7-PART SERIES · FREE COMPANION

Testimonial collector

A serverless collector that asks happy customers for a testimonial at the right moment, collects their reply, tidies it into a clean quote with their permission, and files approved ones ready to use on your site. It never nags, and it never publishes a word without the customer's yes and your sign-off. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Testimonial collector

01 A testimonial collector on AWS for a few dollars a month

02 How a testimonial request goes out

03 How a testimonial reply becomes a quote

04 How a testimonial gets sign-off

05 How a testimonial gets published

06 What the testimonial collector costs

07 Engineering reference: the testimonial collector architecture

PART 1 OF 7

MAY 12, 2026 PART 1 OF 7 · TESTIMONIAL COLLECTOR SERIES ~5 MIN READ

A testimonial collector on AWS for a few dollars a month

Every small business has happy customers it never quotes. The client who emailed “honestly the best decision we made this year” and got a thank-you and nothing else. The 5-star review that nobody turned into a line on the homepage. The project that wrapped beautifully and was forgotten the next Monday. Those moments are the cheapest marketing a business will ever have — and they evaporate because nobody had a system to catch them, ask nicely, and tidy the reply into something usable. This post walks through the design of a small collector that spots the good moment, asks once, tidies the reply, and never publishes a word without sign-off.

KEY TAKEAWAYS

- Three sources for happy moments: a Drive list, an inbox forwarding lane, and a ratings webhook lane.
- Every customer ends in one of four moves on each tick: wait, first ask, one reminder, or stop.
- Per-moment rules: a 5-star rating asks after 1 day, a finished project after 3, a renewal after 7.
- Asks respect quiet hours and a never-nag cool-down. Anyone who declines is left alone for a year.
- Designed on AWS for about $2/month at typical small-business volume.

