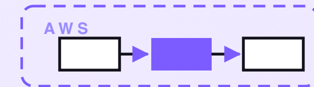
7-PART SERIES · FREE COMPANION



FAQ builder

A serverless builder that turns the questions customers actually ask into a self-updating FAQ. It reads your support email and chat, groups the repeat questions, drafts a clear answer from your own help docs that cites its source, and proposes new or updated FAQ entries for a human to approve before anything publishes. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

BUILD IT FOR REAL

Workflow guide $19 · Deployable AWS CDK starter $79 · Bundle $89

Free lite starter + this PDF · paid tiers at

shop.allanninal.dev/w/faq-builder

CONTENTS

FAQ builder

01 A self-updating FAQ builder on AWS for a few dollars a month

02 How the question pile gets built

03 How repeat questions get grouped

04 How the FAQ answer gets drafted

05 How a FAQ entry gets approved

06 What the FAQ builder costs

07 Engineering reference: the FAQ builder architecture

PART 1 OF 7

MAY 11, 2026 PART 1 OF 7 · FAQ BUILDER SERIES ~5 MIN READ

A self-updating FAQ builder on AWS for a few dollars a month

Every small business answers the same questions over and over. “Do you ship to Canada?” “How do I reset my password?” “What’s your refund window?” “Can I change my plan mid-month?” The answers live in someone’s head, or in a help doc nobody links to, or in a hundred past replies that all said roughly the same thing. Your FAQ page, meanwhile, was last touched a year ago. This post walks through the design of a small builder that reads the questions people actually ask, spots the ones that keep coming back, drafts a clear answer from your own help docs, and hands you a proposed FAQ entry to approve before anything goes live.

KEY TAKEAWAYS

- Three sources of questions: a support-inbox lane, a chat-export lane, and a manual lane.
- Questions that keep repeating get grouped into clusters; a cluster that crosses a threshold earns a FAQ entry.
- Answers are drafted from your own help docs and must cite a source — no made-up facts.
- Nothing publishes on its own. Every entry is approved, edited, or rejected by a person first.
- 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.

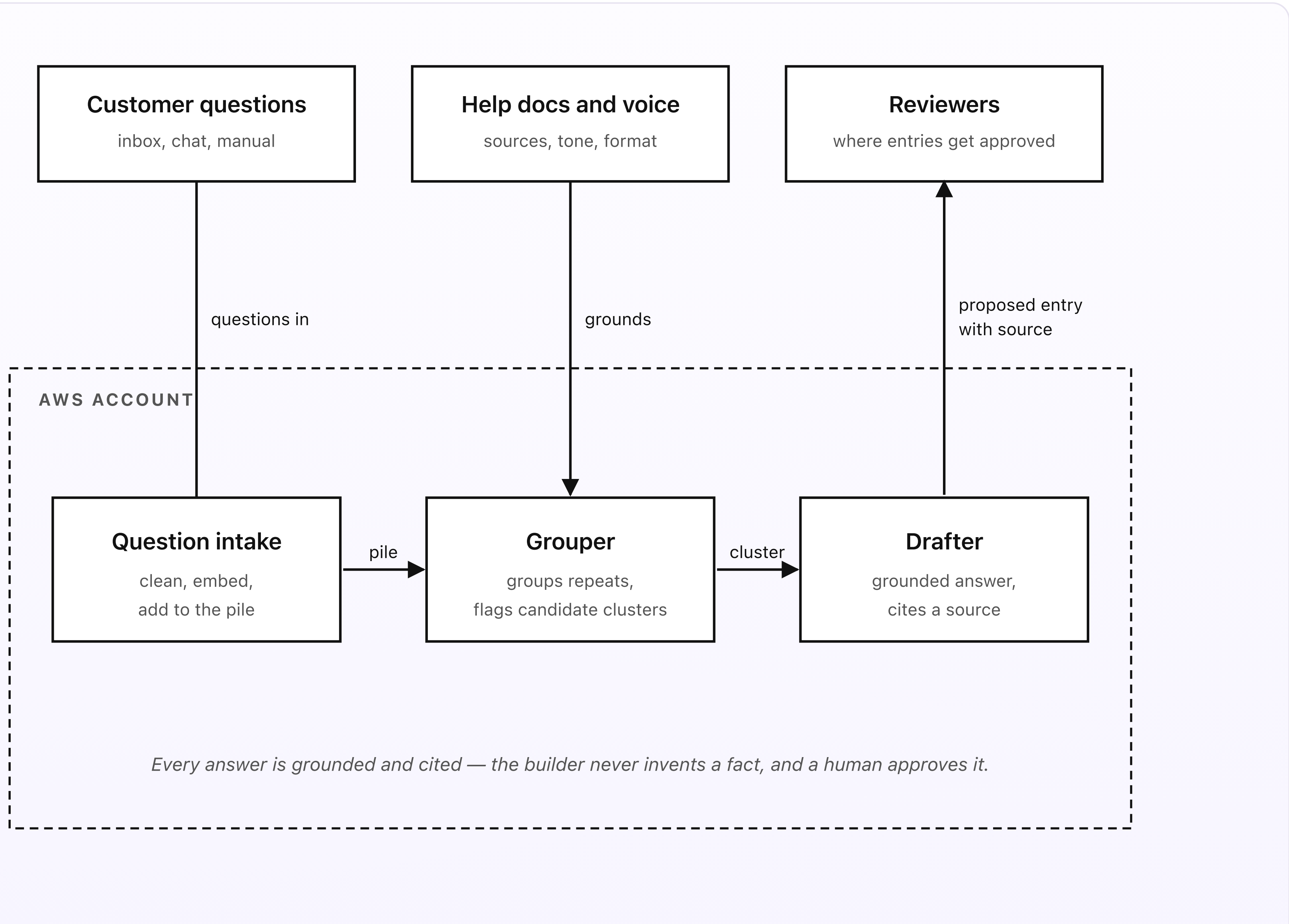

Customer questions
inbox, chat, manual
Help docs and voice
sources, tone, format
Reviewers
where entries get approved
questions in
grounds
proposed entry
with source
AWS ACCOUNT
Question intake
clean, embed,
add to the pile
pile
Grouper
groups repeats,
flags candidate clusters
cluster
Drafter
grounded answer,
cites a source
Every answer is grounded and cited — the builder never invents a fact, and a human approves it.

Fig 1. Three sources outside, three pieces inside AWS. Questions flow in from a support inbox, a chat export, and a manual lane. The Grouper runs daily and finds the repeats. The Drafter writes a grounded answer for each cluster that earns one, and a reviewer approves it.

What you set up once (the outside)

- **Customer questions.** The raw material. Three lanes feed the pile, covered in Part 2: a support-inbox forwarding lane (forward or auto-route support email to a dedicated address and the builder reads the question out of it), a chat-export lane (drop your chat transcripts in a Drive folder and the builder picks them up), and a manual lane (a rep types in a question they keep getting asked). You don't have to wire up all three; one is enough to start.
- **A help-docs folder.** Your existing help content in a Drive folder — setup guides, policy pages, product notes, whatever you already wrote. The builder reads these to ground every answer, so the FAQ says what your docs say, not what a model guessed. A short *voice* doc sits alongside them: the tone and length the FAQ should follow ("friendly, two or three sentences, no jargon") plus any phrases to avoid.
- **Reviewers.** The people who approve entries. Each proposed FAQ entry lands in a review queue — a Slack message or a simple web list — with the question, the drafted answer, the source passage it was grounded in, and three buttons: approve, edit, reject. Nothing reaches the live FAQ doc until somebody taps one.

What runs each day (the inside)

- **The question intake.** As questions arrive, a small Lambda cleans each one (strips signatures, greetings, and order numbers so the question stands on its own), turns it into a vector with Titan Text Embeddings V2 (a vector is just a list of numbers that places similar questions near each other), and writes it to the question pile. A vector means the grouper can tell “do you ship to Canada?” and “can you deliver to Toronto?” are the same question without any exact word match.
- **The grouper.** Runs once a day. Reads the new questions, compares each against the existing question vectors, and groups the near-duplicates into clusters. A cluster that crosses a repeat threshold — say, asked five times this month and not already covered by an approved FAQ entry — becomes a candidate. The grouping itself is plain Python over the vectors; no model writes anything here.
- **The drafter.** For each candidate cluster, a Lambda pulls the help-doc passages that best match the question, then asks Claude Haiku 4.5 for a short answer using only those passages, with an instruction to cite the source passage and to say “not covered in the docs” rather than guess. The proposed entry — question, answer, source link — goes to the review queue. A weekly digest lists what published, what’s waiting, and which clusters had no grounded answer.

