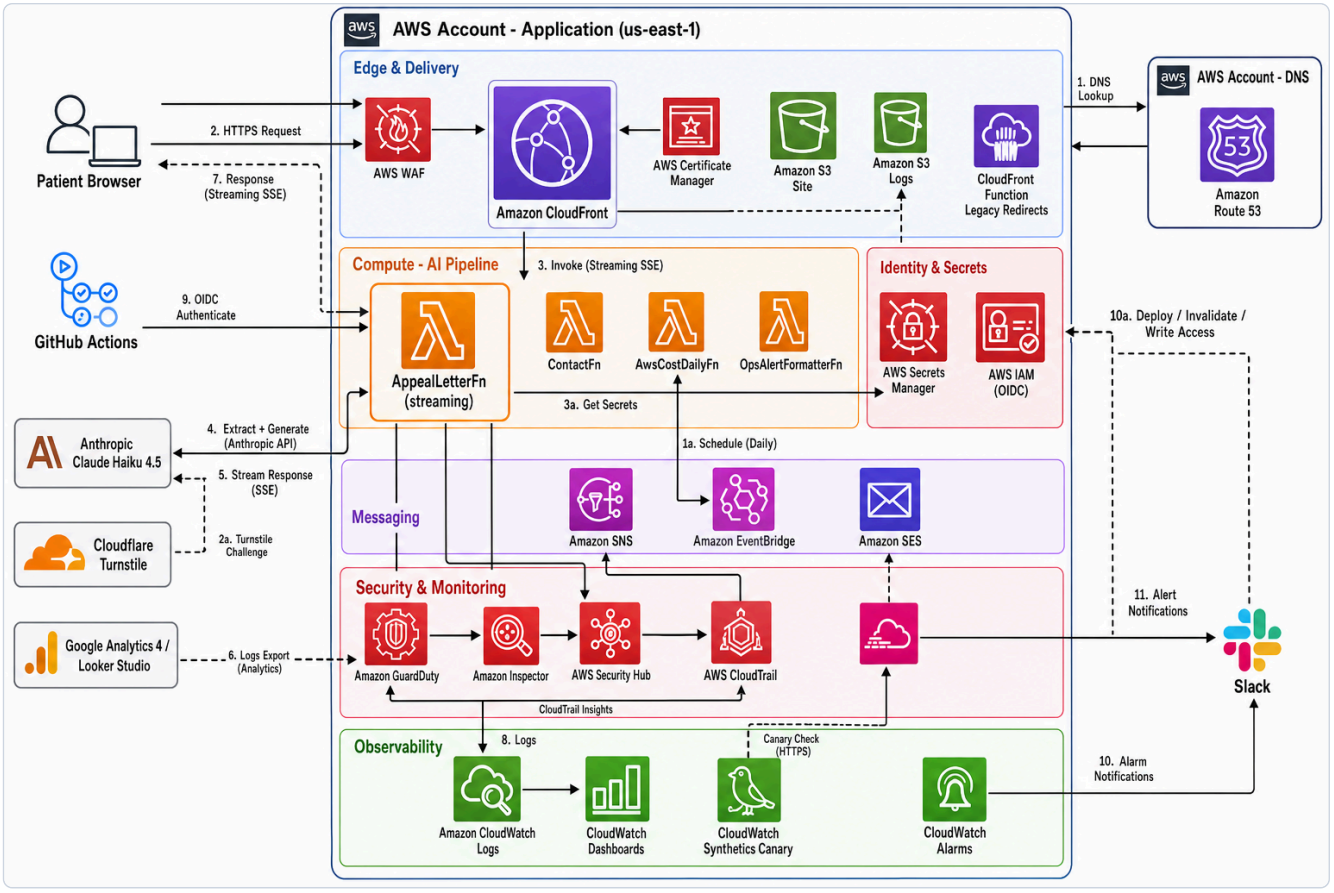
type2.wiki: A Production AI Capability for Patient Advocacy



Architecture at a Glance

1. Executive Summary

People with Type 2 Diabetes are routinely denied coverage for the medications and devices that keep them healthy — GLP-1 and dual-incretin therapies such as Mounjaro and Ozempic, insulin, continuous glucose monitors, and test strips. The appeal process that could overturn those denials is opaque, state-specific, and intimidating. Most patients give up.

type2.wiki set out to change that with a free, no-login tool that turns a denial into a polished, state-correct, evidence-backed insurance appeal letter in minutes. The vision came from the patient community itself. Turning it into a secure, observable, cost-controlled production system was an engineering problem — and that is where **Jacobian Engineering** came in.

