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...

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 1,000?"
- "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?

To paraphrase the Capability Maturity Model (CMM), maturity is how formal and actively optimized your processes are, from ad hoc to formally defined.

I've been asking around at ngrok, too.

- "The API won't change underneath me."
- "When I get paged because something is broken, it's because of a bug in my software and not the infrastructure breaking down. :"
- "Don't conflate maturity with quality."
- "Wide adoption, but sometimes crusty."

Maturity isn't always a good thing.

- There's a fine line between *mature* and *legacy*.
- Mature platforms are difficult for you to modify and others to build upon.
- No one understands how a mature platform works anymore, whether that's lost institutional knowledge or it just worked once so leave it alone.
- Hard to steer the ship quickly if you need to do something complex... like multicloud (yikes).

But it *can* be a conscious effort to get you:

- Reliability without surprises.
- Velocity without chaos.
- Scalability without rip-and-replace (again).
- Governance without bureaucracy.

As teams mature, they shift *away* from building API gateways for themselves and *torward* enabling others to ship fast without giving up control over policy and governance.

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 and API governance without slowing devs down... and not making more incidents for myself."

Why should you care?

API gateways are a core 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.

Everybody *loves* a caveat!

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 or service.

Where do you think you stand?

- 🚀 "Our API gateway is in good shape."
- 🏗️ "We’re duct-taping stuff together."
- 🔥 "NGINX config copy-pasted from Stack Overflow. Send help."

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 I make this thing multi-region, multi-team, multicloud, 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 & reliability
- Authentication & security
- Observability & debugging
- Developer/team experience
- Governance & compliance

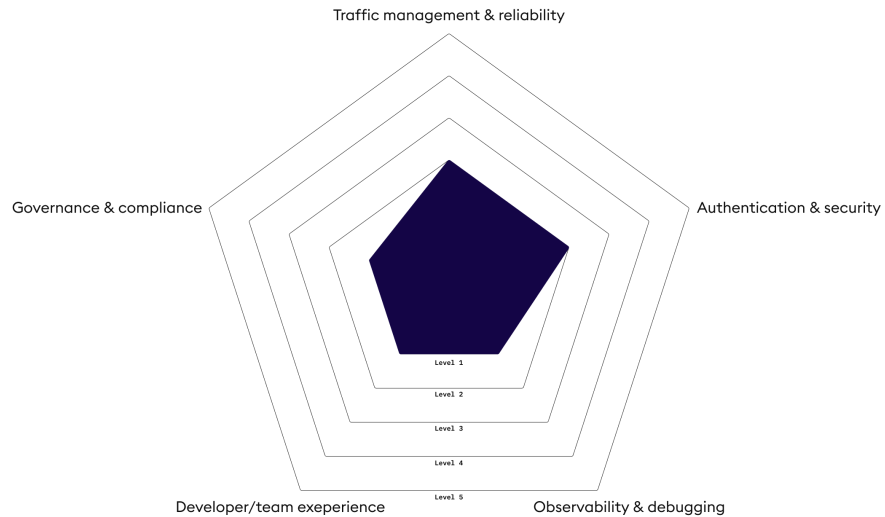
Traffic management & reliability

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 (+ redirects!)	Basic rate limits (e.g. 100 req/min across all endpoints)	Simple load balancing and manual failover
Operate	Our APIs can handle high traffic safely and without overloading our upstream services, and we can handle new versions or changed paths.		
	Client- and service-specific rate limiting	Geoblocking and IP restrictions	DDoS protection and global load balancing
Scale	We now optimize traffic for performance and availability environments, with automatic failover when things go wrong, and support more sophisticated deployments.		
	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 archeticture... and no need for complex automation. Manual deploys and NGINX, TYVM!

