Apache APISIX API Gateway Architecture

Apache APISIX API Gateway Architecture – Data Plane, Control Plane, and API Types

Apache APISIX is a **dynamic, real-time, high-performance API gateway** and **cloud-native** platform that provides rich traffic management features such as load balancing, dynamic upstream, authentication, observability, and more.

Apache APISIX is a **high-performance, cloud-native API gateway** built on a robust stack of technologies designed for **scalability, dynamic configuration, and extensibility**. Below is a detailed breakdown of its core components and architecture:

**Foundation: APISIX is built on top of OpenResty, which extends Nginx with Lua scripting.**

**Meaning:**

* **Nginx** is a high-performance web server and reverse proxy commonly used for serving websites and balancing traffic.
* **OpenResty** is a platform that extends Nginx by adding support for the Lua scripting language, allowing you to write custom logic that runs inside the Nginx server.
* **APISIX** uses OpenResty as its foundation. This means:
  + APISIX inherits Nginx’s high performance, reliability, and event-driven architecture.
  + By using OpenResty, APISIX can run custom Lua scripts to implement advanced API gateway features like dynamic routing, plugin hot-loading, real-time traffic policies, and more.

**In short:**

APISIX leverages the speed and scalability of Nginx, and the flexibility of Lua scripting (via OpenResty), allowing powerful customizations and plugin development not possible in plain Nginx.

**1. Core Technologies Behind APISIX**

**a) Nginx (OpenResty)**

* **Foundation**: APISIX is built on top of **OpenResty**, which extends **Nginx** with Lua scripting.
  + **Why Nginx?**
    - Handles **high concurrency** with low resource usage (event-driven architecture).
    - Supports **L4/L7 traffic** (TCP/UDP/HTTP/HTTPS, gRPC, WebSockets, etc.).
  + **Why OpenResty?**
    - Adds **LuaJIT** for dynamic scripting inside Nginx.
    - Enables **plugin system** (APISIX plugins are mostly written in Lua).

**b) etcd (Configuration & Service Discovery)**

* **Role**: APISIX uses **etcd** (a distributed key-value store) as its **primary configuration backend**.
  + **Why etcd?**
    - Provides **real-time updates** (no reloads needed for config changes).
    - Supports **high availability** (distributed consensus via Raft).
    - Used for storing:
      * Routes
      * Upstreams (load balancing configs)
      * Plugin configurations
      * SSL certificates

**c) Lua & Plugin System**

* **Lua** is the primary scripting language for APISIX plugins.
  + **Why Lua?**
    - Lightweight, fast (JIT-compiled via LuaJIT).
    - Seamless integration with Nginx/OpenResty.
  + **Plugins** can modify requests/responses, add auth, logging, etc.

**d) Additional Dependencies**

* **Prometheus** (metrics collection).
* **Zipkin/SkyWalking** (distributed tracing).
* **Redis** (rate limiting, caching).
* **Kafka** (log streaming).

**2. APISIX Architecture (Deep Dive)**

**a) Data Plane vs. Control Plane**

| **Component** | **Role** | **Technology** |
| --- | --- | --- |
| **Data Plane** | Handles actual API traffic (L4/L7 proxy). | Nginx + Lua |
| **Control Plane** | Manages configs (routes, plugins, etc.). | etcd + Admin API |

**b) How APISIX Processes a Request**

1. **Request hits APISIX** → Matched against **routes** (stored in etcd).
2. **Plugins execute** (auth, rate limiting, transformations).
3. **Load balancing** → Forwarded to upstream (dynamic or static).
4. **Response returned** with optional logging/metrics.

**c) Dynamic Updates (No Reloads!)**

* Traditional Nginx requires reloads (nginx -s reload) for config changes.
* **APISIX avoids this** by:
  + Storing configs in **etcd** (real-time sync).
  + Using **Lua code** to apply changes on-the-fly.

**3. Why This Stack?**

**a) Performance**

* **Nginx** (C + LuaJIT) → Handles **millions of requests/sec** with low latency.
* **etcd** → Ensures configs are distributed and consistent.

**b) Extensibility**

* **Lua plugins** → Custom logic without recompiling.
* **Multi-language plugins** (Go, Java, Python via WASM).

