7-PART SERIES · FREE COMPANION

Staff policy answerer

A serverless assistant that answers staff questions about company policy from your own handbook, in chat. An employee asks “how many sick days do I have left?” or “what’s the travel expense rule?” and it answers in plain words using only your real policy docs, links the exact section it used, and says “I’m not sure, ask HR” when the answer isn’t in the docs. It never makes up policy. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Staff policy answerer

PART 1 OF 7

MAY 27, 2026 PART 1 OF 7 · STAFF POLICY ANSWERER SERIES ~5 MIN READ

A staff policy answerer on AWS for a few dollars a month

Every small business answers the same handful of staff questions over and over. How many sick days do I have left? What's the rule on travel expenses? Can I carry leave into next year? Who approves overtime? Today those questions land in HR's inbox, or a manager's DMs, or they get a half-remembered answer from a colleague who read the handbook once in 2023. This post walks through the design of a small assistant that answers them in chat — in plain words, from your own real policy docs — links the exact section it used, and says "I'm not sure, ask HR" when the answer isn't in the handbook. It never makes up policy. Think of it as the inside-the-company twin of the website chat assistant: same idea, but pointed at staff and the handbook instead of customers and the website.

KEY TAKEAWAYS

- Staff ask in Slack; answers come from your own handbook, never from the model's own opinions.
- Every answer ends in one of three moves: a grounded answer with a section link, "ask HR," or a clean handoff.
- The handbook lives in Drive. Edit a policy doc and the answerer re-reads it within minutes — no deploy.
- If the answer isn't in the docs, the assistant says so. It does not guess.
- Designed on AWS for about $3/month at typical small-business volume.

The whole system on one page

Before any code, here's the shape of what we're designing.

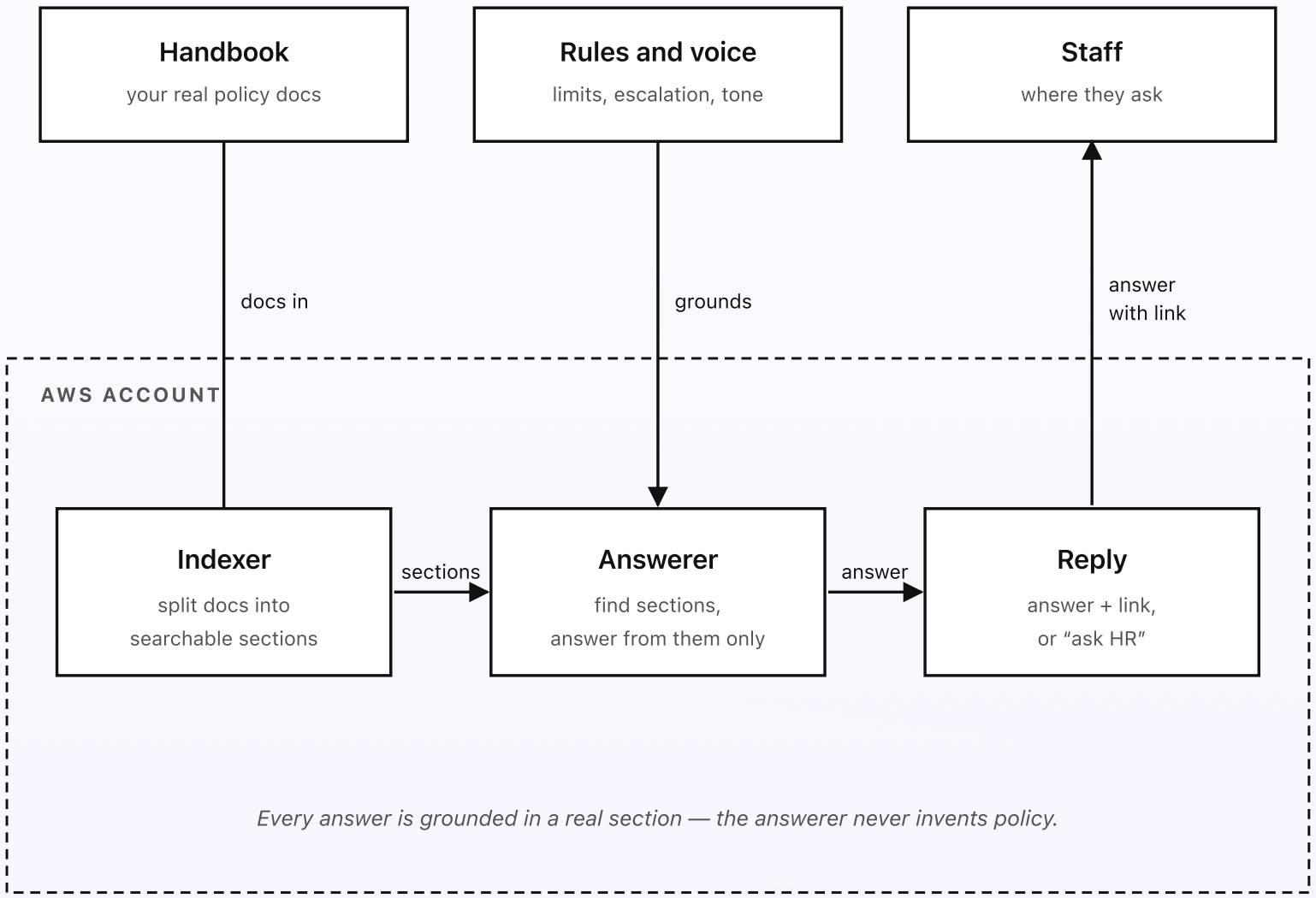

Handbook
your real policy docs
Rules and voice
limits, escalation, tone
Staff
where they ask
docs in
grounds
answer
with link
AWS ACCOUNT
Indexer
split docs into
searchable sections
sections
Answerer
find sections,
answer from them only
answer
Reply
answer + link,
or “ask HR”
Every answer is grounded in a real section — the answerer never invents policy.

Fig 1. Three sources outside, three pieces inside AWS. The handbook, the rules, and the staff. The Indexer makes the docs searchable. The Answerer answers from those docs only. The Reply sends a grounded answer with a link, or an honest "ask HR."

What you set up once (the outside)

- **The handbook.** A Google Drive folder holding the policy docs you already have: the employee handbook, the leave policy, the expense policy, the code of conduct, the remote-work rules, whatever your business actually runs on. These are the only source of truth the assistant is allowed to answer from. You don't rewrite them for the machine. You drop the same documents your team already reads into one folder and point the assistant at it.
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc sets the boundaries: which topics the assistant may answer (leave, expenses, hours, conduct) and which it must always hand to a human (anything about pay disputes, terminations, grievances, or legal questions). It also names the escalation contact per topic — who "ask HR" actually points to. The *voice* doc holds the tone: short, kind, plain words; always link the section; never sound like a lawyer.
- **Staff.** The people who ask. They ask the way they already talk to each other — a Slack DM to the assistant, or an @-mention in a shared #ask-hr channel. The answer comes back in the same place, with a link to the exact handbook section so anyone can check the source for themselves.

What runs on every question (the inside)

- **The indexer.** Reads the handbook docs from Drive, splits each one into small sections (usually a heading plus its paragraphs), and stores every section in a

way that can be searched by meaning, not just exact words. This matters because nobody asks a question using the handbook's exact words. Somebody types "how many sick days do I have left"; the handbook section is titled "Paid Personal Leave Accrual." The indexer is what lets the right section come back anyway. It re-runs whenever a doc changes, so the assistant is always answering from the current version.

- **The answerer.** Takes the staff question, finds the few handbook sections most likely to hold the answer, and asks the model to write an answer using *only* those sections. The prompt is strict: answer from the provided sections, quote the rule, and if the sections don't contain the answer, say so plainly. The model never gets to fall back on what it "knows" about leave policy in general — only what's in your docs counts.
- **The reply.** Sends the answer back into the same Slack thread. A grounded answer comes with the plain-words explanation and a link to the section it used. When nothing in the handbook matched, the reply is honest: "I couldn't find this in the handbook — here's who to ask," with the right HR contact for that topic. Every question and answer is logged so HR can see what staff are actually asking.