Jacobian designed and delivered the complete AI/ML capability end to end:

- A **generative-AI pipeline** built on Anthropic's **Claude Haiku 4.5**, using a two-stage extract-then-generate orchestration grounded in a curated medical-evidence corpus.
- **Real-time token streaming** to the browser over Server-Sent Events, so patients watch their letter being written.
- **Privacy by design** — patient PII never leaves the browser unprotected; the model works on redacted placeholders.

- **Cost engineering** — every letter's token spend is measured, logged, and alarmed, with a hard daily ceiling.
- **Multi-layer abuse prevention** — bot mitigation, rate limiting, geo-fencing, and an instant kill switch.
- A **fully serverless AWS architecture** delivered as infrastructure-as-code, with end-to-end observability, security monitoring, and analytics.

The result is a live production workload that serves a real, underserved patient community — delivered with the engineering discipline (spec-driven development, eval-gated AI quality, 485+ automated tests, and FinOps from day one) that AWS partner-grade work demands.

Headline characteristics of the delivered system

Dimension	What Jacobian delivered
AI model	Claude Haiku 4.5, two-call orchestration (extract + generate)
Latency UX	Sub-second time-to-first-token via SSE streaming
Grounding	10-document curated medical-evidence corpus, deterministically selected
Privacy	Client-side PII tokenization; no PII persisted anywhere
Cost control	Per-letter cost telemetry + composite cost-spike alarm + kill switch
Infrastructure	100% serverless, 5-stack AWS CDK, zero standing servers
Quality	Spec-driven delivery, eval harness, 485+ automated tests
Operations	2 CloudWatch dashboards, synthetic canary, SNS→Slack alerting

2. The Customer & the Challenge

2.1 A community problem

type2.wiki grew out of the **Mounjaro for Diabetes** patient community — tens of thousands of people comparing notes on diagnoses, dosing, and, constantly, insurance denials. The site's contributors are patients and patient-advocates first: they wrote the Mounjaro guide, the state-by-state appeal guidance, and the dosing references that the site is known for.

The recurring pain point was always the same. A patient is prescribed an effective therapy. The insurer denies it — citing step-therapy requirements, "not medically necessary," formulary exclusions, or quantity limits. The patient has the *right* to appeal, but:

- The **process differs in all 51 jurisdictions** (50 states + D.C.), with different deadlines, forms, and Independent Review Organizations.
- Winning an appeal requires citing the **right clinical evidence** in the right language — material most patients have never seen.
- The emotional and administrative load arrives exactly when the patient is least equipped to handle it.

> *"I've been a Type 2 patient since 2021 and on Mounjaro since 2022, and I've watched our community fight the same denial letters over and over. People who needed their medication just walked away because the appeal felt impossible. We knew what a winning appeal looked like — we just couldn't get it into every patient's hands. That's the gap we wanted to close." — Gwynne Sullivan, co-founder, type2.wiki; co-author of the original Mounjaro for Diabetes FAQ*

2.2 The engineering challenge

The community knew *what* a good appeal looked like. The challenge Jacobian was engaged to solve was *how* to deliver one to any patient, on demand, safely:

1. **Clinical accuracy without hallucination.** A letter that invents a study or misstates an effect size is worse than useless. The system had to be *grounded* in real, curated evidence and *checked* before a patient ever sees it.
2. **Privacy for sensitive health data.** Names, member IDs, claim numbers, diagnoses — this is exactly the data patients are most afraid to hand to "an AI." The architecture had to make leakage structurally impossible, not merely unlikely.
3. **Cost predictability for a free tool.** type2.wiki has no revenue and no logins. An unbounded LLM bill — whether from genuine demand or abuse — could end the project overnight.
4. **Trust and availability.** The tool serves people in a vulnerable moment. It had to be observable, alertable, and gracefully degradable.

