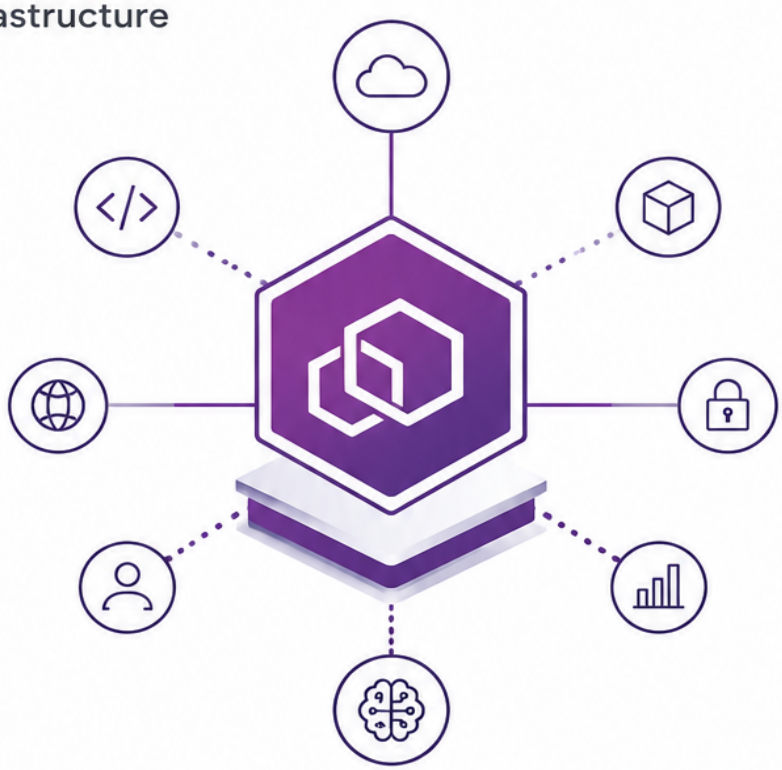
ENVOY PROXY

The Definitive Production Guide

Architecture, Configuration, Operations, and Advanced Patterns for Cloud-Native Infrastructure and AI Workloads

- ARCHITECTURE AND DESIGN
- TRAFFIC MANAGEMENT AND ROUTING
- SECURITY AND POLICY
- OBSERVABILITY AND OPERATIONS
- EXTENSIBILITY AND FILTERS
- AI AND LLM WORKLOADS
  Multi-Provider Routing, Model Selection, Streaming, Cost Optimization, and Observability



ASAHI SHINGEKI

Envoy Proxy: The Definitive Production Guide

Architecture, Configuration, Operations, and Advanced Patterns for Cloud-Native Infrastructure and AI Workloads

Steve Publications

This book is available at
https://leanpub.com/envoyproxythedefinitiveproductionguide

This version was published on 2026-07-21

Contents

Architecture, Configuration, Operations, and Advanced Patterns for Cloud-Native Infrastructure and AI Workloads

Envoy Proxy has become the foundational data plane for modern cloud-native infrastructure, powering service meshes, API gateways, edge proxies, and increasingly, AI inference platforms. This book provides a comprehensive, production-focused guide to understanding, deploying, and operating Envoy at scale. Beginning with core architectural concepts and progressing through advanced traffic management, security, observability, and extensibility patterns, every chapter delivers detailed explanations alongside real-world configuration examples. A dedicated section covers modern AI and LLM workloads, including multi-provider routing, intelligent model selection, streaming responses, cost optimization, and inference-specific observability. Whether you are a platform engineer building service meshes, an SRE operating critical infrastructure, or an architect designing AI-native systems, this book equips you with the depth needed to use Envoy effectively in production environments.

Introduction: Why Envoy Matters

The Proxy Problem in Modern Infrastructure

Every distributed system depends on proxies. They sit between clients and services, handling the unglamorous but essential work of routing requests, terminating TLS, load balancing across backends, enforcing rate limits, and collecting telemetry. In monolithic architectures, a single reverse proxy at the edge was sufficient. The world looked simple: users hit a web server, the web server talked to a database, and if traffic grew, you added another web server behind a load balancer.

Modern infrastructure is nothing like that. Microservices decompose functionality into dozens or hundreds of independently deployed components. Each service calls other services, creating complex dependency graphs where a single user request might traverse five or ten different backends before completing. Services run across multiple availability zones, multiple cloud regions, and sometimes on-premises data centers. They communicate over HTTP, HTTP/2, gRPC, WebSockets, and raw TCP protocols simultaneously.

