APACHE APISIX

BUILDING INTELLIGENT API GATEWAYS FOR THE AI ERA

DESIGN, SECURE, AND OPERATE HIGH-PERFORMANCE GATEWAYS FOR APIS, MICROSERVICES, AND AI WORKLOADS AT SCALE

ROUTING
SMART TRAFFIC.
RIGHT DESTINATION.

SECURITY
ZERO TRUST.
BUILT IN.

OBSERVABILITY
REAL-TIME INSIGHTS.
BETTER DECISIONS.

KUBERNETES
CLOUD NATIVE.
PRODUCTION READY.

AI GATEWAY
MULTI-LLM ORCHESTRATION.
INTELLIGENT ROUTING.

PATRICK HOCHHEIMER

Apache APISIX: From API Gateway to AI Gateway

Steve Publications

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

This version was published on 2026-07-20

Contents

The Complete Guide to Building Scalable API Gateways and AI Infrastructure

About this book: Apache APISIX has evolved from a high-performance API gateway into a comprehensive platform for managing APIs, microservices, and AI workloads at scale. This book takes you from zero knowledge to production mastery, covering architecture, routing, security, observability, Kubernetes integration, and the powerful new AI Gateway capabilities for multi-LLM orchestration. Every chapter includes runnable, production-ready examples in YAML, JSON, Docker, Kubernetes, Python, Go, Java, and TypeScript. Whether you are a developer building your first route, a platform engineer designing a multi-cluster gateway, or an AI engineer routing traffic across dozens of models, this book gives you the depth and practical guidance to do it right.

Introduction

In late 2024, an e-commerce platform found itself in a familiar crisis. Their microservices architecture had grown from a dozen services to over two hundred, each with its own authentication requirements, rate limits, and versioning policies. Developers were spending more time wiring together cross-cutting concerns than building features. Then came the AI wave: their product team wanted to integrate GPT-4 for customer support, Claude for content generation, and an open-source model hosted on-premises for cost-sensitive workloads. Suddenly the infrastructure problem had a second dimension. They needed not just an API gateway, but an AI gateway.

This is not a unique story. Modern organizations face the same dual challenge: managing complex API ecosystems while simultaneously orchestrating traffic to multiple large language models. The solution has become increasingly clear: you need a platform that does both, and it must be fast enough, flexible enough, and reliable enough for production at scale. That platform, for an growing number of teams, is Apache APISIX.

Why This Book Exists

API gateways are one of those technologies that are easy to underestimate until you need them in production. A naive implementation might route requests and terminate TLS, which works fine until you have ten thousand routes, three authentication methods, per-consumer rate limits, canary deployments, distributed tracing, and compliance requirements all running in the same cluster. Suddenly the gateway is no longer a simple proxy; it is the nervous system of your entire architecture.

At the same time, AI infrastructure has its own unique demands. LLM providers have different APIs, pricing models, latency characteristics, and reliability profiles. A production AI application cannot afford to be locked into a single provider, yet managing multiple providers directly in application code creates a maintenance nightmare. You need a layer that normalizes the interface, handles failover, tracks token usage, enforces guardrails, and

provides observability. This is the AI gateway problem, and it is fundamentally similar to the API gateway problem: you need centralized, programmable traffic management with rich extensibility.

Apache APISIX addresses both problems with a single architecture. Originally created by API7.ai and open-sourced in June 2019, APISIX entered the Apache Incubator later that year and graduated to become an Apache top-level project in September 2023. It is built on Nginx and OpenResty, using LuaJIT for its plugin runtime, which gives it the raw performance of a battle-tested web server with the flexibility of a dynamic scripting language. Its latest stable release, version 3.16.0, shipped in April 2026 with over one hundred built-in plugins, native AI Gateway capabilities, and deep Kubernetes integration [1].

This book is written for the people who will actually operate APISIX in production: developers, DevOps engineers, platform engineers, solution architects, and AI engineers. It assumes you understand HTTP and containerization, but it does not assume any prior knowledge of APISIX. Every concept is explained from first principles, every configuration is accompanied by runnable examples, and every recommendation is grounded in real-world production experience.

What You Will Learn

By the end of this book, you will be able to:

- Explain APISIX architecture, including its control plane and data plane separation, etcd-based configuration management, and plugin execution model.
- Install and configure APISIX in multiple environments: Docker, Kubernetes with Helm, bare metal, and standalone mode.
- Design routing, upstream, service, and consumer configurations that form the foundation of your API infrastructure.
- Implement advanced traffic management patterns including weighted routing, canary deployments, blue-green releases, and fault injection.
- Secure your APIs with authentication plugins (API keys, JWT, OAuth2, OIDC, mTLS), authorization, rate limiting, and web application firewall rules.
- Extend APISIX functionality through its plugin ecosystem, including writing custom Lua plugins and integrating multi-language plugin runners.

- Set up comprehensive observability with Prometheus metrics, Grafana dashboards, distributed tracing via OpenTelemetry, and structured logging.
- Deploy and manage APISIX on Kubernetes using the official Ingress Controller, custom resources, and GitOps workflows.
- Build AI infrastructure that routes traffic intelligently across multiple LLM providers (OpenAI, Anthropic, Gemini, Azure OpenAI, AWS Bedrock, Ollama, and others) with load balancing, failover, token-based rate limiting, caching, guardrails, and cost tracking.
- Tune APISIX for maximum performance, design high-availability clusters, and troubleshoot production issues systematically.