These four constraints shaped every decision that follows.

---

3. Why Jacobian Engineering

Jacobian Engineering is a systems and cloud engineering practice that builds production-grade software on AWS. type2.wiki engaged Jacobian to take the capability from "a great idea and a pile of community knowledge" to "a secure, monitored, cost-bounded production service."

Jacobian's operating principles on this engagement:

- **Spec-driven delivery.** Every capability shipped as a numbered, reviewed specification (Specs A through J) with an implementation plan, an eval gate, and a handoff document. Nothing shipped on vibes.
- **Eval-first AI.** AI quality is treated as a measurable, regression-tested property — not a one-time prompt-tuning exercise.
- **Cost-aware architecture.** Model selection, token budgets, and spend alarms were designed in from the first AI spec, not bolted on.
- **Security and privacy by construction.** Least-privilege IAM, no-PII-persistence, secrets isolation, and edge protections were part of the baseline, not a later audit.
- **Operate what you build.** Dashboards, synthetic canaries, and on-call alerting shipped as a first-class capability (Spec J), not an afterthought.

The sections below walk through the delivered capability in that frame.

---

4. The AI/ML Pipeline

The heart of the platform is a generative-AI pipeline that converts a short, structured intake plus a pasted denial into a complete, citation-backed appeal letter. Jacobian designed it as a deterministic *scaffold* wrapped around tightly-scoped *model calls* — the model supplies fluent, persuasive language; the system supplies structure, evidence, and guardrails.

4.1 The intake: a four-step wizard

Patients interact with a four-step React wizard (an Astro "island" embedded in an otherwise static site):

1. **Medication & diagnosis** — what was prescribed, for what condition.
2. **The denial** — paste the denial text, or upload the denial letter for OCR extraction; flag step-therapy and urgency.
3. **Patient & plan details** — name, member ID, claim number, state, and insurance type (commercial, Medicare, Medicaid, and other variants drive different filing paths).
4. **Review & generate** — a final review with bot-verification before submission.

The wizard captures everything the letter needs while keeping the model's job narrow and well-specified.

4.2 Two-call orchestration: extract, then generate

Rather than hand the model one sprawling prompt, Jacobian split the work into two purpose-built Claude **Haiku 4.5** calls (`claude-haiku-4-5-20251001`):

Call 1 — Extraction (temperature 0.1, max 1,024 tokens). The raw wizard input and pasted denial text are parsed into a clean, structured JSON context: medication, diagnosis, denial rationale, appeal stage, jurisdiction, and insurance type. Low temperature keeps this deterministic and faithful — this is comprehension, not creativity.

Call 2 — Generation (temperature 0.3, max 4,096 tokens). The structured context, the selected medical evidence, and a deterministic letter scaffold are composed into the generation prompt. The model writes the letter HTML, following a fixed section structure and citing the supplied evidence. Slightly higher temperature gives the prose a human, non-templated voice without letting it wander.

This separation is what makes the output reliable: the model never has to *both* interpret messy input *and* compose a persuasive argument in a single pass, and each call can be evaluated and tuned independently.

4.3 Grounding: a deterministic medical-evidence corpus

The system does **not** ask the model to recall clinical facts from training. Instead, Jacobian curated a versioned corpus of ten evidence documents — the standards-of-care anchor, GLP-1 randomized-controlled-trial citations, step-therapy rebuttal playbooks, Independent Medical Review winning-language patterns, dual-incretin mechanism of action, cardiovascular and renal evidence, off-label evidence, and Medicare/Medicaid appeal procedures.