1. Planes in APISIX

Data Plane (DP)

- Location in Diagram: Left side

- Purpose: Handles all actual API request and response processing.

- Responsibilities:

- Receives requests from clients/services

- Applies plugins (rate limiting, authentication, logging, etc.)

- Proxies and routes traffic to backend services

- Returns responses to clients

Control Plane (CP)

- Location in Diagram: Right side

- Purpose: Manages configuration, orchestration, and observability of the API gateway.

- Responsibilities:

- Stores and manages configuration (routes, plugins, credentials)

- Provides admin UI (dashboard/Manager API)

- Synchronizes configuration to Data Plane nodes

- Exports data to monitoring/observability tools

2. Data Plane Components

- Client/Service:

End-users or client applications making API requests.

- Apache APISIX Gateway:

The main API gateway engine (orange box), built on top of NGINX, handling incoming traffic, enforcing policies, and proxying requests.

- Plugin Layer:

- Rate Limit: Controls request rates.

- Auth: JWT, OAuth2, Key-auth, etc.

- Security: Access controls, IP restrictions.

- Logging: Captures API access logs.

- Custom Plugins: Extend APISIX functionality as needed.

- Public / Private / Partner:

Categories of APIs/services exposed via APISIX, each with different access and security policies.

3. Control Plane Components

- etcd Cluster:

Distributed, highly available storage for all APISIX configuration and metadata.

- Dashboard / Manager API:

Admin interfaces to configure and manage APISIX, which send updates to etcd.

- Observability & Monitoring Stack:

- Apache Skywalking, Prometheus, Grafana:

For tracing, monitoring, and visualizing logs/metrics collected from APISIX.

4. Flow Overview

API Request Flow:

1. Clients send API requests to APISIX.

2. APISIX processes the request via enabled plugins.

3. Requests are routed to the appropriate backend service (public, private, or partner APIs).

Configuration Flow:

1. Admins manage configs via Dashboard/Manager API.

2. Configurations are stored in etcd.

3. APISIX dynamically pulls configs from etcd—no restart needed.

Monitoring Flow:

1. APISIX exports monitoring data to Prometheus, Grafana, and Skywalking for real-time operational visibility.

5. Public, Private, and Partner APIs in APISIX

| API Type | Exposed To | Example Security | Example Use |

|-----------|-----------------------|--------------------------|------------------------------|

| Public | Anyone/external users | Basic Auth, Rate Limit | Open data, public search |

| Private | Internal teams/apps | mTLS, JWT, LDAP | Microservices, internal ops |

| Partner | Trusted third parties | API Key, OAuth, Quotas | B2B integration, supply chain|

How APISIX Handles These:

- Routing: Defines which APIs are public, private, or partner.

- Policy Enforcement: Applies appropriate plugins for auth, rate limiting, logging, etc., for each API category.

- Access Control: Ensures only authorized users or partners can access non-public APIs.

6. Summary Table

| Plane | Primary Role | Typical Components |

|---------------|----------------------------------|-------------------------------------|

| Data Plane | Handles API requests & policies | APISIX Gateway, Plugins |

| Control Plane | Config & monitoring management | etcd, Dashboard/API, Monitoring tools|

7. Summary Points

- Data Plane: Real API traffic processing and enforcement of policies/plugins.

- Control Plane: Configuration, orchestration, and monitoring of gateway nodes.

- etcd: Central configuration store.

- Observability: Real-time monitoring and visualization via Skywalking, Prometheus, and Grafana.

One-Line Summary

Apache APISIX is a next-gen cloud-native API gateway, built on NGINX and etcd, supporting dynamic routing, hot plugin loading, and deep integration with monitoring stacks.

(Use as documentation, speaker notes, or slide content as needed.)

**What is LuaJIT?**

**LuaJIT** is a **Just-In-Time (JIT) compiler** for the [Lua](https://www.lua.org/) programming language.

**What does this mean?**

* LuaJIT takes Lua code and compiles it **on the fly** (just-in-time) into very fast machine code, instead of interpreting it line-by-line.
* As a result, **LuaJIT runs Lua scripts much faster** than the standard Lua interpreter—often close to the speed of programs written in C.