How This Book Is Structured

The book is organized into thirteen chapters that build on each other progressively:

Chapter 1 establishes the problem space and explains why API gateways matter in modern architectures. Chapter 2 dives deep into APISIX internals so you understand how it works under the hood. Chapters 3 and 4 cover installation and core concepts, giving you the foundation to start building configurations. Chapter 5 explores traffic management patterns, while Chapter 6 covers security comprehensively.

Chapter 7 is dedicated to the plugin ecosystem, which is where APISIX truly shines. Chapter 8 covers observability, and Chapter 9 dives into Kubernetes integration with the Ingress Controller. Chapters 10 and 11 focus on AI Gateway capabilities, from fundamental LLM routing to advanced patterns like cost-aware orchestration and semantic caching. Chapter 12 addresses performance, scalability, and high availability. The final chapter brings everything together with operational best practices, troubleshooting guides, and real-world case studies.

Throughout the book, code examples are designed to be production-ready, not toy demonstrations. You will see complete configurations, full API requests, realistic Kubernetes manifests, and client code in multiple languages. Where relevant, I include comparisons with other gateways (Kong, Envoy, Traefik) to help you understand where APISIX fits in the landscape.

A Note on Versions and Accuracy

This book is based on Apache APISIX version 3.16.0 and the development trajectory visible in the codebase as of mid-2026. The AI Gateway plugins (ai-proxy, ai-proxy-multi, ai-rate-limiting, and related) are actively evolving, so some features may change in future releases. Where I describe behavior that is new or experimental, I note it explicitly. The official documentation at apisix.apache.org/docs should always be your final reference when configuring production systems.

Now let us begin.

Chapter 1: What Is Apache APISIX and Why It Matters

The API Gateway Problem: Why You Need One

In the early days of web development, a single application served all requests. Users hit one URL, the application processed the request, queried its database, and returned a response. This monolithic architecture was simple to understand, deploy, and operate. As organizations grew and applications became more complex, this simplicity broke down.

Microservices emerged as the solution: decompose the monolith into smaller, independently deployable services, each responsible for a specific business capability. But with decomposition came complexity. A single user action might now trigger calls to five or ten different services. Each service had its own authentication requirements, rate limits, versioning policies, and deployment schedules. Clients needed to know about multiple endpoints, handle multiple authentication flows, and manage partial failures when some services were unavailable.

This is where the API gateway pattern emerged. The API gateway sits between clients and backend services as a single entry point for all API traffic. It handles cross-cutting concerns that would otherwise need to be duplicated across every service:

- **Routing**: Directing requests to the correct backend service based on URL path, headers, methods, or other criteria.
- **Authentication and authorization**: Verifying client identity and permissions before forwarding requests.
- **Rate limiting**: Protecting backends from overload by controlling request rates per client, per API, or globally.
- **TLS termination**: Decrypting incoming HTTPS traffic so backend services can communicate over plain HTTP internally.
- **Request transformation**: Modifying headers, body content, or query parameters to match what each backend expects.

- **Observability**: Collecting metrics, logs, and traces for all API traffic in one place.
- **Caching**: Storing responses to reduce load on backends and improve response times.
- **Version management**: Routing different clients to different API versions during transitions.

Without an API gateway, each of these concerns becomes distributed code scattered across services. With a gateway, they become centralized policies that are easier to understand, modify, and audit.

The problem is not just theoretical. Consider a real scenario: you have fifty microservices, each with its own rate limiting logic implemented in application code. A new compliance requirement demands that all API keys be rotated every ninety days and logged for audit purposes. Without a gateway, you need to update and redeploy all fifty services. With a gateway, you change two plugins and the policy is enforced immediately.

Enter Apache APISIX: A Modern Approach

Apache APISIX is an open-source API gateway that addresses the limitations of earlier generations of gateways while adding capabilities for modern cloud-native and AI workloads. It was originally created by API7.ai and donated to the Apache Software Foundation in 2019. The project graduated from the Apache Incubator in July 2020 and became an Apache top-level project in September 2023, reflecting its maturity and community adoption [1].

APISIX is defined by several characteristics that distinguish it from other gateways:

Dynamic configuration without restarts. APISIX stores all routes, upstreams, plugins, and consumers in etcd, a distributed key-value store. When configuration changes are made through the Admin API, they propagate to all gateway nodes within milliseconds without requiring reloads or restarts. This is not just a convenience feature; it enables use cases like automated canary deployments, real-time rate limit adjustments, and programmatic traffic management that would be impractical with static configuration files.

High performance at scale. Built on Nginx and OpenResty with LuaJIT, APISIX inherits the event-driven architecture that makes Nginx one of the

most widely deployed web servers in the world. Official benchmarks show single-core throughput of approximately 18,000 queries per second with 0.2 millisecond average latency, and multi-core deployments reaching 140,000 QPS on an eight-core server [2]. The routing engine uses a radix tree data structure that maintains performance even with thousands of routes, unlike traversal-based approaches that degrade as route counts grow.

Rich plugin ecosystem. APISIX ships with over one hundred built-in plugins covering authentication, security, traffic management, observability, and AI integration. Plugins can be enabled, disabled, and reconfigured dynamically without restarting the gateway. The plugin architecture supports multiple languages: Lua (native), Go, Java, Python, JavaScript, and experimental WASM support through separate plugin runners.

Cloud-native design. APISIX is designed from the ground up for Kubernetes and containerized environments. It integrates with Kubernetes service discovery, supports the official Ingress Controller for declarative configuration via CRDs, and offers standalone deployment mode that works without etcd for simpler setups or custom control planes.

