

# The API Gateway Maturity Matrix

*Where Do You Rank?*

Joel Hans – Senior Developer Educator at ngrok

April 2, 2025

# Let's start with a story..



# NGINX



# Is this Matrix right for you?

## Tech leads and architects

→ "I need to make build vs. buy decisions and define an API strategy that scales."

## Platform engineers

→ "I'm building an internal developer platform and need to define how API gateways fit into our developer experience."

## DevOps and infrastructure engineers

→ "I want to standardized ingress, security, and observability while also enabling others with self-serfice... and not making more incidents to deal with."

# Why should you care?

API gateways are the front door of your platform and your business. You need to:

- Think critically about where you are and what you could bring to the table for improvements.
- Evaluate new technologies or what capabilities your current stack provides that you haven't yet turned on.
- Find places where you can invest time as the users of your platform—or the scale of your business—demand.

That requires:

A way to reflect on yourself, your team, and your offering... then find ways to improve or figure out where you're falling behind.

## This model is:

- Designed to help you self-assess where your API gateway implementation is today and plan ahead for what's next.
- A way to focus on solving problems and enabling value or return on investment.
- A collaborative effort that needs contributions from *you*!

## This model is not:

- A way to judge your implementation or tell you how to do your job.
- A prescriptive, "all or nothing" approach to building a mature API gateway.
- Designed to cover *all* the cultural and technological complexity of building an internal developer platform.
- A way to pitch any particular API gateway product, service, or capability *without a business need*.

# **Detroit, Michigan.**

**October 27, 2022.**

# Our starting point: CNCF's Cloud Native Maturity Model

A framework for starting—and succeeding!—in a cloud native journey, starting with inception into full adoption of cloud native technologies across the CNCF landscape.

- Launched in 2021, reached 3.0 in Autumn 2023.
- Includes five Levels: **Build, Operate, Scale, Improve, Adapt.**
- Started with 4 dimensions of **People, Process, Policy, and Technology**, with **Business Outcomes** added later and since promoted to top spot on the model with 3.0.

# Our lens: Where maturity and a specific technology meet

Most maturity models, like CNCF's, are built for sweeping cultural change. "Digital transformation" and so on.

What about tactical decisions about a *single* technology?

- "My API gateway offers 100 features. Which ones do I need Day 0, Day 1, and Day 2?"
- "What value does my API gateway offer as part of an internal developer platform?"
- "Things are actually quiet right now... what should I think about building next?"
- "Woof. That outage was a bad one. What do I need to build right now to make sure I never have to experience that again?"



# Hold up: What is maturity, anyway?

Well, it's *not* about:




- How old an API gateway product is or how long you've been using it.
- Whether you've written down your processes or not.
- How many features an API gateway offers.

Instead, let's think about:

- How you've adopted new features based on new requirements or inciting incidents.
- Whether your API gateway and processes are flexible enough to deal with new problems.
- How good the ergonomics are when you *do* turn on new features.

As adoption of an API gateway matures, engineers shift *away* from building API gateways for themselves and *torward* enabling others to ship fast without giving up control over policy and governance.

# Where do you think you stand?

-  "I think there's something in front of my APIs... but I'm not totally sure what it's doing."
-  "Building on the API gateway involves lots of tickets and paying someone still remembers how it works."
-  "I'm just here to validate how much further along we are than everyone else."

# Onto the API Gateway Maturity Matrix™

**Build** **Is this thing on?** We have a basic ingress with a reverse proxy masquerading as an API gateway. The infra team (or the one person who understands it) owns the manual configuration, holds all the knowledge, and manages things ad-hoc.

---

**Operate** **Time to make this API gateway thing a little less Swiss cheese-y.** We've upgraded to a dedicated API gateway, managed by our infra or DevOps team, which is declaratively configured with CI/CD and supported by minimal-to-acceptable documentation.

---

**Scale** **How do we make this thing get 10X (or 100X!) bigger... and not terrible to work on at the same time?** Our API gateway now standardizes services from deployment to observability to incident response, and offers reusable configs for developers to ship quickly while our newfangled platform team manages the scale.

---

**Improve** **All the things are important now. And how do I help devs move quick without losing control?** The platform team constantly modulates how the API gateway works to create a golden path for self-service while isolating different teams' work, automating policy enforcement, and providing built-in observability for every service.