In plain words

Over two weeks, eleven different customers email some version of “can I switch from monthly to annual without losing my current discount?” The intake cleans and embeds each one as it arrives. On the daily pass, the grouper notices they’re all the same question and the cluster crosses the threshold. The drafter pulls the

two paragraphs from your billing help doc that cover plan changes and proration, and writes: “Yes — you can switch to annual at any time. Your current discount carries over, and we prorate the difference on your next invoice. *(Source: Billing & Plans → Changing your plan.)*” That proposal lands in your reviewer’s Slack. They read it, see it matches the doc, and tap Approve. The entry is written to the live FAQ doc, and the next customer who searches your help center finds the answer without emailing anyone.

The cost of running this is about $3 a month at SMB volume. The cost of *not* running it is the same question answered by hand a hundred more times, a help center that quietly goes stale, and the support queue that never quite gets shorter.

DESIGN RULES THAT SHAPED EVERY DECISION

- Answers are grounded in your own docs and cite a source. The builder never speaks for facts it can't point to.
- If nothing grounds an answer, the builder says so — it does not fill the gap with a guess.
- Nothing publishes on its own. A person approves, edits, or rejects every entry.
- Repeats earn an entry; one-offs don't. The threshold keeps the FAQ about what people actually ask.
- The FAQ and help docs live in Drive. Editing an answer doesn't need a deploy.
- Every action is logged. Audit any entry later and you can see who approved it and from which source.

Why this shape

Most teams keep a FAQ in one of three states: missing, stale, or guessed-at. The missing one sends every question to a human. The stale one answers questions nobody asks anymore and misses the ones they do. The guessed-at one — the tempting shortcut of pointing a chatbot at your site and letting it answer freely — is the most dangerous, because it will confidently tell a customer something that isn't true, and you won't find out until they act on it.

The setup above keeps the source of truth in docs your team already edits, but adds a small system that *watches what people ask*, *only writes answers it can ground*, and *never publishes without a human*. The FAQ stays current because it's fed by real questions. It stays correct because every answer points at a source. And it stays yours because a person signs off on every word before it goes live.

The next four posts walk through each piece in turn: how the question pile gets built, how repeat questions get grouped, how the answer gets drafted and grounded, and how an entry gets approved and published. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 11, 2026 PART 2 OF 7 · FAQ BUILDER SERIES ~4 MIN READ

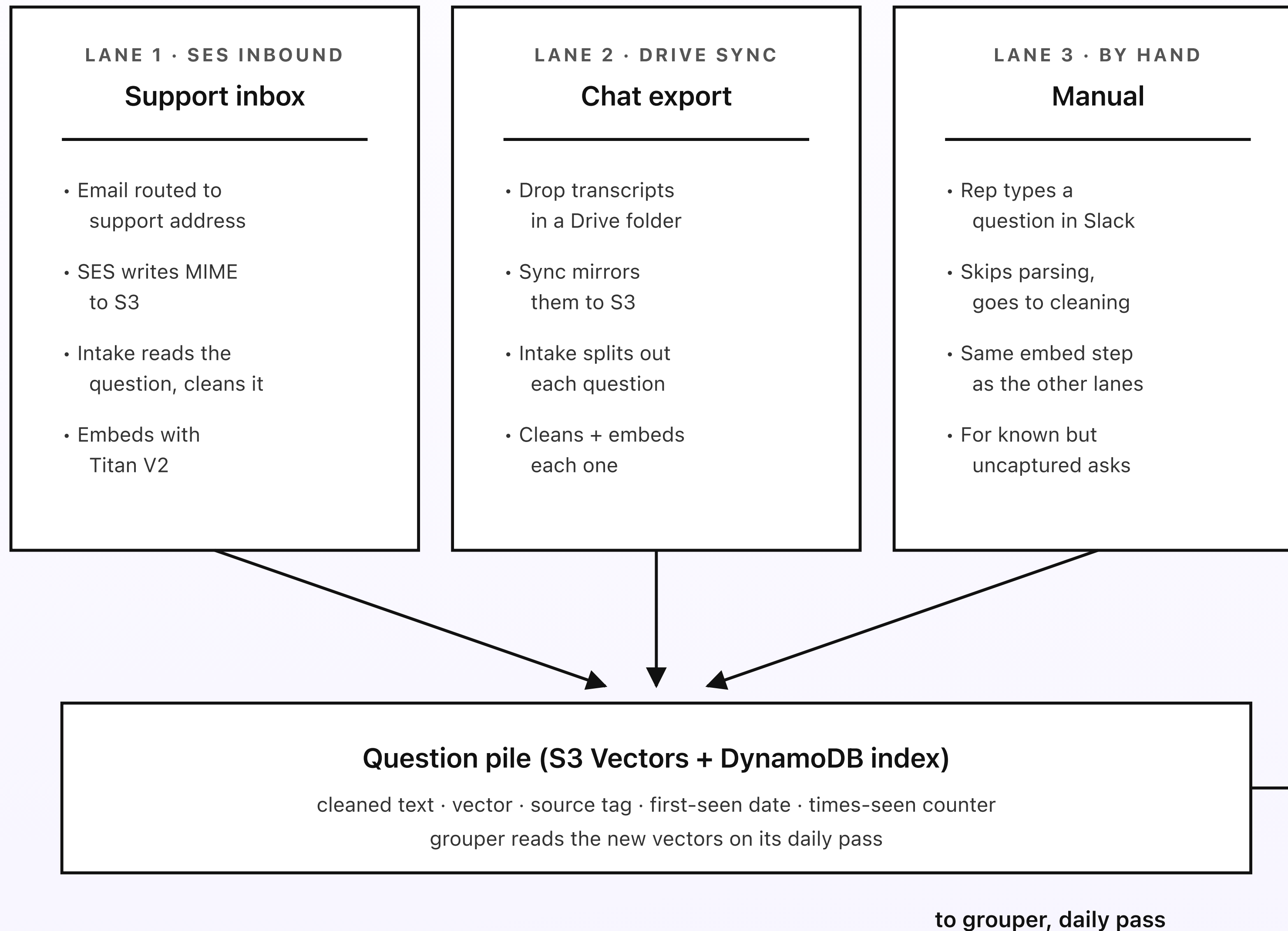
How the question pile gets built

The builder can only find repeats in questions it has seen. So the first job is feeding it the questions your customers actually ask — not the ones you think they ask. There are three ways a question gets into the pile: support email forwarded to a dedicated address, chat transcripts dropped in a Drive folder, or a rep typing in a question by hand. The first two are where most of the volume comes from. The third is for the question a rep knows is common but hasn't been captured yet.

KEY TAKEAWAYS

- Three intake lanes feed one pile: a support inbox, a chat export, and a manual lane.
- Each question is cleaned (signatures, greetings, and order numbers stripped) so it stands on its own.
- The cleaned question is embedded with Titan Text Embeddings V2 and written to S3 Vectors.
- Personal details are removed before anything is stored — the pile holds questions, not customer data.
- The pile is the one place the grouper looks. The lanes are just ways of filling it.

Three lanes into one pile



Personal details are stripped before anything is stored — the pile holds questions, not customer data.

Fig 2. Three lanes converge on one question pile. The pile holds the cleaned question text and its vector; the inbox, chat, and manual lanes are just three ways of filling it. The grouper reads the new vectors once a day.

Lane 1: the support inbox (most of the volume)

Set up a dedicated inbound address — something like `questions@your-company.com` — via Amazon SES, and forward or auto-route your support email there. (If you already run support out of a shared inbox, a single forwarding rule does it.) SES writes the raw MIME to `s3://fb-raw-mime/`. The S3 PUT triggers an intake Lambda that walks the MIME to the message body and pulls out the actual question.

