# ingress-nginx Alternatives: Cilium, Traefik, Kong, AKS, GKE

2026-07-18

How Cilium, Traefik, and Kong score against the Part 3 matrix, plus a survey of AKS, GKE, STACKIT, and OVH ingress and Gateway API paths on managed Kubernetes.

---

Part 4 covered the five candidates source-verified with the same depth as the Part 3 matrix. This part covers the rest of what this series set out to compare: three candidates this series tests but doesn't deep-dive — Cilium, Traefik, Kong — and a survey of what migrating off ingress-nginx looks like on four managed Kubernetes platforms that were never part of the empirical lab at all.

**► TL;DR**

Cilium (6 green / 4 yellow / 7 red), Traefik (7/8/2), and Kong (2/10/5) score against the same 17-row matrix as Part 4′s candidates, at context level rather than deep-empirical. Cilium's reds cluster around features that simply don't exist on its Ingress/Gateway path (no rate limiting, no WAF, no session affinity); Kong's yellows cluster around its own generic plugin bridge standing in for almost everything; Traefik's actual strength is a dedicated ingress-nginx-compatibility provider most of the other candidates in this series don't have at all. On managed clouds: AKS and GKE both ship a Gateway API path built specifically as an ingress-nginx successor; STACKIT has no managed Gateway API offering at all, only a managed Ingress-API alternative; OVH is represented here by this blog's own self-managed Gateway API setup behind a plain LoadBalancer service; no vendor-managed ingress or Gateway API path was found in the docs reviewed for this series.

## Context-level candidates: tested, not deep-dived

"Context-level" is this series' own term for a specific claim: these three candidates get scored against the same matrix as Part 4′s five, but the architecture facts behind each score come from a single source-verification pass, not the same depth as Part 4 (or the 8-agent audit Part 3 got). The distinction matters because "tested with this series' full rigor" and "worth naming" are different claims — see the series hub for how the prompt driving this series scopes the two tiers.

| Candidate | Green | Yellow | Red | Config layer | Proxy engine |
|---|---|---|---|---|---|
| Traefik | 7 | 8 | 2 ages bridged by `ExtensionRef`→`Middleware` | `Ingress`/ CRDs and `HTTPRoute`, separate provider packages | Traefik's own native Go proxy |
| Cilium | 6 | 4 | 7 | `Ingress` or `HTTPRoute`, converging into one shared translation model | Embedded per-node Envoy (DaemonSet) — except TCPRoute/UDPRoute, which bypass it via eBPF |
| Kong | 2 | 10 | 5 firmed: both `networking.k8s.io` and `HTTPRoute` handlers present in source) | `Ingress` or `HTTPRoute`, one codebase for both (con- | NGINX/OpenResty + Lua (Kong Gateway) |

## Cilium

Cilium's ingestion code genuinely treats `Ingress` and `HTTPRoute` as separate paths, but both converge into one shared `model.Model` before either reaches Envoy — the same embedded, per-node Envoy instance the eBPF dataplane already runs, confirmed from source (`operator/pkg/model/ingestion/{ingress,gateway}.go`). The one exception is the newest addition: TCPRoute/UDPRoute reconciliation landed around June 2026 and deliberately bypasses that Envoy entirely, routing weighted traffic through Cilium's own eBPF Maglev load-balancer instead (`l4HasWeights`, `operator/pkg/model/translation/gateway-api/translator.go`) — the one row in this whole matrix where a single candidate's proxy-engine answer forks by route kind.

The seven reds are concentrated, not scattered: no rate-limiting code anywhere in the Ingress/Gateway L7 path, no WAF or ModSecurity integration, and — despite Gateway API's experimental `SessionPersistence` field existing in Cilium's own vendored types — it is never actually read by the ingestion code, confirmed by a zero-hit grep. Where Cilium does implement a feature, though, it tends to be the standards-native answer: its `ExternalAuth` `HTTPRoute` filter support (auth-url/ auth-signin) is the most spec-faithful of any target in the whole series, reaching a real Envoy `ext_authz` filter through the actual core field rather than a vendor Policy CRD.