**Why is LuaJIT important for APISIX and OpenResty?**

* **OpenResty** (which APISIX is built on) uses LuaJIT to execute Lua plugins and scripts.
* This enables **high performance**, allowing custom logic and plugins to run in the API gateway with very low latency.

**Key Points:**

* **JIT Compiler**: Converts Lua scripts to fast native machine code at runtime.
* **Performance**: Offers a huge speed boost for Lua-based applications (like APISIX plugins).
* **Compatibility**: Supports most Lua 5.1 code and many Lua libraries.

**In one line:**

**LuaJIT** is a lightning-fast version of the Lua interpreter, enabling high-performance scripting and plugin execution in systems like APISIX.

**What is etcd?**

* **etcd** is a fast, distributed, highly-available key-value store.
* It’s open-source and widely used for storing configuration and metadata in cloud-native systems (like Kubernetes, APISIX).

**Why is etcd used in APISIX?**

**1. Central Configuration Store**

* All API gateway configurations—routes, upstreams, plugin settings, consumer info—are stored in etcd.
* This allows for **dynamic, real-time configuration** (APISIX doesn’t need to restart when configs change).

**Why is etcd Required by Both Data Plane (DP) and Control Plane (CP)?**

**Control Plane (CP):**

* The **CP** (e.g., Dashboard, Admin API) is responsible for writing or updating config in etcd.
* Whenever an admin adds, edits, or deletes a route or plugin, that information is written into etcd.

**Data Plane (DP):**

* The **DP** (the APISIX nodes handling live traffic) **reads config from etcd**.
* The DP watches etcd for any config changes and applies them instantly, without downtime.

**What is etcd?**

* **etcd** is a fast, distributed, highly-available key-value store.
* It’s open-source and widely used for storing configuration and metadata in cloud-native systems (like Kubernetes, APISIX).

**Why is etcd used in APISIX?**

**1. Central Configuration Store**

* All API gateway configurations—routes, upstreams, plugin settings, consumer info—are stored in etcd.
* This allows for **dynamic, real-time configuration** (APISIX doesn’t need to restart when configs change).

**Why is etcd Required by Both Data Plane (DP) and Control Plane (CP)?**

**Control Plane (CP):**

* The **CP** (e.g., Dashboard, Admin API) is responsible for writing or updating config in etcd.
* Whenever an admin adds, edits, or deletes a route or plugin, that information is written into etcd.

**Data Plane (DP):**

* The **DP** (the APISIX nodes handling live traffic) **reads config from etcd**.
* The DP watches etcd for any config changes and applies them instantly, without downtime.

In **Apache APISIX**, **Grafana** and **Prometheus** are used for **monitoring, observability, and real-time analytics** of API traffic, system performance, and business metrics. Here's a detailed breakdown of their roles:

**1. Prometheus in APISIX**

**What is Prometheus?**

Prometheus is an open-source **time-series database** and monitoring system that collects, stores, and queries metrics.

**How APISIX Uses Prometheus**

* APISIX exposes **metrics in Prometheus format** (HTTP endpoint /apisix/prometheus/metrics).
* Prometheus **scrapes** this endpoint at regular intervals (e.g., every 15s).
* Metrics collected include:
  + **API Traffic**: Requests, responses, latency.
  + **System Metrics**: Bandwidth, CPU/memory usage.
  + **Plugin-Specific Metrics**: Rate-limiting hits, failed auth attempts.
  + **Upstream Health**: Success/error rates for backend services.

**Key Metrics Exported by APISIX**

| **Metric** | **Description** |
| --- | --- |
| apisix\_http\_requests\_total | Total HTTP requests |
| apisix\_http\_request\_latency\_ms | Request latency (milliseconds) |
| apisix\_bandwidth | Ingress/egress traffic (bytes) |
| apisix\_http\_status\_5xx | Number of 5xx errors |
| apisix\_plugin\_requests\_total | Plugin-specific calls (e.g., limit-count) |

**Example Prometheus Configuration**

yaml

Copy

Download

scrape\_configs:

- job\_name: 'apisix'

scrape\_interval: 15s

static\_configs:

- targets: ['apisix:9091'] # APISIX Prometheus endpoint

**2. Grafana in APISIX**

**What is Grafana?**

Grafana is a visualization tool that **dashboards** Prometheus (and other) metrics.

