WHITEPAPER · V1.0 · AUGUST 2026

The State of Custom Domain Infrastructure

Why DNS onboarding breaks, and how programmatic domain connection fixes it

---

A technical reference for engineers and product teams who let customers bring their own domain.

customdomain.ai

About this paper

This paper is written for engineers and technical decision-makers at B2B SaaS companies who have to let customers point their own domains at a product, and who have discovered that the difficulty is not in documenting the records.

It is organised so the sections are useful independently. The executive summary states the argument. The problem and landscape sections are vendor-neutral and should be readable by someone who intends to build this themselves. The reference architecture is the technical core. The evaluation checklist near the end is deliberately written so that it remains useful to a reader who chooses a competitor, or who chooses to build.

On sourcing. Every external claim carries a citation. Where a figure comes from a vendor-published case study, it is labelled as an attributed customer claim rather than presented as independent research — including where that vendor is a competitor. Where a widely-repeated number could not be traced to a primary source, that is stated rather than quietly reproduced. Product capabilities described here are ones that exist and run; where a capability is limited, the limit is given.

On the numbers about our own coverage. This paper reports that a majority of DNS providers have no automated write rail and fall back to manual record entry. That is not a competitor's weakness we are pointing at; it is the shape of the ecosystem, and it applies to us. A paper that claimed otherwise would be worth less to the reader.

Executive summary + The problem

Executive summary

Before Fourthwall automated its DNS onboarding, "nearly half of Fourthwall's support tickets were related to DNS configuration," and each ticket took 15 minutes to an hour of back and forth with the customer to close.[1] Letting a customer point their own domain at your product is table stakes in B2B SaaS. The step that delivers it is a manual write into a zone the vendor does not control, performed by a person who frequently cannot name their DNS provider, verified through caches the vendor also does not control.

The failures here are mechanical. A CNAME cannot sit at a zone apex (RFC 1034 §3.6.2, RFC 2181 §10.1). A pre-existing CAA record that does not authorize your CA blocks issuance inside the CA rather than at the DNS write (RFC 8659). Recursive resolvers cache independently, so there is no observable moment of propagation. A second SPF record breaks mail instead of adding a feature (RFC 7208 §3.2).

Where the provider exposes a write path a third party can drive, the human comes out of the loop; where it does not, verification is the only lever left. That split produces four components: nameserver-based provider detection, a census that routes each domain across three automated write rails over a manual floor, a bounded propagation state machine, and fail-closed certificate issuance. We also report where automation is unavailable. In our 63-provider census, 38 providers have no working automated write rail and fall back to copy-paste. Domain Connect, the open standard built to fix this, lists nine live DNS provider implementations on its own site;[2] Kowalik's APNIC post puts the number at roughly twenty, covering 35% of the `.com` zone as of May 2024.[3] We use the standard's own list because it is the only one with a checkable membership.

Why bring-your-own-domain is table stakes

Customers want `app.theircompany.com`, not `theircompany.yourproduct.io`. The reasons are functional as often as they are cosmetic: cookie and session scope, link equity on published pages, and security review checklists that treat a vendor-owned hostname as third-party data handling.

Email pushed it from preference to requirement. Since 2024-02-01, Google requires senders of more than 5,000 messages per day to personal Gmail accounts to publish SPF and DKIM on the sending domain and to have a DMARC policy, which may be `p=none`.[4] Any product that sends mail on a customer's behalf now has to get records into that customer's zone. There is no version of the feature that skips DNS.

The cost of that step lands on the vendor. Crisp reported going from "five or six DNS tickets per day" to "maybe one" after automating the flow, an 83% reduction.[5] Fourthwall had hired part-time contractors purely to handle DNS support.[6] Both figures come from vendor-published case studies and should be read as attributed customer claims.

We looked for independent research on custom-domain onboarding drop-off and found none. The most-quoted number in the category, that Microsoft 365 domain setup takes 7 to 15 DNS entries across 16 help sites and 40 minutes of training with 50% of users abandoning, appears in an APNIC blog post by Pawel Kowalik of DENIC with no source given for those specific figures.[7] We cite it as an attributed estimate and not as research.

Where DNS onboarding breaks

The user does not know their DNS provider. Registrar and DNS operator are separate roles. The registrar holds the registration and submits the delegation to the registry, which publishes the NS records in the parent zone; whoever operates those nameservers answers queries. A domain registered at GoDaddy but delegated to `ns1.digitalocean.com` is managed at DigitalOcean, and the GoDaddy DNS panel is inert. Onboarding asks the user to recall a delegation decision a contractor made three years ago.

Records get pasted wrong. GoDaddy and Namecheap panels take a host relative to the zone origin and append the domain for you. Paste `_acme-challenge.example.com` into a panel rooted at `example.com` and you create `_acme-challenge.example.com.example.com`. Trailing dots, the `@` convention for the apex, and quoting rules all vary by provider. TXT strings above 255 bytes must be split into multiple character-strings (RFC 1035 §3.3.14), and providers disagree about whether they do that for you. The vendor sees none of this, because the write happens in a UI the vendor has never rendered.

Propagation is invisible. There is no propagation event. Authoritative servers have the record immediately; each recursive resolver learns it whenever its own cached entry expires. If anyone queried the name before the record existed, the resolver cached the negative answer for the zone's negative TTL, which RFC 2308 §5 takes from the lower of the SOA MINIMUM field and the SOA record's own TTL: 300 seconds for `cloudflare.com`, 1,800 for `digitalocean.com` and 3,600 for `godaddy.com` when we measured with `dig` on 2026-08-02. Checking too early makes the wait longer. Meanwhile the customer refreshes, sees nothing, and opens a ticket.

CAA blocks certificate issuance silently. CAA checking is mandatory under the CA/Browser Forum Baseline Requirements (RFC 8659). Before issuing, the CA queries CAA at the FQDN and walks up the tree. A leftover `example.com. CAA 0 issue "digicert.com"` from a previous vendor relationship forces Let's Encrypt to refuse. Every record the user added is correct and resolving. The failure occurs inside the CA, minutes later, and reaches end users as a TLS handshake error. A related trap: Let's Encrypt notes a provider "does not need to specifically support CAA records; it only needs to reply with a NOERROR response for unknown query types."[8] Providers that answer SERVFAIL for unknown types break issuance without ever holding a CAA record.

Apex domains cannot take a CNAME. RFC 1034 §3.6.2 says no other data should be present at a name that holds a CNAME, and RFC 2181 §10.1 turns that into a hard prohibition; the apex must carry SOA and NS. So `example.com. CNAME edge.vendor.net.` is illegal, and vendors want a CNAME precisely because their edge addresses change. Providers invented non-standard workarounds: Cloudflare flattens the CNAME at query time, others expose ALIAS or ANAME pseudo-types resolved

server-side. Coverage is uneven and the spelling differs per provider. Route 53 is a hard case: its alias is an `<AliasTarget>` element in the API rather than a record type, and it cannot point at an arbitrary external hostname.