In this environment, proxies are no longer optional infrastructure. They are the nervous system of distributed applications. And the demands placed on them have grown accordingly:

- Dynamic service discovery: services scale up and down continuously; proxies must discover healthy endpoints in real time without restarts
- Fine-grained traffic management: canary deployments, A/B tests, shadow traffic, and progressive rollouts require per-route, per-header, and per-identity routing decisions
- Resilience at scale: circuit breakers, retries with backoff, outlier detection, and timeout policies must prevent cascading failures across hundreds of services
- Zero-trust security: mutual TLS between every service pair, JWT validation, role-based access control, and external authorization for thousands of endpoints

- Observability by default: structured access logs, metrics with percentiles, distributed tracing with span propagation, all without application changes
- Extensibility: custom authentication logic, protocol translation, request/response mutations, and AI-specific features that change faster than any proxy release cycle

Traditional proxies were not built for this world. They assumed relatively stable configurations, perimeter-based security, and a clear distinction between edge traffic and internal service communication. When applied to modern microservice architectures, they required workarounds: configuration reloads causing brief outages, manual certificate management, application-level retry logic duplicated across services, observability bolted on through sidecar agents.

Envoy was designed from the ground up to solve these problems differently.

How Envoy Got Here: From Lyft to CNCF

Envoy began at Lyft in May 2015, during a critical period in the company's infrastructure evolution. Lyft was migrating away from a monolithic architecture toward microservices, and the engineering team recognized that their existing proxy infrastructure could not support the demands of what they were building.

The requirements were ambitious: a proxy that could handle all traffic at Lyft scale, support dynamic reconfiguration without restarts, provide comprehensive observability, and be extensible enough to adapt as the platform evolved. The team chose C++ for performance, designed a non-blocking event-driven architecture, and built a modular filter chain model that would become Envoy's defining abstraction.

By the end of 2015, before Envoy was even publicly announced, it was serving 100 percent of Lyft's traffic. This rapid internal adoption validated the design under real production conditions: millions of requests per second, complex service meshes, aggressive reliability requirements, and continuous deployment cycles.

On September 14, 2016, Lyft open-sourced Envoy. The project attracted immediate attention from the cloud-native community. Companies evaluating service mesh architectures saw in Envoy a data plane that could actually deliver

on the promises of zero-trust networking, fine-grained traffic management, and comprehensive observability without requiring application modifications.

In 2017, Envoy joined the Cloud Native Computing Foundation as an incubating project. The CNCF provided a neutral home for the growing community, legal support, and event infrastructure that Lyft alone could not sustain. Major technology companies began contributing: Google integrated Envoy into Istio, Amazon built App Mesh around it, and numerous others adopted it as their service mesh data plane or edge proxy.

On November 28, 2018, Envoy graduated from the CNCF, becoming only the third project to reach that status after Kubernetes and Prometheus. Graduation signaled that Envoy had achieved production maturity, sustainable governance, and broad enterprise adoption. It was no longer a Lyft project or even primarily a service mesh component; it was infrastructure in its own right.

Since graduation, Envoy's ecosystem has expanded significantly. Envoy Gateway, launched as a Kubernetes-native control plane, makes deploying Envoy as an ingress gateway straightforward through the Kubernetes Gateway API. Envoy AI Gateway, reaching version 1.0 in early 2026, extends Envoy's capabilities to generative AI workloads with multi-provider routing, token-based rate limiting, and inference-specific observability. The Gateway API Inference Extension, a joint effort between SIG-Network and WG-Serving, standardizes how gateways route traffic to self-hosted AI models using Envoy's external processing protocol.

Today, Envoy runs in production at thousands of organizations worldwide, from startups to Fortune 500 companies, handling everything from web traffic at the edge to service-to-service communication in massive microservice deployments and AI inference requests across multiple model providers.

What Envoy Is (and What It Is Not)

Understanding what Envoy is designed for helps avoid misapplication. Envoy is a high-performance, L7-capable proxy written in C++ with a modular architecture centered on filter chains. It excels at:

- Edge and ingress proxying: terminating TLS, routing based on host headers and paths, load balancing across backend services

- Service mesh data plane: deployed as sidecars alongside application containers, handling all inbound and outbound traffic with mTLS, observability, and traffic management
- API gateway functionality: rate limiting, authentication, request/response transformation, protocol translation
- Internal load balancing: distributing requests across service instances with sophisticated algorithms including consistent hashing and locality-aware routing
- Extensibility through WebAssembly filters: adding custom logic without recompiling the proxy

Envoy is not a web server. It does not serve static files, render templates, or execute application code. While it can technically do these things with sufficient configuration, that is not what it was designed for, and you will get better results using a purpose-built web server like NGINX or Caddy for those tasks.

Envoy is not a batteries-included API management platform out of the box. Unlike Kong or Apigee, which ship with built-in developer portals, analytics dashboards, monetization features, and extensive plugin ecosystems, Envoy provides the underlying proxy capabilities but expects you to compose additional services for complete API management. This is both a strength and a trade-off: you get exactly what you need without bloat, but you must invest in the integration work.

Envoy is not simple to configure directly. The raw Envoy configuration model, based on Protocol Buffers and the xDS API, has a steep learning curve. YAML configurations for production deployments can be hundreds or thousands of lines long. This complexity is what enables Envoy's power, but it is real. Most production deployments use a control plane (Istio, Envoy Gateway, Consul, etc.) that translates higher-level abstractions into Envoy configuration, sparing operators from writing raw xDS configs.

