ingress-nginx Replacement Candidates: Envoy, HAProxy, NGINX

2026-07-18

A source-verified look at Envoy Gateway, two HAProxy controllers, F5′s NGINX controller, and NGINX Gateway Fabric, scored against the Part 3 matrix.

Part 3 answered which annotations survive a migration and which don't. This part answers the question that matrix exists to feed into: given those findings, which target do you actually move to? Five candidates get the same source-verified depth as Part 3′s matrix here — not because the other seven considered for this series are less real, but because "tested with the same rigor" and "worth a mention" are different claims, and this guide tries not to blur them.

► TL;DR

Five candidates, scored against the same 17-row matrix from Part 3: Envoy Gateway (9 green / 8 yellow / 0 red) and F5 NGINX Ingress Controller (8/9/0) come out with the broadest clean coverage; HAProxy Unified Gateway (6/5/6) and NGINX Gateway Fabric (8/5/4) trade broad coverage for a narrower, spec-clean core; HAProxy Kubernetes Ingress Controller (6/10/1) covers almost everything, mostly through its own annotation family rather than standard fields. Only three real proxy engines sit behind these five config-layer choices — NGINX, HAProxy, and Envoy — which matters more than which vendor's CRD you write, since two pairs of these candidates literally share a data plane. Cilium, Traefik, Kong, and the managed-cloud paths (AKS, GKE, STACKIT, OVH) get the same treatment in Part 5, not skipped, just not conflated with these five.

Scope: five candidates, not twelve

This series originally planned five candidates for Part 4. By the time Part 3′s matrix was done, fairness and legitimacy arguments — the same annotation family deserves an Ingress-API answer *and* a Gateway-API answer where both exist, and an official `ingress2gateway` provider earns a seat at the table — had grown that list to twelve: five candidates source-verified against the Part 3 matrix, three tested only at a context level (Cilium, Traefik, Kong), and four managed-cloud-provider paths (AKS, GKE, STACKIT, OVH) that were never part of the empirical lab at all.

Writing all twelve into one part would have repeated Part 3′s own problem — a page too long and too dense to be useful — so this part keeps the scope this series' prompt doc set from the start: only candidates tested with the same depth as the matrix appear here. Cilium, Traefik, Kong, and the managed-cloud survey get their own part next, not a shorter treatment bolted onto this one.

HAProxy, disambiguated

> **Three HAProxy-adjacent names, not one project**
>
> This part covers two of them in depth. Keeping them straight matters, because Part 3's own research caught this exact conflation once already (the `auth-url`/`auth-signin` row's corrected finding):

- **HAProxy Kubernetes Ingress Controller** (`haproxytech/kubernetes-ingress`) — HAProxy Technologies' own controller, covered below. Config layer: `Ingress` + annotations + its own CRDs.
- **`haproxy-ingress/haproxy-ingress`** — a different, unrelated community project, despite the near-identical name. Not covered in this series at all.
- **HAProxy Unified Gateway** ("HUG", `haproxytech/haproxy-unified-gateway`) — also HAProxy Technologies, but a separate, newer codebase (first commit 2025-02-13) that speaks only Gateway API, covered below as its own candidate.

The author has been a contributor to HAProxy and part of the HAProxy community since 2003 (see the about page's background table). Stated openly so the HAProxy sections below can be read with that in mind — the comparison stays on fit and trade-offs, not a ranking, for every candidate in this part, HAProxy included.

Config layer vs. proxy engine

Which CRD or annotation configures a candidate and which process actually proxies the traffic are two different questions — and across these five candidates, the answer to the second one collapses onto only three real engines. Confirmed directly from each candidate's source (module paths, CRD definitions, and — for Envoy Gateway — the Kubernetes `Deployment` its own control plane manages):

Candidate	Config layer	Actual proxy engine
Envoy Gateway	`HTTPRoute`/Gateway API only, no `Ingress` support (confirmed: zero `networking.k8s.io` references in source) + its own Policy CRDs	Envoy — a separate `Deployment` created and driven by Envoy Gateway's own control plane over xDS, not embedded in-process
HAProxy Kubernetes Ingress Controller	`Ingress` + annotations + its own CRDs (`Backend`/`Defaults`/`Global`/`TCP`, confirmed from `crs/definition/*.yaml`)	HAProxy
HAProxy Unified Gateway	`HTTPRoute`/Gateway API only, no `Ingress` support (confirmed: zero `networking.k8s.io` references in source) + its own CRDs (`Global`/ `HugConf`/ `Defaults`/ `HugGate`/ `Backend`, confirmed from `api/definition/*.yaml`)	HAProxy, via its own Data Plane API / `client-native` model
F5 NGINX Ingress Controller	`Ingress` + annotations + its own CRDs (`VirtualServer`/ `VirtualServerRoute`/ `Policy`/`TransportServer`, confirmed from `docs/crd/*.md`), no `HTTPRoute` support at all (confirmed: zero `HTTPRoute` references in source)	NGINX (plain)
NGINX Gateway Fabric	`HTTPRoute`/Gateway API only + its own Policy CRDs (`RateLimitPolicy`, `ClientSettingsPolicy`, `ProxySettingsPolicy`, `WAFPolicy`) — no local clone for this series, confirmed against NGF's own compatibility docs and CRD reference instead	NGINX (plain)