In plain words

It's Tuesday and Dev wants to book Friday off but isn't sure he has the days. He DMs the assistant: "how many vacation days do I have left this year?" The assistant can't answer that one from the handbook — his personal balance isn't a policy, it's a number in the HR system — so it says: "I can't see your personal balance, that lives in the HR portal. But the policy is 20 days a year, accrued

monthly, and up to 5 can carry into next year — *[link: Leave Policy §3]*. For your exact remaining balance, ask Priya in HR." Honest about what it knows, useful about what it can.

An hour later someone else asks "can I expense an airport lounge pass?" The handbook does cover that — the expense policy lists lounge access as not reimbursable unless a flight is delayed over three hours. The assistant answers exactly that, in one short paragraph, with the section link. No back-and-forth, no waiting for HR to get to the email, no guessing.

The cost of running this is about $3 a month at SMB volume. The cost of *not* running it is HR answering the same twenty questions every week, and the slow drift where half the team operates on a policy that changed a year ago.

DESIGN RULES THAT SHAPED EVERY DECISION

- Answers come only from your handbook. The model is told to ignore its own general knowledge.
- Three moves, always. Grounded answer, “ask HR,” or a clean handoff on off-limits topics. There is no fourth.
- Every grounded answer links the exact section, so staff can check the source themselves.
- “I’m not sure” is a valid, encouraged answer. Guessing about policy is the one thing it must never do.
- The handbook lives in Drive. Editing a policy doesn’t need a deploy — the indexer re-reads it.
- Every question is logged. HR sees what staff ask and where the handbook has gaps.

Why this shape

Most teams answer policy questions in one of three ways: HR fields them one at a time, a manager guesses, or somebody searches a 40-page PDF and gives up. HR fielding them works until HR is busy, on leave, or fielding the same question for the fortieth time. A manager guessing is how two people on the same team end up with two different answers about carry-over leave. And the 40-page PDF is technically the source of truth, but nobody reads it — it’s a search problem wearing a document costume.

The setup above keeps the source of truth where it belongs — in the policy docs HR already maintains — but adds a small system that *reads* those docs every time someone asks, and answers only from them. Answers come back in seconds, in plain words, with the exact section attached so anyone can verify. When the handbook genuinely doesn't cover something, the assistant says so instead of inventing a plausible-sounding rule. The assistant is invisible most of the time; useful the moment somebody has a question they'd otherwise have to interrupt a person to answer.

The next four posts walk through each piece in turn: how a staff question reaches the answerer, how a policy answer gets grounded in the right section, how a policy answer stays honest, and how the handbook stays current. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 27, 2026 PART 2 OF 7 · STAFF POLICY ANSWERER SERIES ~4 MIN READ

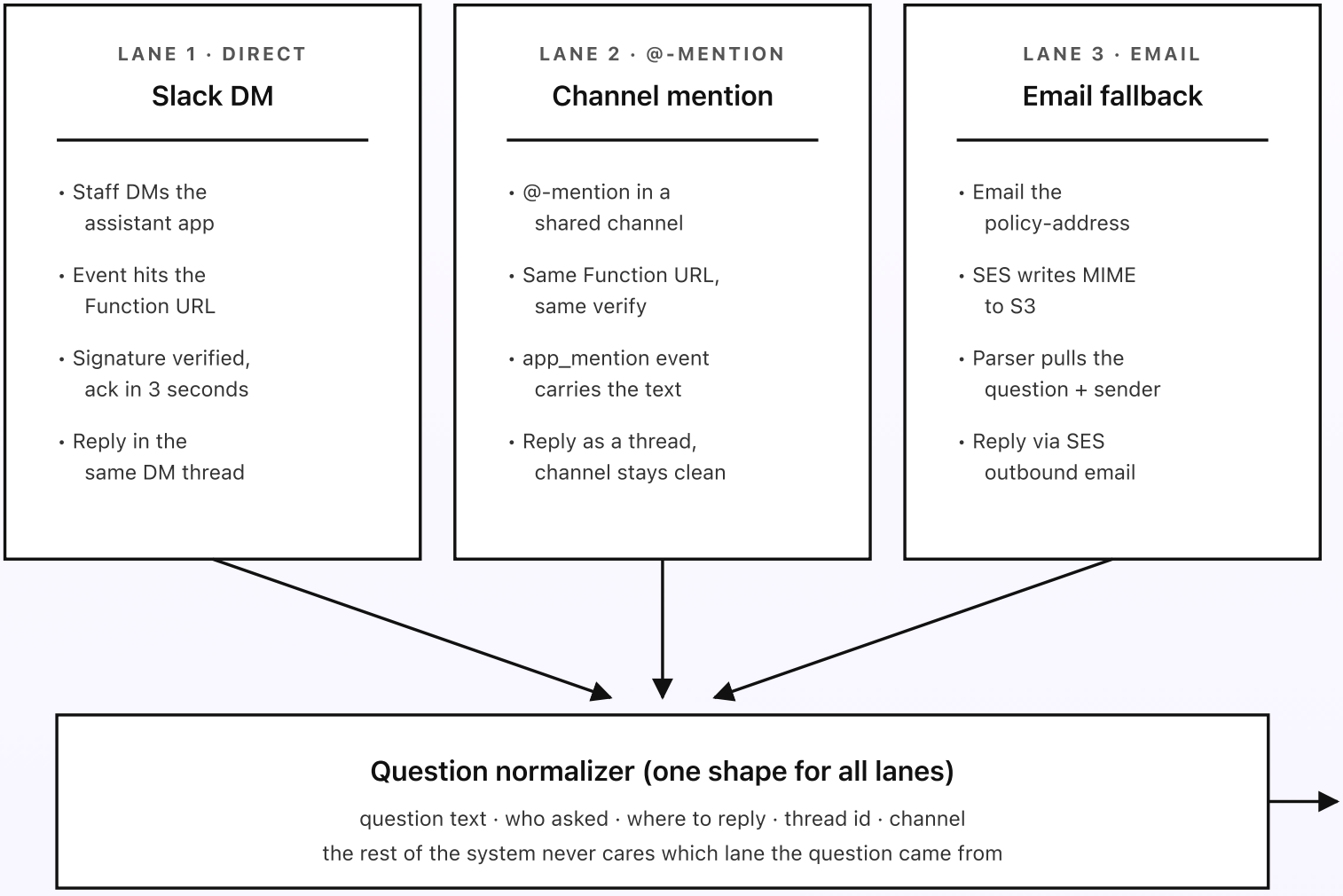
How a staff question reaches the answerer

The answerer can only help with questions it actually receives. So the first job is meeting people where they already are. There are three ways a question gets in: somebody DMs the assistant in Slack, somebody @-mentions it in a shared channel like #ask-hr, or somebody emails a dedicated address. The first two are how most teams will use it. The third exists because not everyone lives in Slack — the warehouse, the shop floor, the contractor who only has email. All three end up in the same place and run through the same answer flow.

KEY TAKEAWAYS

- Three intake lanes feed one answer flow: a Slack DM, a channel @-mention, and an email fallback.
- Slack events hit a Lambda Function URL; the request is verified before anything else runs.
- The assistant replies in the same thread, so the question and answer stay together.
- Email questions come in via SES inbound and get answered with a plain-text reply.
- One question normalizer means the rest of the system never cares which lane a question came from.

Three lanes into one answer flow



The DM and the channel are how most teams ask — email is the fallback so nobody is left out.

Fig 2. Three lanes converge on one question normalizer. Slack DMs and channel mentions hit a Function URL; email comes in through SES. After the normalizer, every question is the same shape, so the answerer never has to know where it came from.

Lane 1: the Slack DM

The most private lane, and the one people reach for when the question feels personal — leave, a benefit, a rule they're slightly embarrassed to not know. The staff member opens a direct message with the assistant's Slack app and types their question. Slack sends a `message` event to a Lambda Function URL (a plain web address that runs a Lambda — no API Gateway in front of it). The Lambda's very first job is to verify the request really came from Slack, using the signing secret stored in Secrets Manager. Slack expects an acknowledgment within three seconds, so the Lambda confirms receipt immediately and does the actual answering in the background, posting the reply back into the same DM thread when it's ready.

Keeping the reply in the same thread matters: the question and the answer stay together, so if the person asks a follow-up the assistant has the context, and if they want to forward the answer to a teammate, it travels as one tidy exchange.