Who Should Read This Book

This book targets engineers who deploy, configure, operate, or architect systems using Envoy Proxy. Specifically:

- Platform and infrastructure engineers building service meshes, API gateways, or ingress solutions on top of Envoy

- Site reliability engineers responsible for the performance, reliability, and observability of Envoy deployments in production
- Cloud architects designing multi-region, multi-cloud infrastructure that uses Envoy as a foundational component
- DevOps practitioners integrating Envoy into CI/CD pipelines, Kubernetes clusters, or container orchestration platforms
- API platform teams building developer-facing services with rate limiting, authentication, and traffic management
- AI infrastructure engineers deploying generative AI workloads with Envoy AI Gateway or building custom inference routing solutions

The book assumes familiarity with networking fundamentals (TCP/IP, HTTP, TLS), basic understanding of microservices architecture, and some experience with containers or Kubernetes. It does not assume prior knowledge of Envoy.

How This Book Is Organized

The book is structured to take you from foundational concepts through advanced production patterns, with each chapter building on the previous ones.

The Introduction (this chapter) establishes why Envoy exists, how it evolved, and what problems it solves.

Chapter 1 covers Envoy's internal architecture: the threading model, event loop design, memory management, and the filter chain abstraction that defines everything in Envoy. It also compares Envoy to alternative proxies like NGINX, HAProxy, and Traefik.

Chapter 2 traces a request through Envoy from connection acceptance to response delivery, explaining the networking stack, protocol handling, listener filters, and connection lifecycle management.

Chapter 3 teaches the core configuration model: bootstrap files, listeners, filter chains, HTTP connection managers, routes, and virtual hosts. This is where you learn how to actually configure Envoy.

Chapter 4 covers clusters, endpoints, and service discovery mechanisms including static configurations, DNS-based discovery, and the EDS/xDS ecosystem.

Chapter 5 dives into load balancing: all supported algorithms, their trade-offs, locality-aware routing, consistent hashing strategies, and production tuning.

Chapter 6 addresses reliability patterns: active and passive health checking, outlier detection, retry policies with exponential backoff, circuit breakers with retry budgets, and timeout configuration.

Chapter 7 covers advanced traffic management: local and global rate limiting, fault injection for chaos testing, canary deployments, shadow traffic, and request mirroring.

Chapter 8 provides comprehensive security coverage: TLS termination, mutual TLS for service meshes, certificate management with SDS, JWT authentication, RBAC policies, and external authorization.

Chapter 9 explains observability: access logging formats, structured logging, Prometheus metrics, OpenTelemetry tracing integration, and building effective monitoring dashboards.

Chapter 10 covers extensibility: the filter architecture in depth, WebAssembly filter development, Lua scripting, external processing filters, and choosing the right extension mechanism.

Chapter 11 explains dynamic configuration through xDS APIs: all discovery services (LDS, RDS, CDS, EDS, SDS, and beyond), control plane patterns, versioning, and rollout strategies.

Chapter 12 covers service mesh integration with Istio and other platforms, sidecar deployment models, Kubernetes Gateway API with Envoy Gateway, and ingress/egress gateway patterns.

Chapter 13 addresses edge proxy deployments, API gateway capabilities, multi-region routing, global load balancing, geo-routing, and CDN integration strategies.

Chapter 14 is dedicated to AI and LLM workloads: the AI gateway pattern, routing to multiple providers, intelligent model selection, prompt-aware routing, A/B testing models, streaming responses, caching strategies, cost optimization, latency-aware routing, multi-region AI deployments, inference-specific observability, and securing AI workloads.

Chapter 15 provides production operations guidance: performance tuning, scalability patterns, high availability architectures, debugging with the admin

interface, troubleshooting common issues, migration strategies from other proxies, and best practices.

The Conclusion synthesizes key principles and looks ahead at where Envoy and proxy architecture are heading, particularly in the context of AI-native infrastructure.

Read the book sequentially for the complete journey, or jump to specific chapters based on your immediate needs. The configuration examples throughout are designed to be reusable: copy them, adapt them to your environment, and deploy them with confidence.

Chapter 1: Architecture and Core Design Principles

Understanding Envoy's architecture is not optional for production operators. The design decisions baked into Envoy affect how you configure it, how it performs under load, how failures propagate, and what patterns are natural versus forced. This chapter establishes the architectural foundation that every subsequent chapter builds upon.

The Filter Chain: Envoy's Central Abstraction

If you understand one concept about Envoy, make it the filter chain. The filter chain is the central abstraction through which all traffic flows and all functionality is implemented. It is not a feature of Envoy; it is the architecture of Envoy.

A filter chain is an ordered list of filters that process data as it passes through Envoy. Each filter has a specific responsibility and can inspect, modify, terminate, or forward data. Filters are processed sequentially: when one filter completes its work, control passes to the next filter in the chain. On the response path, filters are processed in reverse order.

Envoy uses two types of filter chains:

Network filter chains operate at the TCP layer. They process raw bytes from connections before protocol decoding. Examples include TLS termination (the transport socket), TCP proxying, and rate limiting at the connection level. The HTTP connection manager is itself a network filter that sits near the end of the network filter chain and handles HTTP protocol parsing.