---

**Adapt** **How do we keep innovating... without over-engineering?** The API gateway is now fully dynamic, policy-driven, and responsive to signals like traffic patterns, active threats, or cost. It's the foundation of a unified platform that everyone not just relies on, but actually enjoys using.

# Five problems to solve (capability threads)

- Traffic management
- Authentication & security
- Observability & debugging
- Developer/team experience
- Governance & compliance

# Traffic management

## Build

Our APIs are protected from unauthorized access, but the application of AuthN is inconsistent—sometimes at our gateway, sometimes embedded in your services.

Static routing on paths, subdomains, etc

Hardcoded rewrites and redirects

Simple load balancing and manual failover

## Operate

We centrally manage, configure, and turn on features like routing on behalf of development teams, some driven by their asks and some driven by protecting our infrastructure.

Basic per-client rate limits for fairness and abuse

Geoblocking and IP restrictions

DDoS protection and global load balancing

## Scale

Our APIs can handle high traffic safely and without overloading our upstream services, and we're optimizing for performance and availability... and automatically fail over when things go wrong.

Multi-environment and -region routing (multicloud?)

Weighted traffic splitting for blue/green or canaries

Load balancing: latency-based, or sticky, round robin

## Improve

Teams can self-manage routing, rate limits, and other traffic management rules while staying within platform guardrails.

Dynamic throttling based on load or error rates

Custom LB: PEWMA+weighting, proximity+load

Request hedging for latency optimization

## Adapt

We can route traffic dynamically based on cost, latency, congestion, usage patterns, availability, and beyond.

Dynamic routing for user priority or slow starts

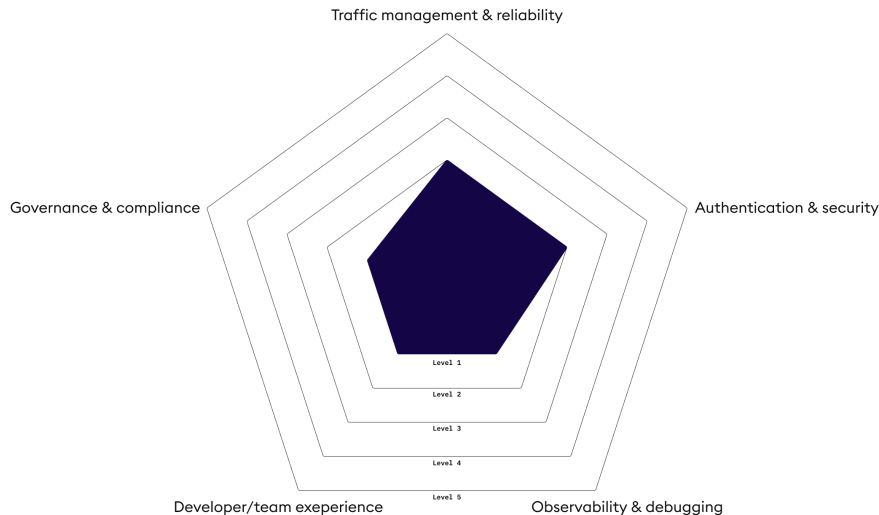
Dynamic limits based on predicted workload patterns

Multicloud LB based on cost and capacity

# A two-person... *thing*?

With a single service and two technical co-founders, it's easy to collaborate and write directly into the app layer rather than blow up the architecture... and no need for complex automation. Manual deploys and a basic API gateway, TYVM!

- ☒ Can we route API requests correctly?
- ☒ Do we know if things crash because we have basic monitoring?
- ☒ Do we have *any* authentication in place?





# **Authentication & security**

## Build

Our APIs are protected from unauthorized access, but the application of AuthN is inconsistent—sometimes at our gateway, sometimes embedded in your services.

API keys and basic authentication

Mix-and-match of AuthN/AuthZ services

TLS termination at the gateway edge

## Operate

Security enforcement is increasingly centralized at the API gateway, reducing per-service misconfigurations that lead to risk or breaches.

JWTs or OAuth2

Centralized AuthN/AuthZ via the API gateway

Basic role-based access control

## Scale

Our API gateway is a central hardening point, and we have a unified and repeatable security model that supports multiple distributed teams.

Geoblocking and IP reputation filtering

mTLS for service-to-service AuthN/AuthZ

Multi-tenant isolation in gateway routes

## Improve

Developers can implement new Zero Trust fundamentals via the API gateway without writing tickets or waiting for approvals.