Most support emails are mostly noise: a greeting, the question, a thank-you, a signature, a legal footer. The intake strips all of that down to the question itself, and removes anything personal — names, order numbers, account IDs, email addresses — because the pile is about *what* people ask, not *who* asked. The cleaned question is then embedded with Titan Text Embeddings V2 and written to the pile. If a single email contains two unrelated questions, the intake splits them so each can be grouped on its own.

Lane 2: chat export

If you run a live chat or a help widget, you already have a record of what people type in. Most chat tools can export transcripts on a schedule or let you drop them in a folder. Point that at a Drive folder the builder watches. A `drive-sync` Lambda mirrors the folder to S3 every fifteen minutes; new transcripts trigger the same intake Lambda.

Chat is messier than email — a transcript is a back-and-forth, not a single question — so the intake does a little more work here: it picks out the customer's turns, keeps the ones that are actually questions, and drops the small talk. Each surviving question is cleaned and embedded exactly like the inbox lane. From the pile's point of view, a question from chat and a question from email look identical; only the source tag differs.

Lane 3: manual entry

Sometimes a rep just knows. "Everyone asks whether the warranty covers water damage, and it's not written down anywhere." They don't need to wait for five emails to prove it. Lane 3 is a small Slack form (or a row in the Drive sheet) where a rep types the question directly. It skips the parsing step — there's no email or transcript to read — and goes straight to cleaning and embedding.

A manually entered question can be marked "priority," which tells the grouper in Part 3 to treat it as a candidate even if it hasn't been asked five times yet. That's the escape hatch for the obviously-common question that just hasn't shown up in the data yet.

Why everything funnels into one pile

Three lanes in, but only one place the grouper looks. That's deliberate. If chat questions and email questions lived in separate stores, "how often do people ask this?" would mean counting across two places and hoping the dedup worked. Funneling everything into one pile of cleaned, embedded questions means there is exactly one count per question, regardless of where it came from. The lanes are

first-class for getting questions in, but they always pass through the same cleaning and embedding on the way.

Next post: how the grouper reads the pile, finds the questions that keep repeating, and decides which clusters earn a FAQ entry.

PART 3 OF 7

MAY 11, 2026 PART 3 OF 7 · FAQ BUILDER SERIES ~5 MIN READ

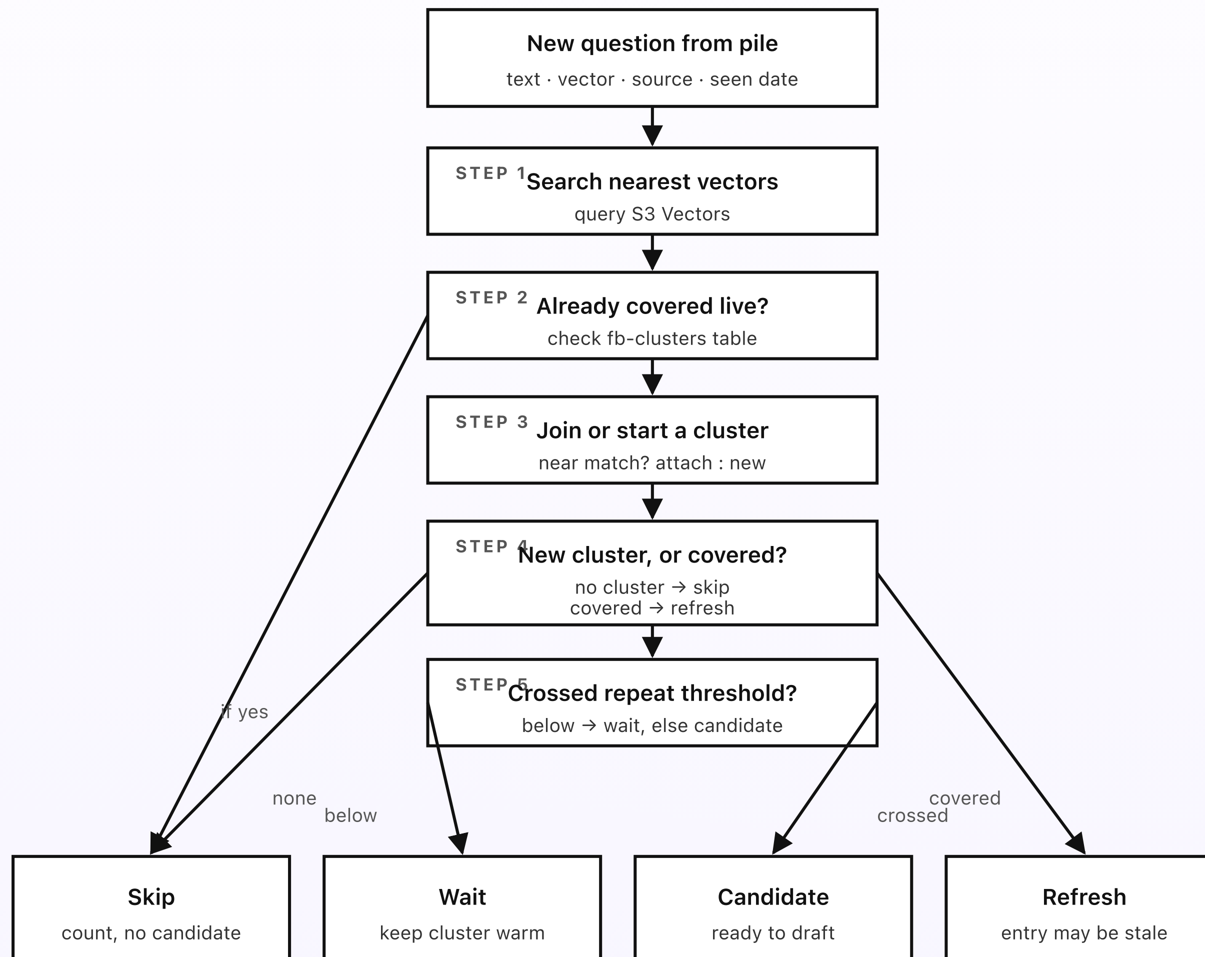
How repeat questions get grouped

Once a day, an EventBridge Scheduler rule fires the grouper Lambda. It reads the new questions added since the last pass, looks at one at a time, finds the questions already in the pile that mean the same thing, and either joins an existing group or starts a new one. Then it decides which groups have been asked enough to deserve a FAQ entry. The whole pass is plain Python over the vectors. No model writes anything here — the only AI involved is the embedding from Part 2.

KEY TAKEAWAYS

- The grouper runs once a day via EventBridge Scheduler.
- Each new question is matched against existing clusters by vector nearness — same meaning, not same words.
- A near-match joins a cluster; nothing close enough starts a new one.
- A cluster that crosses the repeat threshold (default five asks, not already covered) becomes a candidate.
- The grouper writes no answer — it only decides what is worth answering.

The grouping flow, per question



The threshold lives in the rules doc — change the minimum-asks and tomorrow's pass uses it.

Fig 3. The grouper's decision tree, per question, per daily pass. Five steps decide whether a question is skipped, kept warm, or turned into a candidate (or marks a live entry as needing a refresh). The rules doc holds the threshold; the grouper only enforces it.

Same meaning, not the same words

The whole reason questions are embedded in Part 2 is so the grouper can match them by meaning. "Do you ship to Canada?", "Can you deliver to Toronto?", and "Is international shipping available north of the border?" share almost no words, but they're the same question. Their vectors sit close together, so a nearest-neighbor search in S3 Vectors finds them as a group. A keyword match would put them in three different buckets and you'd never see that it's one popular question asked three ways.

The grouper queries S3 Vectors for the closest existing questions to each new one. If the closest is within the *join threshold* (a distance the rules doc sets, with a sensible default), the new question joins that cluster and bumps its times-seen counter. If nothing is close enough, the question starts a new single-question cluster of its own. Over days, popular questions accrete into big clusters and rare one-offs stay as singletons.

Four outcomes, every pass

Every new question, every pass, lands in exactly one of four buckets. The names are plain on purpose.

- **Skip.** The question is already covered by a live FAQ entry. Count it — knowing a covered question is still being asked is useful — but don't make a new candidate. Most questions, once the FAQ matures, land here.
- **Wait.** The question joined a cluster, but the cluster hasn't been asked enough times yet. Keep it warm. A cluster sitting at three asks this month is one good week away from becoming a candidate; the grouper remembers it so nothing has to start over.
- **Candidate.** The cluster just crossed the repeat threshold and isn't covered. Mark it ready for the drafter in Part 4. This is the whole point of the pass — turning "people keep asking this" into "let's answer it once, well."
- **Refresh.** The cluster *is* covered by a live entry, but the asks keep coming — which often means the published answer is unclear, incomplete, or out of date. Flag the entry for a refresh so a reviewer can decide whether the answer needs a rewrite.

