For instance, if you deploy new APIs, or you require additional API management features (like ingress or load balancing) that you did not use when you initially deployed the solution, can the product adapt with ease? Or would you need to roll out a new version from scratch in order to support new capabilities or business requirements?

Day 1 Considerations:

When assessing a tool's value on Day 1, consider:

Ease of use: How simple will the solution be for your team to use on an ongoing basis? For example, does it support centralized, Git-based configuration management? How much manual effort will the tool require from your team? Will they view it as a solution that provides real value, or one that becomes just another tool to manage?

GitOps compatibility: We said it before, but we'll say it again, because it's so critical on Day 1: Consider the extent to which the solutions you are evaluating enable transformative GitOps practices, making it possible to leverage Git as a "single source of truth" that drives collaboration across the organization while also providing benefits like transparent configuration auditing and easy configuration rollbacks.

Extensibility: If you need additional functionality beyond the core features built into the gateway, how easy is it to extend functionality by installing plugins or add-ons? Is there a large ecosystem of plugins to connect your gatway to other tools? How much time and effort will it take your engineers to enable and configure the extensions you want to use?

Day 2 Considerations

Finally, assess how much value your gateway will deliver on Day 2 by considering:

Reliability: Which levels of availability (in other words, how many "9s") does the gateway vendor promise, and what is the solution's track record of delivering high availability and performance? Consider, too, how many resources the gateway consumes to operate and whether its overhead may negatively impact your workloads by depriving them of the resources they require.

Observability: Once your gateway is up and running, how easy is it to observe and troubleshoot the performance of the gateway itself? Does the vendor provide observability tools, visualizations, and integrations for this task, or are you on your own to figure out how to support the gateway? Can you easily run traces to pinpoint the source of errors in the event of a gateway performance issue?

Change management: How can admins implement changes, such as upgrading the API gateway to a new version or modifying its configuration? Is the change management process largely automated, or do admins need to apply changes manually? Do changes require shutting down the gateway—which disrupts operations—or can they be applied dynamically, with no downtime?

Conclusion

API gateways share common features, but no two are created equal when it comes to their day-to-day operation. This is why making the right decision is so critical for success. When evaluating solutions your goal should be to think not just about short-term needs and costs, but also which role your gateway will play in enabling efficient Day-2 operations over the long term. Make sure that your solution can fully support modern automation and cloud native computing needs, while providing compatibility with any hybrid or legacy environments that you operate. Last but not least, consider how your needs will change in the future and how easily your gateway can adapt to them.

See the most modern, GitOps-driven API gateway in action.

WATCH DEMO VIDEO

træfiklabs