HTTP filters operate after the HTTP connection manager has decoded the request into structured data. They work with headers, bodies, trailers, and metadata rather than raw bytes. Examples include JWT authentication, RBAC authorization, routing decisions, fault injection, and WebAssembly-based custom logic.

The power of the filter chain model is composability. Instead of building monolithic features into the proxy core, Envoy provides a framework where each capability is an independent filter. To add rate limiting to your deployment, you insert the rate limit filter into the chain. To add authentication, you insert the JWT or ext_authz filter. The filters do not need to know about each other; they operate on shared request/response objects and can communicate through dynamic metadata.

This design has several important implications:

Order matters: filters are processed in sequence, so placement affects behavior. Authentication should typically occur before authorization. Rate limiting should occur early to reject abusive requests before expensive processing. The router filter, which selects the upstream cluster, is almost always the last HTTP filter because it terminates the downstream request path and initiates the upstream connection.

Figure 1: Envoy Filter Chain Architecture

A visual representation of how traffic flows through Envoy's layered architecture:

```
[Client] --> [TCP Connection] --> [Listener]
                                        |
                                  [Listener Filters]
                                  (TLS Inspector, Proxy Protocol)
                                        |
                                  [Network Filter Chain]
                                  - Transport Socket (TLS termination)
                                  - HTTP Connection Manager
                                        |
                                  [HTTP Filters] (in order):
                               1. Local Rate Limit        <-- Early rejection
                               2. JWT Authentication      <-- Verify identity
                                  3. RBAC Authorization       <-- Check
                                  ↪ permissions
                                 4. External Authz          <-- Fine-grained
                                  ↪ policy
                                5. WASM Custom Filter       <-- Business logic
                                  6. Router                    <-- Always last
                                  ↪ (selects upstream)
                                        |
                                  [Cluster Manager] --> [Connection Pool] -->
                                  ↪ [Upstream Backend]

Response path flows in reverse through HTTP filters.
```

Key observations from this diagram:

- Listener filters run first, before the network filter chain is selected. They inspect raw bytes.
- Network filters process transport-level data (TLS, TCP proxying). The HTTP Connection Manager is itself a network filter.
- HTTP filters operate on decoded request/response objects, not raw bytes.
- The Router filter terminates the downstream path and initiates upstream communication.
- Each layer can independently inspect, modify, or reject traffic.

Failure isolation: if one filter fails or times out, it does not necessarily crash the entire proxy. Filters can return errors that propagate back to the client, and well-behaved filters handle partial data gracefully. This is essential for reliability in production.

Extensibility: new functionality can be added without modifying Envoy's core codebase. WebAssembly filters run within the same filter chain as native C++ filters, giving you the ability to deploy custom logic dynamically.

Consider a typical HTTP request path through an Envoy deployment configured as an API gateway:

1. The TLS transport socket decrypts the incoming HTTPS connection
2. The local rate limit filter checks if the client has exceeded their quota
3. The JWT authentication filter validates the bearer token
4. The RBAC filter checks if the authenticated user has permission to access this path
5. The ext_authz filter calls an external authorization service for fine-grained policy decisions
6. The router filter matches the request against route rules, selects the backend cluster, and forwards the request upstream

Each step is an independent filter. You can add, remove, or reorder them based on your requirements. This modularity is what makes Envoy powerful and is why it can serve so many different use cases from edge proxying to service mesh sidecars to AI gateways.

Threading Model and Event Loop Architecture

Envoy uses a single-process, multi-threaded architecture designed for high concurrency with minimal locking. Understanding this model is critical for performance tuning, debugging, and capacity planning.

The main thread handles coordination tasks: processing configuration updates from the control plane via xDS APIs, managing statistics, handling hot restart coordination, and running the admin interface. It does not process application traffic. This separation ensures that configuration changes and management operations do not interfere with request processing.

Worker threads handle all data plane work. The number of worker threads is configured via the –concurrency flag, which defaults to the number of available hardware threads on the host. Each worker thread runs an independent, non-blocking event loop based on libevent. On Linux systems, Envoy also integrates io_uring for asynchronous I/O, with each worker maintaining its own io_uring instance.

When a client connection is accepted by a listener, it is assigned to one worker thread and remains with that worker for the entire lifetime of the connection. All I/O for that connection, including reading requests, processing through filters, writing responses, and managing upstream connections, happens on the same worker thread. This "connection sticks to thread" model eliminates the need for locking around connection state: since only one thread ever touches a given connection's data structures, there is no contention.

This design has important consequences:

Scalability: because each worker operates independently with its own event loop and connection pools, Envoy scales efficiently across many CPU cores. Adding more workers (up to the number of hardware threads) increases throughput linearly for most workloads.

No listener sharding: all workers listen on all configured listeners. When a new connection arrives, the kernel assigns it to one worker. On Linux, the default SO_REUSEPORT behavior distributes connections relatively evenly, though perfect distribution is not guaranteed. On Windows, Envoy enforces manual balancing.