Existing records conflict with the new ones. SPF permits exactly one policy record per domain (RFC 7208 §3.2); a second one makes the check return permerror, and under a DMARC policy of quarantine or reject that costs delivery for mail that authenticated fine the day before. The correct operation is a merge of an `include:` into the existing string, staying inside the 10-lookup limit of §4.6.4. An existing A record at `www` cannot coexist with a new CNAME at `www` either. Some panels reject that write; others replace it and quietly take down the customer's marketing site.

Showing the correct records is a documentation problem. Getting them into a stranger's zone and confirming they arrived is the engineering problem, and it is the one that generates the tickets.

How the DNS record gets written today, and what each way costs

Three patterns cover how products ask a customer to write the DNS record: a support document, an in-app record table, and an integration vendor.

The first is a support document. The product displays a target hostname, links to a help-center article, and the customer leaves to find their registrar. Whoever wrote the article now owns a matrix problem: the article has to be correct for GoDaddy's panel, Squarespace's panel, Namecheap's two different panels, and whatever the customer's agency set up in 2019. Screenshots rot faster than the text around them.

The second is a record table rendered inside the app. Type, host, value, TTL, a copy button, and a verify poll. This is better, because the values are generated rather than transcribed, but the customer still has to log in somewhere else, find the right zone, and interpret what "host: @" means in a panel that calls it something else. Apex records are where this pattern breaks. A root domain cannot hold a CNAME under RFC 1034 rules, so the correct instruction depends on whether the provider offers flattening, ALIAS, or ANAME, and the product usually does not know which.

The third is an integration vendor: a JavaScript widget that detects the provider and writes the record on the customer's behalf.

What the failure costs

The public numbers here are thin. We found no independent, peer-reviewed measurement of custom-domain onboarding drop-off or support cost. Everything in circulation traces back to vendor case studies or to a single uncited figure.

That figure is the most-quoted one in the category: configuring Microsoft 365 on a custom domain as a six-step process needing 7 to 15 DNS entries, 16 help sites, and 40 minutes of training, with 50% of users failing and abandoning.[1] It appears in an APNIC post by Pawel Kowalik of DENIC, who cites CENTR elsewhere in the same post but gives no source for these numbers. Treat it as an attributed claim from a standards author, not as research.

The customer-attributed numbers are more specific and come with named people. Fourthwall reported that nearly half its support tickets concerned DNS configuration, that each took 15 minutes to an hour of back-and-forth, and that it had hired part-time contractors to absorb the load. After adopting Entri, it reported a reduction of as much as 97%.[2] Crisp reported going from five or six DNS tickets a day to about one, an 83% cut, with CEO Baptiste Jamin quoted directly.[3] Both are published by the vendor. They are consistent with each other and with the shape of the problem, and they are not independent evidence.

One structural change makes this harder to avoid. Since 1 February 2024, Gmail requires bulk senders above 5,000 messages a day to have SPF, DKIM, and DMARC on the sending domain.[4] Any product that sends mail as its customer now writes DNS on the customer's domain, whether or not it wanted to be in the DNS business.

The standards, and where they stop

Domain Connect is the direct attack on this problem. It is MIT-licensed, originated at GoDaddy in 2016, and is now on the IETF Standards Track as draft-ietf-dconn-domainconnect-03 in the dconn working group, co-authored by Kowalik (DENIC), A. Blinn, Jody Kolker (GoDaddy), and Sami Kerola (Cloudflare), with a milestone targeting IESG submission in December 2026.[5] A service provider publishes a template; the DNS provider hosts the consent UI and applies the records. There are two rails: a signed browser redirect (sync) and an OAuth-based API flow (async).

Coverage is the limit, and the public numbers disagree. domainconnect.org's own live-implementation list names nine DNS providers: IONOS, Cloudflare, Domain Chief, Glauca Digital, GoDaddy, NameSilo, Plesk, Vercel, and WordPress.com.[6] The APNIC post claims roughly twenty implementations covering 35% of the .com zone as of May 2024.[1] We measured the official template repository on 2 August 2026 and counted 998 root-level templates from 596 distinct provider IDs, well above the "over 300 templates from more than 120 providers" quoted in that post.[7] Templates published is not the same as providers honoring them.

The deeper limit is structural. In Domain Connect the DNS provider is the only party that writes records, runs discovery, and issues or revokes tokens. A service provider can request. A DNS provider can decline every apply and revoke every token, unilaterally.[8] That asymmetry has already been litigated: Entri sued GoDaddy in the Eastern District of Virginia (1:24-cv-00569), Judge Trenga denied the motion to dismiss on 11 October 2024 on a negative-tying theory under Sherman Act §1, the parties settled with prejudice on 25 February 2025, and by 18 June 2025 they had announced a multi-year agreement restoring access.[9][10][11] Eight months separated the motion-to-dismiss ruling from the partnership announcement; the docket shows the complaint filed 8 April 2024, fourteen months before it. Access came back through a private bilateral agreement between the two companies.

Provider OAuth APIs are the other native rail. Where one exists, the customer authorizes once and the platform writes the record with no value copied by hand. Few providers offer one. Building adapters for 38 DNS providers, we found six that expose a provider-hosted OAuth flow usable for third-party DNS writes: Cloudflare, DigitalOcean, DNSimple, Netlify, Vercel, and WordPress.com. GoDaddy is not among them; its one-click path is Domain Connect async, not OAuth. Most of the rest need a scoped API token the customer creates by hand, which is a DNS panel visit with extra steps. Four are exceptions: GoDaddy, IONOS and NameSilo we route to Domain Connect, and Namecheap to manual records, because its API replaces the entire zone in a single call.

ACME (RFC 8555, March 2019) issues certificates; it does not configure the records that route traffic.[12] Its HTTP-01 and TLS-ALPN-01 challenges assume the customer's DNS already points at you. Its DNS-01 challenge drops that assumption and instead requires a TXT record on the customer's domain, which is the DNS-writing problem again under a different name. Let's Encrypt's rate limits

shape the design of anything issuing at volume: 300 new orders per account per 3 hours, 5 certificates per identical identifier set per 7 days, 5 authorization failures per account per hostname per hour, 50 certificates per registered domain per 7 days, 100 identifiers per certificate.[13] For a custom-domain platform the 300-orders-per-3-hours account limit binds first, because every new customer domain is a separate registered domain but shares one ACME account. The authorization-failure limit is scoped per account per hostname, so it does not aggregate across the fleet; it turns one misconfigured customer domain into a one-hour retry lockout on that domain alone.

CAA (RFC 8659) is the quiet one. CAA checking is mandatory under the CA/Browser Forum Baseline Requirements, so a customer's pre-existing CAA record can block issuance for a hostname that is otherwise configured correctly. The DNS provider does not need to support the CAA record type; it needs to return NOERROR for unknown query types.[14] The failure surfaces as a certificate that never issues, with no obvious cause in the product's own logs.

Where the vendors actually operate

Approach	Who writes the DNS record	Solves	Leaves open
Support doc	The customer, in their panel	Nothing; documents the work	Full support load, apex ambiguity
In-app record table	The customer, from generated values	Transcription errors	Panel navigation, provider variance
Domain Connect	The DNS provider	One-click on supported providers	Coverage; provider can revoke
Provider OAuth	The platform, with a user-scoped token	Clean writes where offered	Six of 38 providers we integrated
Edge/TLS service	The customer, unaided, pointing one record at your edge	TLS, routing, cert lifecycle	The DNS entry itself
Config aggregator	Aggregator, via DC or OAuth	The onboarding UX	Price floor, standards dependency

Table 1. Custom-domain approaches classified by which party writes the DNS record, since that determines what each one can and cannot fix.

