ingress-nginx Migration Mechanics: ingress2gateway and Cutover

2026-07-18

An ingress2gateway walkthrough, coexistence patterns, DNS cutover, cert-manager, external-dns, and a rollback plan for a flag-day-free ingress-nginx migration.

Parts 3 through 5 answered what migrates cleanly and to which target. This part answers the question every migration guide owes its reader eventually: how do you actually move traffic without a flag day? A converter tool, a coexistence window, a staged cutover, and — the section every migration plan needs but few write down — an actual rollback plan.

► TL;DR

`ingress2gateway` v1.0.0 (released 2026-03-21) recognizes 47 distinct `nginx.ingress.kubernetes.io/*` annotations in its ingress-nginx provider — more than the "30+" figure sometimes quoted, confirmed by reading the source directly. It converts most of Part 3's "clean" annotations syntactically without issue, but flatly skips almost every hard case this series has covered: no rate limiting, no `auth-url`/`auth-signin`, no snippets, no `tcp-services`/ `udp-services`. Even where it does convert cleanly — the `Timeouts` field is the sharpest example — a successful conversion is not proof of equivalent behavior; Part 4 already found three different failure modes hiding behind that one field, per target. The rest of this part is the mechanics around the tool: coexistence, DNS-staged cutover, cert-manager and external-dns on Gateway API, monitoring, smoke tests, and a rollback plan as its own section, not an afterthought.

⚠ A migration is not done when manifests apply

It is done when traffic, logs, redirects, headers, TLS, auth, timeouts, canaries, and rollback have all been verified — not when `kubectl apply` exits zero. Every section below assumes that bar.

`ingress2gateway`: what it actually converts

`ingress2gateway` v1.0.0 shipped 2026-03-21 (confirmed via the tag's own commit date — "reportedly 2026-03-20" was one day off). Its ingress-nginx provider recognizes 47 distinct `nginx.ingress.kubernetes.io/*` annotation strings, confirmed by reading the provider's source directly rather than trusting a release-notes figure — one dedicated Go file per feature (`canary.go`, `rewrite.go`, `session_affinity.go`, `backend_protocol.go`, `timeouts.go`,

`cors.go`, `ssl_passthrough.go`, `bodysize.go`, `iprange.go`, `redirect.go`, `approot.go`, `backend_tls.go`, `headers.go`).

What converts cleanly, syntactically: weight- and header/cookie-based canary, simple prefix/full-path rewrites, custom headers, CORS, body size, `backend-protocol`, session affinity (the core `SessionPersistence` field), the three timeout annotations (into core `Timeouts`), SSL passthrough, allow/deny source ranges, redirects, and app-root. Each has real test coverage in the tool's own repo, not just a mapping table.

Where it gives up entirely: none of Part 3′s hardest cases appear anywhere in that annotation list — no rate-limit family, no `auth-url`/`auth-signin`, no `modsecurity-snippet` or any other snippet annotation, no `tcp-services`/`udp-services`. If your own `Ingress` objects depend on any of these, `ingress2gateway` has nothing to convert; the manual work Part 3′s per-annotation findings already describe is the actual plan for those rows, not a tool run.

What output still needs review: the tool has a real notification mechanism, not silent best-effort guessing — `notifications.WarningNotification` and `notifications.ErrorNotification` fire per-annotation for exactly the cases that need a human look (an unsupported value, a malformed `canary-weight`, a `canary-weight` exceeding `canary-weight-total`). Read the tool's own output, not just the generated YAML.

> **Syntactic conversion is not semantic equivalence**
>
> The `Timeouts` field is the sharpest example of why a clean conversion still needs verification. `ingress2gateway` parses `proxy-connect-timeout`/`proxy-read-timeout`/`proxy-send-timeout` (unitless seconds, confirmed from `timeouts.go`) straight into Gateway API's core `Timeouts` field without complaint. But Part 4 already found three different failure modes hiding behind that exact field, per target: Cilium collapses both sub-fields via `min()`, Envoy Gateway's `BackendRequest` unconditionally overwrites `Request` without a `Retry` block, and NGINX Gateway Fabric rejects the field outright. A successful `ingress2gateway` run proves the annotation was *recognized* — it proves nothing about what your specific target actually does with it. Re-check the relevant part's per-target findings after every conversion, not just after picking a target.

Coexistence patterns

Running the old and new controller side by side is the standard way to avoid a flag day: a separate `IngressClass` for the retiring controller and a separate `GatewayClass` for the new one, both pointed at different subsets of traffic (by hostname, by namespace, or by an explicit migration label) while the cutover is staged. This is general Kubernetes practice, not something this blog's own infrastructure has done for an ingress-nginx migration specifically — none of this blog's own production clusters ever ran `ingress-nginx` (confirmed in the previous part's research), so this section is deliberately generic guidance, not a retold migration story.

DNS-staged cutover and the LoadBalancer swap

A staged cutover shifts traffic gradually rather than repointing DNS all at once: weighted DNS records (where the provider supports it) or a temporary dual-listener setup that accepts traffic on both the old and new `LoadBalancer` Service during the observation window. Only after the new path has carried real production traffic without incident does the old `LoadBalancer` get decommissioned — not before, and not automatically on a timer.

cert-manager with Gateway API

cert-manager's Gateway API support is a real, separate controller — the `gateway-shim`, gated behind the `--enable-gateway-api` flag — confirmed from source (`internal/controller/feature/features.go`, `cmd/controller/app/options/options.go`). It watches `Gateway` resources for `cert-manager.io/*` annotations the exact same way the core controller watches `Ingress` — the annotation-driven certificate-request model carries over, only the watched resource type changes.