Per-worker resources: each worker thread maintains its own connection pools for upstream clusters. If you have two workers and a cluster supporting

both HTTP/1 and HTTP/2, there will be at least four connection pools (two workers times two protocol types). This means memory usage increases with the number of workers, as does the total number of connections Envoy can maintain.

Blocking operations are dangerous: because each worker runs an event loop, any blocking operation on a worker thread stalls all connections handled by that worker. DNS resolution, file I/O, and other potentially blocking operations are offloaded to dedicated thread pools or extension threads. If you write a custom filter or WebAssembly plugin, you must ensure it never blocks.

The default recommendation for –concurrency is to set it equal to the number of hardware threads available to the process. In containerized environments, this typically means matching the CPU limit configured for the pod. Setting concurrency higher than available cores adds context-switching overhead without improving throughput. Setting it lower underutilizes available CPU capacity.

Envoy also runs a file flusher thread for asynchronous log writing and other file operations, ensuring that I/O to disk does not block worker threads. For extensions like GeoIP database reloading or cache eviction, dedicated extension threads handle the work.

A watchdog mechanism monitors thread liveness. If a worker thread fails to check in within a configurable interval (default 200 milliseconds), the watchdog can escalate from logging warnings to killing the process, depending on configuration. This prevents hung workers from silently degrading service.

Memory Management and Resource Constraints

Envoy is written in C++ and manages memory explicitly rather than relying on garbage collection. This choice enables predictable performance but requires careful configuration to prevent memory exhaustion in production.

Connection buffers are one of the largest memory consumers. Each connection has buffers for reading incoming data and writing outgoing data. Under normal conditions, these buffers are small because data flows through quickly. However, if a downstream client sends data faster than the upstream backend can consume it, or vice versa, buffers grow to absorb the difference. Unbounded buffer growth can exhaust memory during traffic spikes or slow consumer scenarios.

Envoy provides several mechanisms to control memory usage:

Connection limits: listeners can be configured with max_connection_limits to cap the total number of simultaneous connections. When the limit is reached, new connection attempts are rejected with a connection refused error rather than consuming more memory.

Buffer limits: the HTTP connection manager supports configures for request and response buffer sizes. These control how much data Envoy will buffer for a single stream before applying back pressure or terminating the connection.

Overload manager: the overload manager monitors resource usage and triggers protective actions when thresholds are exceeded. It can shrink memory allocations, stop accepting new requests, or shed connections based on configurable policies. This is Envoy's primary defense against cascading memory exhaustion.

Fixed heap configuration: the bootstrap configuration supports a fixed_heap_size_bytes setting that limits total heap allocation. When this limit is approached, the overload manager takes action. This provides a hard ceiling on memory usage, though it must be set appropriately for your workload.

Circuit breakers also indirectly protect memory by limiting the number of concurrent connections and requests to upstream clusters. If a backend becomes slow or unresponsive, circuit breakers prevent Envoy from accumulating thousands of pending requests waiting for that backend.

For production deployments, you should:

Set resource limits in your container orchestrator (Kubernetes requests and limits) to bound the maximum memory available to each Envoy instance.

Configure the overload manager with thresholds appropriate to your workload. A common pattern is to trigger heap shrinking at 90 percent of the fixed heap limit and stop accepting new requests at 95 percent.

Monitor memory metrics (server.memory.heap_used_bytes, server.memory.page_faults) and set alerts for sustained high usage.

Test under realistic load conditions to understand how buffers behave with your specific traffic patterns, especially for large request/response bodies or streaming workloads.

In service mesh deployments where Envoy runs as a sidecar alongside

each application pod, memory management is particularly important because every pod has its own Envoy instance. A misconfigured sidecar that consumes excessive memory can trigger OOM kills across your entire cluster.

Process Isolation and Sandboxing Patterns

Envoy runs as a single process, which means all traffic, all listeners, and all filter chains share the same address space. This is efficient but introduces risk: a bug in one filter or a malformed request affecting one listener can potentially crash the entire process.

For production deployments, several patterns mitigate this risk:

Hot restart: Envoy supports hot restart, where a new instance takes over connections from an existing instance without dropping traffic. During a hot restart, the old process continues serving existing connections while the new process accepts new ones. If the new process crashes, the old process resumes handling all traffic. This provides safety during configuration changes and version upgrades. The trade-off is that both processes run simultaneously during the transition, temporarily doubling resource usage.

Sidecar isolation: in service mesh deployments, each application pod has its own Envoy sidecar. A crash in one sidecar affects only the traffic for that single pod, not the entire cluster. This natural isolation is one reason sidecar-based architectures are resilient.

Replication: at the edge or gateway level, run multiple Envoy instances behind a load balancer. If one instance crashes, others continue serving traffic. This is standard practice for any critical infrastructure component.

WebAssembly sandboxing: WASM filters run in an isolated virtual machine (V8 or Wasmtime) within the Envoy process. The sandbox prevents WASM code from accessing arbitrary memory or performing uncontrolled I/O. While a buggy WASM filter can still cause issues through the official API, the sandbox provides meaningful protection against malicious or unstable code.