**How APISIX Uses Grafana**

* Grafana connects to Prometheus as a **data source**.
* Pre-built **APISIX dashboards** display:
  + **Real-time API traffic** (RPS, latency, errors).
  + **Plugin performance** (rate-limiting, auth failures).
  + **System health** (CPU, memory, bandwidth).
* Supports **alerts** (e.g., "Alert if 5xx errors > 1%").

**Key Grafana Dashboards for APISIX**

1. **APISIX Overview Dashboard**
   * Requests/sec, latency percentiles, upstream health.
2. **Plugin Monitoring Dashboard**
   * Rate-limiting hits, JWT auth failures, etc.
3. **Business Metrics Dashboard**
   * Custom metrics (e.g., "Checkout API calls per user").

**Example Grafana Dashboard (APISIX Official)**

<https://apisix.apache.org/assets/images/dashboard-example.png>  
*(Shows requests, latency, and error rates over time.)*

**3. Why Use Prometheus + Grafana with APISIX?**

| **Feature** | **Benefit** |
| --- | --- |
| **Real-time Monitoring** | Detect API issues instantly (e.g., spikes in 5xx errors). |
| **Historical Analysis** | Track performance trends (e.g., "Latency increased after v2 deploy"). |
| **Alerting** | Slack/Email alerts if thresholds are breached (e.g., "Too many 429s"). |
| **Troubleshooting** | Debug plugin behavior (e.g., "Why are auth requests failing?"). |
| **Capacity Planning** | Scale APISIX nodes based on traffic patterns. |

**4. How to Set Up Prometheus + Grafana with APISIX**

**Step 1: Enable Prometheus Plugin in APISIX**

curl http://127.0.0.1:9080/apisix/admin/routes/1 -X PUT -d '

{

"uri": "/metrics",

"plugins": {

"prometheus": {}

}

}'

**Step 2: Configure Prometheus to Scrape APISIX**

# prometheus.yml

scrape\_configs:

- job\_name: 'apisix'

static\_configs:

- targets: ['apisix:9091']

**Step 3: Import APISIX Dashboards into Grafana**

1. Add Prometheus as a **data source** in Grafana.
2. Import the **official APISIX dashboard** (ID: **11719**).  
   *(Grafana Dashboard URL:*[*https://grafana.com/grafana/dashboards/11719*](https://grafana.com/grafana/dashboards/11719)*)*

**5. Advanced Use Cases**

**a) Custom Business Metrics**

* Use APISIX **prometheus-plugin** to expose custom metrics (e.g., orders\_api\_calls\_total).

**b) Multi-Cluster Monitoring**

* Aggregate metrics from **multiple APISIX instances** into a single Grafana view.

**c) Alerting Rules**

**yaml**

# prometheus.yml

rule\_files:

- 'alert.rules.yml'

**Example alert rule (alert.rules.yml):**

**yaml**

groups:

- name: apisix-alerts

rules:

- alert: HighErrorRate

expr: sum(rate(apisix\_http\_status\_5xx[1m])) by (service) / sum(rate(apisix\_http\_requests\_total[1m])) by (service) > 0.01

for: 5m

labels:

severity: critical

annotations:

summary: "High error rate on {{ $labels.service }}"

**6. Alternatives**

| **Tool** | **Use Case** | **APISIX Integration** |
| --- | --- | --- |
| **Prometheus** | Metrics collection | ✅ Native support |
| **Grafana** | Visualization | ✅ Pre-built dashboards |
| **Elasticsearch + Kibana** | Log analytics | Requires http-logger plugin |
| **OpenTelemetry** | Distributed tracing | Supports Jaeger/Zipkin |

**Summary**

* **Prometheus** → Collects and stores APISIX metrics.
* **Grafana** → Visualizes metrics for monitoring/debugging.
* **Key Benefits**:
  + Real-time observability.
  + Historical trend analysis.
  + Proactive alerting.