Lane 2: the channel @-mention (the one HR will love)

A shared channel — call it #ask-hr — is where the answerer earns its keep, because answers become public knowledge. Someone @-mentions the assistant: "@policy-bot what's the work-from-home rule on Mondays?" The same Slack app sends an `app_mention` event to the same Function URL, runs the same verification, and the answer is posted back as a threaded reply so the channel

itself stays scannable. The next person with the same question can search the channel and find it, or just ask again and get the same grounded answer.

This is the lane that quietly takes load off HR. The first time a question is answered in the channel, it's answered for everyone, with the section link right there for anyone who wants to check it.

Lane 3: email fallback

Not everyone is in Slack all day. The warehouse team, the shop floor, a part-time contractor — for them, a dedicated email address is the lane that works. Set up `policy@your-company.com` via Amazon SES. An email to that address gets written as raw MIME to an S3 bucket; the S3 write triggers a small parser Lambda that walks the MIME, pulls out the plain question text and the sender's address, and hands it to the same answer flow as the other two lanes. The answer comes back as a plain-text reply via SES outbound — same grounded answer, same section link, just delivered as an email instead of a Slack message.

Email is the most opt-in of the three lanes. A team that lives in Slack may never use it; a team with staff off the keyboard needs it on day one.

Why everything passes through one normalizer

Three lanes in, but only one shape of question after the front door. That's deliberate. The moment a question lands, the intake Lambda turns it into the same small record regardless of source: the question text, who asked, where to reply (Slack thread or email address), and a timestamp. Everything downstream — finding the right handbook section, asking the model, deciding whether to hand

off to HR — works off that one record. If you later add a fourth lane (a Microsoft Teams app, a kiosk on the shop floor), you write one more small intake adapter and the rest of the system doesn't change.

Next post: how the answerer takes that normalized question, finds the few handbook sections most likely to hold the answer, and grounds its reply in them.

PART 3 OF 7

MAY 27, 2026 PART 3 OF 7 · STAFF POLICY ANSWERER SERIES ~5 MIN READ

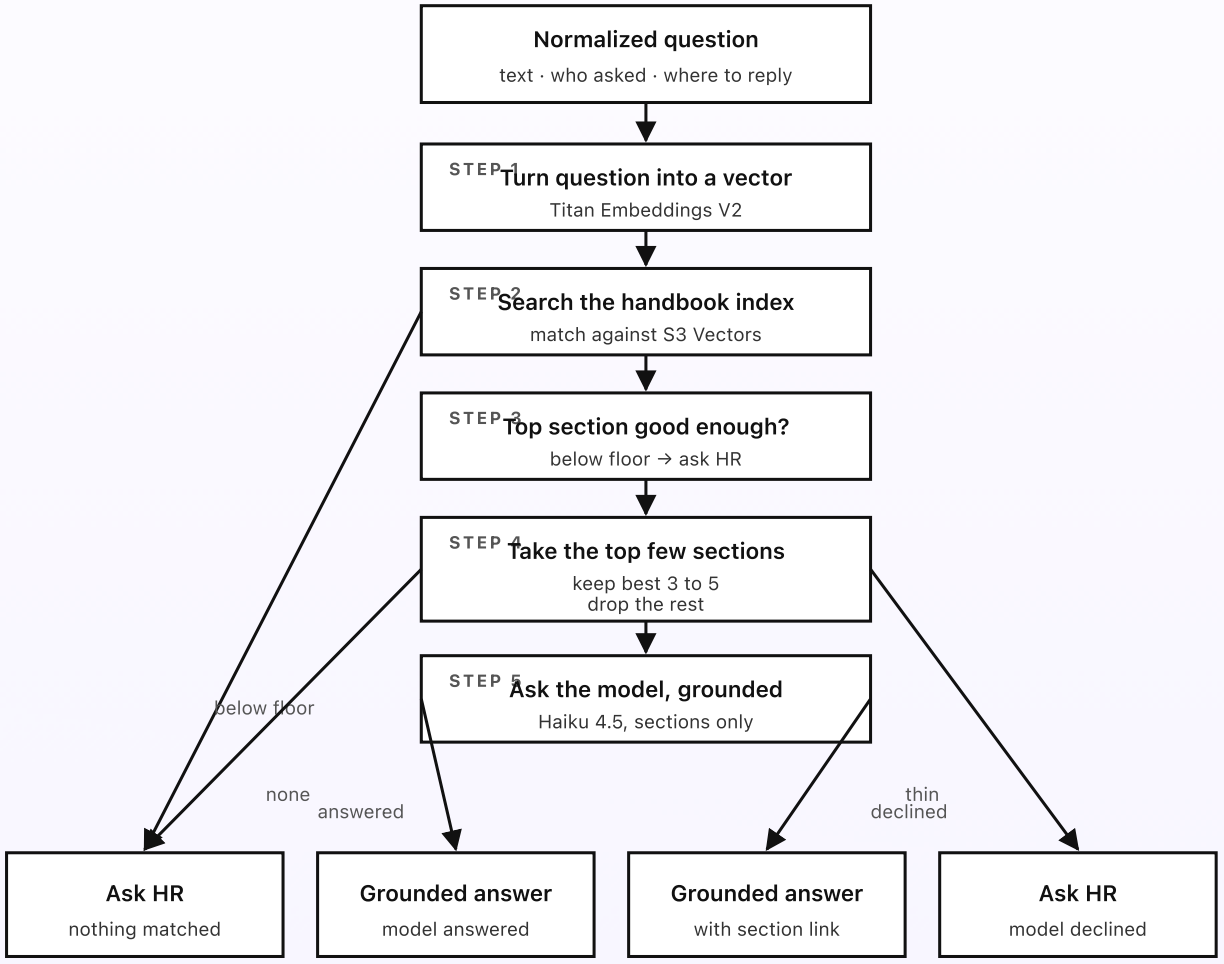
How a policy answer gets grounded

A normalized question arrives: “how many sick days do I get?” The answerer’s whole job is to answer it from your handbook and nowhere else. To do that, it turns the question into a search over the handbook, pulls back the few sections most likely to hold the answer, and asks the model to write a reply using only those sections. If the sections don’t contain the answer, the model is told to say so. The search step is what keeps the model honest — it can only talk about what was handed to it.

KEY TAKEAWAYS

- The question is searched against the handbook by meaning, not by exact words.
- Only the top few sections are pulled — enough to answer, few enough to stay precise.
- If nothing scores well enough, the answerer routes to “ask HR” without calling the model at all.
- The model answers from the pulled sections only and must cite the section it used.
- This is retrieval: search first, then answer. The model never works from memory.

The grounding flow, per question



The search is the guardrail — the model can only talk about the sections it was handed.

Fig 3. The grounding flow, per question. Five steps turn a question into a search, keep the best sections, and let the model answer only from them. If nothing matches, or the model declines, the answer is an honest "ask HR."

Search by meaning, not by exact words

Here's the problem the search step solves. The staff member types "how many sick days do I get?" The handbook section is headed "Paid Personal Leave — Accrual and Use," and the words "sick days" might not appear in it at all. A plain keyword search would miss it. So instead of matching words, the answerer matches *meaning*.

It does that with embeddings. An embedding turns a piece of text into a long list of numbers — a kind of fingerprint of what the text is about. Texts that mean similar things get similar fingerprints, even when they share no words. The answerer runs the question through Amazon Titan Text Embeddings V2 to get the question's fingerprint, then compares it against the fingerprints of every handbook section (which the indexer computed ahead of time — that's Part 5). The sections whose fingerprints are closest to the question's are the ones most likely to hold the answer. "Sick days" and "paid personal leave" land close together, so the right section comes back.

Those fingerprints live in Amazon S3 Vectors — a store built for exactly this kind of search, where you ask "which of my stored fingerprints are nearest to this one?" and get an answer in milliseconds. There's no always-on search server to pay for; you query it only when a question comes in.

Keep the context thin on purpose

The search can return ten or twenty sections, ranked by how close they are. The answerer keeps only the best three to five. That's a deliberate constraint. Hand the model too many sections and two bad things happen: the answer gets vaguer (it tries to reconcile sections that don't all apply), and the call costs more (more text in means more to pay for). Hand it the few sections that actually matter and the answer is sharp, cheap, and easy to cite.