Two convergences worth naming explicitly: the two HAProxy candidates share a data plane despite being separate codebases and separate config surfaces, and F5′s NGINX Ingress Controller shares its engine with NGINX Gateway Fabric despite the two projects choosing opposite config layers (Ingress API vs. Gateway API only) for the same underlying software. "Which proxy does this actually run" is often the more useful migration question than "which vendor," precisely because of pairs like these.

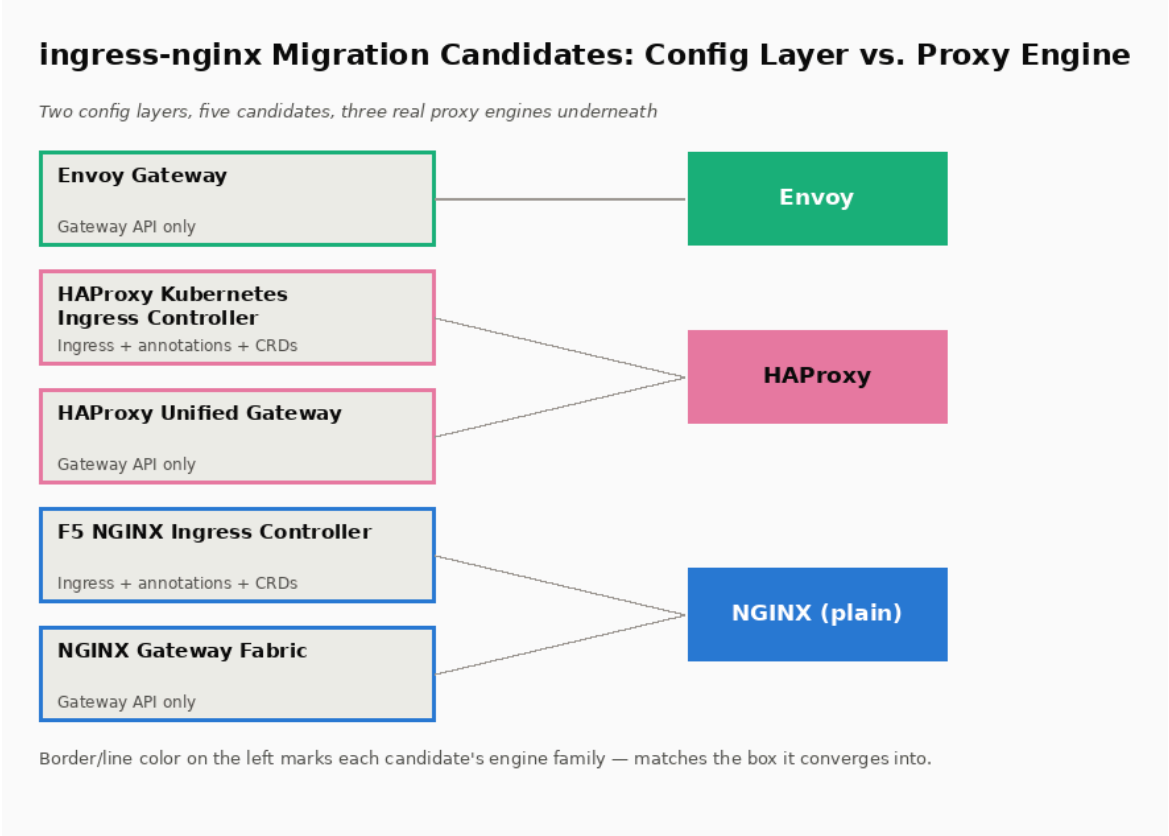

Figure 1: Five ingress-nginx migration candidates grouped by config layer and by which of three proxy engines actually serves traffic

How the five score against Part 3's matrix

Counted directly from the same 17-row CSV Part 3 built, not re-estimated for this part:

Candidate	Green	Yellow	Red	Config layer
Envoy Gateway	9	8	0	Gateway API only
F5 NGINX Ingress Controller	8	9	0	Ingress + annotations
NGINX Gateway Fabric	8	5	4	Gateway API only
HAProxy Kubernetes Ingress Controller	6	10	1	Ingress + annotations
HAProxy Unified Gateway	6	5	6	Gateway API only

Zero reds for Envoy Gateway and F5's controller does not mean either is a clean drop-in — most of their rows land yellow, meaning a policy, extension, or CRD stands in for what used to be a plain annotation, and each of those needs its own validation pass. The two Gateway-API-only newcomers (HUG, NGF) carry the most reds of the five, concentrated in exactly the features neither project has built yet: no snippet mechanism, no rate limiting on HUG, no external auth on either

at the time of Part 3's research. That is a maturity gap, not a design flaw — both are younger, narrower-scoped projects than the other three.

Envoy Gateway

Config layer: Gateway API only. **Proxy engine:** Envoy, via a dedicated `Deployment` the control plane manages over xDS — the standard control-plane/data-plane split, unlike, say, Cilium's per-node embedded model (covered in Part 5).

Envoy Gateway's matrix score (9 green / 8 yellow / 0 red) is the broadest clean coverage of the five, and it earns the two most standards-native answers in the whole matrix: it's the reference implementation team for the Envoy `ext_authz` filter that several other candidates also end up calling, and its `SecurityPolicy` (`ExtAuth`) is the most battle-tested external-auth path Part 3 found anywhere. It also owns this series' sharpest translation-layer finding: `BackendRequest` unconditionally overwrites `Request` in the core `Timeouts` field whenever no `Retry` block is configured — last-write-wins, not the spec-intended composition — a bug class worth checking before trusting Gateway API's core timeout field on any target, not just this one.

For teams already running Istio or considering a service mesh alongside ingress, the Istio vs. Envoy Gateway comparison covers the mesh-vs-gateway question this part doesn't — Envoy Gateway and Istio's own Gateway API surface solve overlapping problems from different starting points, and that choice is orthogonal to picking a migration target here.

Last verified 2026-07-17 against `envoyproxy/gateway` main @ `b68043e7` (2026-07-15).

HAProxy Kubernetes Ingress Controller

Config layer: `Ingress` + annotations + its own CRDs. **Proxy engine:** HAProxy.

The closest lineage to `ingress-nginx` of any Ingress-API candidate in terms of migration shape — teams staying on the classic `Ingress` object get a controller with a rich annotation family of its own (`rate-limit-*`, `cookie-persistence`, `timeout-*`, `route-acl` covering both weight- and header/cookie-based canary through one mechanism) rather than a thin adapter. That breadth shows up in the matrix score too: only one red in 17 rows, the widest raw coverage of any candidate here, concentrated almost entirely through this controller's own annotations rather than standard fields.

Its own Gateway API support is real but frozen: `documentation/gateway-api.md` states TCPRoute-only support on Gateway API v0.5.1, a pre-GA release from before Gateway API reached v1.0, and the one commit that ever touched that file is dated 2023-02-23 — no feature growth since, only maintenance-level commits (RBAC handling, error-message cleanup) confirmed in the same clone. Read together with HAProxy Unified Gateway's first commit landing in 2025-02-13, this looks like HAProxy Technologies' real Gateway API investment moving into that separate, purpose-built project instead of this controller's own stalled attempt — an inference from the evidence, not a confirmed vendor statement.

Last verified 2026-07-17 against `haproxytech/kubernetes-ingress` v3.2.12.

HAProxy Unified Gateway

Config layer: Gateway API only. **Proxy engine:** HAProxy, via its own Data Plane API / `client-native` model.

The youngest project in this part (first commit 2025-02-13) and the only one of the five reaching double digits on neither green nor yellow — its matrix score (6/5/6) carries the most reds here, concentrated exactly where you'd expect from a project still building out feature parity: no snippet mechanism of any kind, no rate-limit policy or filter, no external-auth filter implemented at all. What it does implement, it implements natively against the actual Gateway API spec fields rather than bolting a vendor Policy CRD on top — its `SessionPersistence` (GEP-1619) and weighted `backendRefs` support both come from the core spec, not a workaround, and it's one of only two targets in the whole Part 3 matrix reaching session affinity that way (the other is Envoy Gateway).

For a team specifically committed to Gateway API as the long-term interface and comfortable with a narrower current feature set, that standards-native core is the actual trade-off HUG offers — not a smaller HAProxy, a different bet on what "done" means for a Gateway API implementation.