The whole system on one page

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

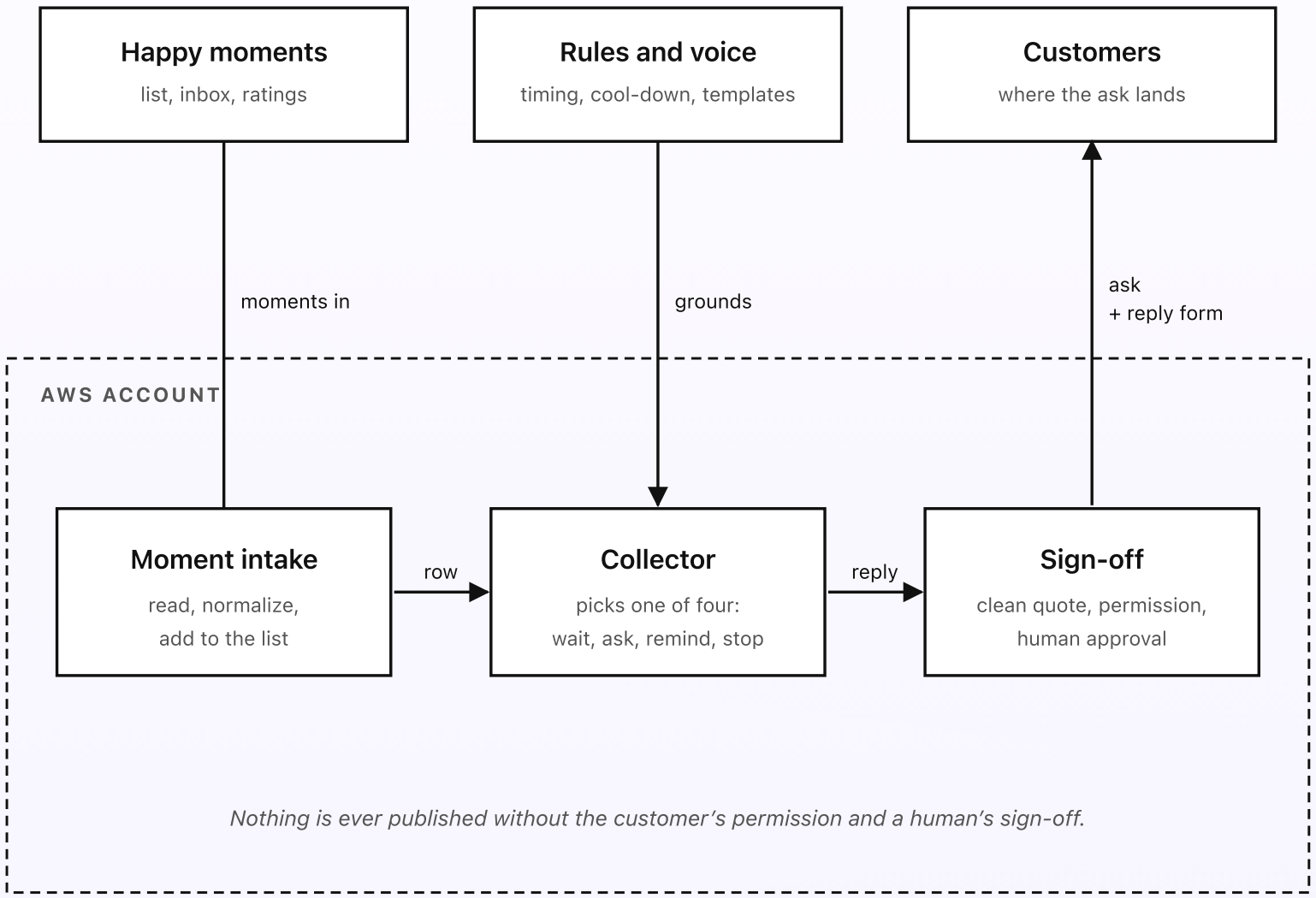

Happy moments
list, inbox, ratings
Rules and voice
timing, cool-down, templates
Customers
where the ask lands
moments in
grounds
ask
+ reply form
AWS ACCOUNT
Moment intake
read, normalize,
add to the list
row
Collector
picks one of four:
wait, ask, remind, stop
reply
Sign-off
clean quote, permission,
human approval
Nothing is ever published without the customer's permission and a human's sign-off.

Fig 1. Three sources outside, three pieces inside AWS. Moments flow in from a Drive list, an inbox forwarding lane, and a ratings webhook lane. The Collector runs daily and picks one of four moves. Sign-off turns a reply into a clean quote and holds it for permission and approval.

What you set up once (the outside)

- **Happy moments.** A Google Sheet in a Drive folder, one row per customer worth asking: name, email, the moment that made them a candidate (5-star rating, glowing reply, finished project, renewal), the date of that moment, and a link to where it came from. You can fill it in once and keep adding by hand; new moments can also enter via two other lanes covered in Part 2 — an inbox-forwarding lane (forward a glowing email to a dedicated address and the collector proposes a row for one-tap approval) and a ratings webhook lane (a 5-star rating in your review tool pings a small endpoint and lands as a candidate automatically).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the timing for each moment — how many days after the moment to send the ask, and the cool-down windows. A 5-star rating might ask after 1 day; a finished project after 3; a renewal after 7. The doc also holds the never-nag policy (no second ask within 90 days, no ask at all for a year after someone declines), the quiet hours, and the holiday calendar. The *voice* doc holds one ask message template per moment — what the email actually says.
- **Customers.** The happy people themselves. The ask is one short, warm email with a single link to a reply form. The form has a text box for their words and one permission checkbox: “You may use my words on your website and

marketing." No login, no account, no ten-question survey. The whole point is that it takes thirty seconds.

What runs on every tick (the inside)

- **The moment intake.** Three sources feed the list. The Drive sheet itself is the canonical store. New moments can also be added via the inbox forwarding lane (forward a glowing email to `kudos@your-company.com`, the collector uses Bedrock Haiku 4.5 to read the praise and pull out the customer name, email, and a one-line summary, then drops a one-tap approval card in the team's Slack to confirm before the row is added) and the ratings webhook lane (a 5-star rating in your review tool calls a small Function URL that adds the customer as a candidate).
- **The collector.** Runs once a day at 9am local. Reads the list. For each candidate, computes days-since-the-moment. Checks the never-nag cool-down. Picks one of four moves. *Wait*: the moment is too fresh, or a cool-down is active — do nothing. *First ask*: the timing window just opened and the customer hasn't been asked — send one warm ask with the reply form. *One reminder*: the first ask got no reply after a set gap — send a single gentle nudge. *Stop*: the reminder also got no reply, or the customer declined — close the candidate quietly and never ask again this cycle. The collector itself doesn't call a model on the daily tick — the move logic is plain Python.
- **Sign-off.** When a customer replies, a Bedrock Haiku 4.5 call tidies their words into a short clean quote — fixing typos and trimming filler, never changing the meaning. The quote is held until two gates pass: the customer ticked the permission box, and a human on the team reviewed the clean quote against the original reply and approved it. Only then does the quote land in an "approved"

sheet, ready to drop on a page. A monthly summary writes a short paragraph: how many asks went out, how many replied, how many were approved, and the best new quotes.