The pricing gap follows the mechanism split. Cloudflare for SaaS includes 100 custom hostnames on Free, Pro, and Business, then charges $0.10 per hostname per month up to 50,000, with custom certificates, mTLS, and wildcard custom hostnames reserved for Enterprise.[15] Approximated charges about $0.20 per domain per month with a $20 minimum and 400 GB of bandwidth included, and describes its setup as one A record pointed at your cluster's dedicated IP.[16] SaaS Custom Domains prices at $0.29 per domain with a 100-domain minimum and automatic discounts up to 50% as volume scales.[17] All three terminate TLS for arbitrary hostnames and reverse-proxy them to an origin. None of them help the customer find the DNS panel.

Entri prices on plans instead: Startup at $249/month for 600 domains a year, Growth at $749/month, with everything above Startup demo-gated.[18] The premium sits on the configuration UX. The entrijs npm package recorded 236,511 downloads in the week ending 1 August 2026.[19] IONOS took a minority stake of under 20% in August 2025, booked in its FY2025 consolidated statements as €5,028k of additions at FVOCI, Level 3, which is the only hard financial number in Entri's public record.[20][21] Entri is also the standard's largest template publisher, with 77 templates in the Domain-Connect repository against GoDaddy's 27 across its godaddy and secureserver provider IDs.[7] Its own public provider counts differ by surface: the Connect product page markets 50+ DNS providers, while the developer provider list enumerates 64.[22] Domainee, a newer entrant, competes directly on the price axis with 50 domains free and $0.20 per domain after, while conceding in its own comparison post that Entri's widget is one of the best onboarding components in the market.[23]

Two jobs are being priced here. Terminating TLS for an arbitrary hostname sells at ten to thirty cents per hostname per month. Getting a non-technical customer's DNS record written correctly on the first attempt, in a panel neither party controls, has no settled price at all: the incumbent charges $249 a month before the first domain, and a challenger selling the same widget charges nothing under 50 domains and $0.20 after. The spread between those two floors is a claim about provider coverage and the bilateral agreements behind it, not a measurement of what the work costs.

Sources

[1] P. Kowalik, "Domain Connect: a missing piece in the DNS toolbox," APNIC Blog, 24 Dec 2025. https://blog.apnic.net/2025/12/24/domain-connect-a-missing-piece-in-the-dns-toolbox/ [2] Entri, "Fourthwall case study." https://www.entri.com/case-studies/fourthwall-case-study [3] Entri, "Crisp case study." https://www.entri.com/case-studies/crisp-case-study [4] Google, "Email sender guidelines." https://support.google.com/mail/answer/81126 [5] IETF, draft-ietf-dconn-domainconnect-03, rev. 3 Jul 2026. https://datatracker.ietf.org/doc/draft-ietf-dconn-domainconnect/ [6] Domain Connect, "DNS providers." https://www.domainconnect.org/dns-providers/ [7] Domain-Connect/Templates, counted via the GitHub git-trees API (untruncated) on 2 Aug 2026. https://github.com/Domain-Connect/Templates [8] Domain Connect specification draft. https://github.com/Domain-Connect/spec [9] Entri LLC v. GoDaddy.com LLC, No. 1:24-cv-00569 (E.D. Va.). https://www.courtlistener.com/docket/68429734/entri-llc-v-godaddycom-llc/ [10] Domain Name Wire, "Judge denies GoDaddy's motion to dismiss in antitrust lawsuit," 11 Oct 2024. [11] Domain Name Wire, "Entri regains DNS update access at GoDaddy under new agreement," 18 Jun 2025. [12] RFC 8555, "Automatic Certificate Management Environment (ACME)," Mar 2019. https://www.rfc-editor.org/rfc/rfc8555 [13] Let's Encrypt, "Rate limits," updated 12 Jun 2025. https://letsencrypt.org/docs/rate-limits/ [14] RFC 8659 and Let's Encrypt, "Certificate Authority Authorization (CAA)." https://letsencrypt.org/docs/caa/ [15] Cloudflare for SaaS plans. https://developers.cloudflare.com/cloudflare-for-platforms/cloudflare-for-saas/plans/ [16] Approximated. https://approximated.app/ [17] SaaS Custom Domains. https://saascustomdomains.com/ [18] Entri pricing. https://www.entri.com/pricing [19] npm registry downloads API, `entrijs`, week of 26 Jul to 1 Aug 2026. [20] IONOS Group, "IONOS secures strategic minority stake in Entri," 18 Aug 2025. [21] IONOS Consolidated Financial Statements 2025. [22]

https://www.entri.com/products/connect, https://developers.entri.com/provider-list (both retrieved 2 Aug 2026; the provider list renders two tables carrying the same 64 providers). [23] Domainee, "Entri alternative 2026." https://domainee.dev/blog/entri-alternative-2026

A reference architecture for programmatic domain connection

A domain connection is three operations wearing one button: a write to a zone the platform does not own, a wait for that write to become globally visible, and a proof that the result is what was asked for. The common implementation collapses all three into a static instructions page that lists the records and trusts the user to copy them. The architecture below separates them, and treats the failure of any one as a state rather than an error page.

Two planes, one system of record

Split the system before writing any DNS code. A control plane holds tenant state, computes records, writes them once, and watches propagation. A data plane terminates TLS and reverse-proxies live custom-domain traffic. In the implementation described here they are separate services, and only one of them is stateful: the control plane owns the single Postgres, the edge holds no persistent state beyond its certificate cache, and the API gateway is never in the proxied request path. At request time the edge calls back to exactly one control-plane endpoint, `POST /internal/ask`.

The reason is blast radius. A bad deploy of a REST API should degrade onboarding, not drop every customer's production traffic.

Figure 1. The connection flow from domain entry to post-launch monitoring. Every rail converges on the same state machine and the same propagation check, so the manual path is verified identically to the automated ones.

Detection runs before the interface offers a path

The cascade has two probes and a floor. Domain Connect discovery goes first, per the spec: a TXT lookup of `_domainconnect.<domain>` yields a host, then `GET https://<host>/v2/{domain}/settings` returns `providerId`, `urlSyncUX`, `urlAsyncUX` and `urlAPI`.[9] Nameserver-pattern matching against each registered adapter runs second. A domain matching neither classifies as manual.

Discovery is best-effort on purpose. Any failure returns `Supported: false` rather than raising, because the caller's next move is identical either way: drop a rail.

Detection answers who runs the zone. A separate capability layer answers what that provider will accept: apex and subdomain support, wildcards, CNAME flattening at the apex, SPF-override tolerance, CAA support, TXT values over 255 bytes, and a conflict-tolerance threshold that predicts when an automated write silently degrades to manual. Route 53 is the instructive case. It has no ALIAS record *type*, only an `<AliasTarget>` XML shape that cannot point at an arbitrary external hostname, so its flattening flag is false. Reporting that at check time is better than promising an apex and refusing it at apply time.