AI Gateway capabilities. Beginning with version 3.12.0 in early 2025, APISIX introduced a suite of AI-focused plugins that transform it into an AI gateway capable of routing traffic to multiple LLM providers, enforcing token-based rate limits, applying content guardrails, caching responses, and tracking usage across models [3].

Core Design Principles That Set It Apart

Understanding APISIX's design philosophy helps explain why it behaves the way it does and where it excels.

Separation of control plane and data plane. APISIX cleanly separates configuration management (control plane) from request processing (data plane). The control plane consists of the Admin API and etcd cluster; it handles CRUD operations for routes, plugins, consumers, and other resources. The data plane is the actual proxy that receives client requests, matches them against routes, executes plugins, and forwards traffic to backends. This separation means:

- You can scale the data plane independently by adding stateless APISIX nodes behind a load balancer.

- Configuration changes are centralized and versioned in etcd, making rollback straightforward.
- The data plane never queries a database during request processing; all configuration is cached in memory after being synced from etcd.

This architecture contrasts with gateways like Kong that traditionally used PostgreSQL or Cassandra as their configuration store, requiring the data plane to poll the database for changes. APISIX's etcd-based approach provides more consistent latency because there are no database queries on the request path [4].

Explicit over implicit. APISIX favors explicit configuration over magic. Routes explicitly define matching conditions and target upstreams. Plugins are explicitly enabled on routes, services, or globally with clearly defined configurations. While this means slightly more verbose configuration than some alternatives, it makes behavior predictable and debuggable. When something goes wrong in production, you can trace exactly which route matched, which plugins executed, and what configuration was active at the time.

Stateless data plane. Each APISIX node is stateless with respect to routing decisions and plugin logic. State lives in etcd (for configuration) or external systems like Redis (for distributed rate limiting). This statelessness makes horizontal scaling trivial: add more nodes behind a load balancer and traffic automatically distributes across them. It also simplifies upgrades because you can drain and restart individual nodes without affecting the cluster as a whole.

Hot-reload everything. Not just configuration, but plugins themselves can be hot-reloaded. If you deploy a new version of a custom Lua plugin to all APISIX nodes, the changes take effect immediately for subsequent requests without requiring an Nginx reload or worker process restart. This is made possible by Lua's dynamic module loading and APISIX's careful management of plugin lifecycle.

How APISIX Compares to Other API Gateways

When evaluating an API gateway for production use, it is important to understand how APISIX compares to established alternatives. This section contrasts APISIX with Kong, Envoy, and Traefik across the dimensions that matter most to architects and platform engineers.

Kong. Kong was the first major open-source API gateway built on Nginx and Lua. It remains widely deployed, particularly in organizations that adopted it early. The key differences:

- **Configuration store:** Kong traditionally uses PostgreSQL or Cassandra, requiring database queries on the request path for plugin execution. APISIX uses etcd with watch-based synchronization, keeping all configuration in memory after initial sync. This gives APISIX more consistent latency under load.
- **Plugin architecture:** Both use Lua plugins, but APISIX's plugin lifecycle management is more aggressive about hot-reload. Kong Enterprise adds features that are open in APISIX (such as certain rate limiting algorithms).
- **AI Gateway capabilities:** As of mid-2026, APISIX has native AI plugins (ai-proxy, ai-proxy-multi) built into the core distribution. Kong offers AI capabilities primarily through third-party plugins or enterprise add-ons.
- **Performance:** Independent benchmarks generally show APISIX with a slight edge in raw throughput and lower p99 latency, attributed to its radix tree routing and etcd-based config sync.

Kong is a mature choice with a large ecosystem. APISIX is the better fit when you need the highest performance, native AI Gateway features, or want to avoid database dependencies in your data plane.

Envoy. Envoy is a L7 proxy and communication bus originally built by Lyft, now maintained by the CNCF. It is the data plane for Istio and many service meshes. The comparison here is nuanced because Envoy and APISIX serve overlapping but different roles:

- **Architecture:** Envoy is a pure proxy written in C++. APISIX is Nginx with Lua scripting layered on top. Envoy has no built-in scripting; customization happens through filters (written in C++ or via WASM) or external control planes.
- **Ease of use:** APISIX provides a higher-level abstraction for API management tasks like authentication, rate limiting, and consumer management. Envoy requires assembling these capabilities from individual filters and an external control plane (xDS server).
- **Performance:** Envoy is extremely performant for its class, particularly for gRPC and mTLS workloads. For pure HTTP routing with plugins, APISIX's LuaJIT approach is competitive and often simpler to operate.

- **AI Gateway:** Neither has deep native AI features out of the box, though APISIX's ai-proxy plugins provide a ready-made solution that would require custom development on Envoy.

Envoy is the right choice when you need a service mesh data plane or fine-grained network-level control. APISIX is better for API management scenarios where you want built-in plugins, consumer management, and a simpler operational model. Many production architectures use both: Envoy for east-west traffic between services, APISIX for north-south ingress.

Traefik. Traefik is a cloud-native edge router popular in Kubernetes environments. It is written in Go and has deep Kubernetes integration.

- **Kubernetes integration:** Both Traefik and APISIX have strong Kubernetes support through Ingress Controllers. Traefik's configuration model is simpler for basic use cases; APISIX's custom resources provide more granular control over plugins and traffic management.
- **Plugin ecosystem:** Traefik has a growing plugin system (primarily Go-based). APISIX's plugin ecosystem is larger, with one hundred plus built-in plugins and support for Lua, Go, Java, Python, and WASM.
- **Performance:** Traefik is optimized for Kubernetes discovery speed and dynamic routing. APISIX generally has higher raw throughput due to its Nginx foundation.
- **AI Gateway:** APISIX's native AI plugins give it a clear advantage for LLM traffic management. Traefik would require custom middleware development.

