ingress-nginx Migration: A Field Guide

2026-07-18

A practical field guide to migrating off the retired ingress-nginx controller: inventory, annotation compatibility, migration mechanics, and a companion lab.

This is the index for a field guide to migrating away from `ingress-nginx`, the Kubernetes Ingress controller that was archived and retired in March 2026. Each part stands on its own, but together they cover the whole path: deciding whether to act, finding out what you actually depend on, knowing which annotations translate cleanly, choosing a target, and doing the migration itself without a flag day.

The differentiator here is not another "here are the alternatives" list — plenty of those already exist. It's an inventory-first approach, an annotation compatibility matrix built from observed behaviour rather than name-matching, real migration mechanics, a reproducible companion lab, and gotchas documented from actually running the migration, not just reading about it.

How to read this guide

If you haven't yet decided whether this affects you, start with Part 1 — it takes ten minutes and answers that question directly. If you already know you need to migrate and want the practical work, the later parts will get progressively more hands-on: inventory, then a compatibility matrix, then candidate controllers, then the migration mechanics themselves.

What's retired, and what isn't

Retired?	Item	Meaning
Yes	`ingress-nginx` controller	Archived read-only on GitHub since 2026-03-24. No further releases, bug-fixes, or security patches.
No	Kubernetes Ingress API	Still part of Kubernetes; other controllers keep implementing it.
No	NGINX (the software)	Unaffected — `ingress-nginx` was one controller built on top of it, not NGINX itself.
No	NGINX Gateway Fabric	A separate, actively maintained F5/NGINX project implementing Gateway API. Not the retired controller, despite the similar name.
No (separate project)	Chainguard EmeritOSS fork	A maintained fork for stability and CVE backports, started in response to the retirement — not upstream `ingress-nginx`, and not framed by its maintainers as a permanent substitute.

Part 1 covers the reasoning and the sourcing behind this table in full.

Publication status

Part	Status	Audience	Topic
Part 1 — Retirement Decision	Published	Decision-makers, architects	Why it matters, what's retired, four honest options
Part 2 — Inventory Before You Migrate	Published	Platform engineers	Finding out what your clusters actually depend on
Part 3 — Annotation Compatibility Matrix	Published	Platform engineers	Which `nginx.ingress.kubernetes.io/*` annotations translate cleanly, and which don't
Part 4 — Replacement Candidates	Published	Platform engineers, architects	Source-verified look at Envoy Gateway, HAProxy Kubernetes Ingress Controller, HAProxy Unified Gateway, F5 NGINX Ingress Controller, and NGINX Gateway Fabric, evaluated against the Part 3 matrix
Part 5 — Alternatives	Published	Platform engineers, architects	Context-level look at Cilium, Traefik, and Kong, plus a managed-cloud-provider survey (AKS, GKE, STACKIT, OVH)
Part 6 — Mechanics	Published	Platform engineers	`ingress2gateway`, coexistence patterns, cutover, rollback
Part 7 — Hard Cases + Full Migration	Published	Platform engineers	A deliberately nasty sample app, migrated live on OVH MKS, with every failure documented

Part 1 — Retirement Decision

Audience: decision-makers and architects.

`ingress-nginx` isn't retiring soon — it already has. Part 1 covers what exactly that means, what is and isn't affected, the CVEs disclosed since the archival date, and four honest options: stay and accept the risk, switch controllers, move to Gateway API, or use a maintained fork as a bridge. No code, no vendor preference, no panic.

Key takeaways:

- The `kubernetes/ingress-nginx` repository was archived read-only on 2026-03-24 — this is a completed fact, not an upcoming deadline.
- Existing deployments keep running unchanged; what stops is future fixes, including for whatever security issue gets found next.
- There isn't one right answer among the four options — the right one depends on annotation usage and risk tolerance that later parts help you actually measure.

Read Part 1 — Retirement Decision →

Part 2 — Inventory Before You Migrate

Audience: platform engineers.

Part 2 answers the second question: what do you actually depend on? A `kubectl`/`jq` one-liner inventories every `nginx.ingress.kubernetes.io/*` annotation in the cluster, plus the two other surfaces that annotation-only inventories miss — the global controller `ConfigMap` and controller flags that gate certain annotations.

Key takeaways:

- Comparing replacement controllers before inventorying your own cluster is the wrong first step.
- TCP/UDP services configured through the `tcp-services`/`udp-services` ConfigMaps are invisible to any annotation-based inventory — a common red-migration case if you rely on them.
- An annotation being present in a chart doesn't mean it's load-bearing; only observed behaviour tells you that.

Read Part 2 — Inventory Before You Migrate →

Part 3 — Annotation Compatibility Matrix

Audience: platform engineers.

Part 3 answers the third question: for each annotation you depend on, does it migrate cleanly, or does it silently break? A traffic-light matrix (Green/Yellow/Red, plus a more precise behaviour classification) covers the hard cases named in this guide — snippets, external auth, canary, rewrites, rate limiting, session affinity, timeouts, TCP/UDP services — with real per-target findings, not name-matching guesses.

Key takeaways:

- `rewrite-target` is green for a simple prefix rewrite and red for a regex capture group — same annotation, opposite verdict depending on the value, not the name.
- Some targets fail loudly when a standard field isn't supported (NGINX Gateway Fabric rejects Gateway API's `Timeouts` field outright); at least one fails silently (Traefik accepts the field and ignores it).
- The full matrix is a living CSV in the companion repo, not a finished 100+-row reference — this part covers the groups that matter most first.