Last verified 2026-07-17 against `haproxytech/haproxy-unified-gateway` v1.0.5.

F5 NGINX Ingress Controller

Config layer: `Ingress` + annotations + its own CRDs, no Gateway API support at all (confirmed: zero `HTTPRoute` references in source). **Proxy engine:** NGINX (plain).

The closest lineage of any target to the retired `ingress-nginx` in the literal sense — several annotations carry over under a different prefix (`nginx.org/rewrite-target` mirrors `nginx.ingress.kubernetes.io/rewrite-target` exactly), and its `VirtualServerRoute` CRD's `rewritePath` field explicitly documents `$1` through `$9` capture-group support, the one case in this matrix where a regex rewrite carries over cleanly on the Ingress-API side. Its matrix score (8/9/0) matches Envoy Gateway's for breadth, with the same caveat: no reds, but the majority of rows are yellow, meaning "your own CRD, not the retired controller's annotation" more often than "the exact same config."

One gap worth flagging on its own: no ModSecurity integration exists in this controller at all — F5′s commercial NGINX App Protect is the vendor's own answer, a genuinely different product with a different rule syntax, not a ModSecurity-compatible drop-in. Teams relying on `modsecurity-snippet` or `enable-owasp-core-rules` on `ingress-nginx` should treat that as a hard stop for this candidate, not a configuration detail to work around.

Last verified 2026-07-17 against `nginx/kubernetes-ingress` v5.5.3.

NGINX Gateway Fabric

Config layer: Gateway API only. **Proxy engine:** NGINX (plain) — the same engine as F5′s controller above, reached through the opposite config-layer choice.

No local clone exists for this series; every claim below traces to NGF's own public compatibility docs and CRD reference rather than source, flagged the same way Part 3 flagged it. Its matrix score (8/5/4) is the second-broadest green coverage of the five, driven by a fast-moving policy-CRD roadmap: `ClientSettingsPolicy` (body size, June 2024), `ProxySettingsPolicy` (buffering,

January 2026), and `WAFPolicy` (May 2026, F5 WAF for NGINX, Plus-only) each landed within roughly the last two years, closing gaps this matrix would have scored red a year earlier. Its four reds cluster around licensing, not missing engineering: cookie-based session persistence and the WAF policy both require an NGINX Plus subscription, with no free-tier equivalent the way F5's own Ingress Controller offers one for sticky sessions.

That licensing split is the concrete trade-off here: NGF's free tier is narrower than either NGINX Ingress Controller's or the two HAProxy candidates', for annotations that gate on a Plus license rather than on unfinished code.

Last verified 2026-07-17 against NGF's public compatibility docs and CRD reference (no local clone).

What this doesn't answer yet

None of the above ranks these five candidates — "broadest coverage" and "best fit" are different questions, and which one matters depends on which of Part 3's rows your own `Ingress` objects actually touch, whether Gateway API as an interface is itself a goal or an incidental means, and how much a licensing gate versus an unfinished feature changes the calculus for your own budget. Cilium, Traefik, Kong, and the managed-cloud-provider paths (AKS, GKE, STACKIT, OVH) that this series originally planned to fold into this same part get exactly that same fit/trade-off treatment next, not a shorter version of it.

Sources

- envoyproxy/gateway — Envoy Gateway (source, main branch @ `b68043e7`, 2026-07-15; confirms no Ingress API support, the managed-`Deployment` proxy architecture, and the `Timeouts` overwrite behavior)
- haproxytech/kubernetes-ingress — HAProxy Kubernetes Ingress Controller (source, v3.2.12)
- haproxytech/haproxy-unified-gateway — HAProxy Unified Gateway "HUG" (source, v1.0.5)
- nginx/kubernetes-ingress — F5 NGINX Ingress Controller (source, v5.5.3)
- NGINX Gateway Fabric Gateway API compatibility — docs.nginx.com
- kubernetes-sigs/gateway-api — Gateway API (source, main branch)
- kubernetes-sigs/ingress2gateway — the provider list that motivated including Kong in this series (source, v1.2.0)
- Part 3 — Annotation Compatibility Matrix — the 17-row matrix every score in this part is drawn from

What's next

Part 5 covers the three context-level candidates this series tests but doesn't deep-dive (Cilium, Traefik, Kong) and the managed-cloud-provider survey (AKS, GKE, STACKIT, OVH) — content-level, not empirically lab-tested, the same distinction this part drew for the five above. The series hub tracks what's published; Part 3 is the matrix every score above is drawn from.