Self-service policy enforcement via OPA/Kyverno

Automated API posture checks

Fine-grained access control per team or service

## Adapt

Our API security model is adaptive and capable of preventing breaches before they happen.

Risk-based authentication and rate limiting

Threat detection/intelligence feeds

Just-in-time access control

# Observability & debugging

## Build

We can see *what* our API services are doing, mostly via logs, in a deployed environment to help us identify issues.

Basic logs generated at the API gateway

Error rate monitoring (5xx API errors)

Basic health check endpoints

## Operate

We have a unified view *how* our APIs are doing via dashboards, making troubleshooting a lot easier and helping us resolve issues faster.

Distributed tracing with Jaeger, OpenTelemetry, etc

Structured logs

Unified monitoring for one gateway+environment

## Scale

Our API gateway is seen as the first place to look (and often to blame!) when we're monitoring the API landscape or developers need to do distributed debugging.

Unified monitoring across clusters/regions/clouds

Tracing to help find distributed bottlenecks

<Real-time incident alerting for relevant stakeholders

## Improve

Other teams can extend the API gateway to add their own metrics or tracing based on new requirements without having to change the main API surface beneath everyone's feet.

Automated remediation runbooks

Editable request replay for debugging

Team-specific gateway traffic dashboards

## Adapt

API issues are fully self-healing, minimizing the need for us (platform team) or others (API developers) to intervene manually.

AI-driven anomaly detection and automated RCA

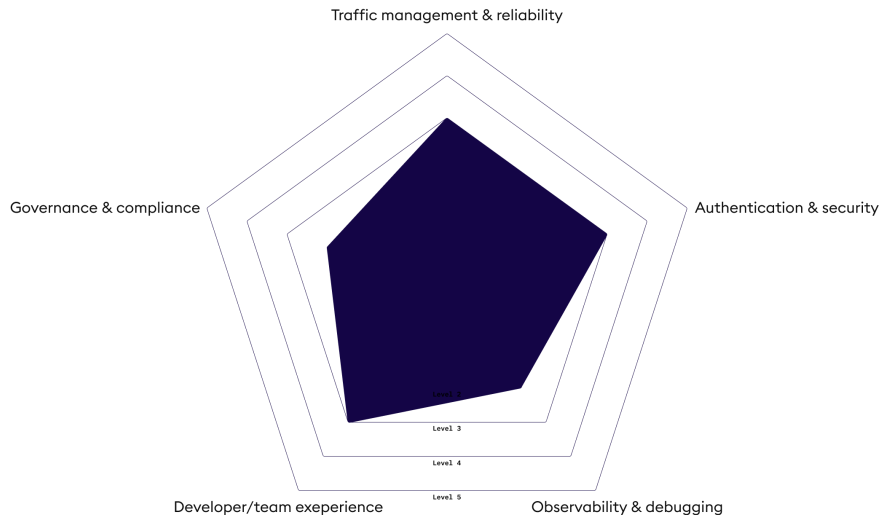
Automated incident response and rerouting

Business outcome correlation with ingress patterns

# A 50-person startup?

Now they're bigger—like hundreds of thousands of requests every day bigger. They need rate limiting and ways to figure out error rates or percentage of late responses, and the small-but-mighty infra team is getting tired of pulling logs and running queries. All this optimization requires better deployment strategies, too.

- ☒ Can we protect our APIs from excessive traffic or bad actors?
- ☒ Can we deploy API changes without manual intervention?
- ☒ Have we moved past basic metrics and into structured logs?



**Developer/team experience**

**Build** API configurations are managed with manual changes, with developers cobbling together ways to ship to prod. It works, but it's slow.

No IaC for API gateway configs

Ticket-based (or YOLO) configuration changes

Little to no documentation

**Operate** API changes have a clearly-defined process, streamlining Linear/Jira/etc board of tickets and freeing up our team's engineering time.

Standardized API definitions

IaC + CI/CD for all gateway configurations

Basic API catalog derived from routing topology

**Scale** API management (via automation and IaC) helps us manage APIs at scale across multiple clusters, environments, and growing teams.

GitOps and CI/CD++ (regions, canaries, rollbacks)

Many deployment options (K8s, hybrid, multicloud)

API versioning strategies at the API gateway

**Improve** Developers can self-service isolated API gateway configurations while we (now a platform team!) enforce policy, reducing operational overhead on a golden path.