The threshold is a number you own

How many asks make a FAQ entry? That's a judgment call, and it lives in the rules doc as a plain number — `min_asks_for_candidate`, default five. Set it lower if you want a thorough FAQ that captures the long tail; set it higher if you only want the genuine top questions. The grouper reads the number each pass, so changing it doesn't need a deploy. The priority flag from a manual entry (Part 2) is the one override: a rep can mark a question important and skip the counting entirely.

There's also a time window. The count that matters is "asked N times in the last 30 days," not "asked N times ever," so a question that was hot last year but nobody

asks now doesn’t keep nagging to be answered. The window length is configurable too.

Why the grouping uses no model

The grouper could ask a model “are these two questions the same?” on every pair. It doesn’t. Two reasons. First, the embedding already captured the meaning in Part 2 — a nearest-neighbor search over those vectors does the same job for a fraction of a cent, and it’s consistent: the same two questions always land the same distance apart. Second, a model in this loop would cost money on every question on every pass, most of which just join an obvious cluster or get skipped. The model earns its place in Part 4, drafting the answer for a cluster that’s already been judged worth answering — not here, sorting questions into bins.

Next post: how the drafter takes a candidate cluster, pulls the matching passages from your help docs, and writes a short answer that cites its source — or admits when it can’t.

PART 4 OF 7

MAY 11, 2026 PART 4 OF 7 · FAQ BUILDER SERIES ~5 MIN READ

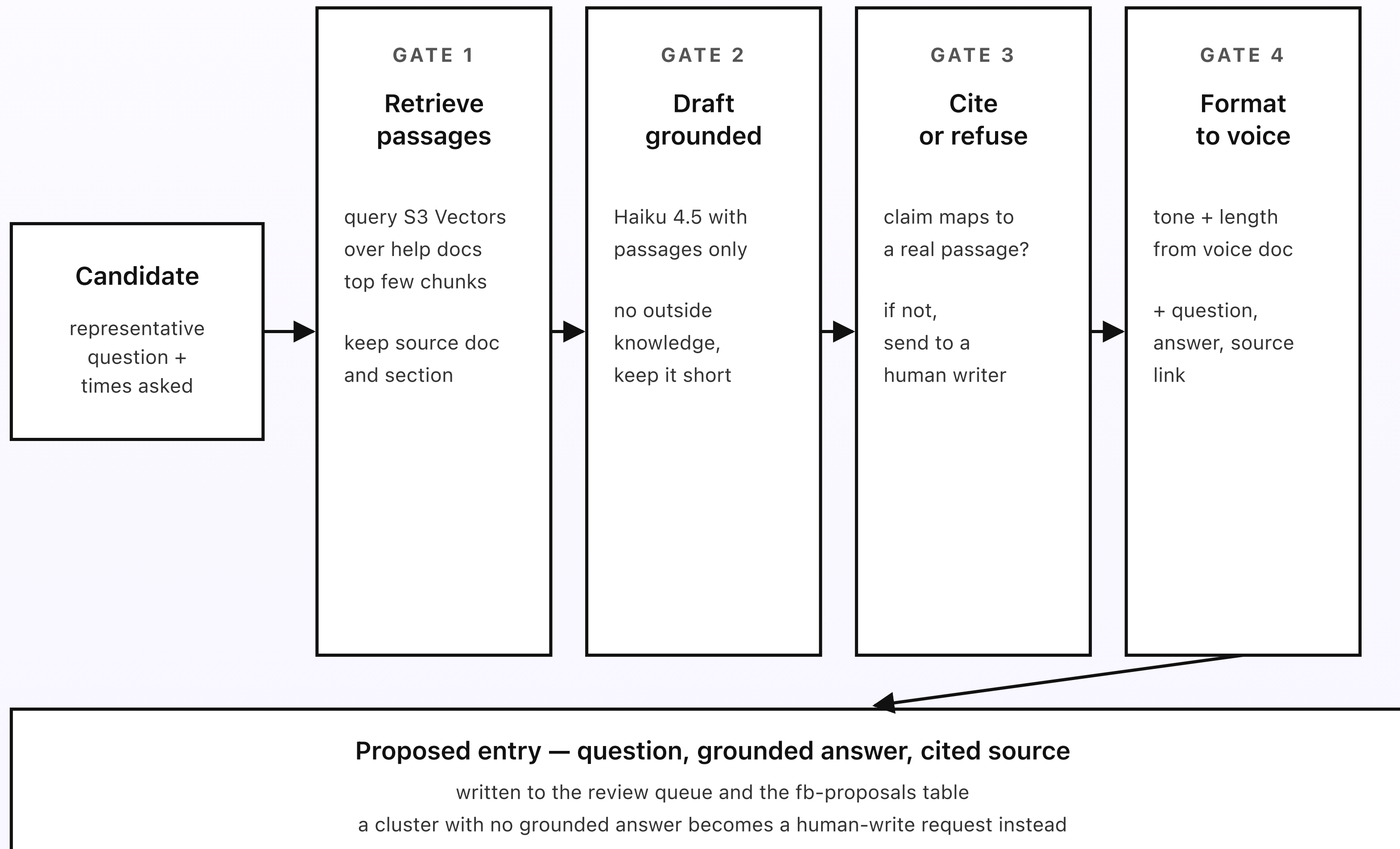
How the FAQ answer gets drafted

The grouper handed over a candidate cluster — a question people keep asking, not yet answered. Now the drafter has to write a clear answer. The temptation is to just ask a model the question and publish whatever comes back. That's how a FAQ ends up confidently wrong. Instead, the drafter pulls the relevant passages out of your own help docs, asks the model to answer using only those passages, makes it cite the source, and accepts "not covered" as a valid answer. Four small gates sit between the cluster and the proposed entry.

KEY TAKEAWAYS

- The drafter retrieves the help-doc passages closest to the question before any model runs.
- The model answers using only those passages — it is told not to use outside knowledge.
- Every claim must cite the source passage it came from; the citation rides along with the answer.
- If nothing grounds the answer, the drafter returns “not covered” instead of guessing.
- The answer is formatted to the voice doc — short, plain, in your house style — then proposed.

Four gates on every draft



A draft with no source never becomes a proposal — it becomes a request for a human to write one.

Fig 4. Four gates between a candidate cluster and the proposed entry. Retrieve the passages. Draft using only them. Check the citation or refuse. Format to your voice. Then write the proposal to the review queue — or, if nothing grounded it, ask a human to write the answer.

Gate 1: retrieve the passages

Before the model sees anything, the drafter goes and finds the parts of your help docs that bear on the question. Your help docs were chunked into short passages and embedded once, up front, and they live in S3 Vectors alongside the question vectors. The drafter embeds the cluster's representative question and queries for the closest passages — usually the top three to five. Each comes back with its text, the doc it's from, and the section heading, so a citation can point at exactly where the answer came from.

This is the step that makes the whole thing grounded. The model never gets to roam your entire site or its own training; it gets a handful of passages from your docs and is asked to work within them. If your docs don't cover the question, the retrieval comes back thin — and that's a signal, not a problem to paper over.

Gate 2: draft using only those passages

Now the model runs. The drafter calls Claude Haiku 4.5 with the question and the retrieved passages, and a system prompt that's blunt about the rules: "Answer the question using only the passages below. Do not use any outside knowledge. Keep it to two or three sentences. For each claim, name the passage it came from. If the passages don't answer the question, reply exactly 'NOT COVERED'." Haiku is the

cheap path and it's plenty for short, grounded answers; the heavier model isn't needed here because the thinking is "restate what the passage says, plainly," not open-ended reasoning.

Keeping the answer short isn't just style. A short answer is easier for a reviewer to check against the source, and a FAQ entry that runs three paragraphs usually means the question should have been split into two.