The server computes the records

The browser never derives a record. The control plane emits the rrset and the widget renders it, which keeps one authority for what "correct" means across every rail.

Apex connections use a provider-agnostic `APEXCNAME` type, rewritten immediately before the write into each provider's native spelling: flattened CNAME, ALIAS, or ANAME. Twelve providers currently have that mapping. The persisted and verified record stays `APEXCNAME`, so verification does not have to know which dialect was used. Cloudflare records are forced to `proxied: false`, because the platform's own edge already terminates TLS and a second proxy in front of it breaks routing.

Rails, with manual as a real path

Three write rails sit above a manual floor. Domain Connect sync is a signed browser redirect where the provider performs the write. Domain Connect async exchanges an OAuth code for a grant and applies the template server-side. When a grant-encryption key is configured the grant is stored encrypted, so the template can be re-applied or reverted later without re-consent; without a key the apply is use-once. The third rail is the platform writing records directly with a scoped token or provider OAuth credential. Every adapter implements the same libdns-shaped contract: `GetRecords`, `SetRecords` as an idempotent create-or-update to an exact rrset, `DeleteRecords`.

The census records the best rail each provider is routed to, which is not the same as the rail that executes in a given deployment. Across 63 providers in the parity target, the census routes 38 to manual, 16 to a scoped API token, 6 to native provider OAuth, and 3 to Domain Connect. Six adapters expose provider-hosted OAuth: Cloudflare, DigitalOcean, DNSimple, Netlify, Vercel and WordPress.com.

The shipped configuration is narrower than the routing table, and the gap is worth stating precisely. Of the three Domain Connect providers, only NameSilo can complete a redirect out of the box: the sync rail is disabled by default and re-enabled only for the providers measured to serve the template catalog, which excludes GoDaddy and IONOS because both hand-curate their template sets and serve none of ours, and the async rail ships with no partner credential. The six OAuth providers each need a client id and secret registered with the provider before their rail runs at all. The number to

design around is therefore not 63 routed but at most 7 one-click, and only after per-provider app registration — an owner action, not a code path. Everything else lands on an API token the tenant pastes in, or on manual.

So manual is not a dead end screen. It receives the same computed records, the same guarded state machine, and the same propagation verification. It differs in who performs the write and in its timeout budget: 72 hours pending versus 24 hours propagating. The widget carries a distinct terminal status for manual completion, because a manual success is a success.

Rails degrade rather than fail. A provider with no registered OAuth client reports OAuth unavailable and the flow falls back. The sync-redirect rail is restricted to providers with a published, verified template, since an unpublished template strands the user inside someone else's control panel. Where a rail does run, it runs under fixed invariants: HMAC-signed single-use state bound to the connection, provider, PKCE challenge and return origin; mandatory S256 with no plain fallback; the provider access token used for exactly one `SetRecords` call and never persisted, logged, or returned to a browser; and on the async rail, the provider API base pinned into the signed state at start so no attacker-influenced discovery result can redirect a token exchange.

Propagation is polled, and the state machine is guarded

Statuses are `pending`, `propagating`, `live`, `failed`, with transitions validated by a single owner. The `failed` to `live` edge is deliberately absent, so a retry cannot skip re-verification.

A background poller advances applied connections by reading public DNS on a one-minute tick with geometric backoff to fifteen minutes. It claims batches of up to 100 under a two-minute lease so a horizontally scaled control plane does not double-poll, and runs eight concurrent workers.

One design note from building this: the mock DNS adapter is excluded from the default registry, because it accepts any credential without validation. A caller could persist a written record set with no actual DNS access, and the poller would then promote it to live. Test doubles that reach production registries manufacture false positives in exactly the component whose job is to prevent them.

Certificates issue at the handshake, fail-closed

The edge orders certificates on demand during the TLS handshake using ACME,[10] with TLS-ALPN-01 answered inline on the same `:443` listener and HTTP-01 on `:80`. The host policy defers to a gate that calls the control plane's `/internal/ask`. Unknown host, malformed response, timeout, control plane unreachable: every one resolves to deny.

Fail-closed here does two jobs: it keeps unapproved hosts off the edge, and it keeps unapproved hosts out of the ACME order budget. Let's Encrypt allows roughly 300 new orders per account per three hours and 5 authorization failures per identifier per hour.[11] An open host policy converts any scanner sending random SNI at your edge IP into ACME orders against those budgets. CAA is the

other silent blocker: a pre-existing CAA record that does not authorize the issuing CA blocks issuance at the CA,[12] while a CAA set that lists letsencrypt.org does not, so the pre-flight runs a live CAA lookup and warns only when the existing set omits the issuer.

A positive answer returns everything the proxy needs in one round trip: the allow decision, the upstream origin, a rotatable per-app shared secret injected as an auth header so the origin can verify traffic came from the edge, and the owning tenant id. With a shared KV configured, a fleet of edges uses one certificate cache and a per-host issuance lock, so a newly seen host normally produces one ACME order. The lock is advisory and fail-open: an edge that does not acquire it waits 20 seconds for the winner's cert to land in the shared cache and then issues itself. With no shared KV each instance keeps its own on-disk cache and there is no cross-edge coordination at all.

Drift after go-live

Go-live is not the end of the record's life. Customers migrate registrars, hand DNS to an agency, or clean up records they no longer recognize. A monitor engine computes per-record verdicts against the stored baseline (no change, missing, changed, with restored emitted by the state machine) and drives re-check sweeps through a pluggable resolver seam; the wired deployment reads through the system recursive resolver.

When drift alerting is enabled, findings leave the system as signed webhooks; the sweep otherwise runs in shadow mode, persisting per-record state without delivering. The event set covers the connection lifecycle and drift specifically, including `domain.record_missing` and `domain.record_restored`. Deliveries are HMAC-SHA256 signed over `{timestamp}.{raw_body}`, hex-encoded with a `sha256=` prefix and the timestamp in a companion header, and they queue through a durable outbox rather than a best-effort in-process send. A dropped drift notification is indistinguishable from no drift, which is the failure mode the outbox exists to remove.

Domain connection for AI agents

An agent that provisions a customer's domain hits constraints a human user never notices.

It cannot click. In our provider census of 63 DNS providers, 38 resolve to the manual rail: the record has to be typed into a control panel that only a logged-in human can reach. A browser-using agent might get through some of those panels. It will also get through the delete button. That is not a class of failure worth designing around.

It must not hold the credential. The obvious shortcut is to ask the user for a DNS API token and let the agent call the provider directly. That token is usually zone-wide or account-wide, it lives in a context window, and nothing about it expires when the task ends. A token pasted into a chat is a token you now have to rotate.

It must not spend money on its own. Domain purchase is a real charge against a real card. An agent that can search availability and an agent that can buy are different security objects, and the second one needs a human in the loop by construction.

The tool surface

The MCP server implements protocol revision 2025-06-18 and exposes 12 tools: `search_domain_availability`, `generate_domain_suggestions`, `create_domain_order`, `connect_domain`, `check_connection_status`, `check_order_status`, `reapply_connection`, `disconnect_domain`, `discover_provider`, `forward_domain`, `add_email`, and `list_connections`.