There's also a confidence floor before the model is called at all. If the best-matching section's score is still weak — nothing in the handbook is really close to the question — the answerer skips the model entirely and routes straight to "ask HR." No point paying for a model call to answer a question the handbook plainly doesn't cover. The cheapest, safest answer to "the docs don't have this" is to say so directly.

Answer from the sections only

With the top sections in hand, the answerer calls Claude Haiku 4.5 (a small, fast, cheap model that's plenty for reading a few policy sections and writing a short answer) with a strict prompt. The shape of it: "Here is the staff member's question. Here are the only handbook sections you may use. Answer in plain words. Quote the specific rule. If these sections do not contain the answer, reply exactly that you don't have it in the handbook — do not guess, do not use general knowledge about employment policy."

That last line is the whole game. Models are trained on the public internet, so they have plenty of opinions about how sick leave "usually" works. For a staff policy

answerer, “usually” is poison — your business’s policy is whatever your docs say, not the industry average. By handing the model only your sections and forbidding outside knowledge, the answer is pinned to your reality. If your handbook says 12 sick days, it answers 12; it never drifts to “most companies offer 10.”

Why retrieval, and not just a big prompt

You might wonder why we search at all — why not paste the whole handbook into every model call and let it find the answer? Two reasons. First, cost and speed: a 40-page handbook in every call is slow and adds up fast at any real question volume. Searching first means each call carries only the few relevant sections. Second, and more important, precision: a model handed the entire handbook is more likely to blend unrelated policies or cite the wrong one. Handing it exactly the sections that matched the question keeps the answer tight and makes the citation in Part 4 trustworthy — the section it answers from is the section it was given.

Next post: how a policy answer stays honest — how the citation gets attached, how off-limits topics get handed to a human, and the guardrails that keep the answerer from ever inventing policy.

PART 4 OF 7

MAY 27, 2026 PART 4 OF 7 · STAFF POLICY ANSWERER SERIES ~5 MIN READ

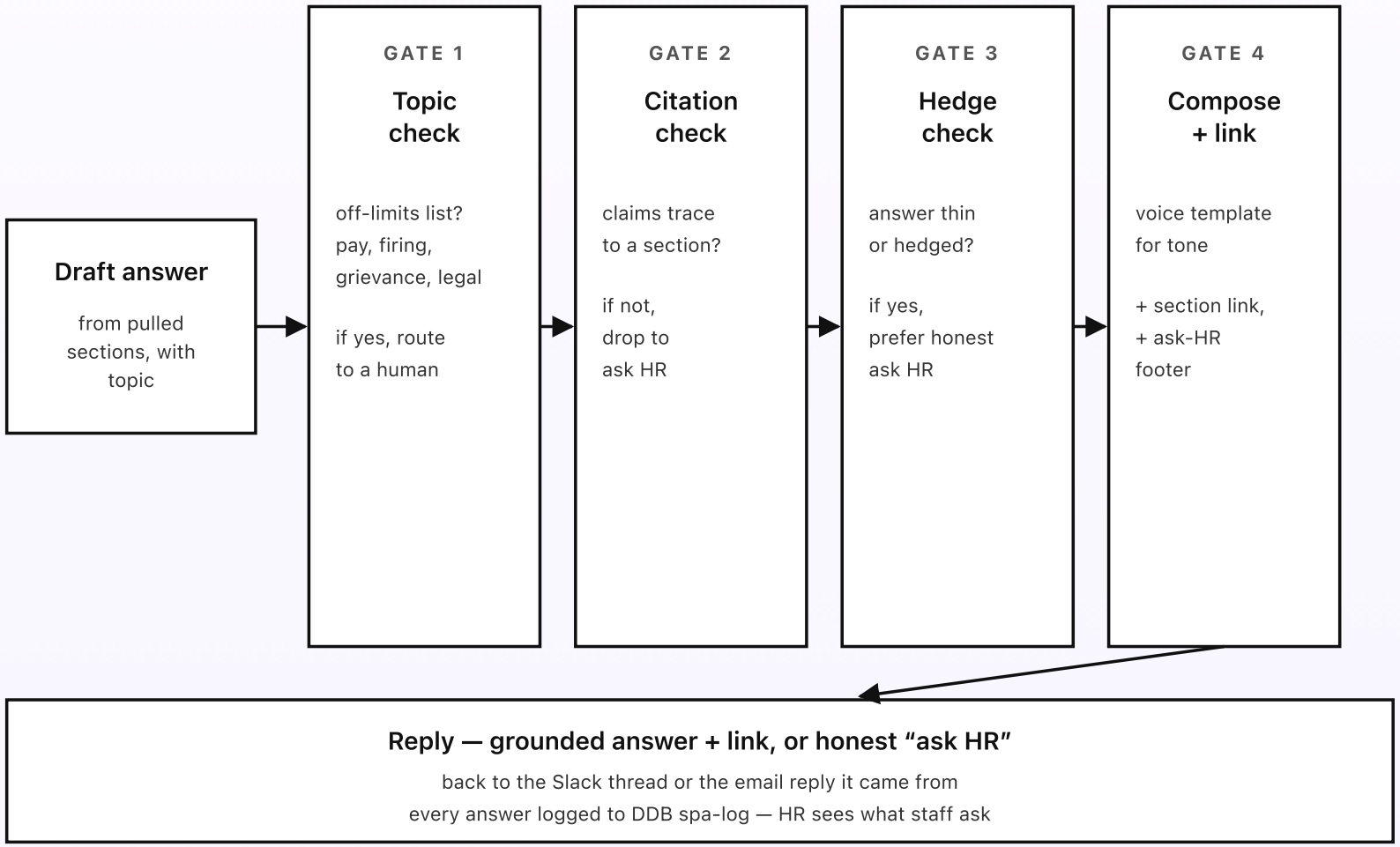
How a policy answer stays honest

The model wrote a draft answer from the sections it was handed. Before that draft reaches the staff member, it passes four small gates. Each one is there to catch a different way an answer could go wrong: it could touch a topic the assistant should never handle alone, it could state a rule that isn't actually in the pulled sections, it could sound more certain than the docs justify, or it could arrive without the section link that lets the reader check it. Four gates, four failure modes, no surprises.

KEY TAKEAWAYS

- Off-limits topics (pay disputes, terminations, grievances, legal) are handed straight to a human.
- The citation check makes sure the answer's claims actually trace back to a pulled section.
- The hedge check turns a thin answer into "ask HR" rather than a confident guess.
- Every grounded answer ships with the exact section link, so the reader can verify it.
- "I'm not sure, ask HR" is a first-class outcome, not a failure.

Four guardrails on every answer



Ask HR is a first-class outcome — every gate would rather hand off than guess.

Fig 4. Four guardrails between the draft and the staff member. Screen off-limits topics. Check every claim traces to a section. Catch thin answers and hedge to "ask HR." Compose with the section link. Then reply, and log it.

Gate 1: the topic check

Some questions should never get an automated answer, no matter how good the docs are. "Can they fire me for this?" "I think my paycheck is wrong." "How do I file a harassment complaint?" These are not handbook-lookups — they're moments where a person needs a person. The rules doc has an off-limits list: pay disputes, terminations, grievances, anything that reads as a legal or safety question. Gate 1 checks the question against that list first. If it matches, the assistant doesn't answer at all — it replies warmly and routes the person to the named human for that topic: "This is something to talk through with Priya in HR directly — here's how to reach her."

This gate runs first on purpose. There's no value in a clever answer to a question the system shouldn't be touching, and a confident wrong answer about a termination is the kind of mistake that ends up in a lawyer's email. When in doubt, the topic check sends it to a human.

Gate 2: the citation check

The model was told to answer only from the pulled sections, but "told to" isn't "guaranteed to." Gate 2 verifies it. The answerer asks the model to return its answer with the supporting section id attached to each claim, then checks that each cited section is actually one of the ones that were pulled. If the draft makes a

claim that doesn't trace back to a pulled section — a sign the model leaned on its own general knowledge — the draft is rejected and the answer becomes "ask HR."

This is the gate that turns the grounding from Part 3 into a promise rather than a hope. The search narrowed the model to a few sections; the citation check confirms the answer actually stayed inside them. An answer that can't point to a section it came from doesn't ship.