Gate 3: cite the source, or refuse

The model can still go off-script — claim something the passages don't support, or cite a passage that doesn't really say what the answer says. Gate 3 checks. It confirms the citation points at a passage that was actually retrieved, and that the cited passage plausibly supports the claim. If the model returned "NOT COVERED," or the citation doesn't map to a real passage, the cluster doesn't become a proposal at all. Instead it goes to a *human-write* queue: "people keep asking this and the docs don't answer it — someone should write the answer (and probably update the docs)."

That refusal path is the most important part of the whole system. A FAQ that quietly invents an answer for a gap in your docs is worse than no FAQ, because customers trust it. Sending the gap to a human keeps the published FAQ honest and surfaces the holes in your help docs as a useful byproduct.

Gate 4: format to your voice, then propose

A grounded, cited answer still has to sound like you. Gate 4 applies the voice doc: the tone (“friendly and direct”), the length cap, the formatting (a lead sentence then a short explanation), and any banned phrases (“simply,” “just,” anything that talks down to the reader). It then assembles the proposed entry — the question as a customer would phrase it, the answer, and a link to the source passage — and writes it to the review queue and the `fb-proposals` table. The source link stays attached all the way through, so the reviewer in Part 5 can click straight to the doc the answer came from.

For a cluster the grouper flagged as a *refresh* (an existing entry that keeps getting asked), the drafter does the same work but presents it as a diff: here’s the live answer, here’s the redrafted one, here’s what changed and why. The reviewer decides whether the update is an improvement.

Why grounding is the whole game

None of these gates are exotic. They’re the discipline a careful person would apply if they were writing the FAQ by hand: look up what your docs actually say, write only what you can back up, cite where it came from, and flag the questions your docs don’t answer instead of bluffing. Putting that discipline in code — retrieve, draft-from-passages, cite-or-refuse, format — makes it a property of the system, not something you’re hoping a model remembers to do on a given Tuesday.

Next post: how a proposed entry gets approved — the review queue, the three actions a reviewer can take, and how an approved entry reaches the live FAQ doc.

PART 5 OF 7

MAY 11, 2026 PART 5 OF 7 · FAQ BUILDER SERIES ~5 MIN READ

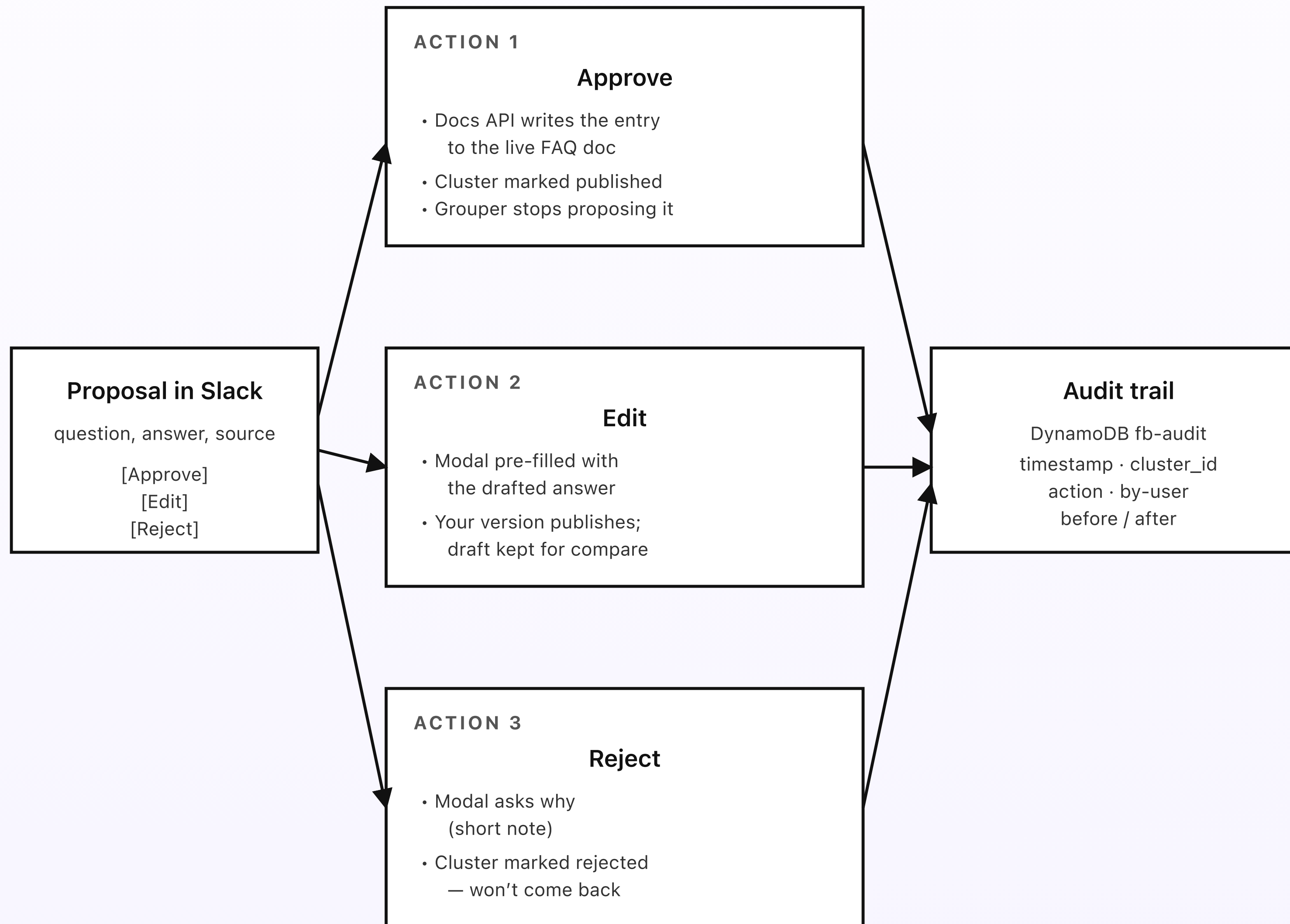
How a FAQ entry gets approved

A proposed entry lands in the reviewer's Slack at the end of the day's run. The question, a short grounded answer, a link to the source it came from, and three buttons. What happens when they tap one? This post walks through the three things a reviewer can do — approve, edit, reject — and how the live FAQ doc, the cluster state, and the audit trail all stay in sync. This is the gate that keeps a wrong answer from ever reaching a customer.

KEY TAKEAWAYS

- Three actions per proposal: *approve* (publish as drafted), *edit* (fix it, then publish), *reject* (drop it with a note).
- Approve and edit write the entry to the live FAQ doc via the Docs API and mark the cluster published.
- A published entry is added to the “already covered” set, so the grouper stops proposing it.
- Reject keeps the cluster from coming back unless the question changes meaningfully.
- The approve button is a Slack interactive message backed by a Function URL; every action is logged.

Three actions on a proposal



Nothing reaches the live FAQ without a human tap — approve and edit publish, reject does not.

Fig 5. Three actions per proposal, three different effects. Approve publishes the draft as-is. Edit publishes the reviewer's version. Reject drops it with a reason. Every action writes to the audit trail, and only approve and edit reach the live FAQ.

Action 1: approve (the common case)

The reviewer reads the proposal, clicks the source link, sees the answer matches the doc, and taps *Approve*. The tap submits to a Function URL Lambda. Three things happen, in order. First, the Docs API inserts the entry into the live FAQ doc — a new question-and-answer section, placed in the right category, with a small footnote linking the source passage. Second, the cluster is marked `published` in the `fb-clusters` table and added to the "already covered" set, so the grouper in Part 3 will *skip* the same question from now on instead of proposing it again. Third, an `action: approved` row is written to `fb-audit` with the user, the timestamp, and the published text.

From the customer's side, the next person who searches the help center for that question finds the answer waiting — no email, no wait. From your side, one more recurring question just stopped reaching the support queue.

Action 2: edit (the "almost right")

Most drafts are close but not perfect. The answer is correct but a little stiff, or it's missing a caveat the reviewer knows matters, or the phrasing could be warmer. The reviewer taps *Edit*. A Slack modal opens pre-filled with the drafted answer. They fix the wording and hit Save.