Every tool carries a human title, `readOnlyHint`, and `openWorldHint`; the five state-changing tools add `destructiveHint` and `idempotentHint`, and `disconnect_domain` carries `destructiveHint` without an idempotency hint. The six read-only tools omit the two write hints entirely, because the annotation fields are optional pointers and a nil pointer is dropped from the JSON. The host reads them to decide what to auto-run and what to gate; `disconnect_domain` is the only tool whose `destructiveHint` is true.

Annotations are advisory. Enforcement is separate, and it uses a closed three-scope model:

Scope	Covers	Why it is separate
`domains:read`	availability search, suggestions, provider discovery, status checks, listing connections	observation only, safe to auto-run
`domains:connect`	connect, reapply, forward, add email, disconnect	writes DNS and can remove a live domain
`domains:purchase`	order creation	spends money

The scope map is stamped at construction time by an invariant that rejects any tool lacking exactly one valid scope. A new tool cannot ship unscoped: `NewServer` panics at construction when a registered tool has no entry in the scope map, and `TestToolScopesExactlyCoverRegisteredTools` fails CI before a binary ever runs. Enforcement is live in `tools/call` for delegated agent tokens: a call whose verified ES256 claims lack the tool's required scope returns an `insufficient_scope` error naming the scope it needed. The scope model binds agent tokens only. An integrator credential, meaning an `sk_` API key or a client-credentials JWT, carries no scopes and reaches all twelve tools including `create-domain-order`, so an integrator that hands its own API key to an agent gets none of this.

The authorization flow

The delegated-agent surface described here is off by default. The control plane registers the routes but reports not-configured until agent auth is switched on with a durable ES256 signing key, and the MCP server's agent-token path stays dark until it is given a JWKS URL. The design goal is that the agent never sees a DNS credential and never obtains authority without a human granting it in a browser the agent does not control.

The control plane implements RFC 7591 dynamic client registration at `POST /oauth/agent/register`, a browser authorize endpoint that hands off to a human console consent screen, `POST /oauth/agent/token`, and RFC 7009 revocation.[1][2] Agent clients register as public clients with no secret. PKCE with `code_challenge_method=S256` is mandatory; a request without it is rejected, and the token exchange compares `BASE64URL(SHA256(verifier))` against the stored challenge in constant time.

Three unauthenticated discovery documents describe the server: RFC 9728 protected resource metadata, RFC 8414 authorization server metadata, and a capability manifest at `/.well-known/agent/mcp.json`.[3][4] Today they carry the integrator client-credentials path only; an agent taking the delegated path is pointed at the control plane's own `/.well-known/oauth-authorization-server`, which is where the register, authorize, token and revoke endpoints are advertised.

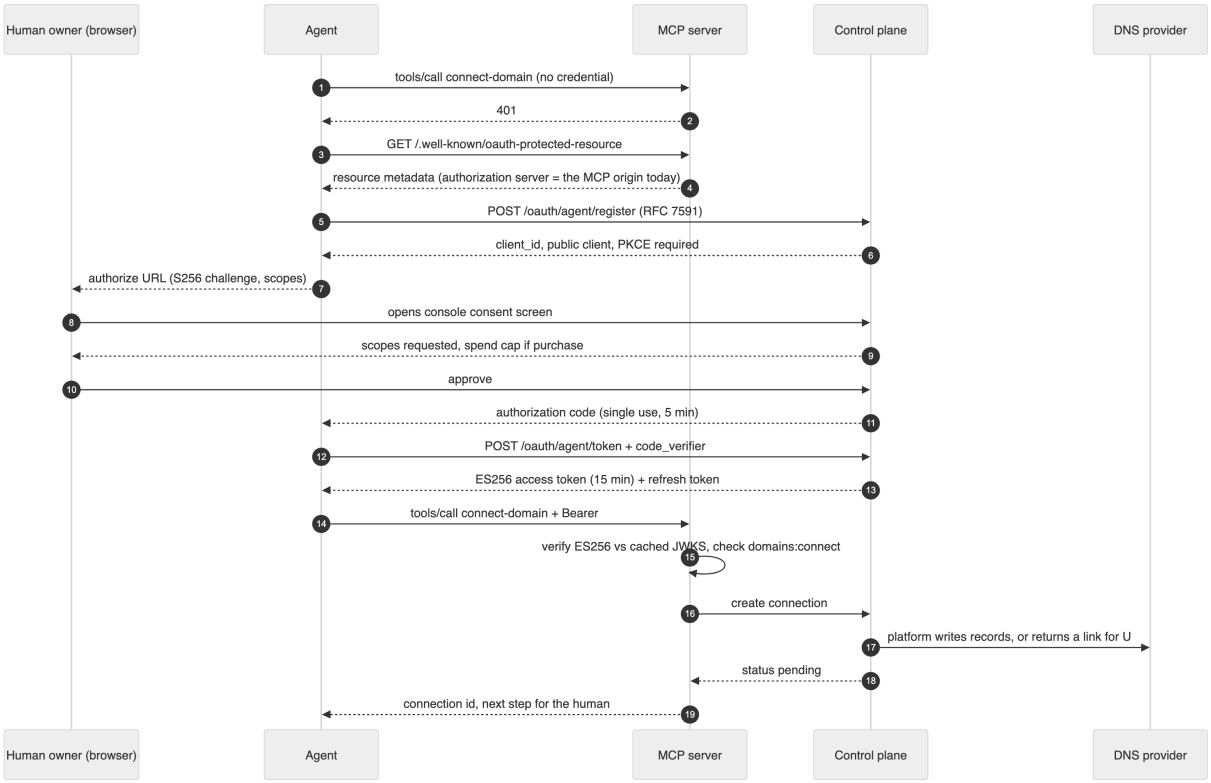

Figure: the agent obtains a scoped, tenant-bound token only after a human approves it in a browser session the agent cannot reach, and the agent's token authorizes tool calls rather than DNS access.

Access tokens are ES256-signed, tenant-scoped, and valid for 15 minutes. The MCP server verifies them locally against the control plane's published JWKS with no per-request introspection, which keeps the hot path cheap. The 15-minute ceiling is what makes that safe: revoking a grant or a refresh token takes effect within one token lifetime. Authorization requests live 10 minutes and codes live 5.

The agent's token authorizes tool calls. It is not a DNS credential and cannot be replayed against Cloudflare or GoDaddy. When the connection needs a human, the tool returns a link the person opens themselves: a Domain Connect authorization URL at their own DNS provider where one exists, otherwise a console page carrying the same records.

Money is gated twice

Consent granting `domains:purchase` is refused unless the human supplies an explicit positive monthly spend cap. At purchase time the cap is enforced by an atomic check-and-record that closes the double-spend race. Over cap, the spend is not recorded, the owner is alerted, and the purchase is denied. A non-positive cap denies outright.

A second gate sits in front of that. Paid orders placed through MCP call back to the integrator over Basic Auth before any order is placed. HTTP 200 authorizes. 401 blocks. Anything else, including timeouts and network errors, denies. With no authorization URL configured, every purchase is denied. Registrar purchase is itself behind a deployment flag that defaults off, and so is the agent auth surface;

in production the control plane refuses to enable agent auth without a durable ES256 signing key, because an ephemeral key would invalidate every live access token and the published JWKS on any restart or second instance behind the load balancer.