*Last verified 2026-07-17 against `cilium/cilium` main @ `eef7b9e` (2026-07-14).*

## Traefik

Traefik is this matrix's one genuine outlier on the proxy-engine axis — its own native Go reverse proxy (`&httputil.ReverseProxy{}` from the stdlib, plus an experimental zero-allocation en-

gine), not a wrapper around NGINX, HAProxy, or Envoy. Its `Ingress`/CRD path and its `HTTPRoute` path are separate provider packages, bridged only by a generic `ExtensionRef`→`kind: Middleware` registry.

What actually lifts Traefik's score above Cilium's and Kong's is a mechanism no other candidate in this series has at all: a dedicated ingress-nginx-annotation-compatibility provider (`pkg/provider/kubernetes/ingress-nginx/`) that parses existing `nginx.ingress.kubernetes.io/*` annotations directly, including a bounded snippet interpreter for the auth path. Several of the matrix's yellow verdicts for Traefik exist specifically because this compat provider's own translated type is narrower than the generic `Middleware` CRD equivalent it resembles — the same "real path is narrower than the generic one" pattern Part 3 and Part 4 both ran into repeatedly. Its two reds are structural, not maturity gaps: no `SessionPersistence` support at all (not even attempted, per Traefik's own checked-in conformance report), and `UDPRoute` confirmed unimplemented on the Gateway API path — no `udproute.go` file exists in the provider package.

*Last verified 2026-07-17 against `traefik/traefik` main branch.*

### Kong

Kong Kubernetes Ingress Controller is architecturally distinct from every other candidate in this series: one codebase answers both the classic `Ingress` API and `HTTPRoute` (confirmed — both a generated `networking.k8s.io` controller and a dedicated `httproute_controller.go` exist in the same source tree), and it's an officially supported provider in `kubernetes-sigs/ingress2gateway` — the first time this series named "is this an `ingress2gateway` provider" as an explicit reason to include a candidate at all, rather than a fairness argument.

Kong's ten yellows almost all funnel through the same generic bridge: a `KongPlugin` attached via either the `konghq.com/plugins` annotation (`Ingress`) or an `ExtensionRef` filter (`HTTPRoute`), confirmed from source (`generateExtensionRefKongPlugin`, `internal/dataplane/translator/subtranslator/httproute.go`). Rate limiting, external auth, and session affinity (via Kong's own `KongUpstreamPolicy` CRD rather than core `SessionPersistence`) all reach Kong Gateway this same way — nearly every hard case in this matrix funnels through one generic mechanism rather than dedicated code, which is either a strength (one thing to learn) or a weakness (the actual logic runs in Kong Gateway, outside this controller's own repository), depending on how much that indirection matters for your own operational model.

*Last verified 2026-07-17 against `kong/kubernetes-ingress-controller` main @ `e47c34066`.*

## Managed cloud provider paths

None of the four platforms below were part of this series' empirical lab — what follows is a survey of what each one's own documentation and, for OVH, this blog's own production repos actually show, not a fourth round of source-verified testing. The headline finding: "managed Gateway API" and "managed ingress" do not imply one consistent runtime model across providers, even before either platform's actual annotation compatibility enters the picture.

## AKS and GKE

Both platforms ship a Gateway API path specifically framed as an ingress-nginx successor — not to be confused with either platform's separate *cloud-native* Gateway API product (Azure's Application Gateway for Containers, GKE's own multi-GatewayClass offering), which the existing Istio vs. Envoy Gateway comparison already covers in depth, including Cilium-CNI specifics and cross-cluster compatibility. The two paths are genuinely different products, not two names for the same thing.