The Function URL Lambda publishes the reviewer's version to the FAQ doc, not the model's. It keeps the original draft alongside the published version in the audit row, so later you can see what the model proposed and what a human changed — useful for tuning the voice doc if the same kind of edit keeps happening. The cluster is marked published exactly like the approve path. Edit is the path that quietly trains the system: when reviewers keep softening the same phrase, that's a hint to add it to the voice doc's banned list so the next draft already avoids it.

Action 3: reject (the "not this")

Sometimes the proposal shouldn't exist. The question is a one-off the grouper over-counted. The answer touches something you'd rather not put in a public FAQ (pricing exceptions, a workaround for a bug you're fixing). The source the model grounded in is itself out of date. The reviewer taps *Reject* and a small modal asks why — a short note, picked from a few common reasons or typed freely.

On save, the Lambda drops the proposal, records the reason in the audit trail, and marks the cluster `rejected`. A rejected cluster won't come back as a candidate unless the underlying question changes meaningfully — the grouper checks whether new asks have drifted far enough from the rejected cluster's center to count as a genuinely different question. That keeps a reviewer from having to reject the same thing every week, while still letting a real new question through.

Every action is logged, every change is reversible

The `fb-audit` table records every approve, edit, and reject with the user who acted, the timestamp, and a snapshot of the entry before and after. If a wrong

answer gets published — a reviewer approved too fast, or a source turned out to be stale — a rep can run an “unpublish” through a small admin command that removes the entry from the FAQ doc and reopens the cluster. The unpublish is itself an audit row, so the trail stays clean.

This reversibility matters because a FAQ is a living document edited by different people over months. The next person to touch the warranty answer might be a teammate who wasn’t here when it was written. The audit trail — who approved it, from which source, and what it said before — is the only memory they have, and it’s the difference between confidently updating an entry and being afraid to touch it.

Next post: the cost breakdown. The whole pipeline above runs in coffee-money territory at SMB volume; Part 6 explains exactly where the dollars go and why the embeddings, not the drafting model, are the biggest line.

PART 6 OF 7

MAY 11, 2026 PART 6 OF 7 · FAQ BUILDER SERIES ~3 MIN READ

What the FAQ builder costs

The builder is one of the cheaper systems in this whole series. It embeds each incoming question, runs a once-a-day grouping pass that calls no model, and only asks the drafting model to write an answer for the handful of clusters that earn one. At typical SMB volume, the bill is a few dollars a month, fixed cost essentially zero. The interesting twist is which line dominates: it's the embeddings, not the drafting model.

KEY TAKEAWAYS

- Around $3/month at typical SMB volume (around 800 support messages a month).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- Embeddings are the biggest variable line — one per incoming question.
- The drafting model is a small sliver — it fires only on candidate clusters, a few a week.
- At light volume the bill is around $1. At heavy volume (5,000 messages) it's around $12.

Cost at three volumes

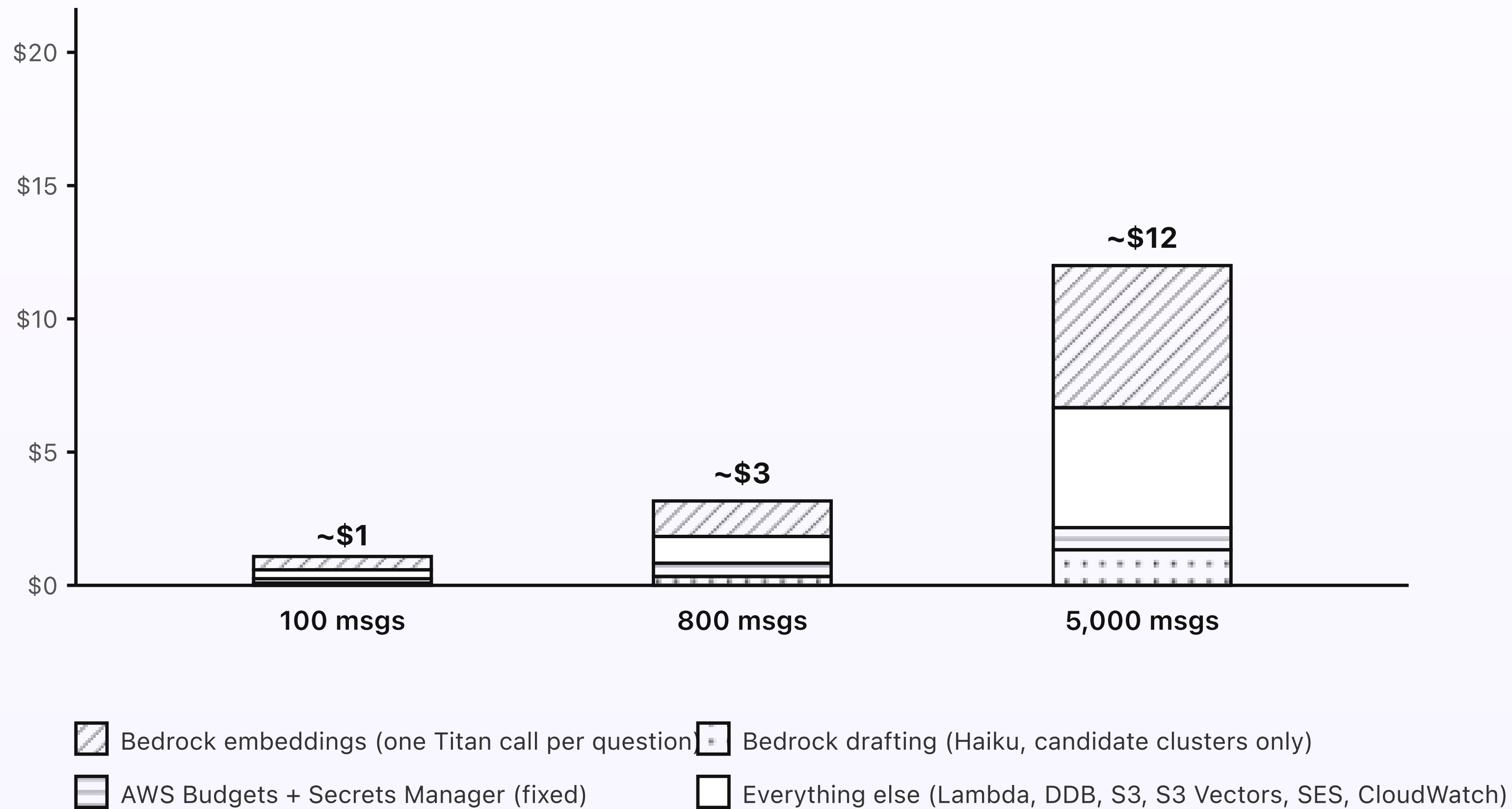


Fig 6. Monthly cost at three message volumes. The embeddings slice dominates because every incoming question is embedded once. The drafting model is small because it only fires on candidate clusters, a few a week.

Where the dollars actually go

Bedrock embeddings (the bulk). Every incoming question is embedded once with Titan Text Embeddings V2 so the grouper can find repeats. Titan embeddings are priced per token and a cleaned question is short, so each embed is a tiny fraction of a cent — but it's the one cost that scales directly with how many questions you get. At 800 messages a month it's the largest single line, and it's still well under a couple of dollars. Your help docs are embedded too, but that's a one-time cost paid up front and a small top-up whenever a doc changes.

Bedrock drafting. Claude Haiku 4.5 only runs on candidate clusters — questions that crossed the repeat threshold and aren't already covered. In steady state that's a few a week, not a few a day, because most questions either join a covered cluster or sit below the threshold. Each draft is a few thousand input tokens (the retrieved passages) and a few hundred output tokens (a short answer), so a fraction of a cent per draft. The drafting line stays a sliver at every volume.

S3 Vectors. The question vectors and the help-doc vectors live here. Storage is cheap and the per-query cost on the daily grouping pass and the drafter's retrieval is small. A few cents a month at SMB volume.

Lambda runtime. The intake fires per incoming question, the grouper runs once a day, the drafter runs per candidate, the `ack-handler` runs per reviewer action, and the `drive-sync` Lambda runs every fifteen minutes. All short. The Lambda total lands under a dollar at all three volumes.

DynamoDB on-demand. Three small tables: `fb-clusters`, `fb-proposals`, `fb-audit`. Reads and writes are dominated by the daily pass and the approval actions. Pennies a month.