In plain words

A client named Dev leaves a 5-star rating on Tuesday. The ratings lane adds him as a candidate. On Wednesday morning (1 day out, per the rule for ratings), the collector sends one short email: “Dev — thank you for the 5 stars. Would you be open to sharing a sentence or two about working with us? Here’s a 30-second form.” Dev is busy and doesn’t reply. Four days later he gets one gentle reminder. This time he clicks, writes “they turned our mess of a launch into something we’re actually proud of, couldnt recommend more,” and ticks the permission box. Bedrock tidies it to “They turned our mess of a launch into something we’re actually proud of — couldn’t recommend them more.” The team lead opens the approval card, sees the clean quote next to Dev’s original, agrees it’s faithful, and taps Approve. The quote lands in the approved sheet with Dev’s name and the date. If Dev had never ticked the box, the quote could never be used — full stop.

The cost of running this is about $2 a month at SMB volume. The cost of *not* running it is a year of happy moments that turned into nothing — the homepage with no quotes on it, the proposal that needed proof and didn’t have any.

DESIGN RULES THAT SHAPED EVERY DECISION

- Ask at the moment of goodwill — right after a 5-star rating or a glowing reply, when the customer actually feels it.
- Four moves, always. Wait, first ask, one reminder, stop. There is no third ask.
- Never nag. A cool-down stops repeat asks; a decline buys a full year of silence.
- Permission is a hard gate. No checkbox, no use — ever, no exceptions.
- A human signs off every quote against the original words before it can be used.
- Every action is logged. Audit a published quote next year and you can see the consent and the approval.

Why this shape

Most teams “collect testimonials” in one of three ways: they mean to and never do, they send a clunky survey that nobody finishes, or they paste a customer’s private email onto the website without asking. The first one is the default and it costs the most — a steady drip of goodwill that turns into nothing. The survey is the false-effort version: it feels like a system but the response rate is near zero because it’s work. And the copy-paste-without-asking version is the dangerous one — it works right up until the customer sees their words used without consent and the goodwill flips to anger.

The setup above asks at the moment the customer is happiest, makes replying a thirty-second job, tidies the reply so a good thought isn't lost to a typo, and refuses to publish anything without two clear yeses — the customer's and a teammate's. It is polite by default and silent the moment someone isn't interested. The collector is invisible most days; visible only when there's a genuinely good moment to act on.

The next four posts walk through each piece in turn: how a testimonial request goes out, how a reply becomes a clean quote, how a testimonial gets sign-off, and how an approved one gets published. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 12, 2026 PART 2 OF 7 · TESTIMONIAL COLLECTOR SERIES ~4 MIN READ