A **deterministic, rules-based selector** chooses which evidence to inject based on the extracted context (medication, diagnosis, insurance type, denial reason, appeal stage). **No AI is in this selection loop** — it is auditable code. The model is then instructed to cite only the evidence it was given, verbatim. This is retrieval-augmented generation reduced to its most defensible form: a small, hand-verified, domain-expert-curated knowledge base, selected by transparent rules.

4.4 Real-time streaming to the browser

A blank screen for ten seconds reads as "broken." Jacobian deployed the generation Lambda in `RESPONSE_STREAM` invoke mode and built a Server-Sent Events protocol so the letter renders token-by-token, with a typing cursor, as the model produces it:

```
meta → body_chunk* → [validation_warning | quality_warnings | filing_instructions | citations] → done | error
```

The browser consumes the SSE stream, renders progressively, and supports cancellation mid-generation. The platform measures **time-to-first-byte**, **model-first-token latency**, and **total stream duration** on every request — the UX metric and the cost metric, captured together.

4.5 Privacy by design: PII never reaches the model in the clear

This is the architectural decision Jacobian is proudest of. Patient PII — name, member ID, claim number, address — is **tokenized in the browser** before anything is sent upstream. The model sees and emits only placeholders like `[PATIENT_NAME]` and `[CLAIM_NUMBER]`. After the stream completes, the browser substitutes the real values back into the rendered letter, locally.

Consequences:

- The model and the Lambda operate on **redacted** content.
- **No PII is persisted anywhere** — there is no database; wizard state is ephemeral browser memory.
- The substitution buffer is hardened against edge cases (a token split across two stream chunks, an unclosed bracket at a chunk boundary) — these paths are explicitly fuzz-tested.

Privacy is enforced by the data flow, not by policy.

4.6 Validation and post-processing

Before and as the letter streams, multiple guardrails run:

- **Bad-phrase scrubbing.** A server-side buffer (which holds the trailing characters of each chunk so it can catch phrases that straddle a chunk boundary) strips language that reads as desperate, threatening, or counterproductive — removing the offending phrase and the remainder of its sentence.
- **Claim checking.** Numerical clinical claims (effect sizes, A1c reductions, trial outcomes) are scanned; unverified claims raise a non-blocking quality warning so the patient and reviewers can sanity-check them.
- **Required-sections assertion.** The letter must contain its core sections ("why this denial should be overturned," "requested action"); a missing section emits a validation warning.
- **State-aware filing instructions.** The deterministic engine appends the correct, jurisdiction-specific filing instructions and deadlines for the patient's state and insurance type.

The model writes; the system verifies. That division of labor is the whole design philosophy.

---

5. How the Model & Pipeline Were Developed

5.1 Spec-driven delivery

Jacobian delivered the platform as a sequence of reviewed specifications, each with a design doc, an implementation plan, an eval/test gate, and a handoff. The AI capability matured across them:

Spec	Capability
F	AI appeal-letter generator V1 — four-step wizard, Haiku 4.5 orchestration, evidence corpus, state data, bot verification, rate limiting
G	V1.5 — PDF-upload + OCR, urgency flag, six insurance-type filing variants, kill switch, effect-size guardrails, evidence-corpus expansion
H	Streaming letter generation — `RESPONSE_STREAM` Lambda, SSE protocol, client-side PII substitution, server-side bad-phrase buffering, streaming telemetry
J	Observability — dashboards, cost visibility, synthetic canary, and alerting

(Specs A–E and I covered the surrounding site relaunch, author/E-E-A-T entities, the 51-jurisdiction external-review audit, the wizard UI redesign, and the contact form.)

5.2 An evaluation harness for AI quality

Because the output is clinical and consequential, AI quality is **regression-tested like any other behavior**. Jacobian built an eval harness with:

- **Synthetic scenarios** — representative wizard requests covering medication × state × denial-reason combinations.
- **A locked baseline** — snapshots of reference letters, captured before major changes, used as a regression anchor.
- **Similarity gating** — generated letters are compared against the baseline using Jaccard similarity, with thresholds calibrated to the real paraphrase variance of Haiku at the production temperature (so the gate catches regressions without flagging acceptable wording differences).
- **Streaming-equivalence checks** — the SSE path is verified to produce the same content as the buffered path, with no PII token leakage, including a chunk-boundary fuzz test that splits every PII token at every position.
- **A real anchor case** — a redacted real denial plus a known-winning appeal, used to benchmark output quality against ground truth.