Traefik is an excellent choice for simple Kubernetes ingress with automatic service discovery. APISIX is better when you need advanced API management features, a rich plugin ecosystem, or AI Gateway capabilities.

Comparison summary:

Dimension	APISIX	Kong	Envoy	Traefik
Core technology	Nginx + LuaJIT	Nginx + Lua	C++	Go
Config store	etcd (in-memory cache)	PostgreSQL/Cassandra	xDS (external)	In-memory / Kubernetes API
Dynamic config speed	Milliseconds (watch-based)	Seconds (polling DB)	Fast (xDS push)	Fast (Kubernetes events)
Plugin count (built-in)	100+	70+	Filters (not plugins)	~30
Plugin languages	Lua, Go, Java, Python, WASM	Lua, Go, JS	C++, WASM	Go
Hot-reload plugins	Yes	Limited	No (requires restart/WASM)	Yes (Go plugins)
AI Gateway native support	Yes (ai-proxy, ai-proxy-multi)	Third-party / Enterprise	Custom development needed	Custom middleware needed
Kubernetes Ingress Controller	Yes (official)	Yes	Yes (via gateway-api)	Yes (primary use case)
Standalone mode (no etcd)	Yes	No	Yes	Yes
Best for	API + AI gateways, high throughput	Established Kong shops	Service mesh, microservice proxy	Simple K8s ingress

The choice ultimately depends on your specific requirements. If you need a gateway that handles both traditional APIs and LLM traffic with maximum performance and minimal operational overhead, APISIX is among the strongest

options available.

Where APISIX Fits in Your Architecture

APISIX can be deployed in several architectural positions depending on your needs:

Edge gateway. At the perimeter of your infrastructure, APISIX terminates TLS, handles authentication, applies rate limits, and routes traffic to internal services. This is the classic API gateway use case. Clients (web browsers, mobile apps, third-party integrations) send requests to a single public endpoint, and APISIX distributes them across your backend ecosystem.

Internal service mesh complement. While service meshes like Istio handle east-west traffic between microservices with mTLS and fine-grained routing, APISIX can sit at the north-south boundary handling external traffic. The two patterns are complementary: APISIX manages the public-facing API layer while the service mesh handles inter-service communication. In some architectures, APISIX replaces the need for a full service mesh by providing sufficient traffic management capabilities without the complexity.

AI gateway. For AI applications, APISIX sits between your application code and LLM providers. Your application sends requests to a single APISIX endpoint using the OpenAI-compatible API format. APISIX then routes to whichever provider or model is appropriate based on routing rules, load balancing policies, token budgets, and health checks. This abstraction means your application does not need to know about multiple API keys, different authentication mechanisms, or provider-specific quirks.

Kubernetes Ingress. When deployed with the APISIX Ingress Controller, it becomes the ingress layer for your Kubernetes cluster. Instead of configuring routes through the Admin API, you define ApisixRoute custom resources that the controller translates into APISIX configuration. This fits naturally into GitOps workflows where infrastructure is managed as code.

How This Book Is Organized

The remaining chapters follow a logical progression from foundations to mastery:

- **Chapter 2: Architecture Deep Dive** explains how APISIX works internally, including the Nginx/Lua foundation, configuration sync mechanisms, and request processing pipeline.
- **Chapter 3: Installation and Getting Started** walks through multiple installation methods with production-ready configurations.
- **Chapter 4: Core Concepts** covers routes, upstreams, services, and consumers in detail with complete examples.
- **Chapter 5: Routing, Load Balancing, and Traffic Management** explores advanced routing strategies, load balancing algorithms, canary deployments, and traffic splitting.
- **Chapter 6: Authentication, Authorization, and Security** provides comprehensive coverage of securing APIs through APISIX.
- **Chapter 7: The Plugin Ecosystem** explains how plugins work and how to use and write them.
- **Chapter 8: Observability, Logging, and Monitoring** covers metrics, tracing, dashboards, and alerting.
- **Chapter 9: Kubernetes Integration and Ingress Controller** dives deep into running APISIX on Kubernetes.
- **Chapters 10 and 11: AI Gateway** cover LLM routing fundamentals and advanced patterns including multi-provider orchestration, cost optimization, caching, guardrails, and token tracking.
- **Chapter 12: Performance, Scalability, and High Availability** addresses running APISIX at scale.
- **Chapter 13: Production Operations** brings everything together with testing strategies, CI/CD, troubleshooting, security hardening, and real-world case studies.

Each chapter builds on the previous ones, but you can also treat chapters as reference material once you are familiar with the basics. The code examples throughout are designed to be copy-paste runnable, so you can experiment alongside the explanations.

The journey ahead will take you from understanding what an API gateway does to mastering one of the most capable open-source gateways available today. Let us start by looking under the hood.

Chapter 2: Architecture Deep Dive

To use Apache APISIX effectively in production, you need to understand how it works under the hood. This chapter explains the architectural foundations that make APISIX fast, dynamic, and extensible. If you only read one chapter before deploying to production, this should be it.

The Nginx Foundation: Lua and OpenResty

APISIX is built on Nginx, specifically the OpenResty distribution, which bundles Nginx with a rich set of Lua libraries. This combination provides the performance characteristics of a battle-tested C-based web server with the flexibility of an embedded scripting language.