For maximum isolation, some deployments run different responsibilities in separate Envoy processes. For example, one Envoy instance handles edge traffic with public-facing listeners and security filters, while another Envoy instance handles internal service mesh traffic. A failure in the edge proxy does not affect internal communications. This pattern adds operational complexity but improves fault containment.

Comparing Envoy to NGINX, HAProxy, and Traefik

Choosing a proxy involves trade-offs. Understanding how Envoy compares to alternatives helps you decide when Envoy is the right tool and when another solution might be better.

Envoy versus NGINX:

NGINX is the canonical reverse proxy, deployed on more websites than any other server software. It excels at serving static content, terminating TLS at scale, and providing reliable, low-overhead proxying for traditional web workloads. Its configuration model is well understood and its memory footprint is small.

Envoy differs in several key ways. NGINX uses a configuration-reload model: changes to routing rules or upstream servers require reloading the configuration, which briefly interrupts new connections. Envoy uses xDS APIs for fully dynamic configuration: routes, clusters, listeners, and certificates can change without any restart or reload. For environments with rapidly changing service topologies (microservices scaling up and down, frequent deployments), this is a decisive advantage.

NGINX's plugin ecosystem uses Lua scripts running in OpenResty. These are easy to write and widely used. Envoy's extensibility relies on WebAssembly filters or C++ native filters, which have a steeper development curve but offer better sandboxing and performance characteristics.

For raw HTTP/1.1 reverse proxy workloads with stable configurations, NGINX may use less memory and be simpler to operate. For dynamic environments requiring fine-grained traffic management, service mesh capabilities, or protocol-native support for HTTP/2 and gRPC, Envoy is better suited.

Envoy versus HAProxy:

HAProxy is legendary for its performance at Layer 4 (TCP) load balancing. It has been battle-tested for decades in some of the world's highest-traffic environments. Its configuration syntax, while idiosyncratic, provides extremely fine-grained control over connection handling.

HAProxy's strength is straightforward, high-performance load balancing with minimal overhead. Envoy's strength is Layer 7 intelligence: understanding HTTP semantics, parsing headers and paths, making routing decisions based on request content, and providing rich observability.

For pure TCP load balancing of databases or legacy protocols, HAProxy remains an excellent choice. For HTTP/gRPC traffic requiring advanced routing, authentication, rate limiting, or service mesh integration, Envoy provides more capabilities out of the box.

Performance comparisons require strict context: hardware, configuration, protocol, and enabled features all dramatically affect results. Envoy's official documentation explicitly states that no single QPS or latency figure characterizes the proxy because performance depends on which features are active and the runtime environment.

In a published benchmark by Solo.io (September 2024), a single Envoy instance with 10 CPUs on an EKS cluster achieved 25,500 requests per second under baseline conditions with zero errors across over 132 million total requests. Enabling OAuth authentication reduced throughput by approximately 24 percent; enabling rate limiting reduced it by 45 percent. This demonstrates that real-world feature usage meaningfully impacts capacity.

Other published tests show HAProxy achieving higher raw throughput for simple TCP and HTTP/1.1 load balancing. An independent benchmark on a 4-core AMD EPYC VPS (Onidel, October 2025) measured HAProxy at approximately 50,000 RPS, Nginx at 40,000+ RPS, and Envoy at 35,000+ RPS for basic proxy workloads. However, Envoy scales better on high-core systems and becomes competitive or superior for HTTP/2, gRPC, and workloads requiring advanced routing, mTLS, or dynamic configuration. Memory usage also differs: HAProxy typically consumes around 50 MB for basic configurations, NGINX around 80 MB, and Envoy around 150 MB, reflecting Envoy's richer feature set and more complex runtime.

The performance difference is rarely the deciding factor in production deployments; feature fit, dynamic configuration capabilities, and operational model matter more. Organizations choosing Envoy typically prioritize its service mesh integration, observability, and extensibility over marginal throughput advantages offered by simpler proxies.

Envoy versus Traefik:

Traefik is a modern reverse proxy designed specifically for containerized environments. It automatically discovers services from Kubernetes, Docker, and other providers, requiring minimal manual configuration. Written in Go, it has a smaller memory footprint than Envoy and integrates seamlessly with dynamic container orchestration.

Traefik's philosophy is simplicity and automation. For teams deploying microservices on Kubernetes who want an ingress controller that just works with minimal configuration, Traefik is attractive. Its middleware model provides authentication, rate limiting, and other features through straightforward YAML.

Envoy provides more advanced capabilities: sophisticated load balancing algorithms, circuit breakers with retry budgets, outlier detection with statistical analysis, comprehensive tracing integration, and the xDS ecosystem for custom control planes. The trade-off is complexity: Envoy requires more configuration knowledge and operational investment.

For typical reverse proxy workloads in Kubernetes, Traefik may be sufficient. For latency-sensitive service mesh scenarios, complex traffic management requirements, or environments where you need the full power of the Envoy ecosystem, Envoy is the better choice.