Golden path templates/recipes for gateway patterns

Rich documentation of gateway best practices

Extensive API catalog/developer portal

**Adapt** We give developers more than guardrails—we support them with best practices on designing and deploying APIs.

Support for advanced customization and plugins

Shift-lefted, AI-driven gateway recommendations

Automatic detection of unused/deprecated services

# Governance & compliance



## Build

APIs meet basic security and compliance standards, but inconsistently, leading to risk and ad-hoc responses.

No enforced API standards at the gateway

Infrequent/ad-hoc audits of gateway compliance

Manual configuration reviews

## Operate

We enforce standards across multiple APIs and teams to reduce the risk of being non-compliant.

Basic API lifecycle management (e.g. versioning)

Standardized security (TLS requirements/versioning)

Semi-regular scanning of API gateway policy

## Scale

Governance of API gateways scales across teams and cloud environments without blocking development velocity.

Configs controlled with policy-as-code and CI/CD

Traffic auditing across multiple environments

Gateway-enforced data sovereignty

## Improve

Anyone can configure APIs within a predefined, platform-wide governance model.

Role- or team-based API gateway management

Fine-grained access control for config changes

Configuration change tracking for compliance

## Adapt

Our compliance is now automated and capable of adapting to new regulations or security risks dynamically.

AI-driven, continuous compliance monitoring

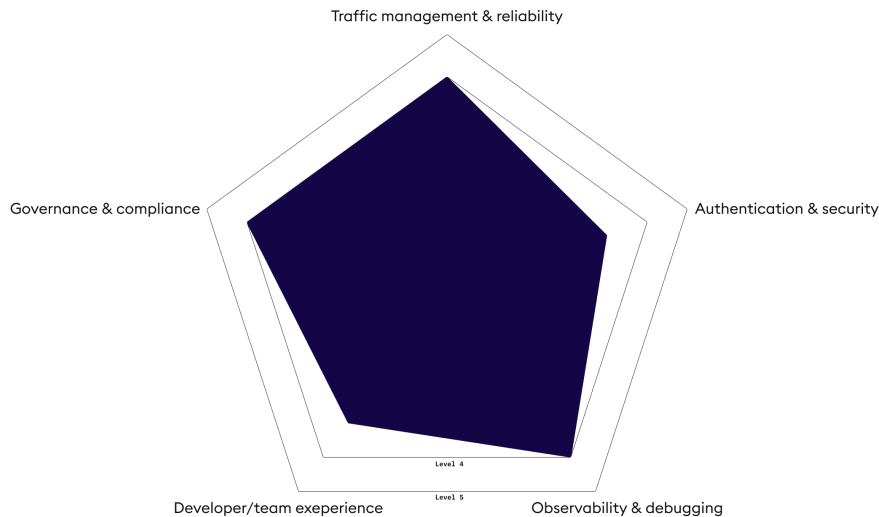
Threat-based security policy updates

Automatic policy updates for regulatory changes

# A 200-developer machine?

They handle 10 million requests and need to stand up a new API every day. *All of a sudden, all the things are important.* DDoSes happen every now and then, but also oddities like a single customer hitting them on a loop—same time every day. On top of that, they're also acquiring a company in a new and scary regulatory environment!

- ☒ Are individual teams configuring and deploying APIs without causing chaos?
- ☒ Can we scale API security across multiple regions and teams?



# Where do you think you stand now?

- 😅 "We just realized no one actually owns the gateway config."
- 🌱 "We're maturing fast, but still figuring out how to scale safely."
- 🏆 "We've got solid foundations and room to grow."