Two gates for one action is deliberate. The spend cap protects the human owner from their own agent. The callback protects the integrator from a compromised grant.

---

[1] RFC 7591, OAuth 2.0 Dynamic Client Registration. https://www.rfc-editor.org/rfc/rfc7591 [2] RFC 7009, OAuth 2.0 Token Revocation. https://www.rfc-editor.org/rfc/rfc7009 [3] RFC 9728, OAuth 2.0 Protected Resource Metadata. https://www.rfc-editor.org/rfc/rfc9728 [4] RFC 8414, OAuth 2.0 Authorization Server Metadata. https://www.rfc-editor.org/rfc/rfc8414

Security, trust and failure modes

A system that writes DNS records on a domain it does not own is holding borrowed authority. Borrowed authority has a shape: how wide the grant is, how long it survives, who can take it back, and what still runs after it is taken back. Those four answers are the security model.

What is actually delegated

Three write rails and four grant shapes, and they are not interchangeable.

On the signed-redirect rail, which ships behind a deployment flag and a per-provider allowlist because an unpublished template dead-ends the user at their provider's error page, the platform holds nothing at all. The control plane signs an apply URL with RS256 using a private key that never leaves it, the DNS provider verifies that signature against the matching public key the platform publishes in DNS by selector (`_dcpubkeyv1`), the user lands on their own DNS provider, and the provider authenticates them and writes. Authority ends at "request".

On the Domain Connect async rail an OAuth code is exchanged for a durable grant, encrypted at rest when a grant-encryption key is configured. That grant exists so a template can be re-applied or reverted server-side without dragging the end user back through consent. With no encryption key configured it degrades to use-once: apply, then discard the credential.

On the provider-OAuth rail the end user authorizes their own DNS provider inside the widget. The resulting access token is used for exactly one `SetRecords` call. It is never persisted, never logged, and never sent to a browser. The same rail also accepts a caller-supplied provider API token passed on the apply call itself, which is the mode the provider census records for 16 of 63 providers.

The widest grant is that caller-supplied provider API token on the BYO-token path: it is whatever the integrator's customer minted at their provider, typically zone-wide or account-wide, and the platform cannot narrow it. It is passed per apply call and not persisted. The Domain Connect grant is the only one the platform stores, and its width is bounded by a template the provider itself chose to honor.

Pinning and the parts of discovery an attacker can reach

Domain Connect discovery starts with a TXT lookup on `_domainconnect.<domain>`, which yields a host, which returns a settings document containing `urlAPI`. Everything in that chain is influenced by whoever controls the domain being connected. So on the async rail the provider API base is captured at start time, signed into the OAuth state, and pinned onto the stored grant. Token exchange, apply and revert go only to that pinned base. A `urlAPI` from fresh discovery is never trusted after the flow begins.

The provider-OAuth rail carries its own invariants: HMAC-signed, single-use, short-TTL state bound to the connection, the provider, the PKCE challenge and the return origin; PKCE S256 required with no plain fallback; callback results posted only to an origin vetted at start time against a configured allowlist. An empty allowlist denies everything rather than defaulting to permissive.

Revocation, and who holds the lever

Revocation from our side depends on which rail wrote the records, and one rail cannot revoke at all. Domain Connect grants support server-side revert when a grant-encryption key is configured; without one the async apply is use-once and there is no stored grant to revert with. Because the provider-OAuth callback never persists the access token, an OAuth-connected domain is never marked managed, so `:reapply` returns 409 for it and there is no server-side revert of those records. The agent authorization surface, which is off by default and enabled per deployment, is designed around RFC 7009 revocation and 15-minute ES256 access tokens, so a revoked grant stops working within one token lifetime even though tokens are verified locally against a JWKS and never introspected per request. The embed uses a 15-minute widget JWT minted server-side by the integrator, so no API key reaches a browser. Integrator API keys are stored hashed with only a short prefix indexed.

Revocation from the provider's side is the uncomfortable half. Under the Domain Connect specification the DNS provider is the only party that writes records, runs discovery, and issues or revokes tokens. A service provider can only ask.[1] Any DNS provider can sever any aggregator unilaterally by declining applies or revoking tokens, and that is not theoretical. GoDaddy revised its terms of use in a way that cut off aggregator DNS access. In Entri LLC v. GoDaddy.com LLC, No. 1:24-cv-00569 (E.D. Va.), Judge Anthony J. Trenga denied GoDaddy's motion to dismiss on 2024-10-10, writing that the alleged negative tie "cuts off all aggregator services ... from competing";[2] the parties settled on 2025-02-25;[3] a multi-year access agreement was announced on 2025-06-18.[4] Eight months separated the ruling from the access agreement, with the aggregator's automation off in between.

The design consequence is blunt. Every automated rail has to degrade to something nobody else can switch off. That floor is manual copy-paste records, and an unrecognized domain classifies to it by default.

Tenant isolation

Credentials are keyed per tenant and every adapter call carries one. The edge's authorization answer returns the owning tenant id alongside the origin URL and a rotatable per-application shared secret, injected as an auth header so the origin can verify that traffic actually came from the edge rather than from the open internet. Where the agent surface is enabled, agent tokens are tenant-scoped. Agent capability is a closed three-scope set (`domains:read` , `domains:connect` , `domains:purchase`) with a construction-time invariant that rejects any tool lacking exactly one valid scope, so a new tool cannot ship unscoped. Purchase is isolated from the other two because it spends money: consent cannot grant `domains:purchase` without an explicit positive spend cap, and the cap is checked and recorded atomically on the purchase path, which closes the double-spend race. Over cap, the spend is not recorded, the human owner is alerted, and the purchase is denied. Both the agent consent flow and registrar purchasing are off by default and must be enabled per deployment.

Fail-closed defaults

Certificate issuance is gated before ACME, not after. The TLS host policy defers to a gate that asks the control plane whether a host is approved. Control plane unreachable, request timeout, malformed response, unknown host: every one of those resolves to deny. A host the control plane has not approved never triggers an ACME order at all.

One decision worth naming because it went the other way first. The connect engine has a Mock adapter that accepts any credential without validating it. It is deliberately excluded from the default registry, because with it registered a caller could persist a "written" record set while holding no DNS access, and the propagation poller would later find matching public DNS and promote that connection to live. The adapter is fine. Shipping it in the default registry would have been a silent authorization bypass.

Honest failure modes

Condition	What the system does	What the integrator sees
Provider unrecognized, no adapter	Classifies as manual, renders the exact rrset	Setup type `manual`; connection stays pending up to 72 hours
Provider API rejects the write	Connection does not advance	Structured error with `{code, title, details}`
Apex requested on Route 53	`cname_flattening` reported false at check time	Capability check fails before apply, not during it
CAA record excludes the issuing CA	ACME order fails; no bypass exists	No certificate for that host; the edge logs the failure and, unless the Secure surface is switched on, re-orders on the next cache-miss handshake
Host not approved	Ask gate denies; no ACME order placed	No certificate, no proxying
Record removed after going live	Monitor verdict `record_missing` against the stored baseline; the hourly sweep persists the transition	`domain.record_missing` webhook, HMAC-SHA256 signed, from a durable outbox, on deployments where drift alerting is switched on
Records never propagate	`propagating` expires at 24 hours	Status `failed`

Table 1. Seven conditions the system cannot fix, and the observable behaviour for each.