5.3 Test-driven, end to end

The platform ships with **485+ automated tests** (279 on the front end, 206 on the infrastructure and Lambda code), plus type-checking and formatting gates, all green. AI behavior, infrastructure synthesis, PII handling, and business logic are all covered. The eval harness runs against the live streaming endpoint using a dedicated bypass token so quality can be measured in production conditions without tripping the abuse controls.

---

6. Cost Engineering & FinOps

A free tool with an LLM backend lives or dies on cost control. Jacobian engineered FinOps into the platform from the first AI spec.

6.1 Model selection as a cost decision

Claude Haiku 4.5 was chosen deliberately. The two-call extract/generate design plus deterministic grounding means the heavy lifting (structure, evidence, validation) is done by code, leaving the model responsible for language — a task Haiku handles at a fraction of the cost and latency of larger models. This is cost-aware model routing applied at the architecture level: use the smallest model that clears the quality bar, and make the quality bar measurable (Section 5.2).

6.2 Per-letter cost telemetry

Every generation logs its exact token usage and computed dollar cost, using the model's known pricing ($0.80 per million input tokens, $4.00 per million output tokens). That figure is emitted as a structured metric on every request — so cost is a first-class, dashboarded signal, correlated with latency and volume.

6.3 Daily spend ingestion and a composite cost-spike alarm

- A scheduled **Lambda pulls AWS Cost Explorer** data daily (via EventBridge), publishing per-service and total daily spend as CloudWatch metrics.
- A **composite cost-spike alarm** fires when combined spend (Anthropic API + AWS) exceeds a hard daily ceiling **and** runs materially above the trailing seven-day average — distinguishing a genuine surge from normal growth — routing to Slack.

6.4 The kill switch

When cost, abuse, or an upstream model outage demands it, a single Lambda environment variable (`APPEAL_LETTER_DISABLED=true`) disables generation in seconds. Critically, the tool **degrades gracefully** rather than breaking: patients still receive a state-aware fallback — a fill-in-the-blank template letter, their state's filing instructions, and a link to an alternative tool. The mission is served even when the AI is paused.

---

7. Abuse Prevention & Anti-Scraping

A public, unauthenticated, free AI endpoint is an attractive target. Jacobian layered defenses so no single failure is catastrophic:

- **Bot mitigation at the edge.** Cloudflare Turnstile gates every submission; the token is verified server-side before any model call.
- **Rate limiting.** Per-IP limits (3 requests / 5 minutes, 10 / hour) reject abusive bursts before they reach the model.
- **Geo-fencing.** Generation is restricted to the United States and its territories (the only jurisdictions the tool's legal guidance covers), enforced from the CloudFront viewer-country signal — which also shrinks the global abuse surface.
- **The kill switch** (Section 6.4) as the ultimate backstop.
- **Eval bypass token.** A secret token lets Jacobian's eval harness exercise the live endpoint without weakening any of the above for real traffic.

Together these protect both the patient experience and the cost ceiling, and they double as anti-scraping controls: the endpoint resists automated harvesting of generated content.

---

8. The AWS Architecture

Jacobian built type2.wiki as a **100% serverless** workload — no standing servers, no patching treadmill, pay-per-use economics, and elastic scaling that suits a free public tool with spiky demand. Everything is defined as **infrastructure-as-code in AWS CDK** (TypeScript), across five stacks, deployed into a dedicated AWS account (`805815160763`) in `us-east-1` .