SES + S3. Inbound for the support-inbox lane: $0.10 per thousand received messages, so a couple of cents a month for an SMB. S3 holds the mirrored docs and raw MIME — a few hundred KB, effectively free.

What doesn’t cost money

- **API Gateway.** Replaced by Lambda Function URLs for the approve and edit endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The builder only runs when there’s a question to embed or a pass to make.
- **A model on the grouping pass.** Grouping is plain Python over the vectors. The model only drafts answers.
- **A separate vector database.** S3 Vectors holds the vectors; there’s no always-on vector store to pay for.

How the cost scales

Embeddings grow linearly with message count, because every question is embedded once. The drafting line grows much more slowly, because the number of *new* questions worth answering tapers off as the FAQ matures — the more you cover, the more incoming questions land on “already covered” and skip the drafter entirely. So the bill at 10,000 messages a month is around $22; at 20,000 it’s around $42, still embeddings-dominated. Past those volumes you’d batch the embedding calls and maybe sample rather than embed every duplicate, but those are optimizations for high-traffic support desks — not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. The builder’s 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, DynamoDB schemas, the S3 Vectors config, the SES rule set, and the EventBridge Scheduler config.

PART 7 OF 7

MAY 11, 2026 PART 7 OF 7 · FAQ BUILDER SERIES ~8 MIN READ

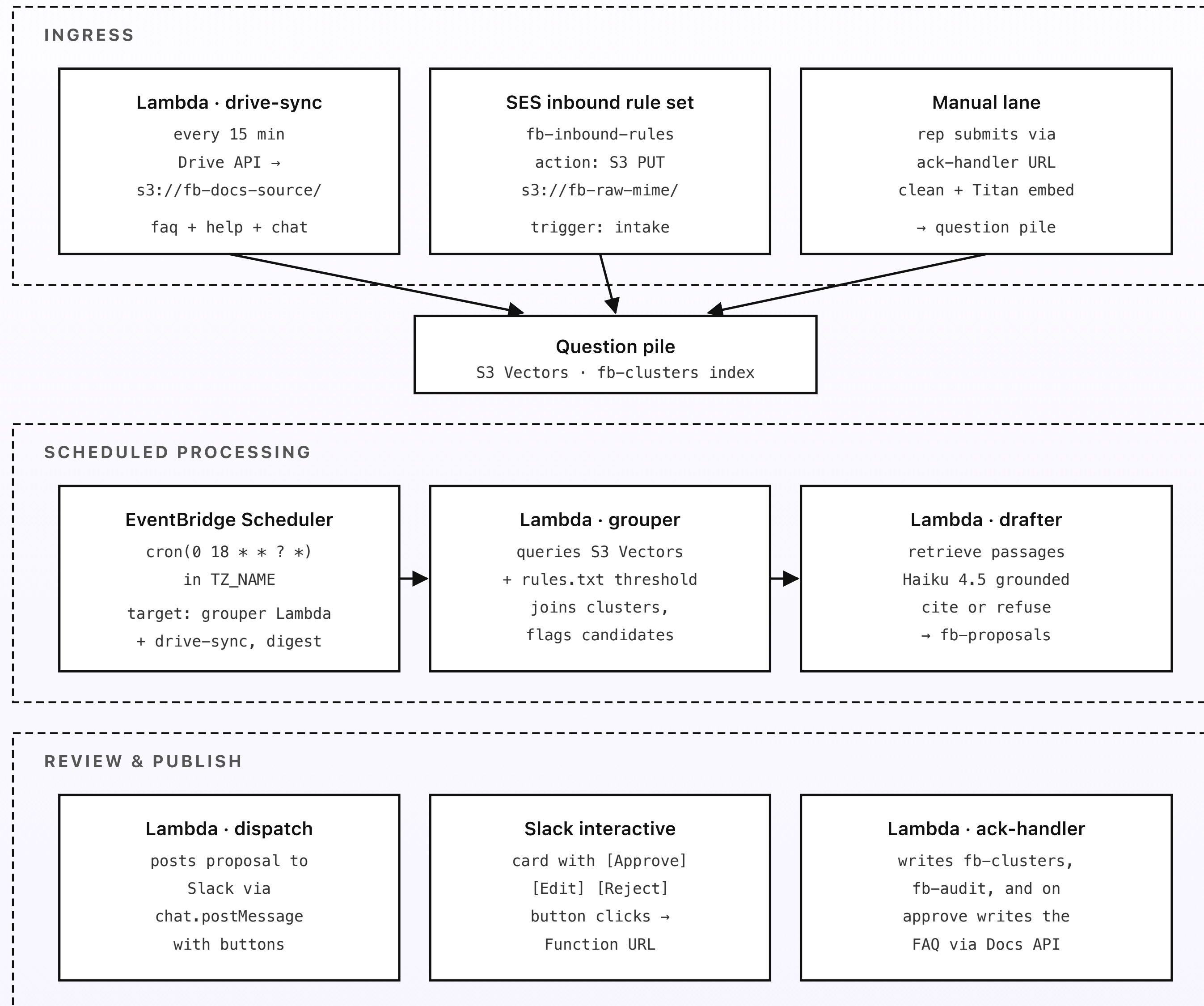
Engineering reference: the FAQ builder architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, Lambda inventory, IAM scopes, the SES inbound rule set, S3 Vectors config, EventBridge Scheduler config, the DynamoDB schemas, and the Slack interactive flow. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). SES inbound, S3 Vectors, Bedrock cross-Region inference (Titan Text Embeddings V2 and Claude Haiku 4.5), and EventBridge Scheduler are all available there. A second region for multi-region resilience isn't worth the extra setup work at SMB volume — the failure mode for an SMB is a delayed FAQ proposal, not a regional outage. One AWS account dedicated to the builder (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Every answer is grounded and cited — and every interaction is logged to fb-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the question pile), scheduled processing (the daily grouper flagging candidates and the drafter writing grounded answers), review and publish (the proposal ships to Slack and the reviewer's decision is recorded and published). Every Lambda is event- or schedule-driven; nothing is synchronous-chained.

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.

- **`drive-sync`** — EventBridge Scheduler target, fires every 15 minutes. Uses the Google Drive API + Docs API (service-account credentials in Secrets Manager under `fb/drive/sa`) to export the live FAQ doc, the help docs, and any new chat transcripts and write them to `s3://fb-docs-source/` only if changed since the last sync. The same pattern syncs the rules and voice docs to `s3://fb-rules-source/`. On a help-doc change, it enqueues the changed doc for re-chunking and re-embedding. Memory: 256 MB. Timeout: 30 s.
- **`intake`** — S3 PUT trigger on `s3://fb-raw-mime/` (support email) and on new transcripts under `s3://fb-docs-source/chat/`. Parses MIME or transcript, extracts the customer's question(s), strips signatures/greetings and PII (regex plus a small allowlist), then calls Bedrock Titan Text Embeddings V2 (`amazon.titan-embed-text-v2:0`, 1024-dim) to embed each cleaned question and writes the vector to the `fb-questions` S3 Vectors index with metadata in `fb-clusters`. The manual lane reaches the same cleaning/embedding code path via `ack-handler`. Memory: 512 MB. Timeout: 60 s.

- **`doc-embed`** — triggered by `drive-sync` when a help doc changes. Chunks the doc into short passages (heading-aware, ~500 tokens), embeds each with Titan V2, and upserts to the `fb-helpdocs` S3 Vectors index keyed by `(doc_id, chunk_id)` with the source section as metadata. Old chunks for that doc are deleted first so retrieval never returns stale passages. Memory: 512 MB. Timeout: 120 s.
- **`grouper`** — EventBridge Scheduler target, daily at 6pm local (the schedule runs in `TZ_NAME`, e.g. `Asia/Singapore`). Reads the questions added since the last pass, queries `fb-questions` S3 Vectors for nearest neighbors, joins or starts clusters by the `join_threshold` distance, increments times-seen counters over the configured window, and applies `min_asks_for_candidate` from the rules doc. Writes cluster state to `fb-clusters` and invokes `drafter` per candidate. *No Bedrock calls* — the embedding already happened at intake. Memory: 512 MB. Timeout: 120 s.
- **`drafter`** — invoked by `grouper` per candidate (or refresh). Embeds the cluster's representative question, queries `fb-helpdocs` S3 Vectors for the top *k* passages, calls Bedrock 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 system prompt (answer from passages only; cite source; return `NOT COVERED` otherwise). Validates the citation maps to a retrieved passage; on failure routes the cluster to the human-write queue. Formats to the voice doc and writes the proposal to `fb-proposals`. For ambiguous, multi-part clusters where Haiku's grounding is weak, it can escalate a single retry to Claude Sonnet 4.6 (`global.anthropic.claude-sonnet-4-6-20250930-v1:0`) — justified only because that retry is rare and a wrong public answer is expensive. Memory: 512 MB. Timeout: 60 s.