Why Nginx? Nginx uses an event-driven, non-blocking architecture that handles thousands of concurrent connections with minimal memory overhead. Unlike traditional web servers that create a thread or process per connection, Nginx uses a small number of worker processes that multiplex I/O across all active connections using epoll on Linux (or kqueue on BSD systems). This design scales efficiently to very high concurrency with predictable resource usage.

For an API gateway, this matters enormously. Gateways sit in the critical path of every request; they must handle connection spikes without queuing or dropping traffic. Nginx's event loop architecture is ideal for this workload because it keeps latency low even under heavy load.

Why OpenResty and Lua? Pure Nginx configuration is powerful but static. Changing routing rules, enabling plugins, or adjusting rate limits requires editing configuration files and reloading the server, which introduces downtime risk and operational complexity. OpenResty solves this by embedding LuaJIT, a just-in-time compiler for Lua that compiles hot code paths to native machine instructions at runtime.

LuaJIT typically runs 2 to 10 times faster than standard Lua interpreters and approaches C performance for tight loops. This means APISIX can execute complex routing logic, plugin chains, and protocol transformations with

minimal overhead. In practice, the latency added by running Lua plugins is measured in microseconds rather than milliseconds.

APISIX uses the ngx_lua module (provided by OpenResty) to hook Lua code into Nginx's request processing lifecycle. When a request arrives, Nginx handles the low-level socket I/O and HTTP parsing, then calls into APISIX's Lua code for routing decisions, plugin execution, and proxying. This hybrid approach gives you the best of both worlds: C-level performance for I/O and protocol handling, plus dynamic scripting for business logic.

The role of etcd. While Nginx handles the data plane and Lua provides the scripting layer, APISIX needs a distributed configuration store to enable dynamic updates across multiple nodes. It uses etcd, a strongly consistent key-value store originally developed by CoreOS.

etcd serves three critical functions in APISIX:

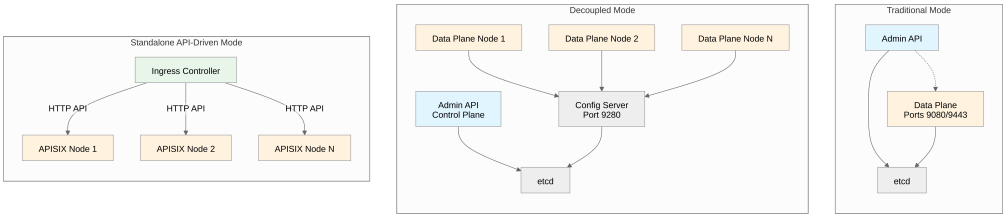
1. **Single source of truth**: All routes, upstreams, services, consumers, plugins, SSL certificates, and stream routes are stored as JSON documents in etcd under the /apisix prefix.
2. **Watch-based synchronization**: APISIX nodes subscribe to watch streams on etcd keys. When configuration changes, etcd pushes updates to all subscribed nodes within milliseconds, eliminating the need for polling or manual reloads.
3. **High availability**: etcd runs as a cluster using the Raft consensus algorithm. As long as a majority of nodes are available, the cluster remains operational and configuration changes can be committed.

This architecture means APISIX achieves millisecond-level configuration propagation without database queries on the request path, which is a key differentiator from gateways that use relational databases [1].

Control Plane vs Data Plane Separation

APISIX supports three deployment modes that represent different tradeoffs between simplicity and scalability: traditional, decoupled, and standalone. Understanding these modes is essential for designing your production architecture.

The following diagram illustrates the three deployment modes side by side:



Traditional mode. In traditional mode, each APISIX instance runs both the control plane (Admin API) and the data plane (traffic proxy) in the same process. The configuration looks like this:

```yaml
# conf/config.yaml
deployment:
  role: traditional
  role_traditional:
    config_provider: etcd

admin:
  admin_listen:
    host: 0.0.0.0
    port: 9180
  admin_key:
    - name: "admin"
      key: "your-write-key-here"
      role: admin

apisix:
  node_listen:
    - port: 9080      # HTTP
    - port: 9443      # HTTPS

etcd:
  host:
    - "http://127.0.0.1:2379"
  prefix: "/apisix"
  timeout: 30
```

Traditional mode is simple to deploy and operates well for small to medium clusters. The Admin API listens on port 9180, the data plane on ports 9080 (HTTP) and 9443 (HTTPS), and all configuration is stored in etcd. However, running both planes together means the Admin API shares resources with the traffic proxy, which can be a concern under heavy load.

Decoupled mode. Decoupled mode separates the control plane into dedicated instances that manage configuration, while data plane instances focus

solely on request processing. The control plane instance runs the Admin API and syncs configuration to etcd; data plane instances subscribe to etcd watches but do not expose the Admin API.

Control plane configuration:

```yaml
deployment:
  role: control_plane
  role_control_plane:
    config_provider: etcd
    conf_server:
      listen:
        host: 0.0.0.0
        port: 9280

admin:
  admin_listen:
    host: 0.0.0.0
    port: 9180
```

Data plane configuration:

```yaml
deployment:
  role: data_plane
  role_data_plane:
    config_provider: conf_server
    conf_server:
      host: "http://control-plane:9280"
      sync_interval: 1

apisix:
  node_listen:
    - port: 9080
    - port: 9443
```

Decoupled mode offers several advantages for production environments:

- The Admin API is isolated from traffic processing, reducing attack surface and resource contention.
- You can scale data plane nodes independently based on traffic volume.
- Data plane nodes do not need direct access to etcd, simplifying network security policies.