- ✅ Can we route API requests correctly?
- ✅ Can we prevent our backend from crashing under load?
- ✅ 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	TWe 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 proper 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	AI-powered threat detection/intelligence feeds	Just-in-time access control

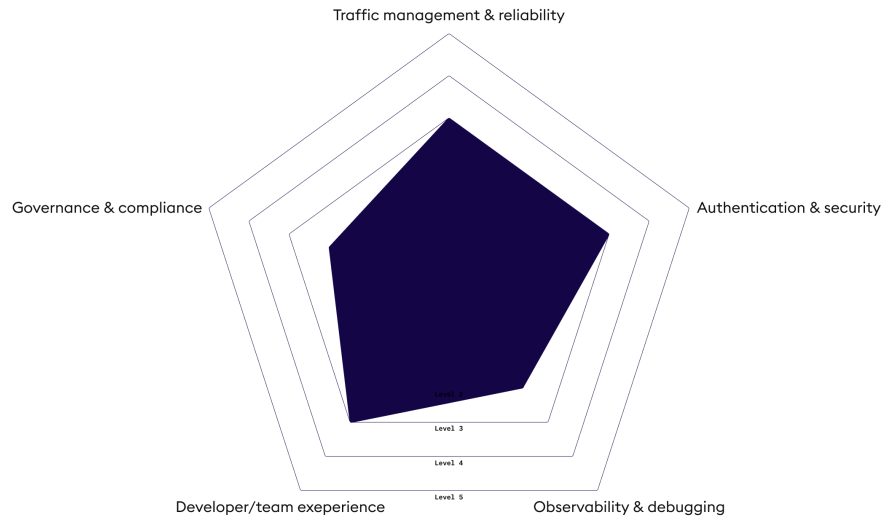
Observability & debugging

Build	We can see what our API services are doing 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 of API health via dashboards, making troubleshooting a lot easier and helping us resolve issues faster.		
	Distributed tracing with Jaeger, OpenTelemetry, etc	Structured logs	Unified monitoring across multiple gateway instances
Scale	Our API gateway is seen as the first place to look in incident response.		
	Real-time incident alerting for relevant stakeholders	Unified API monitoring across clusters/regions/clouds	Anomaly detection for traffic patterns
Improve	Our API gateway can react to its monitoring on its own and prevent a person from having to step in, while also offering way more observability features for those who opt-in to the platform.		
	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 me 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.	API configs in YAML	Ticket-based (or YOLO) configuration changes	Little to no documentation
Operate	API changes are automated, clearing our Linear/Jira/etc board of tickets and freeing up our team's engineering time.	Standardized API definitions	Config and deploy with GitOps, IaC, and CI/CD	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.	Devs provision new gateways per project/function	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

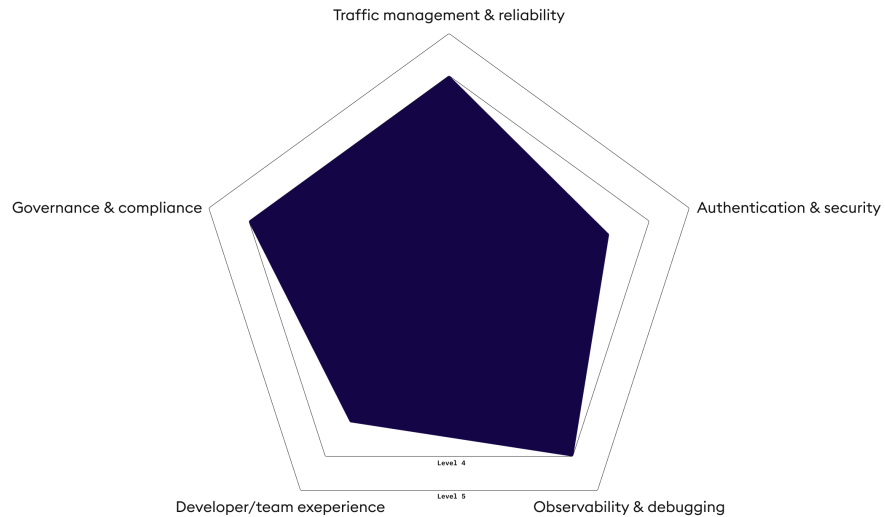
Governance & compliance

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 API security policies	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*. They're getting DDoSed on the regular, but 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?
- ✅ Do we have a way to automate governance instead of enforcing it manually?



We've built and tested the algorithm. Let's run it through some real-world data!

(This is a surprise.)

Where do you think you stand now?