**AKS.** The application-routing add-on's Gateway API implementation (`approuting-istio` GatewayClass) deploys an in-cluster Istio control plane per cluster and is, as of AKS 1.36, the default ingress path on new AKS Automatic clusters. Microsoft's own migration guidance is direct: managed-NGINX users on the application-routing add-on must migrate to this Gateway API implementation (or another supported one) by November 2026, when Azure support for the legacy NGINX-based add-on ends. Managed Gateway proxies get an autoscaling `HorizontalPodAutoscaler` and a `PodDisruptionBudget` per `Gateway`. Two concrete constraints worth knowing before adopting it: it cannot run alongside the separate Istio service-mesh add-on without first deleting that add-on's CRDs and `GatewayClass`, and `TLSRoute`-based SNI passthrough isn't supported until AKS adds Istio 1.30.

**GKE.** The GKE Gateway Controller is not hosted on the cluster's control plane or in the user's project at all — it runs externally and programs Google Cloud Load Balancer resources directly from `HTTPRoute` objects, an architecturally different model from AKS's in-cluster-Istio approach. GKE's own docs state every existing `Ingress` resource on GKE is directly convertible to `Gateway`/`HTTPRoute`, and that the two can run simultaneously without conflict during a migration. GKE Ingress, the older controller, remains supported and was never ingress-nginx-based — it is unaffected by the retirement, same as AKS's legacy NGINX add-on is a separate support question from the retirement itself.

*Last verified 2026-07-17 against Microsoft Learn's AKS application-routing Gateway API docs (`ms.date: 2026-07-01`) and Google Cloud's GKE Gateway API concepts docs.*

## STACKIT

STACKIT SKE's Application Load Balancer (ALB) Ingress controller (`stackit.cloud/alb-ingress`, opt-in via `extensions.applicationLoadBalancer.enabled: true`) is built on the native Kubernetes `Ingress` API — as of this writing, **STACKIT has no managed Gateway API offering at all**, unlike AKS and GKE. Customers migrating off ingress-nginx on STACKIT get a vendor-managed `Ingress`-API alternative, not a vendor-managed Gateway API path; STACKIT's own docs do include a "Moving from Ingress to Gateway API" section, signalling a transition is planned, not yet shipped as a managed product.

Two STACKIT-specific gotchas worth carrying over from this blog's own AuthorizationPolicy/HTTP-2- coalescing research: the ALB selects TLS certificates purely by SNI, so — exactly the same shared-listener, shared-certificate failure mode covered in the AuthorizationPolicy and HTTP/2 connection coalescing post — a certificate from one `Ingress` can affect others sharing the same ALB. And three of the ALB's own backend-TLS annotations still ship with a typo in STACKIT's live docs, unchanged since first noted: `alb.stackit.cloud/traget-pool-tls-enabled`, `-`

`custom-ca`, and `-skip-certificate-validation` (sic) — quoted here exactly as STACKIT's own documentation states them; verify at deploy time whether the misspelling is docs-only or baked into the controller itself before writing a manifest around it.

STACKIT's Kubernetes engine has its own broader constraints worth knowing before treating it as an ingress-only decision — see Who Builds the Platform? Ownership vs. Stack for SKE's audit-logging gap and Gardener-based architecture.

*Last verified 2026-07-17 against STACKIT's own application-load-balancing docs (`docs.stackit.cloud`, page states last updated 2026-07-10).*

### OVH

What's actually confirmed, empirically, across two of this blog's own production repositories — `ovh-example-observability`, and the LLM Inference on OVH series' own gateway setup — is that every one of them self-manages Istio Ambient plus upstream `kubernetes-sigs/gateway-api` CRDs behind OVH's plain `LoadBalancer` Service (OpenStack Octavia underneath), with zero `ingress-nginx` references and zero OVH-managed-ingress references anywhere in any of them. None of OVH's own vendor documentation surfaced a managed ingress or Gateway API product in any of this series' research either — worth stating as "not found in the repos this blog operates or in OVH's own docs," not as a vendor-wide claim that no such product exists anywhere, since a newer offering could exist that none of that research happened to touch. What the evidence does support is the real-world counter-example to AKS, GKE, and STACKIT's managed paths in this survey: on OVH, "migrating off ingress-nginx" and "standing up your own Gateway API control plane" are, in practice, the same project, not two separate decisions. The flip side of that same absence is flexibility: without a vendor-managed ingress or Gateway API product to work around, OVH imposes no platform-level constraint on which Ingress controller or Gateway API implementation you run — any of the candidates covered across this series, or a mix, is equally viable, in a way none of AKS's, GKE's, or STACKIT's own managed defaults are.