Standalone mode. Standalone mode eliminates etcd entirely, loading configuration from a local file or receiving it through an API-driven interface. This mode has two variants:

File-driven standalone:

```
deployment:
  role: standalone
  role_standalone:
    config_provider: yaml
    config:
      path: "/usr/local/apisix/conf/apisix.yaml"
      reload_interval: 1
```

API-driven standalone (for Ingress Controller integration):

```
deployment:
  role: standalone
  role_standalone:
    config_provider: api
    api:
      listen:
        host: 0.0.0.0
        port: 9280
```

Standalone mode is designed for Kubernetes environments where the Ingress Controller manages configuration, or for simpler deployments where etcd adds unnecessary complexity. In file-driven mode, APISIX checks the configuration file every second and applies changes to memory without restarts. The YAML file must end with a `#END` marker so APISIX can detect truncation during writes.

For most production Kubernetes deployments using the Ingress Controller, API-driven standalone mode is recommended because it provides versioned configuration updates that prevent race conditions when multiple controllers are running.

Configuration Storage and Dynamic Updates

APISIX stores all runtime configuration as JSON documents in etcd. Understanding this data model helps you debug issues and design automation tools.

The etcd key structure follows a predictable pattern:

- `/apisix/routes/`: Route definitions
- `/apisix/upstreams/`: Upstream server pools
- `/apisix/services/`: Service abstractions
- `/apisix/consumers/`: Consumer identities and credentials
- `/apisix/plugins/`: Global plugin configurations
- `/apisix/plugin_configs/`: Reusable plugin configuration sets
- `/apisix/ssl/`: SSL certificates for TLS termination
- `/apisix/stream_routes/`: TCP/UDP stream routes
- `/apisix/global_rules/`: Plugins applied to all routes
- `/apisix/secrets/`: Encrypted secrets (for use with vault integration)

Each resource is stored under a UUID key, and the JSON value contains the full configuration. For example, a route might look like:

```
Key: /apisix/routes/a1b2c3d4-e5f6-7890-abcd-ef1234567890
Value:
{
  "uri": "/api/v1/users",
  "methods": ["GET", "POST"],
  "upstream_id": "u1a2b3c4-d5e6-f789-abcd-ef0987654321",
  "plugins": {
    "key-auth": {},
    "limit-count": {
      "count": 100,
      "time_window": 60
    }
  },
  "status": 1
}
```

How dynamic updates work. When you create or modify a resource through the Admin API, the following sequence occurs:

1. The Admin API validates the request against a JSON schema and persists it to etcd.
2. APISIX nodes subscribed to watches on the relevant prefix receive the update notification from etcd.
3. Each node parses the new configuration, compiles route matching rules into its radix tree, and updates in-memory data structures.
4. Subsequent requests use the updated configuration immediately.

This entire process completes within milliseconds for typical configurations. There is no reload, no restart, no brief period of unavailability. The radix tree rebuild is fast because it operates entirely in memory and only affects the changed routes.

etcd connection modes. APISIX supports two modes for communicating with etcd:

- **HTTP mode**: Uses HTTP/2 to poll and watch etcd. Simple but can introduce latency under heavy configuration churn.
- **gRPC mode**: Uses etcd's gRPC API for more efficient streaming watches. Recommended for large clusters or environments with frequent configuration changes.

Enable gRPC mode in config.yaml:

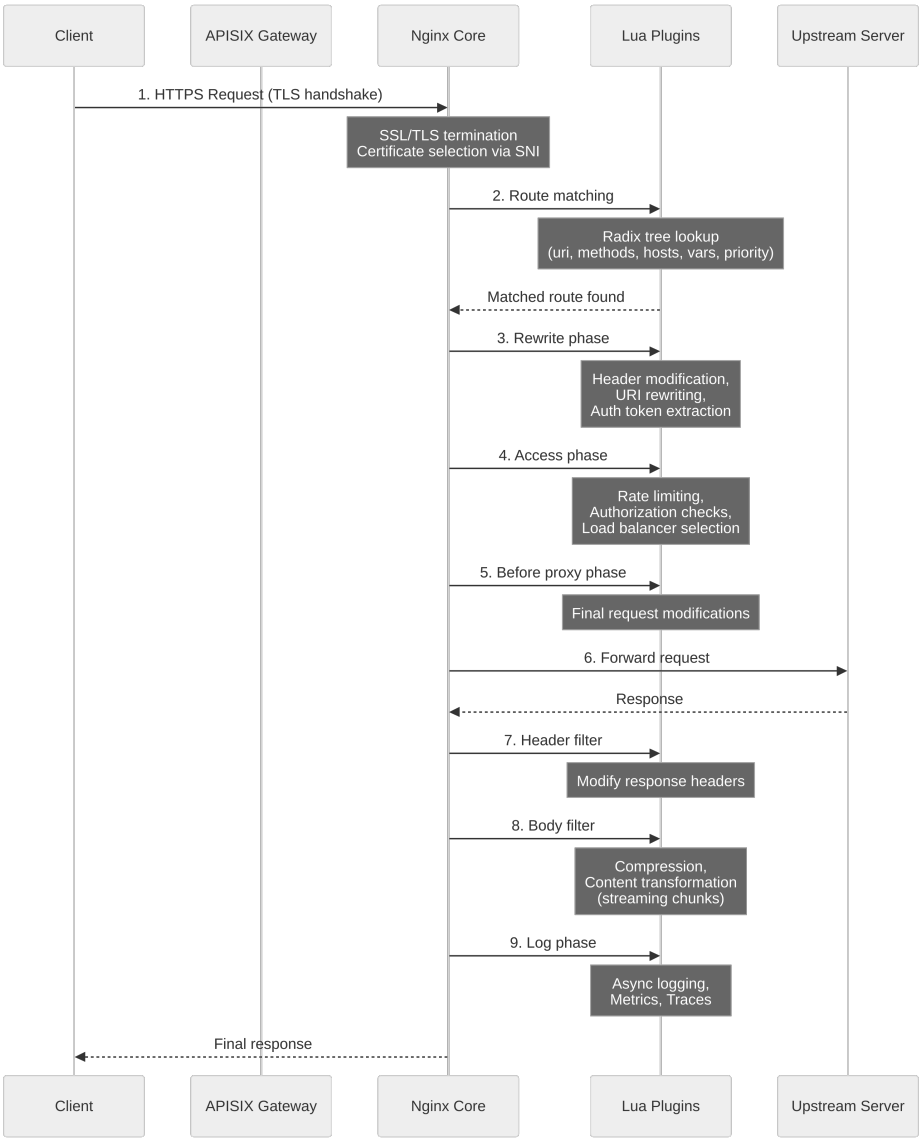
```yaml
etcd:
  host:
    - "http://127.0.0.1:2379"
  use_grpc: true
```