Gate 3: the hedge check

Sometimes the right sections came back, the model stayed grounded, but the answer is still shaky — the search confidence was middling, or the model itself wrote something like "it appears that…" or "this may depend on…". Gate 3 reads those signals. A thin or hedged answer about company policy is worse than no answer, because the reader can't tell the difference between "the handbook clearly says X" and "the assistant is guessing X." So when the signals are weak, the gate downgrades the reply to an honest "I'm not certain this is covered — please check with HR," optionally pointing at the closest section so the person has a head start.

The bar is set on purpose: the assistant should sound certain only when the docs make it certain. Everywhere else, it should sound like a careful colleague who says "let me not guess at that."

Gate 4: compose with the section link, then ship

The answer survived the first three gates. Gate 4 dresses it for delivery. The voice doc sets the tone — short, kind, plain words, no legalese — and the gate formats the answer to match. Then it attaches the exact section link: a deep link back to the handbook section the answer came from, so the reader can open the source and confirm it in one click. Below that goes a small standing footer: “Not what you needed? Ask HR.” with the right contact. Even a perfect grounded answer offers the human path, because the assistant’s job is to help, not to be the last word.

Then it ships — back into the Slack thread or out as an email reply, whichever lane the question arrived on — and the whole exchange is written to a log. That log is quietly valuable: it’s the list of what staff actually ask, which questions hit “ask HR” most often (a sign the handbook has a gap), and which sections answer the most questions. Part 5 uses that log to keep the handbook sharp.

Why the guardrails exist

None of these gates are clever. They’re the care a thoughtful HR person would take if they were answering each message themselves — don’t touch the question that needs a real conversation, don’t state a rule you can’t point to, don’t pretend to be sure when you’re not, and always show your work. Writing them as four small sequential gates makes that care part of the system instead of something you hope the model remembers. The result is an assistant that’s genuinely useful on the easy 80% of questions and genuinely safe on the hard 20%, because the hard ones get a human.

Next post: how the handbook stays current — how a policy edit in Drive flows into the index within minutes, so the answerer never quotes a rule that changed last

week.

PART 5 OF 7

MAY 27, 2026 PART 5 OF 7 · STAFF POLICY ANSWERER SERIES ~5 MIN READ

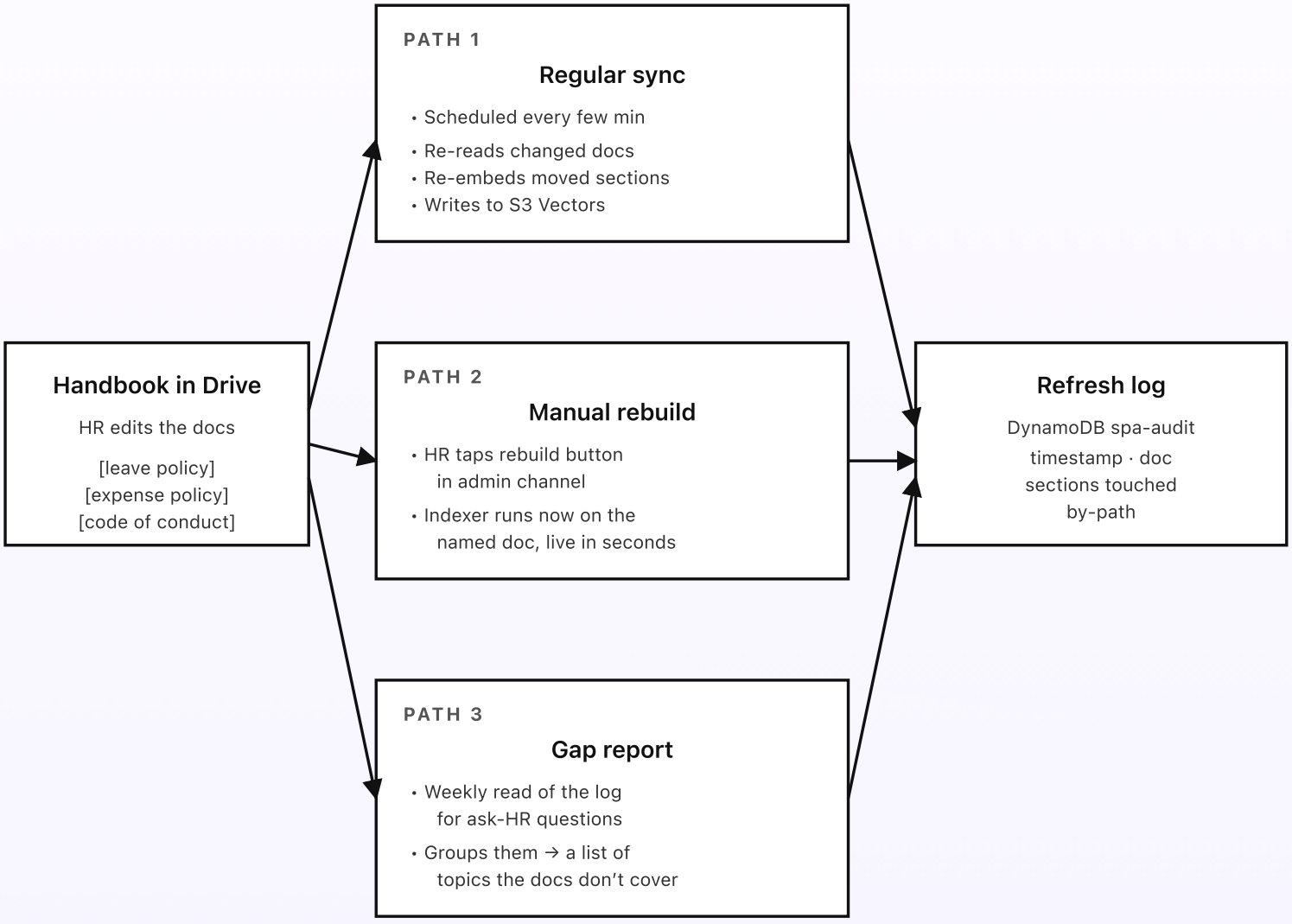
How the handbook stays current

HR rewrites the leave policy on a Monday. By Monday afternoon, the assistant should be answering from the new version — not still quoting last year's carry-over rule. An answerer that drifts out of date is worse than no answerer, because it's confidently wrong. This post walks through the three ways the searchable index gets refreshed — a regular sync, a one-tap manual rebuild, and a gap report — and how each one keeps the answers honest about what the handbook actually says today.

KEY TAKEAWAYS

- Three refresh paths: *regular sync* (changed docs), *manual rebuild* (urgent edit), *gap report* (the ask-HR list).
- A sync re-reads only the docs that changed and re-embeds only the sections that moved.
- A one-tap rebuild lets HR push an urgent policy change live in seconds.
- The gap report turns unanswered questions into a to-do list for the handbook.
- Every refresh is logged, so you can see exactly when a section's answer last changed.

Three ways to refresh the index



The gap report doesn't edit the handbook — it tells a human what to add. People own policy.

Fig 5. Three refresh paths, three effects. The regular sync keeps the index in step with edits. The manual rebuild pushes an urgent change live now. The gap report turns unanswered questions into a to-do list. Every refresh is logged.

Path 1: the regular sync (the one that runs itself)

The workhorse. A small scheduled job runs every few minutes. For each handbook doc, it asks Drive a cheap question: has this changed since I last looked? Drive keeps a revision marker on every file, so the job can tell at a glance which docs are new, which were edited, and which are untouched. For the untouched ones, it does nothing — no re-reading, no re-embedding, no cost. For a changed doc, it pulls the new text, splits it back into sections, and figures out which sections actually moved.

That last part matters for cost and speed. If HR fixed a typo in section 4, only section 4's fingerprint needs recomputing; the other thirty sections are identical and get left alone. The job re-embeds only the moved sections through Titan Text Embeddings V2, writes the new fingerprints into S3 Vectors, and removes the fingerprints for any sections that were deleted. Within a few minutes of HR hitting save, the answerer is searching the new version. Nobody had to deploy anything; editing a Google Doc was the whole workflow.

Path 2: the manual rebuild (for "this changed today")

A few-minute sync is fine for routine edits. But sometimes a policy changes and everyone needs the new answer *now* — the office is closing early for a storm, the

expense limit just dropped, the on-call rule changed this morning. Waiting even five minutes feels wrong when the question is already coming in.