8.1 Compute — AWS Lambda

Function	Role
AppealLetterFn	The AI pipeline — `RESPONSE_STREAM` mode, 512 MB, 60 s timeout
ContactFn	Contact-form handler (sends mail via Amazon SES)
AwsCostDailyFn	Daily Cost Explorer ingestion (EventBridge-scheduled)
OpsAlertFormatterFn	Formats SNS alarms into Slack messages

All functions are bundled and deployed via CDK; none require a VPC, NAT, or always-on capacity.

8.2 Edge & delivery — Amazon CloudFront, S3, ACM, Route 53

- **Amazon S3** hosts the static Astro site (origin-access-control locked, versioned, encrypted), with a second bucket for access logs under a 90-day lifecycle.
- **Amazon CloudFront** is the single front door: it serves the static site and routes `/api/*` paths to the Lambda Function URL origins, enforcing **TLS 1.2+** and HTTP/2. A CloudFront Function performs legacy-URL redirects and directory rewrites at the edge.
- **AWS Certificate Manager** provides the TLS certificate for `type2.wiki` and its subdomains.
- **Amazon Route 53** hosts DNS in a separate Jacobian-managed account (`959279893127`) — a deliberate separation of the DNS root of trust from the application account.

8.3 Secrets — AWS Secrets Manager

All credentials live in Secrets Manager and are injected at runtime — the Anthropic API key, the Turnstile secret, the eval-bypass token, the Slack webhook, and the GA4 Measurement Protocol secret. **No secret is ever in source, in environment files, or in the bundle.**

8.4 Messaging & scheduling — Amazon SNS, EventBridge, SES

- **Amazon SNS** is the alarm bus; all operational alarms publish to a single topic consumed by the Slack-formatter Lambda.
- **Amazon EventBridge** schedules the daily cost-ingestion job.
- **Amazon SES** sends contact-form mail.

8.5 CI/CD — GitHub Actions via IAM OIDC

Deploys are keyless. An **IAM OpenID Connect provider** trusts GitHub Actions, which assumes a least-privilege deploy role (scoped to the site bucket and CloudFront invalidations) to publish the static site on every push to `main` . Infrastructure changes are applied with `cdk deploy` under a scoped SSO profile. There are **no long-lived AWS keys** in CI.

---

9. Security & Compliance Posture

Security is layered across the edge, the application, the data path, the identity plane, and a continuous monitoring plane.

9.1 Application & data-path controls (delivered)

- **No PII persistence** — the strongest control is structural: there is no datastore to breach, and PII is tokenized client-side (Section 4.5).
- **Secrets isolation** — AWS Secrets Manager, runtime injection only (Section 8.3).
- **Least-privilege IAM** — each Lambda and the CI role hold only the permissions they need (e.g., the contact Lambda's send-mail scope is restricted to its SES identity).
- **Encryption everywhere** — S3 server-side encryption at rest; TLS 1.2+ in transit via CloudFront and ACM.
- **Origin Access Control** — the S3 origin is reachable only through CloudFront, never directly.
- **Edge protections** — Turnstile, rate limiting, and geo-fencing (Section 7).

9.2 Continuous security monitoring

The platform's threat-detection and posture-management plane is built on AWS-native security services:

- **AWS WAF** — a web ACL in front of CloudFront applies managed rule groups (common exploits, known-bad inputs, rate-based rules) as a first filter on inbound traffic, complementing the application-level controls.
- **Amazon GuardDuty** — continuous threat detection across the AWS account, analyzing CloudTrail management events, S3 data events, and DNS/network telemetry for anomalous or malicious behavior (credential misuse, reconnaissance, exfiltration patterns).
- **Amazon Inspector** — automated, continual vulnerability assessment of the Lambda functions and their dependency trees, surfacing CVEs in the deployed code packages so they can be remediated quickly.
- **AWS Security Hub** — the aggregation and scoring plane: it ingests findings from GuardDuty, Inspector, and AWS configuration checks against best-practice standards (such as the AWS Foundational Security Best Practices standard), producing a single prioritized security posture view for the account.

Security Hub findings of high severity are routed into the same SNS→Slack alerting path used for operational alarms (Section 10), so security and reliability share one on-call surface.