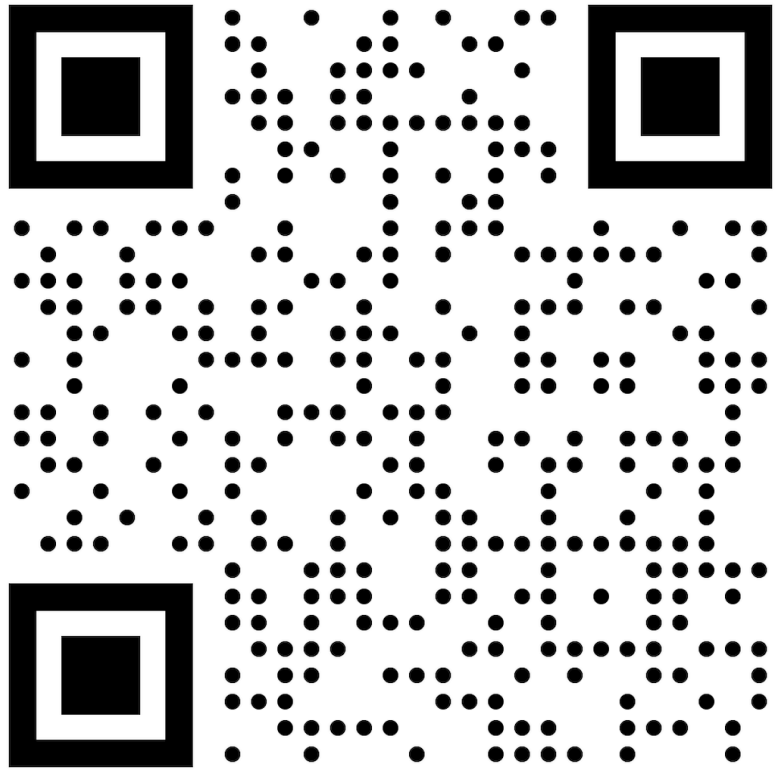
- 🏆 "Feeling mature, but not legacy!"
- 🚧 "We've got work to do."
- 🤡 "I just realized we don't actually have an API gateway."

The Matrix is ready for you!

A single-page website for focused self-assessment:
`https://api-gateway-maturity.joelhans.xyz`

Your action items:

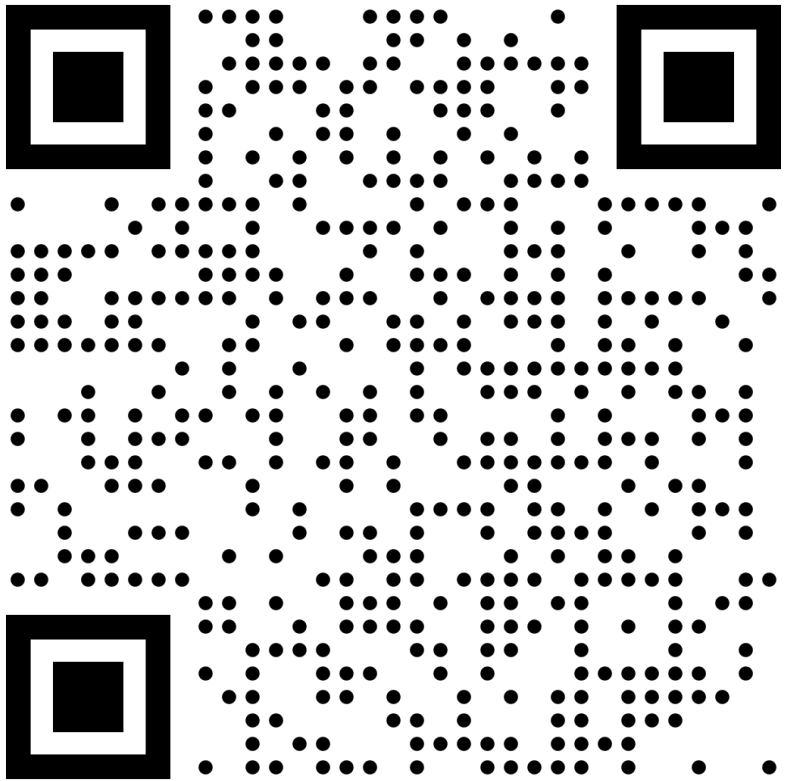
- Take *2 minutes* here at KubeCon to rate yourself across the five problem threads.
  - Where are you strongest? Weakest?
- Take the model back to your team next week and explore each thread+level to explore where you stand on the:
  - Problems you're solving
  - Value you deliver to your organization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



Help make the Matrix better!

Full text + OSS project on GitHub: `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
- 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