So there's a manual rebuild. In a private admin channel, HR taps a "rebuild now" button and picks the doc (or rebuilds everything). The same indexer runs immediately on that doc instead of waiting for the next scheduled pass. The new sections are live in seconds. It's the same machinery as the regular sync, just triggered by a person instead of a clock — the escape hatch for the day a policy can't wait.

Path 3: the gap report (the handbook's to-do list)

The first two paths keep the index in step with the docs. The third path keeps the docs in step with reality. Every week, a job reads the question log from Part 4 and pulls out the questions that landed on "ask HR." Those are the questions the handbook couldn't answer — either because the policy genuinely isn't written down, or because it's written in a way the search couldn't find. The job groups the similar ones (lots of people asking about parental leave in different words become one line) and posts HR a short, ranked list: "these are the topics staff asked about this week that the handbook doesn't cover well."

Crucially, the gap report doesn't edit the handbook. It can't — writing policy is a human's job, and a system that quietly invented a parental-leave rule because people kept asking would be the exact failure this whole series is built to avoid. The report just surfaces the gap. A person decides whether to add a section, and when they do, Path 1 picks it up. Over a few months, the handbook gets

measurably better at answering the questions people actually have, because the questions people actually have are now visible.

Every refresh is logged, every answer is traceable

Each of the three paths writes a row to a refresh log: which doc, which sections were touched, which path triggered it, and when. That log answers a question that comes up more than you'd think: "when did the answer to *this* change?" If someone says "the assistant told me 15 days last month and 12 today," you can look and see that the leave policy was edited on the 3rd and re-indexed at 9:14am, and the answer changed because the policy did — not because the assistant drifted. The index is never a black box; it's a mirror of the docs, with a timestamped history of every time the mirror was wiped clean.

Next post: the cost breakdown. The whole pipeline above — intake, search, answer, and keeping the index fresh — runs in coffee-money territory at SMB volume. Part 6 explains exactly where the dollars go.

PART 6 OF 7

MAY 27, 2026 PART 6 OF 7 · STAFF POLICY ANSWERER SERIES ~3 MIN READ

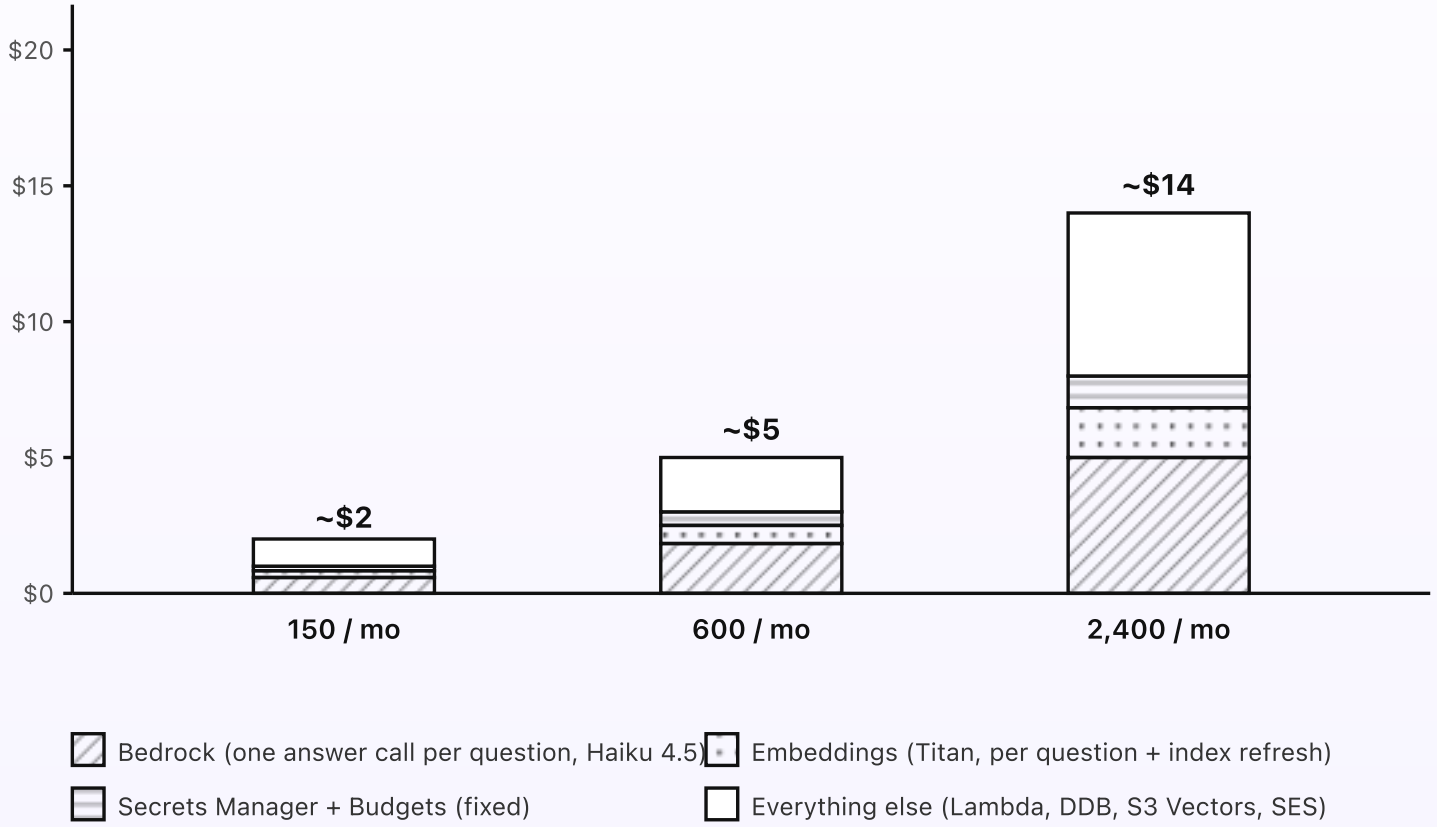
What the staff policy answerer costs

This is a cheap system to run. There's no always-on server, no database humming at 3am, no model call unless somebody actually asks a question. The cost scales with how many questions staff ask — not with how big your handbook is — because the handbook is read once and only re-read when it changes. At typical SMB volume, the bill is a couple of dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2/month at a small team's volume (roughly 150 questions a month).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway, no search server.
- The biggest variable cost is Bedrock — one small answer call per question.
- Keeping the index fresh is cheap: only changed sections get re-embedded.
- At ~600 questions/month the bill is around $5. At ~2,400 it's around $14.

Cost at three question volumes



Cost tracks the number of questions asked — a bigger handbook costs almost nothing extra.

Fig 6. Monthly cost at three question volumes. Bedrock is the largest slice because the answer call fires once per question, and questions are what grow. Embeddings are smaller, and the fixed cost is a rounding error.

Where the dollars actually go

Bedrock (the biggest slice). Each question that clears the confidence floor triggers one Claude Haiku 4.5 call: a few hundred tokens for the prompt and the pulled sections in, a couple of hundred tokens of answer out. That's a fraction of a cent per question. Questions that fail the confidence floor or hit an off-limits topic skip the model entirely and cost nothing. At 150 questions a month it's well under a dollar; at 2,400 it's the dominant line item but still single-digit dollars. Haiku is the right model here — reading a few short policy sections and writing a plain answer doesn't need a heavier model, and the cheaper call keeps the per-question cost tiny.

Embeddings (Titan). Two places. One small embedding per question (to turn it into a vector for search) — cents per thousand questions. And the index refresh: re-embedding only the handbook sections that changed. Since a typical handbook changes a few sections a week, the refresh embeddings are negligible. Embedding the whole handbook once on day one is a one-time cost of a few cents.

S3 Vectors. Stores the section fingerprints and answers search queries. You pay for the stored vectors (a small handbook is a few hundred to a few thousand vectors — pennies) and per query (one query per question). No always-on search cluster to pay for, which is the whole reason to use it over a hosted vector database. Pennies a month at these volumes.

Lambda runtime. The intake Lambda, the answerer, the indexer sync, and the gap-report job. All event- or schedule-driven, all small. Even at 2,400 questions a month the Lambda total lands under a dollar.

DynamoDB on-demand. Two small tables: the question-and-answer log and the refresh audit log. A handful of writes per question and per refresh. Pennies a month.