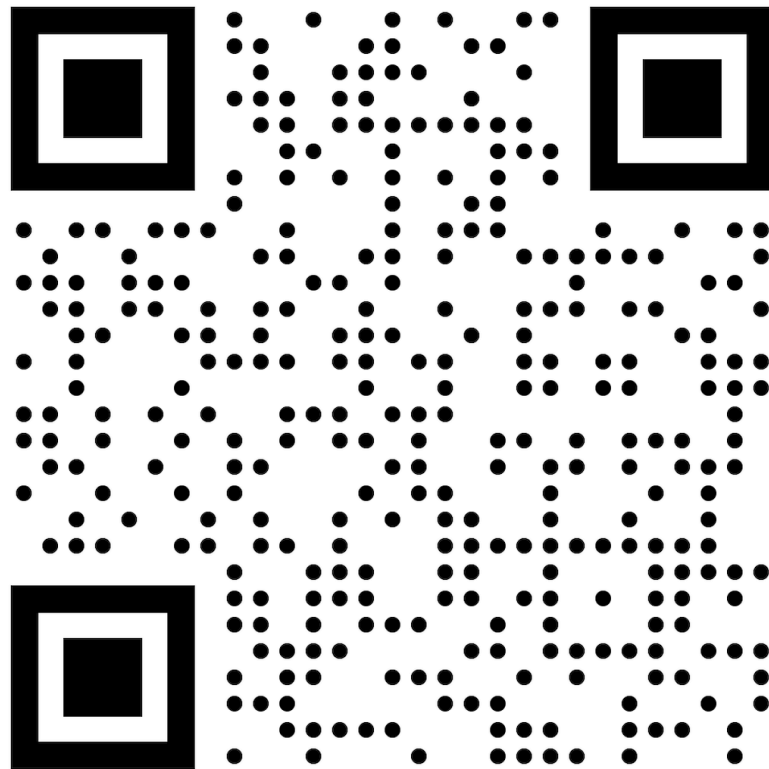
# The Matrix is ready for you!

A single-page website for self-assessment:

<https://api-gateway-maturity.joelhans.xyz>

Your action items:

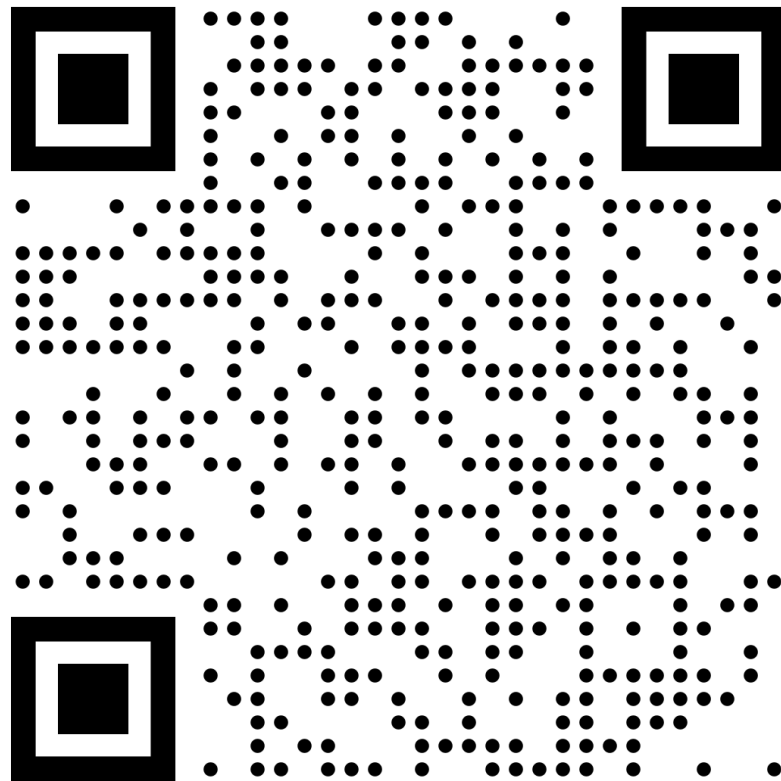
- Take *2 minutes* here at KubeCon to rate yourself across the five problem threads.
  - Where are you strongest? Weakest?
- Take the model back to your team next week and explore each thread+level to explore where you stand on the:
  - Problems you're solving
  - Value you deliver to your organization
  - Capabilities you've mastered
- Let me know about your experience! [j.hans@ngrok.com](mailto:j.hans@ngrok.com)



# Help make the Matrix better!

Full text + OSS project on GitHub: [joelhans/api-gateway-maturity](https://github.com/joelhans/api-gateway-maturity)

- Contribute your experiences on running this "algorithm" or add new illustrations
- Add more example capabilities to each thread+level
- Help develop a Myers-Briggs-esque questionnaire for even smoother self-assessment



# Questions?

Find me *beyond the Matrix*:

- [j.hans@ngrok.com](mailto:j.hans@ngrok.com)
- At the ngrok booth at **N611** (where I have to go right after this and finish up my shift [instead of hiding like I would probably like to])
- Wandering around KubeCon in an ngrok shirt

Thank you to the folks who helped me along the way!

- Alex
- Jeremy
- Alice
- Abdallah
- Peter
- Euan
- Jack