As an API gateway specifically, Envoy competes with purpose-built solutions like Kong and APISIX. These platforms provide built-in developer portals, analytics, monetization, and extensive plugin ecosystems that Envoy does not include. If you need a complete API management platform, Kong or APISIX may reduce integration work. If you prefer composing your own platform from specialized components with maximum control over performance and behavior, Envoy is the foundation to build on.

The right choice depends on your environment. There is no universally best proxy, only the proxy that best fits your specific requirements, team skills, and operational constraints.

Decision matrix for proxy selection:

The following comparison summarizes key operational dimensions to help architects make data-driven choices. Ratings use a scale of Low, Medium, or High.

Operational Overhead:

- NGINX: Low. Mature ecosystem, widely understood configuration model, extensive third-party tooling and documentation. Requires reloads for configuration changes, which adds operational complexity in dynamic environments but is straightforward to manage.

- HAProxy: Low to Medium. Simple configuration for basic load balancing; complex configurations require deep understanding of HAProxy-specific syntax. Minimal external dependencies.
- Traefik: Low. Designed for minimal configuration with automatic service discovery from Kubernetes and Docker. Ideal for teams that want an ingress controller that works out of the box.
- Envoy: High. Rich feature set requires significant configuration effort. Raw xDS configuration is complex; most production deployments use a control plane (Istio, Envoy Gateway) that adds its own operational overhead. Justified when advanced capabilities are required.

Upgrade Strategy:

- NGINX: Rolling restarts with configuration reloads. Mature upgrade paths with extensive backward compatibility. Open-source version upgrades are straightforward; Plus version requires vendor coordination.
- HAProxy: Rolling restarts. Excellent backward compatibility; configuration syntax changes rarely between major versions. Minimal disruption during upgrades.
- Traefik: Rolling restarts in Kubernetes. Go-based binary makes upgrades simple (replace the container image). Good backward compatibility for core features.
- Envoy: Hot restart capability enables zero-downtime upgrades without dropping connections. Control plane coordination required for xDS-based deployments. More complex upgrade process but superior safety guarantees when properly implemented.

Ecosystem Maturity:

- NGINX: Very High. Over 15 years of production use, massive community, extensive commercial support options (F5 NGINX Plus), and integration with virtually every platform and tool.
- HAProxy: Very High. Over 20 years of production use, battle-tested in some of the world's highest-traffic environments. Strong community but smaller commercial ecosystem than NGINX.
- Traefik: Medium to High. Rapidly growing ecosystem focused on container-native deployments. Strong Kubernetes integration. Smaller overall user base compared to NGINX and HAProxy.

- Envoy: High. Graduated CNCF project with broad enterprise adoption. Deep integration with service mesh ecosystem (Istio, Consul, Linkerd). Growing WASM plugin ecosystem. Less mature for standalone edge proxy use cases compared to NGINX.

Learning Curve:

- NGINX: Low to Medium. Configuration syntax is straightforward for basic use cases; advanced features (Lua scripting, complex routing) require more study. Abundant learning resources available.
- HAProxy: Medium. Configuration syntax is unique and requires dedicated learning. Once understood, provides extremely fine-grained control. Steeper initial curve but rewarding for experts.
- Traefik: Low. Designed for simplicity with sensible defaults and automatic discovery. Kubernetes-native teams can be productive immediately. Advanced middleware configuration adds complexity.
- Envoy: High. Configuration model is comprehensive but intricate. Understanding listeners, filter chains, clusters, routes, and xDS APIs requires dedicated study. Control planes (Istio, Envoy Gateway) abstract some complexity but introduce their own learning requirements.

Feature Summary by Use Case:

For simple reverse proxy with static configuration: NGINX or HAProxy. Both provide reliable, low-overhead proxying with minimal configuration. Choose NGINX for broader ecosystem support and documentation; choose HAProxy for raw TCP performance.

For Kubernetes ingress with automatic service discovery: Traefik or Envoy Gateway. Traefik provides the simplest experience with minimal configuration. Envoy Gateway offers more advanced features (circuit breaking, retries, mTLS) at the cost of additional complexity.

For service mesh data plane: Envoy is the clear choice. No alternative matches its dynamic configuration capabilities, observability, or integration with mesh control planes. Istio, Consul Connect, and Linkerd all use Envoy as their data plane.

For API gateway with developer portal and management features: Kong, APISIX, or Gloo Edge. These platforms provide complete API management

capabilities on top of proxy functionality. Raw Envoy requires composing additional services for a complete solution.

For AI inference gateway: Envoy AI Gateway (for Kubernetes-native deployments) or vanilla Envoy with WASM/ext_proc (for custom solutions). No alternative proxy currently provides native multi-provider LLM routing, token-based rate limiting, and inference-specific observability.

The decision should weigh your specific requirements against your team's existing expertise. A simpler proxy operated by a team that understands it thoroughly is often better than a more powerful proxy that the team struggles to configure and debug.