9.3 Identity & supply chain

- **OIDC-federated CI** — no static cloud credentials anywhere in the pipeline (Section 8.5).
- **Dependency hygiene** — a locked dependency graph (pnpm) plus Inspector's continual scanning closes the loop on third-party risk.

---

10. Observability & Operations

Jacobian shipped observability as a first-class capability (Spec J) so the team can *operate* the platform, not just deploy it.

10.1 Dashboards

Two **Amazon CloudWatch dashboards** give a single-pane view:

- **Appeal-letter dashboard** — letters generated, Lambda invocations and errors, streaming time-to-first-byte and total duration, upstream model errors, output tokens, and computed cost.
- **Site dashboard** — CloudFront requests, bytes, 4xx/5xx rates, daily AWS spend, and abuse signals (Turnstile rejections, geo-blocks, rate-limit hits).

These are driven by **CloudWatch metric filters** over the Lambda's structured JSON logs, plus the cost metrics from the daily ingestion job.

10.2 Synthetic monitoring

A **CloudWatch Synthetics canary** runs every five minutes, loading both the home page and the appeal-letter page and asserting that expected content is present — catching outages and broken deploys before patients do.

10.3 Alerting

Three alarms publish to **Amazon SNS**, which fans out to **Slack** via a formatter Lambda:

- **Canary failure** — the site or tool is unreachable/broken.
- **AI generation broken** — upstream model errors exceed threshold in a 15-minute window.
- **Cost spike** — the composite spend alarm (Section 6.3).

High-severity **Security Hub** findings join the same path (Section 9.2).

10.4 Product analytics

Google Analytics 4 captures the wizard funnel client-side (start → step completion → submit → print) and receives **server-side events** via the GA4 Measurement Protocol — including a `letter_generated` event carrying the per-letter cost, and the daily AWS cost. A **Looker Studio** report turns this into a funnel / cost-correlation / breakdown view, so product and spend are analyzed together.

---

11. Architecture detail

type2.wiki is a 100% serverless workload defined as AWS CDK across five stacks. A patient's browser — which tokenizes PII locally before anything leaves the device — reaches the platform through **CloudFront + WAF**; the streaming `AppealLetterFn` Lambda fetches secrets, makes two Claude Haiku 4.5 calls (extract, then generate) against redacted input, and streams the letter back over Server-Sent Events. Supporting planes wrap the core pipeline: **Secrets Manager + IAM (OIDC)** for credentials and keyless CI, **SNS / EventBridge / SES** for messaging and scheduling, **GuardDuty / Inspector / Security Hub / CloudTrail** for continuous security, and **CloudWatch** (logs, two dashboards, a five-minute synthetic canary, and alarms) for observability. DNS is deliberately isolated in a separate AWS account.

---

12. Outcomes & Impact

12.1 What the platform delivers

- **A complete, state-correct, evidence-backed appeal letter in minutes**, free and with no login, for any patient in the U.S. and its territories.
- **A privacy posture patients can trust** — their personal data never leaves their browser in the clear, and is never stored.
- **Graceful degradation** — even with the AI paused, patients receive a usable fallback letter and filing instructions.
- **Predictable economics** — per-letter cost is measured and capped; a spend anomaly pages the team within a day.
- **Operational confidence** — a five-minute canary, two dashboards, and Slack alerting mean issues are caught proactively.

12.2 Engineering outcomes

- 100% serverless, infrastructure-as-code, zero standing servers to patch or pay for idle.
- 485+ automated tests plus an AI-specific eval harness gating quality.
- Security monitoring (WAF, GuardDuty, Inspector, Security Hub) feeding a unified alerting plane.

- Cost, latency, and product funnel analyzed together.