- **`dispatch`** — triggered when a proposal is written. Posts the proposal to the configured Slack channel via `chat.postMessage` with Block Kit Approve/Edit/Reject buttons and the source link. Writes nothing to the FAQ; it only surfaces the proposal for review. Memory: 256 MB. Timeout: 30 s.
- **`ack-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a Slack signature on the request body. Triggered by Slack button clicks (Approve/Edit/Reject), by the Edit modal submission, and by the manual-lane question form. On approve or edit, writes the entry to the live FAQ Google Doc via the Docs API, marks the cluster published in `fb-clusters`, and adds it to the covered set. On reject, records the reason and marks the cluster rejected. Every action writes to `fb-audit`. Memory: 256 MB. Timeout: 15 s.
- **`digest`** — EventBridge Scheduler target, weekly Friday 5pm. Reads `fb-proposals` and `fb-clusters` for the past week; posts a Slack summary of what published, what's waiting in review, and which clusters hit the human-write queue (gaps in the docs). No Bedrock; a plain summary. Memory: 256 MB.

Storage

- **S3 Vectors** · **`fb-questions`** — one vector per cleaned question. 1024-dim (Titan V2). Metadata: `cluster_id`, `source` (email/chat/manual), `first_seen`, `seen_count`. Used by the grouper's nearest-neighbor search.
- **S3 Vectors** · **`fb-helpdocs`** — one vector per help-doc chunk. 1024-dim. Metadata: `doc_id`, `chunk_id`, `section`, `source_url`. Used by the drafter's retrieval. Re-embedded on doc change.
- **DynamoDB** · **`fb-clusters`** — one row per cluster. PK `cluster_id`; attributes: `centroid_ref`, `seen_count`, `window_counts`, `state`

(warm/candidate/published/rejected), `covered_entry_id`, `priority`. On-demand.

- **DynamoDB · `fb-proposals`** — one row per drafted proposal. PK `(cluster_id, draft_ts)`; attributes: `question`, `answer`, `source_ref`, `model`, `status` (pending/approved/edited/rejected). On-demand.
- **DynamoDB · `fb-audit`** — one row per write action of any kind. PK `(cluster_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.
- **S3 · `fb-docs-source`** — mirrored FAQ doc, help docs, and chat transcripts. Versioning enabled. Lifecycle to Glacier at 90 days for transcripts; FAQ and help docs kept hot.
- **S3 · `fb-rules-source`** — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3 · `fb-raw-mime`** — raw inbound MIME from the support inbox. Lifecycle to Glacier at 30 days; expiry at 1 year (questions are already extracted into the pile; the raw MIME is only kept for audit).

Bedrock

- **Embeddings.** `amazon.titan-embed-text-v2:0` (Amazon Titan Text Embeddings V2), 1024-dim. Two callsites: `intake` / `doc-embed` for the question and help-doc vectors. This is the dominant Bedrock cost — one embed per incoming question.
- **Drafting 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: `drafter`, on candidate and refresh clusters only.

- **Escalation model.** `global.anthropic.claude-sonnet-4-6-20250930-v1:0` (Claude Sonnet 4.6), used only on a rare retry when Haiku's grounded draft fails validation on a multi-part question. Capped at one retry per cluster per pass.
- **Quotas.** Default account quotas are more than enough at SMB volume. The grouper itself doesn't call Bedrock; embeddings batch the day's questions, and the drafter fires a few times a week.

EventBridge Scheduler config

- **`fb-daily-group`** — `cron(0 18 * * ? *)` in the SMB's timezone. Target: `grouper` Lambda.
- **`fb-drive-sync`** — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- **`fb-weekly-digest`** — `cron(0 17 ? * FRI *)` in TZ. Target: `digest` Lambda.
- **Re-embed on demand** — `drive-sync` invokes `doc-embed` directly when a help doc changes; no standing schedule needed.

SES inbound

- Set the MX record on a dedicated subdomain (e.g. `questions.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `fb-inbound-rules`: one rule with recipient `questions@your-company.com` → spam scan → S3 PUT to `s3://fb-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake`.

- No SES outbound identity is required — the builder delivers proposals to Slack, not email. If you want email proposals, verify a sender and add an `ses:SendRawEmail` path to `dispatch`.

IAM (least privilege per Lambda)

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

- **intake role:** `s3:GetObject` on `fb-raw-mime` and the chat prefix; `bedrock:InvokeModel` on the Titan ARN; `s3vectors:PutVectors` on `fb-questions`; `dynamodb:PutItem` on `fb-clusters`. *No* access to the drafting model.
- **grouper role:** `s3vectors:QueryVectors` on `fb-questions`; `s3:GetObject` on the rules doc; `dynamodb:Query` + `UpdateItem` on `fb-clusters`; `lambda:InvokeFunction` on `drafter`. *No* `bedrock:*`.
- **drafter role:** `s3vectors:QueryVectors` on `fb-helpdocs`; `bedrock:InvokeModel` on the Haiku and Sonnet ARNs; `s3:GetObject` on the voice doc; `dynamodb:PutItem` on `fb-proposals`.
- **ack-handler role:** `dynamodb:PutItem` on `fb-clusters`, `fb-proposals`, `fb-audit`; `secretsmanager:GetSecretValue` on the Docs-API service-account secret and the Slack signing secret; outbound network access to `docs.googleapis.com` and `slack.com`; `bedrock:InvokeModel` on the Titan ARN for the manual lane's embed; `s3vectors:PutVectors` on `fb-questions`.
- **dispatch role:** `secretsmanager:GetSecretValue` on the Slack bot token; outbound network access to `slack.com`; `dynamodb:GetItem` on `fb-proposals`.

- **drive-sync / doc-embed roles:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the docs and rules buckets; `bedrock:InvokeModel` on Titan and `s3vectors:PutVectors` / `DeleteVectors` on `fb-helpdocs` (doc-embed only); outbound network to `www.googleapis.com`.

Slack interactive flow

Proposals are posted via the `chat.postMessage` Web API with Block Kit blocks containing the Approve/Edit/Reject buttons (the incoming webhook can't carry interactive components). Button clicks are sent by Slack to the configured Interactivity request URL, which is the `ack-handler` Function URL. `ack-handler` verifies the Slack signing secret on the inbound request, parses the `action_id` (`approve`, `edit`, `reject`), opens a modal for Edit (pre-filled draft) and Reject (reason), and processes the response on modal submission. The manual-lane question form is a separate Block Kit modal opened from a slash command, submitting to the same Function URL.

The Slack app needs `chat:write` and the Interactivity URL configured. The bot token lives in Secrets Manager under `fb/slack/bot-token`; the signing secret under `fb/slack/signing-secret`.

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:** grouper Lambda failures > 0 in a day (the daily pass is the one piece that has to run); drafter citation-validation failure rate > 30% in 24h (might mean the help docs drifted from how customers phrase questions); ack-handler signature-verification failures > 5/hour (might mean the Slack secret rotated).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $15/month threshold, alarm at 80% and 100%, posts to SNS topic `fb-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for Drive and Docs APIs live in Secrets Manager under `fb/drive/sa` (one service account scoped to both). Slack bot token and signing secret under `fb/slack/*`. The `join_threshold`, `min_asks_for_candidate`, the count window, retrieval `k`, the configured timezone, and the review Slack channel all live in Parameter Store under `/fb/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 `fb-docs-source` and `fb-rules-source` so a bad Drive edit can be rolled back in one click, and version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily pass in UTC after a CI rotation. Keep the two S3 Vectors indexes (`fb-`

`questions`, `fb-helpdocs`) in their own stack so a re-index never risks the rest. Total deployable surface: around eight Lambdas, three DDB tables, two S3 Vectors indexes, three S3 buckets, the 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**.