One layer below that — the actual DNS-01 challenge against a specific DNS provider — is a separate concern cert-manager delegates to a webhook solver, regardless of whether the certificate request came from an `Ingress` or a `Gateway`. This blog's own cert-manager-webhook-libdns is one such webhook — its provider registry is now generated from the project's own compatibility-report data rather than a small hand-picked list, and compiles in 72 libdns provider implementations from one codebase, confirmed by running `--list-providers` against the current build. Worth knowing about specifically because it's agnostic to this migration: the same webhook keeps working unchanged whether the `ClusterIssuer` it serves is triggered from `Ingress` or `Gateway` resources.

Last verified 2026-07-17 against `git001/cert-manager-webhook-libdns` main @ `06e90ea`, via its own `--list-providers` output.

external-dns with Gateway API sources

external-dns fully supports Gateway API as a DNS source, via five dedicated source flags: `--source=gateway-httproute`, `-grpcroute`, `-tlsroute`, `-tcproute`, and `-udproute`. Version requirements differ by route kind — `HTTPRoute` and `Gateway` need Gateway API v1.0.0+ (the v1 API), `GRPCRoute` needs v1.1.0+, while `TCPRoute`/`UDPRoute` remain on the experimental `v1alpha2` API with no standard-channel release date yet. One concrete limitation worth planning around: `TCPRoute`/`UDPRoute` specs can't carry hostnames at all, so external-dns falls back to the `external-dns.alpha.kubernetes.io/hostname` annotation for those — `HTTPRoute` and `TLSRoute` should specify every intended hostname in the route spec itself instead, since the annotation fallback isn't reliably recognized by the actual Gateway serving the traffic.

Last verified 2026-07-17 against `kubernetes-sigs/external-dns`'s own Gateway API source docs.

Monitoring and access-log comparison before and after

Compare the same signals across both paths during the coexistence window, not just after cutover: request rate, error rate, and latency percentiles per route, plus a direct access-log diff for the specific hard-case annotations Part 3 flagged as risky for your own cluster. If your monitoring stack already

ingests access logs into a queryable store, reuse it here rather than building a one-off comparison — this blog's own SigNoz on OVH MKS: Access Log Reports with Vector and ClickHouse post covers one concrete way to build exactly that kind of before/after reporting.

Smoke tests

Before shifting any real traffic weight, run the same request set — normal paths, the specific hard-case annotations your own inventory (Part 2) flagged, and at least one deliberately malformed request — against the new path directly, bypassing DNS. A smoke-test failure caught here is far easier to act on than the same failure discovered after a weight shift has already moved real traffic.

Rollback plan

This gets its own section deliberately, not a bullet inside the cutover section — a plan that only exists as "shift the DNS weight back" is not a rollback plan, since state that changed underneath the new path (a rotated certificate, a re-provisioned `LoadBalancer` IP, an `external-dns` record update) doesn't necessarily revert just because traffic does. Before cutover starts, confirm: the old `IngressClass`'s `LoadBalancer` and DNS records still exist and are reachable, the old certificate hasn't been allowed to expire during the coexistence window, and the actual rollback step (a DNS weight change, a Service selector flip, or both) has been tested at least once in a non-production environment. A rollback plan that has never been executed is a hypothesis, not a plan.

Security review after snippet removal

Part 3 named ingress-nginx's snippet annotations as this migration's clearest "no" — arbitrary NGINX configuration injected through an annotation is real custom proxy code, not configuration, and every target this series evaluated either has no equivalent at all or gates the same capability behind a structured, narrower mechanism. Removing snippet annotations during a migration is a genuine security improvement, but only if it's actually verified rather than assumed: audit what each removed snippet was doing (the required per-annotation line from Part 3 — "no portable equivalent by design" — should have been landing in this exact review already), and confirm the replacement mechanism (a policy CRD, a vendor extension, or a deliberate "not supported, redesign needed") actually covers the same behavior before treating the snippet as safely gone.

What this doesn't answer yet

None of the mechanics above are specific to any one target from Parts 4 or 5 — the same coexistence/cutover/rollback shape applies whether the destination is Envoy Gateway, HAProxy Unified Gateway, or a managed-cloud Gateway API path. What differs by target is entirely covered in those earlier parts: which annotations survive, and which mechanism replaces the ones that don't. Part 7 puts all of this into one deliberately nasty sample application and migrates it for real.

Sources

- kubernetes-sigs/ingress2gateway — source, main branch (v1.0.0 tag confirmed 2026-03-21)
- cert-manager/cert-manager — source, confirms the `gateway-shim`/`--enable-gateway-api` mechanism
- kubernetes-sigs/external-dns — Gateway API source documentation

- libdns/libdns — the provider interface cert-manager-webhook-libdns implements against
- git001/cert-manager-webhook-libdns — source, main branch @ `06e90ea`
- Part 3 — Annotation Compatibility Matrix — the per-annotation findings this part's conversion coverage is checked against
- Part 4 — Replacement Candidates — source of the `Timeouts` field's three per-target failure modes cited above

What's next

Part 7 covers the hard cases: one deliberately nasty sample application — TLS, regex rewrite, canary, external auth, rate limit, WebSocket, gRPC, a large body, a tight timeout, and one snippet — migrated live, with every failure documented, not smoothed over. The series hub tracks what's published; Part 5 covers the context-level candidates and managed-cloud survey.