gRPC mode reduces the latency of configuration propagation and decreases CPU overhead on etcd nodes, which matters when you have hundreds of APISIX instances watching for changes.

The Request Processing Pipeline

When a request arrives at APISIX, it goes through a well-defined pipeline. Understanding this pipeline is crucial for debugging issues, optimizing performance, and writing plugins.

The following diagram shows the complete request lifecycle from client to upstream and back:



Phase 1: SSL/TLS handshake. If the request arrives on an HTTPS port, Nginx performs TLS termination. APISIX matches the SNI (Server Name Indication) against configured SSL resources to select the appropriate certificate. This step happens before any Lua code runs.

Phase 2: Route matching. APISIX examines the request and searches its radix tree for a matching route. Matching considers:

- URI path prefix or exact match

- HTTP methods (GET, POST, PUT, DELETE, etc.)
- Host headers
- Variables (request headers, query parameters, client IP, consumer name)
- Priority (when multiple routes could match, the one with higher priority wins)

The radix tree algorithm ensures route matching is O(k) where k is the length of the URI path, regardless of how many routes are configured. This is why APISIX maintains performance with tens of thousands of routes while traversal-based gateways degrade.

If no route matches, APISIX returns a 404 Not Found response.

Phase 3: Plugin execution (rewrite phase). Once a route is matched, APISIX executes plugins registered for the `rewrite` phase. Plugins in this phase can:

- Modify request headers
- Rewrite the URI path
- Extract and validate authentication tokens
- Perform early rejections based on security rules

Phase 4: Plugin execution (access phase). Plugins in the `access` phase perform checks that must complete before the request is forwarded. This includes:

- Rate limiting decisions
- Authorization checks
- Request body validation
- Load balancer selection

The access phase is where most of the "business logic" plugins run because they may need to make blocking calls (to Redis for rate limit counters, for example).

Phase 5: Before proxy. The `before_proxy` phase runs just before APISIX forwards the request to the upstream server. Plugins here can do final modifications to the request or customize proxy settings.

Phase 6: Upstream proxying. Nginx sends the request to the selected upstream server and waits for the response. This is handled by Nginx's C code for maximum performance.

Phase 7: Header filter. When the upstream response arrives, APISIX executes `header_filter` plugins that can modify response headers.

Phase 8: Body filter. The `body_filter` phase processes the response body in chunks as it streams from the upstream. This is where compression, encryption, or content transformation plugins operate.

Phase 9: Log phase. Finally, APISIX executes `log` plugins that write access logs, metrics, and traces. These run asynchronously to avoid impacting response latency.

This phased execution model gives you fine-grained control over when plugins run. A rate limiting plugin should run in the access phase (before forwarding), while a logging plugin should run in the log phase (after the response is sent).

Plugin Execution Model and Phases

Plugins are the mechanism by which APISIX extends its core functionality. Each plugin is a Lua module that implements one or more of the phases described above. When you enable a plugin on a route, service, or globally, APISIX registers it for execution in the appropriate phases.

Plugin hierarchy. Plugins can be enabled at three levels with increasing scope:

1. **Route-level**: Applied only to matching requests on that specific route.
2. **Service-level**: Applied to all routes associated with a service.
3. **Global-level**: Applied to all requests regardless of route (via the global_-rules resource).

When a request is processed, APISIX collects plugins from all applicable levels and executes them in order: global first, then service, then route. This allows you to define baseline policies globally (e.g., prometheus metrics collection on all routes) while adding specific policies per route (e.g., stricter rate limits for sensitive endpoints).

Execution order within a phase. Within each phase, plugins execute in a deterministic order based on their registered priority. APISIX assigns priorities that reflect typical usage patterns:

- Authentication plugins run early in the access phase so unauthorized requests are rejected before expensive processing.
- Rate limiting runs after authentication so you can apply different limits per consumer.
- Logging runs last in the log phase to capture all information about the request.

You can inspect the execution order by enabling debug mode and examining the `Apisix-Plugins` response header, which lists plugins in the order they executed.