SES. Inbound for the email lane and outbound for email replies: $0.10 per thousand messages each way. Negligible unless your whole company is on the email lane.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the Slack endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. Nothing runs unless a question comes in or a doc changes.
- **A hosted vector database.** S3 Vectors means no search cluster sitting idle and billing by the hour.
- **Re-reading the whole handbook.** The index refresh touches only the sections that changed, so a big handbook isn't a big bill.

How the cost scales

The bill tracks questions, not handbook size or headcount directly. Bedrock and the per-question embedding grow linearly with how many questions get asked; everything else stays small. So a company asking 5,000 questions a month lands around $28, and 10,000 around $55 — still less than an hour of the HR time those questions would otherwise eat. A bigger handbook barely moves the needle: more

sections means slightly more stored vectors and a slightly larger search, both cheap. The thing that grows your bill is your team getting more answers, which is the point.

Set an AWS Budgets alarm at $20/month so anything unusual — a runaway loop, a misconfigured retry — pages you before the bill matters. The normal-volume bill stays well under that ceiling.

Last post in the series: the engineering reference. Same system, drawn for engineers — service names, Lambda inventory, IAM scopes, the S3 Vectors index config, Bedrock model IDs, the Slack app config, and the DynamoDB schemas.

PART 7 OF 7

MAY 27, 2026 PART 7 OF 7 · STAFF POLICY ANSWERER SERIES ~8 MIN READ

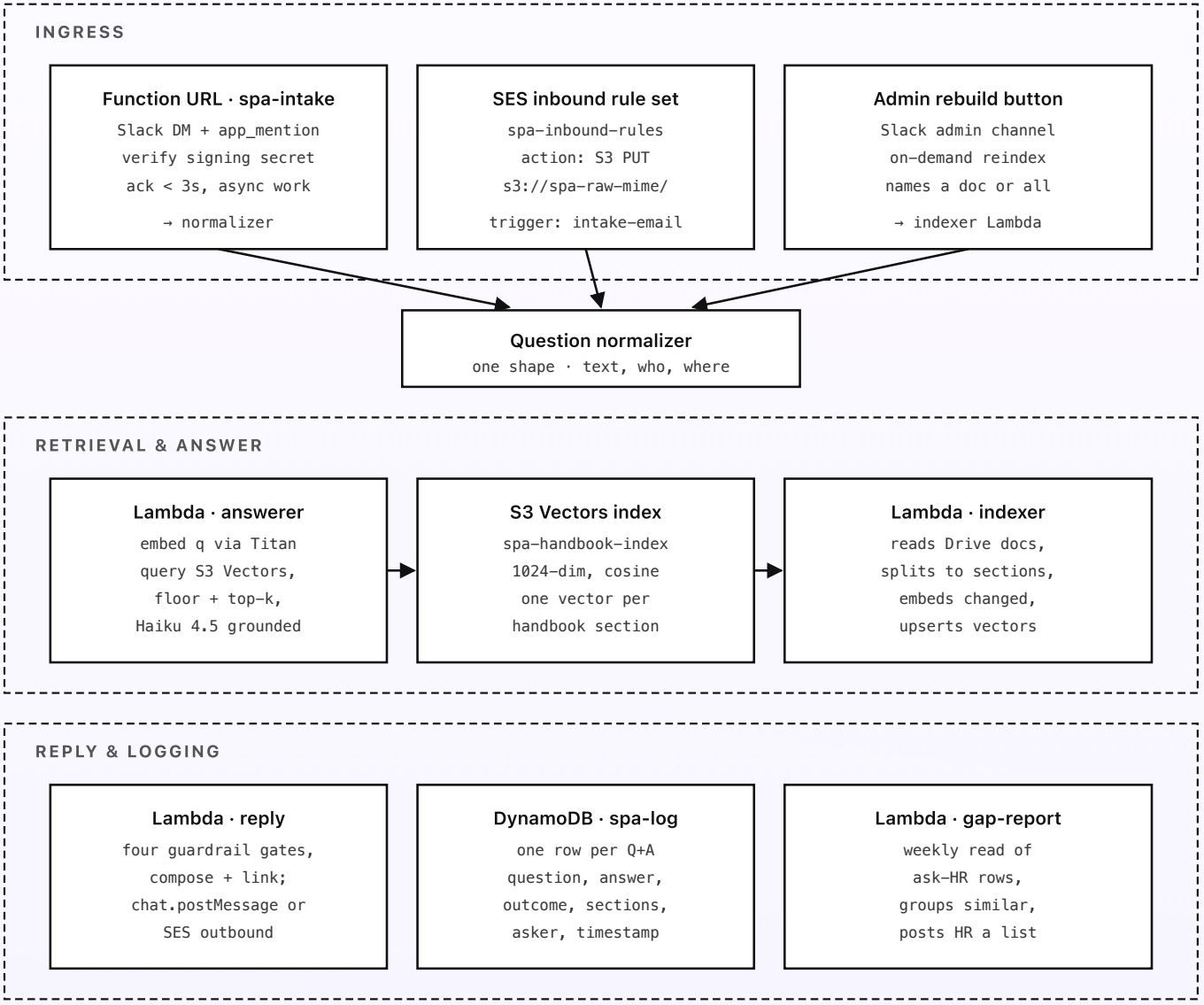
Engineering reference: the staff policy answerer architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, the S3 Vectors index config, Lambda inventory, IAM scopes, the Slack app config, the DynamoDB schemas, and the retrieval pipeline. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). Bedrock (Claude Haiku 4.5 via Global cross-Region inference, Titan Text Embeddings V2), S3 Vectors, Lambda Function URLs, and SES inbound are all available there. A second region for resilience isn't worth the setup at SMB volume — the failure mode is a staff member waiting a few minutes for HR instead of getting an instant answer, not a regional outage. One AWS account dedicated to the answerer (separate from other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system. All handbook content stays inside this account: the only data that leaves is the prompt-and-sections payload to Bedrock, which is not retained for training.

Topology



Every answer is grounded in a pulled section — every question logged to spa-log.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the normalizer), retrieval and answer (embed, search S3 Vectors, ground on Haiku, while the indexer keeps the index fresh), reply and logging (the answer ships, gets logged, and feeds the gap report). Every Lambda is event- or schedule-driven.

Lambda functions

All Lambdas use the `arm64` architecture, the smallest memory size that meets latency targets (typically 256 MB), Python 3.14 runtime, and CloudWatch Logs at 7-day retention. Each function has its own least-privilege IAM role. None run inside a VPC.

- **`spa-intake`** — Lambda Function URL, `AuthType: NONE`, verifies the Slack signing secret (`spa/slack/signing-secret`) on the raw request body before doing anything. Handles Slack URL verification, the `message` (IM) and `app_mention` events. Returns 200 within 3 s, then invokes `answerer` asynchronously with the normalized question. De-dupes on Slack's `X-Slack-Retry-Num` header so a slow downstream doesn't double-answer. Memory: 256 MB. Timeout: 10 s.
- **`intake-email`** — S3 PUT trigger on `s3://spa-raw-mime/`. Parses MIME, strips quoted history and signatures, extracts the question text and sender, and invokes `answerer` with `reply_channel=email`. Memory: 256 MB. Timeout: 30 s.
- **`answerer`** — invoked by the intake functions. Embeds the question with Titan Text Embeddings V2 (`amazon.titan-embed-text-v2:0`, 1024-dim), queries `spa-handbook-index` in S3 Vectors for top-k (k=8) nearest sections by cosine

similarity, applies the confidence floor (drop if top score < `SIM_FLOOR`, default 0.62), keeps the best 3–5, and calls Claude Haiku 4.5 (`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`) with a grounded, citation-required prompt. Returns a structured result (answer text, cited section ids, confidence, off_limits flag) and invokes `reply`. Sonnet 4.6 (`anthropic.claude-sonnet-4-6-...`) is wired as an optional escalation for multi-part or cross-policy questions where Haiku declines, gated behind a flag and off by default. Memory: 512 MB. Timeout: 60 s.