Two of those deserve more than a row. CAA is the one that surprises teams. Checking it is mandatory for every publicly trusted CA under the CA/Browser Forum baseline requirements, so a customer's pre-existing CAA record that omits your CA blocks issuance outright.[5] The block is loud on the ACME side and silent everywhere else: the order fails with a CAA error the platform can read, and

nothing in the customer's DNS or browser says why. There is no workaround on the platform side; the honest move is to detect it, report `failed`, and tell the customer which record to change. Let's Encrypt's own guidance is worth passing to customers: a DNS provider does not need explicit CAA support, only a NOERROR response to unknown query types, and the `accounturi` parameter limits issuance to a specific ACME account.[5]

ACME rate limits are the other. Fifty certificates per registered domain per 7 days, 300 new orders per account per 3 hours, and 5 authorization failures per identifier per hour.[6] The fleet-level answer is a cross-edge issuance lock over a shared KV, so one host newly seen by twenty edge instances produces one ACME order. Configuring that KV is optional: with none configured each instance falls back to a per-instance certificate cache and the fleet can place one order per instance. The per-host answer is conditional, and worth stating precisely because it is the kind of thing a datasheet would round up. The edge carries a 15-minute issuance back-off for any host the control plane reports as `failed`, consulted only on a genuine cache miss so a host holding a valid certificate is never touched. But it reads that verdict off the `secure_status` field of the ask answer, and the control plane only populates that field where the Secure surface is enabled — which is off by default. With Secure off the control plane sends no status, the back-off treats that as "permit", and the shipped behaviour is one ACME order per cache-miss handshake. So on a default deployment the per-host answer is the one nobody controls: a misconfigured domain burns its five authorization failures inside an hour and then waits, and no amount of retry logic changes that.

Recovery has one rule. The connection state machine has no transition from `failed` to `live`. A retry cannot skip re-verification.

---

[1] Domain Connect specification, Domain-Connect/spec. https://github.com/Domain-Connect/spec [2] Memorandum opinion and order, Entri LLC v. GoDaddy.com LLC, No. 1:24-cv-00569 (E.D. Va. Oct. 10, 2024) (Trenga, J.). Docket: https://www.courtlistener.com/docket/68429734/entri-llc-v-godaddycom-llc/ — see also "Antitrust: GoDaddy.com sued for negative tying restriction", Virginia Lawyers Weekly, 2024-10-21, https://valawyersweekly.com/2024/10/21/antitrust-godaddy-com-sued-for-negative-tying-restriction/. Quoted language as reported in "Judge denies GoDaddy's motion to dismiss in antitrust lawsuit", Domain Name Wire, 2024-10-11. https://domainnamewire.com/2024/10/11/judge-denies-godaddys-motion-to-dismiss-in-antitrust-lawsuit/ [3] "GoDaddy and Entri settle antitrust lawsuit", Domain Name Wire, 2025-02-25. https://domainnamewire.com/2025/02/25/godaddy-and-entri-settle-antitrust-lawsuit/ [4] "Entri regains DNS update access at GoDaddy under new agreement", Domain Name Wire, 2025-06-18. https://domainnamewire.com/2025/06/18/entri-regains-dns-update-access-at-godaddy-under-new-agreement/ [5] RFC 8659 (CAA), https://www.rfc-editor.org/info/rfc8659/ and Let's Encrypt CAA documentation, https://letsencrypt.org/docs/caa/ [6] Let's Encrypt rate limits, page updated 2025-06-12. https://letsencrypt.org/docs/rate-limits/

Evaluating a vendor

Run this checklist against any vendor, including the version of the vendor that is your own team with a quarter of runway. The questions are ordered by how expensive they are to discover late.

1. Coverage: of what, exactly, and in which mode

The DNS-configuration aggregators all lead with a provider count. The edge-only services publish none, because they never touch the provider. The number is close to meaningless on its own, because "supported" spans four different mechanics: the provider writes the record itself after a Domain Connect redirect or grant, the end user authorizes the platform through the provider's own OAuth, the platform writes with a scoped API token the tenant supplies, or the user copies and pastes.

Ask for the count broken out by mode. Ask for it as a file, not a slide.

Two public examples show why. One vendor currently publishes three different coverage numbers simultaneously: "50+ DNS providers" and "75% of the market" on its Connect product page, "60+ DNS providers with direct API login" in its developer overview, and 64 named providers on its developer provider list (product page, developer overview, provider list, all fetched 2026-08-02). Separately, the Domain Connect standard's own site lists nine providers under "Implementation is Live" (IONOS, Cloudflare, Domain Chief, Glauca Digital, GoDaddy, NameSilo, Plesk, Vercel, WordPress.com), while the APNIC introduction to the standard says roughly twenty providers have implemented it, covering 35% of the .com zone as of May 2024 (domainconnect.org; Kowalik, APNIC, 2025-12-24). The two counts are not reconcilable from public sources. domainconnect.org lists only implementations it has been told about and carries no last-updated date. The APNIC post gives a number with no list behind it.

Then ask what happens on a domain nobody recognizes. If the answer is an error screen rather than correct copy-paste instructions, the long tail becomes your problem.

2. Apex

Root domains break differently from subdomains, and they break at apply time rather than at check time if nobody planned for it. Providers realize an apex alias three ways: flattened CNAME, ALIAS, or ANAME. Some cannot do it at all.

We modeled apex as a per-adapter implementation detail and were wrong. It only became tractable once we treated the apex target as its own record type and rewrote it to each provider's native spelling immediately before the write, keeping the stored and verified record provider-agnostic. Route 53 forced the correction: it has no ALIAS record *type*, only an `<AliasTarget>` XML shape that cannot point at an arbitrary external hostname, so honest capability reporting means saying no at check time instead of failing at apply.

Ask any vendor for the list of providers where apex works, and ask when the user learns it does not.

3. CAA

CAA is the failure mode nobody demos. The CA/Browser Forum Baseline Requirements make CAA checking mandatory, so if your customer's hostname, or any ancestor of it, publishes a CAA RRset with an `issue` or `issuewild` property that does not name your CA, issuance fails and the surface symptom is a certificate that never appears (RFC 8659 §3; Let's Encrypt CAA guidance).

Ask three things. Does the tool read CAA before ordering. Does it explain the conflict in the failure message. Does it support the `accounturi` parameter, which Let's Encrypt recommends so a temporary domain hijack without the ACME account key cannot yield a certificate.

4. TLS automation

Certificate issuance runs on ACME (RFC 8555), and the constraints that bite are the rate limits, not the protocol. Let's Encrypt allows 50 certificates per registered domain per 7 days, 300 new orders per account per 3 hours, 5 certificates per identical identifier set per 7 days, and 5 authorization failures per identifier per account per hour (limits page, updated 2025-06-12). Onboard more than 300 new hostnames in three hours and a build-it-yourself implementation stalls on the new-order limit. Retry a single misconfigured hostname more than five times in an hour and that hostname is locked out for the rest of the hour, because the authorization-failure budget is counted per identifier per account, not per batch.