> *"What Jacobian built doesn't feel like 'an AI tool.' It feels like having an expert sit down with you and write the letter you didn't know how to write — and it does it without ever holding onto your private information. For our community, that's everything." — Gwynne Sullivan, co-founder, type2.wiki*

> *"The hard part was never getting a model to write fluent text — it was making the output trustworthy, private, and affordable enough to give away for free, and keeping it that way in production. That's an engineering problem, and it's the kind of problem Jacobian Engineering exists to solve." — Erik Jones, Jacobian Engineering*

---

13. Roadmap & The Jacobian Engagement Model

13.1 Architectural direction: Amazon Bedrock

The pipeline is intentionally model-portable: the model call is isolated behind a thin wrapper, the prompts and evidence corpus are provider-agnostic, and the eval harness measures output quality independent of where inference runs. The natural next step is to run Claude inference through **Amazon Bedrock**, keeping the entire data path inside AWS, unifying billing and governance, and inheriting Bedrock's IAM, logging, and guardrail integration. The eval harness already in place is exactly what makes such a migration safe — quality regressions would be caught before release.

13.2 Further roadmap candidates

- Expanding the curated evidence corpus and adding new denial archetypes as the community surfaces them.
- A managed knowledge base / vector store for the evidence corpus as it grows beyond hand-curation.
- Deeper Security Hub automation (auto-remediation playbooks for common findings).
- Multi-region resilience for the static delivery tier.

13.3 The Jacobian engagement model

This project is representative of how Jacobian Engineering works: take a real-world need, deliver it as a sequence of reviewed specs with measurable acceptance criteria, build it serverless-first on AWS, make AI quality and cost *measurable* rather than hoped-for, and hand over a system that is observable, secure, and operable. The same playbook applies whether the workload is patient advocacy, internal automation, or a customer-facing AI product.

---

14. Appendices

Appendix A — AWS services used

Category	Service	Role in the solution
Compute	AWS Lambda	AI pipeline + supporting functions (serverless)
Edge/CDN	Amazon CloudFront	Single front door; static + `/api/*` routing; TLS 1.2+
Edge security	AWS WAF	Managed-rule + rate-based web ACL on CloudFront
Storage	Amazon S3	Static site hosting (OAC) + access logs

Category	Service	Role in the solution
TLS	AWS Certificate Manager	Certificate for type2.wiki
DNS	Amazon Route 53	Hosted zone (separate account)
Secrets	AWS Secrets Manager	All runtime credentials
Identity	AWS IAM (+ OIDC)	Least-privilege roles; keyless CI
Messaging	Amazon SNS	Alarm fan-out to Slack
Scheduling	Amazon EventBridge	Daily cost ingestion trigger
Email	Amazon SES	Contact-form delivery
Threat detection	Amazon GuardDuty	Account-wide threat detection
Vulnerability mgmt	Amazon Inspector	Lambda + dependency CVE scanning
Security posture	AWS Security Hub	Findings aggregation + best-practice scoring
Audit	AWS CloudTrail	API audit trail (GuardDuty source)
Observability	Amazon CloudWatch (Logs, Metrics, Dashboards, Synthetics, Alarms)	Full operational visibility
Cost	AWS Cost Explorer	Daily spend ingestion
IaC	AWS CDK	Five-stack infrastructure-as-code

Appendix B — Non-AWS components

Component	Role
Anthropic Claude Haiku 4.5	LLM inference (extract + generate)
Cloudflare Turnstile	Bot mitigation
Google Analytics 4 + Looker Studio	Product funnel + cost analytics
GitHub Actions	CI/CD (federated to AWS via OIDC)
Astro 5 / React / TypeScript / MDX	Front-end static site + wizard islands

Appendix C — Glossary

- **SSE (Server-Sent Events):** a one-way streaming protocol used to push letter tokens to the browser as they are generated.
- **OAC (Origin Access Control):** restricts S3 origin access so content is reachable only through CloudFront.
- **RAG (Retrieval-Augmented Generation):** grounding model output in selected external documents; here implemented with a deterministic, rules-based evidence selector.
- **FinOps:** the practice of making cloud and AI spend a measured, owned, optimizable engineering concern.
- **Eval harness:** an automated suite that scores AI output quality against scenarios and baselines, gating releases.