- **`reply`** — invoked by `answerer`. Runs the four guardrail gates (topic check against `spa-rules`, citation trace-back, hedge downgrade, compose), formats per the voice template, attaches the deep section link, and ships via Slack `chat.postMessage` (bot token `spa/slack/bot-token`) or SES `SendRawEmail`. Writes a row to `spa-log`. Memory: 256 MB. Timeout: 30 s.
- **`indexer`** — EventBridge Scheduler target every 5 minutes, plus on-demand from the admin rebuild button. Uses the Google Drive + Docs API (service-account credentials in Secrets Manager under `spa/drive/sa`) to detect changed docs via the revision marker, exports each changed doc to text, splits on headings into sections (with a soft 1,200-token cap and overlap), computes a content hash per section, re-embeds only sections whose hash changed via Titan, and upserts them into `spa-handbook-index` (deleting vectors for removed sections). Writes a row to `spa-audit`. Memory: 1024 MB. Timeout: 120 s.
- **`gap-report`** — EventBridge Scheduler target, weekly Monday 9am in `TZ_NAME`. Scans `spa-log` for the past week's `outcome=ask_hr` rows, clusters them (embed each question, group by cosine proximity), and posts HR a ranked

list of uncovered topics to the admin Slack channel. No model needed beyond the embeddings already in `spa-log`. Memory: 512 MB. Timeout: 60 s.

Storage

- **S3 Vectors** · `spa-handbook-index` — one vector per handbook section. 1024-dim (Titan V2), cosine distance. Metadata per vector: `doc_id`, `section_id`, `heading`, `deep_link`, `content_hash`, `updated_at`. Queried with a metadata filter to scope by doc when needed.
- **DynamoDB** · `spa-log` — one row per question. PK `(asker_id, ts)`; attributes: `question`, `outcome` (answered/ask_hr/off_limits), `cited_sections`, `sim_top`, `reply_channel`, `q_embedding` (reused by gap-report). On-demand. TTL 400 days on the raw question text; aggregates kept longer.
- **DynamoDB** · `spa-audit` — one row per index refresh. PK `(doc_id, ts)`; attributes: `sections_touched`, `trigger` (sync/manual), `by_user` (if manual). On-demand. No TTL — long-term freshness trail.
- **S3** · `spa-handbook-source` — mirrored plain-text export of each handbook doc, keyed by `doc_id/revision`. Versioning enabled; this is what the deep link and the citation check resolve against.
- **S3** · `spa-rules-source` — mirrored rules and voice docs as plain text (off-limits list, escalation contacts, tone). Versioning enabled.
- **S3** · `spa-raw-mime` — raw inbound MIME from the email lane. Lifecycle to Glacier at 30 days; expiry at 1 year.

Bedrock

- **Answer model.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0`. One callsite: `answerer`. Prompt is grounded (sections only), citation-required, with an explicit instruction to decline rather than use outside knowledge. `temperature: 0` for deterministic answers.
- **Escalation model.** `anthropic.claude-sonnet-4-6-...` via its Global profile, behind `ESCALATE_TO_SONNET` (default off). Only fires on multi-part questions Haiku declines and the search still has strong sections — the rare case where the reasoning, not the retrieval, is the bottleneck.
- **Embeddings.** `amazon.titan-embed-text-v2:0`, 1024-dim, normalized. Used by `answerer` (query embedding), `indexer` (section embeddings), and `gap-report` (clustering). Embedding dim must match the index dim exactly.
- **Quotas.** Default account quotas are plenty at SMB volume. The hot path is one Haiku call plus one Titan embedding per question.

EventBridge Scheduler config

- **`spa-index-sync`** — `rate(5 minutes)`. Target: `indexer` Lambda.
- **`spa-gap-report`** — `cron(0 9 ? * MON *)` in the SMB's timezone. Target: `gap-report` Lambda.
- **Manual reindex** — not a Scheduler rule; the admin rebuild button invokes `indexer` directly via the Function URL backing the Slack admin action.

Slack app config

The Slack app needs `chat:write`, `im:write`, `im:history`, and `app_mentions:read`. Event subscriptions point at the `spa-intake` Function URL: `message.im` and `app_mention`. Interactivity (the admin rebuild button and the ask-HR footer actions) also points at `spa-intake`, which routes admin actions to the `indexer`. The bot token lives in Secrets Manager under `spa/slack/bot-token`; the signing secret under `spa/slack/signing-secret`. The admin channel id and the per-topic HR contacts live in Parameter Store under `/spa/config/`.

SES inbound and outbound

- Set the MX record on a dedicated subdomain (e.g. `policy.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `spa-inbound-rules`: one rule with recipient `policy@your-company.com` → spam scan → S3 PUT to `s3://spa-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-email`.
- SES outbound for email replies: verify a sender identity at `policy@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request.

IAM (least privilege per Lambda)

Each Lambda has its own role with policies scoped to exact ARNs. Sketch:

- **spa-intake role:** `secretsmanager:GetSecretValue` on the Slack signing secret; `lambda:InvokeFunction` on `answerer` and `indexer`. *No* Bedrock, *no*

DynamoDB.

- **answerer role:** `bedrock:InvokeModel` on the Titan and Haiku ARNs (and the Sonnet ARN if escalation is enabled); `s3vectors:QueryVectors` on `spa-handbook-index`; `lambda:InvokeFunction` on `reply`; `s3:GetObject` on `spa-rules-source`.
- **reply role:** `secretsmanager:GetSecretValue` on the Slack bot token; `ses:SendRawEmail` from the verified identity; `dynamodb:PutItem` on `spa-log`; `s3:GetObject` on `spa-handbook-source` (deep-link + citation resolve); outbound network to `slack.com`.
- **indexer role:** `secretsmanager:GetSecretValue` on `spa/drive/sa`; `bedrock:InvokeModel` on the Titan ARN; `s3vectors:PutVectors` + `DeleteVectors` on `spa-handbook-index`; `s3:PutObject` on `spa-handbook-source` and `spa-rules-source`; `dynamodb:PutItem` on `spa-audit`; outbound network to `www.googleapis.com`.
- **gap-report role:** `dynamodb:Query` on `spa-log`; `secretsmanager:GetSecretValue` on the Slack bot token; outbound network to `slack.com`.

Retrieval and grounding details

Chunking is heading-aware: each section is a heading plus its body, capped near 1,200 tokens with a small overlap so a rule that spans a page boundary isn't split mid-sentence. The `content_hash` per section is what makes the sync incremental — unchanged hashes are skipped, so a one-line edit re-embeds one section, not the whole doc. The query keeps k=8 from S3 Vectors, applies `SIM_FLOOR`, then trims to the top 3–5 by score before the Haiku call. The

grounded prompt requires the model to return JSON: `{answer, cited_section_ids, declined}`. The citation gate rejects any `cited_section_id` not in the pulled set; the hedge gate downgrades when `sim_top < SIM_SOFT` (default 0.70) or the answer contains hedge markers. Every threshold (`SIM_FLOOR`, `SIM_SOFT`, top-k, top-n) lives in Parameter Store so tuning needs no deploy.

Observability and cost gates

- **CloudWatch Logs:** all Lambdas, 7-day retention, structured JSON. Subscription filter on `"error"` + `"throttle"` + `"timeout"` to a CloudWatch metric for alerting.
- **Alarms:** `answerer` error rate > 1% in 24h; `spa-intake` signature-verification failures > 5/hour (might mean the Slack secret rotated); `indexer` failures > 0 in a day (a stale index is a silent correctness bug); ask-HR rate spike > 2× baseline (might mean the handbook export broke).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $20/month threshold, alarm at 80% and 100%, posts to SNS topic `spa-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for the Drive and Docs APIs live in Secrets Manager under `spa/drive/sa`. Slack bot token and signing secret under `spa/slack/*`. SES sender identity lives in IAM and the verified-domain config. The off-limits topic list, per-topic HR contacts, timezone, retrieval thresholds, and the escalation

flag all live in Parameter Store under `/spa/config/`. Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC into a deploy role (no long-lived keys) and AWS SAM for the stack. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning for `spa-handbook-source` and `spa-rules-source` so a bad Drive export can be rolled back, and keep the S3 Vectors index dimension pinned to 1024 to match Titan V2 — a dimension mismatch is a silent retrieval failure. Total deployable surface: six Lambdas, one S3 Vectors index, two DynamoDB tables, three S3 buckets, two EventBridge Scheduler rules, one SES rule set, and one Budgets alarm.

That's the full system. Six narrative posts and this engineering reference. If you want to talk about adapting it for your business, see Work with me.