Ask whether issuance is fail-closed: does a host the control plane has never approved still trigger an ACME order. If it does, one scripted crawl of your edge burns your account limits. Ask whether the certificate cache and the per-host issuance lock are shared across instances, because two edges racing on the same new hostname is how you generate duplicate-certificate failures at exactly the moment you scale out.

5. After the connection goes live

Connection is an event. Custom domains are a subscription to someone else's DNS zone, and that zone changes without telling you. Ask what happens when a customer's record is deleted six weeks later, whether drift is detected from public resolvers or inferred from the vendor's own write log, and whether "verified" means the vendor re-read DNS or simply trusted its API call returned 200.

That distinction matters more than it sounds. Our own connect engine deliberately excludes its mock adapter from the default registry, because an adapter that accepts any credential without validation would let a caller persist a written record set with no DNS access at all, which the propagation poller would then promote to live. A verification pipeline that trusts its own writes will manufacture false positives.

Ask for the webhook event list, the signing scheme, and whether delivery is backed by a durable outbox or fired in-process and forgotten.

6. Agent and API surface

If an agent will ever touch this, the questions get sharper. Is there a machine-readable OpenAPI document served by the running service, or a hand-maintained docs site. If there is an MCP server, are the tools annotated so a host knows which are read-only and which are destructive. Are scopes enforced in the request path, or declared and ignored. If the product can spend money on domain purchases, there must be a spend cap enforced at purchase time with an atomic check-and-record, plus a fail-closed default when no cap is set. A cap checked before an order and recorded after it has a double-spend race.

7. Credentials and data

Where does a customer's DNS provider token live after the flow completes. One-shot tokens used for a single write and never persisted are the safest posture and also the one that quietly removes your ability to re-apply records later. That trade is real, and either answer is defensible, but the vendor should state which one it made. Confirm no API key ever reaches the browser, that the embed is authenticated by a short-lived server-minted token, and that PKCE with S256 is mandatory rather than optional. Ask what leaves your network by default.

Checklist

Question	Answer that should worry you
Coverage, broken out by mode?	A single headline number
Apex support, per provider?	"Yes" with no exceptions list
CAA read before issuance?	"CAA is rare"
Fail-closed cert issuance?	Any host can trigger an order
Drift detection source?	The vendor's own write log
Webhook delivery?	In-process, best effort
Agent scopes?	Declared, not enforced
Provider token lifetime?	Unspecified

Conclusion

The pricing floor in this category is set by edge-only services at $0.10 to $0.29 per hostname per month: $0.10 per additional custom hostname at Cloudflare, 20 cents per custom domain at Approximated, $0.29 per domain at SaaS Custom Domains (Cloudflare for SaaS, Approximated, SaaS Custom Domains). Those services solve TLS and routing. They do not touch the DNS configuration step. The only public quantification of what that step costs is vendor-published: Entri reports

Fourthwall attributed nearly half its support tickets to DNS configuration before automating it (case study). Everything above that floor is priced on bundled products rather than per-hostname volume: Entri's $249 Startup is Connect alone, $749 Growth adds Power, Premium adds Secure (pricing). The checklist above is one question asked eight ways: how much of the DNS problem does this vendor absorb, and how does it behave on the day it cannot.

Disclosure: we build customdomain.ai, and the architecture described earlier in this paper is ours. The reason we published the checklist in vendor-neutral form is that it is the one we run on ourselves. Our census file routes 63 providers, every name on the incumbent's published list but World4You, to a mode: 25 are routed to an automated path (16 BYO-token API, 6 native OAuth, 3 Domain Connect), and the other 38 get provider-specific copy-paste instructions, as does any provider not on the list. Two of those 25 are routing, not working. GoDaddy is one of the three Domain Connect entries, and as of 2026-07-28 it served 0 of our 18 published templates, because it hand-curates its template set and we have not been onboarded; DigitalOcean is the one of the six OAuth entries whose app is still unregistered. So 25 is what the census routes and 23 is what works today. Our own homepage still says "63 DNS providers, auto-configured" without that breakdown, and that is the failure row one is meant to catch. The source is Apache-2.0 licensed and self-hostable, though the repository is not public yet. The OpenAPI 3.1 document is embedded in and served by the running binary at `GET /v1/openapi.json`, with contract tests that fail CI when the served spec and the live handlers disagree. The cloud observability integrations (Sentry, OTLP tracing, PostHog) no-op when their environment variables are unset, so a self-hoster sends no telemetry to us or to any observability vendor. The product still calls Let's Encrypt, public resolvers, and whichever DNS provider the customer uses.

Run row one of the checklist against your own provider table before you run it against anyone's sales deck.

1. Entri, "Fourthwall case study." https://www.entri.com/case-studies/fourthwall-case-study. Vendor-published; treat as an attributed customer claim.
2. Domain Connect, "DNS providers." https://www.domainconnect.org/dns-providers/, retrieved 2026-08-02. Listed under "Implementation is Live": IONOS, Cloudflare, Domain Chief, Glauca Digital, GoDaddy, NameSilo, Plesk, Vercel, WordPress.com.
3. P. Kowalik, "Domain Connect: a missing piece in the DNS toolbox," APNIC Blog, 2025-12-24. https://blog.apnic.net/2025/12/24/domain-connect-a-missing-piece-in-the-dns-toolbox/. The post cites CENTR for its renewal-rate figures but gives no source for the Microsoft 365 numbers.
4. Google, "Email sender guidelines." https://support.google.com/mail/answer/81126. Requirements effective 2024-02-01 for senders above 5,000 messages per day to personal Gmail accounts.
5. Entri, "Crisp case study." https://www.entri.com/case-studies/crisp-case-study. Vendor-published; CEO Baptiste Jamin quoted directly.
6. Entri, "Fourthwall case study." https://www.entri.com/case-studies/fourthwall-case-study. Vendor-published; treat as an attributed customer claim.
7. P. Kowalik, "Domain Connect: a missing piece in the DNS toolbox," APNIC Blog, 2025-12-24. https://blog.apnic.net/2025/12/24/domain-connect-a-missing-piece-in-the-dns-toolbox/. The post cites CENTR for its renewal-rate figures but gives no source for the Microsoft 365 numbers.
8. Let's Encrypt, "Certificate Authority Authorization (CAA)." https://letsencrypt.org/docs/caa/.
9. Domain Connect specification, https://github.com/Domain-Connect/spec and https://www.domainconnect.org/
10. R. Barnes, J. Hoffman-Andrews, D. McCarney, J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, March 2019. https://www.rfc-editor.org/rfc/rfc8555
11. Let's Encrypt, "Rate Limits", page updated 2025-06-12. https://letsencrypt.org/docs/rate-limits/
12. P. Hallam-Baker, R. Stradling, J. Hoffman-Andrews, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 8659, 2019, https://www.rfc-editor.org/info/rfc8659/; Let's Encrypt, "Certificate Authority Authorization", https://letsencrypt.org/docs/caa/