Read Part 3 — Annotation Compatibility Matrix →

Part 4 — Replacement Candidates

Audience: platform engineers, architects.

Part 4 answers the fourth question: given what Part 3 found, which target do you actually move to? Five candidates get the same source-verified depth as the matrix — Envoy Gateway, HAProxy Kubernetes Ingress Controller, HAProxy Unified Gateway, F5 NGINX Ingress Controller, and NGINX Gateway Fabric — each scored against the same 17 rows, not name-matched against a feature list.

Key takeaways:

- Only three real proxy engines sit behind these five candidates' config layers — NGINX, HAProxy, and Envoy — and two pairs of candidates share a data plane despite being separate codebases.
- Zero reds in the matrix doesn't mean a clean drop-in: most of Envoy Gateway's and F5's controller's rows land yellow, meaning a policy or CRD stands in for what used to be a plain annotation.
- The two newest, Gateway-API-only candidates (HAProxy Unified Gateway, NGINX Gateway Fabric) carry the most reds — a maturity gap in what's been built so far, not a design flaw.

Read Part 4 — Replacement Candidates →

Part 5 — Alternatives

Audience: platform engineers, architects.

Part 5 covers what Part 4 deliberately left out: three candidates this series tests but doesn't deep-dive — Cilium, Traefik, Kong — plus a survey of what migrating off ingress-nginx looks like on four managed Kubernetes platforms (AKS, GKE, STACKIT, OVH) that were never part of the empirical lab at all.

Key takeaways:

- Cilium, Traefik, and Kong each score against the same 17-row matrix as Part 4's candidates, just at a lighter verification depth — context-level, not deep-empirical.
- 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.
- "Managed Gateway API" doesn't imply one consistent runtime model — AKS runs in-cluster Istio proxies, GKE programs a cloud load balancer from entirely outside the cluster, and OVH (per this blog's own production repos) self-manages the whole stack.

Read Part 5 — Alternatives →

Part 6 — Mechanics

Audience: platform engineers.

Part 6 answers the question every migration guide owes its reader eventually: how do you actually move traffic without a flag day? An `ingress2gateway` walkthrough, coexistence patterns, a DNS-staged cutover, cert-manager and external-dns on Gateway API, monitoring, smoke tests, and a rollback plan as its own section.

Key takeaways:

- `ingress2gateway` v1.0.0 recognizes 47 `nginx.ingress.kubernetes.io/*` annotations, but skips almost every hard case this series has covered — no rate limiting, no external auth, no snippets, no TCP/UDP services.
- A successful conversion proves an annotation was recognized, not that the target behaves the same way — the `Timeouts` field converts cleanly into every target's core field, yet Part 4 already found three different failure modes hiding behind it.

- A rollback plan that has never been executed is a hypothesis, not a plan — this part treats rollback as its own section, not a bullet point inside the cutover plan.

Read Part 6 — Mechanics →

Part 7 — Hard Cases + Full Migration

Audience: platform engineers.

Part 7 is the one part of this series that's lived, not researched: a real OVH Managed Kubernetes Service cluster, a real domain, one deliberately nasty sample application, migrated live to two Gateway API targets — Envoy Gateway and HAProxy Unified Gateway — against the exact same 10-hard-case test script used for the ingress-nginx baseline. Every result is a real HTTP response or a line read out of a running pod, not a docs claim.

Key takeaways:

- Six of ten hard cases behaved identically across all three targets; rate limiting diverged three different ways, and HAProxy Unified Gateway showed no enforcement at all for both rate limiting and external auth — a coincidental `404`, not a stricter policy.
- The single most structurally interesting finding: HAProxy Unified Gateway needs one `HTTPRoute` per host, not one per hard case — found by reading a generated HAProxy config file inside a running pod after a live 404, not guessed at in advance.
- Envoy Gateway's timeout-overwrite bug documented in Part 4 does not reproduce at v1.8.2; a security-relevant gap that no test case was designed to catch showed up anyway — ingress-nginx's default `Strict-Transport-Security` header disappears on both Gateway API targets unless re-added.

Read Part 7 — Hard Cases + Full Migration →

The through-line

Two things run through every part of this series, even the ones not yet written:

Measure before you migrate. The annotation matrix in Part 3 and the inventory approach in Part 2 exist because guessing what a cluster depends on is how migrations break production. An annotation being present in a Helm chart doesn't mean it's load-bearing; the only way to know is to check observed behaviour, not the annotation's name.

"It depends" is only useful with the dependencies named. Part 4's deep candidate comparison and Part 5's managed-cloud-provider survey exist to make the trade-offs concrete instead of leaving "it depends on your setup" as the final word.

Companion repo

A companion lab repo, `ingress-nginx-migration-lab`, on Codeberg lets you measure your own cluster's behaviour before and after a migration, rather than trusting a written claim about what "should" carry over. It's a skeleton for now — the inventory script from Part 2 and the planned annotation-matrix schema and test-suite layout — and grows into a full reproducible lab as later parts of this series get published.

Where this guide stands

This is a living field guide, not a finished reference — it grows as parts are published, and the publication-status table above always reflects the current state honestly, including the parts that aren't written yet. Everything in Part 1 is dated and sourced; treat later, unpublished parts as planned scope, not as promises about a specific date.

Start with Part 1 — Retirement Decision →