Chapter 2: Request Lifecycle and Networking Model

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Connection Lifecycle: From SYN to FIN

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Protocol Detection and Upgrading

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The Request Path Through Listeners and Filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Response Handling and Connection Reuse

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Graceful Shutdown and Draining

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 3: Configuration Fundamentals — Listeners, Filter Chains, and Routes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The Bootstrap Configuration File

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Listeners: Accepting Connections

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Filter Chains and Protocol Matching

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The HTTP Connection Manager

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Routes, Virtual Hosts, and Route Actions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 4: Clusters, Endpoints, and Service Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Clusters: Logical Upstream Groupings

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Endpoint Types and Resolution Modes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Static and File-Based Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

DNS-Based Service Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

EDS and the xDS Protocol Suite

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 5: Load Balancing Algorithms and Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Round Robin and Weighted Variants

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Least Connection Load Balancing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Ring Hash Consistent Hashing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Maglev and Random Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Locality-Aware and Multi-Zone Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 6: Reliability Patterns — Health Checking, Retries, and Circuit Breaking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Active and Passive Health Checking

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Outlier Detection and Ejection Policies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Retry Policies and Backoff Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Circuit Breaking and Resource Quotas

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Timeout Configuration and Deadline Propagation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 7: Traffic Management — Rate Limiting, Fault Injection, and Canaries

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Local Rate Limiting Filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Global Rate Limiting Service Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Fault Injection for Chaos Testing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Traffic Splitting and Canary Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Shadow Traffic and Request Mirroring

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 8: Security — TLS, mTLS, Authentication, and Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

TLS Termination and Configuration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Mutual TLS (mTLS) in Service Meshes

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Certificate Management and Rotation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

JWT Authentication and Validation

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

RBAC and External Authorization

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 9: Observability — Logging, Metrics, and Distributed Tracing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Access Logging: Formats and Destinations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Structured Logging for Machine Parsing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Metrics Collection with Prometheus

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Distributed Tracing Integration

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Building Observability Dashboards

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 10: Extensibility — WebAssembly, Custom Filters, and Plugins

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The Filter Architecture Deep Dive

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

WebAssembly (WASM) Filter Development

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Lua Scripting for Lightweight Extensions

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

External Processing Filters

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Choosing the Right Extension Mechanism

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 11: Dynamic Configuration and the xDS APIs

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The xDS Protocol Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Discovery Services: CDS, LDS, RDS, EDS, SDS

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Resource Versioning and Incremental Updates

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Control Plane Design Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Building a Custom xDS Control Plane

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Configuration Validation and Rollout Strategies

Chapter 12: Service Mesh Integration and Kubernetes Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Envoy as a Service Mesh Data Plane

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Istio Integration Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Sidecar Proxy Deployment Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Kubernetes Ingress and Gateway API

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Egress Gateway Patterns

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 13: Edge Proxies, API Gateways, and Multi-Region Deployments

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Envoy as an Edge Proxy

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

API Gateway Patterns and Capabilities

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Multi-Region Traffic Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Global Load Balancing and Geo-Routing

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

CDN Integration and Caching Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Chapter 14: AI and LLM Workloads — Building Intelligent Inference Gateways

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

The AI Gateway Pattern: Why Envoy Fits

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Routing to Multiple LLM Providers and Self-Hosted Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Intelligent Request Routing and Model Selection

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Prompt-Aware Routing Strategies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

A/B Testing, Canary Deployments, and Traffic Splitting for Models

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Streaming Responses and Long-Running Inference

Caching Strategies for AI Inference

Cost Optimization and Vendor Diversification

Latency-Aware Routing and Multi-Region AI Deployments

Observability for AI Inference Workloads

Securing AI Workloads with Envoy

Implementing AI Gateway Patterns with Vanilla Envoy

Chapter 15: Performance Tuning, High Availability, and Production Operations

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Performance Tuning: Threads, Buffers, and Connection Pools

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Scalability Patterns and Horizontal Scaling

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

High Availability Architectures

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Debugging Envoy in Production

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Troubleshooting Common Issues

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Advanced Troubleshooting: Complex Incident Walkthroughs

Incident 1: Intermittent Memory Growth Caused by WASM Filter Configuration Updates

Incident 2: xDS Version Skew Causing Silent Routing Failures Across Mixed Envoy Versions

Incident 3: Cascading Timeout Failures from Misconfigured Retry Budgets

Incident 4: Silent Data Loss from HTTP/2 Stream Reset Misconfiguration

Migration Strategies from Other Proxies

Production Best Practices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Real-World Case Studies

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Case Study 1: Lyft — Mitigating Cascading Failures Across Hundreds of Microservices

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Case Study 2: Dropbox — Migrating from NGINX to Envoy at Massive Scale

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Case Study 3: Netflix — Zero-Configuration Service Mesh with On-Demand Cluster Discovery

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Case Study 4: Databricks — Zero-Trust Identity Preservation with mTLS Tunnels

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Case Study 5: SAP — Modernizing Gateway Infrastructure with Envoy Gateway

Conclusion: The Future of Envoy and Proxy Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Key Principles Recap

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Emerging Patterns in Proxy Architecture

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Envoy's Role in AI-Native Infrastructure

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

Staying Current with the Ecosystem

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.

References

This content is not available in the sample book. The book can be purchased on Leanpub at https://leanpub.com/envoyproxythedefinitiveproductionguide.