*Last verified 2026-07-17 against this blog's own `ovh-example-observability` and `ovh-llm-inference` repositories.*

## Four runtime models behind one label

"Managed Gateway API" means something different on each of these four platforms, and the difference is architectural, not cosmetic:

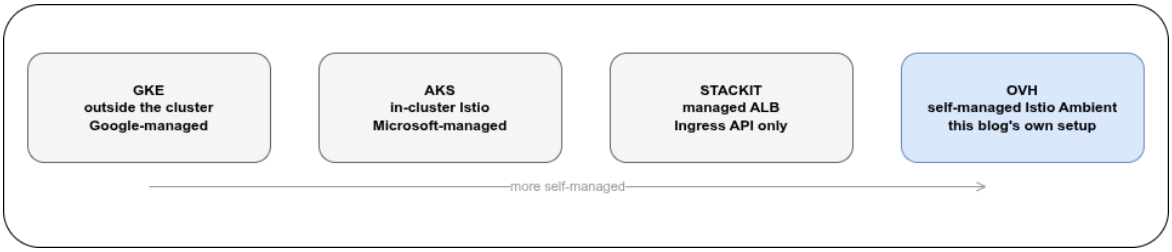


Figure 1: Four Gateway API runtime models: GKE outside the cluster, AKS in-cluster Istio, STACKIT managed ALB, OVH self-managed

| Platform | Where the data plane runs | Who operates it |
|---|---|---|
| AKS (application-routing add-on) | In-cluster Istio proxies, per `Gateway` | Microsoft-managed lifecycle, customer-configured routes |
| GKE (Gateway Controller) | Outside the cluster entirely — Google Cloud Load Balancer | Fully Google-managed, no in-cluster proxy pod at all |
| STACKIT (ALB Ingress) | STACKIT-managed ALB, `Ingress` API only | STACKIT-managed, no Gateway API path yet |
| OVH (this blog's own setup) | In-cluster Istio Ambient, self-deployed | Fully self-managed, upstream CRDs |

Treat "does provider X support Gateway API" as a question with at least two follow-ups before it's useful for a migration decision: does the data plane run inside your cluster or outside it, and who is actually operating the lifecycle of whichever answer you get.

## What this doesn't answer yet

Same caveat as Part 4: none of the above ranks these seven remaining candidates or four cloud paths against each other. Cilium's standards-native auth filter, Traefik's compat provider, and Kong's one-codebase-both-APIs design each solve a different problem well; which one fits depends on constraints this survey can't supply — which annotations your own `Ingress` objects depend on (Part 3), which of Part 4's five deep candidates already covers your case, and which managed platform your workload already runs on. Migration Mechanics, covering `ingress2gateway`, coexistence patterns, and a staged cutover, is next.

## Sources

- cilium/cilium — Cilium (source, main branch @ `eef7b9e`, 2026-07-14)
- traefik/traefik — Traefik (source, main branch)
- Kong/kubernetes-ingress-controller — Kong Kubernetes Ingress Controller (source, main branch @ `e47c34066`)
- kubernetes-sigs/ingress2gateway — official provider list, including Kong's own provider package
- AKS application routing with the Kubernetes Gateway API — learn.microsoft.com (`ms.date: 2026-07-01`)
- GKE Gateway API overview — cloud.google.com
- STACKIT Application Load Balancing — docs.stackit.cloud (last updated 2026-07-10)
- Part 3 — Annotation Compatibility Matrix — the 17-row matrix every score in this part is drawn from
- Part 4 — Replacement Candidates — the five deep-empirical candidates this part's three context-level ones are compared against

## What's next

Part 6 covers Migration Mechanics: an `ingress2gateway` walkthrough, coexistence patterns, DNS-staged cutover, and a rollback plan. The series hub tracks what's published; Part 4 covers the five candidates tested with full empirical depth.