Multi-language plugin runners. While Lua plugins run natively inside APISIX with minimal overhead, some teams prefer to write plugins in other languages. APISIX supports this through separate plugin runner processes:

- **Java Plugin Runner**: For Java/Kotlin plugins
- **Go Plugin Runner**: For Go plugins
- **Python Plugin Runner** (experimental): For Python plugins
- **WASM Runtime** (experimental): For WebAssembly plugins

Multi-language runners communicate with APISIX over HTTP or gRPC. When a request reaches a phase that has a non-Lua plugin, APISIX forwards the relevant context to the runner process, which executes the plugin and returns the result. This adds latency compared to native Lua plugins (typically 0.5 to 2 milliseconds per call) but enables teams to use their preferred language and existing libraries.

Plugin hot-reload. When you deploy a new version of a Lua plugin, APISIX can reload it without restarting workers. The mechanism works as follows:

1. New plugin code is written to the file system.
2. APISIX detects the file change through its internal file watcher.
3. On the next request that uses the plugin, APISIX unloads the old module and loads the new one using Lua's package.loaded clearing mechanism.
4. Subsequent requests use the updated code immediately.

This hot-reload capability is invaluable for production environments where you want to deploy bug fixes or feature updates without service disruption.

Understanding this architecture gives you the mental model needed to reason about how APISIX behaves under load, why configuration changes take effect instantly, and how plugins interact with each other. With this foundation, let us move on to getting APISIX running in your environment.

Chapter 3: Installation and Getting Started

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

Docker and Docker Compose Setup

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

Helm Chart Deployment on Kubernetes

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

Source Build and Binary Installation

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

Verifying Your Installation and Health Checks

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

Initial Configuration and Admin API Access

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

Chapter 4: Core Concepts: Routes, Upstreams, Services, and Consumers

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

Routes: Matching and Directing Traffic

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

Upstreams: Defining Your Backend Servers

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

Services: Abstracting Route and Upstream Relationships

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

Consumers: Identity and Access Management

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

Putting It Together: A Complete API Flow

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

Client Code Examples: Calling APISIX-Protected APIs

Chapter 5: Routing, Load Balancing, and Traffic Management

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

Advanced Route Matching and Priority

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

Load Balancing Algorithms: Round Robin, Least Connections, and More

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

Weighted Routing and Traffic Splitting

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

Canary Deployments and Progressive Delivery

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

Fault Injection and Chaos Testing with APISIX

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

Chapter 6: Authentication, Authorization, and Security

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

Authentication Plugins: Keys, JWT, OAuth2, mTLS, and More

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

Authorization and Access Control Patterns

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

Rate Limiting and Quotas: Protecting Your Backends

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

Web Application Firewall and Request Validation

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

TLS Termination, mTLS, and Certificate Management

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

Chapter 7: The Plugin Ecosystem: Extending APISIX

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

How Plugins Work: Phases, Hooks, and Execution Order

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

Built-in Plugins: A Curated Guide to the Essentials

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

Third-Party and Community Plugins

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

Writing Custom Lua Plugins

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

Plugin Performance Considerations and Best Practices

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

Chapter 8: Observability, Logging, and Monitoring

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

Access Logs, Error Logs, and Structured Logging

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

Prometheus Metrics and Export Configuration

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

Distributed Tracing with Jaeger, Zipkin, and OpenTelemetry

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

Grafana Dashboards and Alerting Setup

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

AI Observability: Tracking Token Usage, Latency, and Costs

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

Chapter 9: Kubernetes Integration and Ingress Controller

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

Deploying APISIX with Helm on Kubernetes

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

The APISIX Ingress Controller: Architecture and Setup

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

Custom Resources: ApisixRoute, ApisixUpstream, and More

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

Managing TLS, Secrets, and ConfigMaps

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

Multi-Tenancy and Namespace Isolation

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

Chapter 10: Apache APISIX AI Gateway: LLM Routing Fundamentals

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

Why You Need an AI Gateway for LLM Traffic

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

The APISIX AI Gateway Architecture

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

OpenAI-Compatible Protocol and Provider Abstraction

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

Multi-LLM Routing: Sending Requests to Different Models

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

Provider Failover and High Availability for AI Workloads

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

Chapter 11: Advanced AI Gateway Patterns and Optimization

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

Cost-Aware and Latency-Aware Intelligent Routing

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

Load Balancing Across LLM Providers and Models

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

Prompt and Response Transformation Pipelines

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

Streaming Responses and SSE Support

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

Token Usage Tracking, Budgeting, and Cost Attribution

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

AI Caching Strategies: Exact Match and Semantic Cache

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

Guardrails, Content Filtering, and Safety Plugins

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

Chapter 12: Performance, Scalability, and High Availability

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

Performance Benchmarks and Capacity Planning

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

Tuning Nginx and Lua for Maximum Throughput

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

Horizontal Scaling and Cluster Topology

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

High Availability Patterns and Failover Strategies

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

Disaster Recovery and Backup Procedures

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

Capstone Project: End-to-End AI-Powered Microservice Platform on Kubernetes

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

Chapter 13: Production Operations: Testing, CI/CD, Troubleshooting, and Best Practices

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

Testing Strategies: Unit, Integration, and E2E Tests for APISIX Configs

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

CI/CD Pipelines for API Gateway Configuration

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

Common Pitfalls and Anti-Patterns to Avoid

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

Troubleshooting Guide: Diagnosing Real Production Issues

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

Security Hardening Checklist and Compliance Considerations

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

Real-World Deployment Case Studies

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

Conclusion: The Future of API and AI Gateways

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

The Convergence of API and AI Gateways

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

Trends to Watch

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

What You Should Take Away

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

Final Thoughts

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

References

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