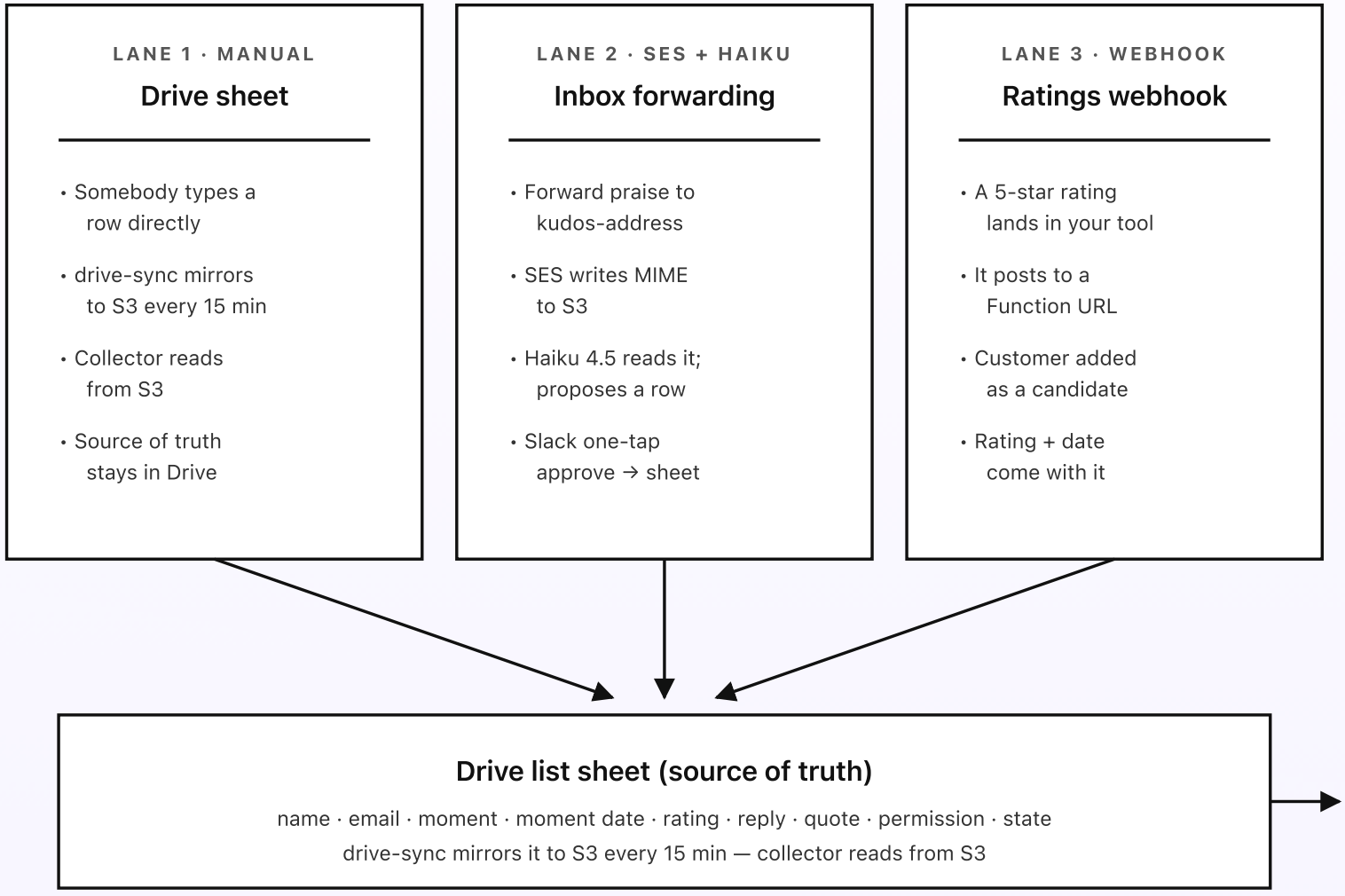
How a testimonial request goes out

The collector only asks customers who are on the list. So the first job is making sure the list actually reflects who's been happy lately. There are three ways a happy moment gets in: somebody types a row in the Drive sheet, somebody forwards a glowing email to a dedicated address, or a 5-star rating in your review tool pings a small endpoint. The first one is obvious. The other two exist because in real life nobody types a row in a sheet for the kind email they just read.

KEY TAKEAWAYS

- Three intake lanes feed one list: the Drive sheet, an inbox-forwarding lane, and a ratings webhook.
- Forwarded praise is read by Bedrock Haiku 4.5, which proposes a row for one-tap approval.
- The ratings lane is a Function URL: a 5-star rating posts a customer straight in as a candidate.
- One warm ask goes out at the right moment, then at most one gentle reminder — never a third.
- Asks respect quiet hours and a never-nag cool-down. No reply is treated as a polite no.

Three lanes into one list



The Drive sheet stays the source of truth — the ask only fires once a candidate clears the cool-down.

Fig 2. Three lanes converge on one Drive sheet. The sheet is the source of truth; the inbox lane and the ratings webhook are conveniences that propose or add candidates. The drive-sync Lambda mirrors the sheet to S3 so the collector can read it without hitting Drive on every tick.

Lane 1: the Drive sheet itself

The simplest lane. Open the list sheet in Drive, add a row, save. The columns are short: name, email, the moment that made them a candidate, the date of that moment, the rating (if any), and later the reply, the tidied quote, the permission state, and the publish state. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://tc-list-source/list.csv` if the sheet has changed since the last sync. The collector reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so a bad bulk-edit can be rolled back in one click.

This lane covers the cases where you already know a customer was happy — the call that ended well, the handshake at the end of a project — and you can spend thirty seconds typing it in.

Lane 2: inbox forwarding (the lane most teams actually use)

Set up a dedicated inbound address — something like `kudos@your-company.com` — via Amazon SES. Anyone on the team forwards a kind email to that address and the collector takes it from there. SES writes the raw MIME to `s3://tc-raw-mime/`. The S3 PUT triggers a parser Lambda. The Lambda walks the MIME tree to the message body and pulls out the plain text of the forwarded note. There's no

document parsing here — praise arrives as email, so no Textract and no per-page charges.

Then a Bedrock Haiku 4.5 call reads the text and emits a structured row: the customer name, their email (from the original sender line), the moment type (set to “glowing reply”), and a one-line summary of what they liked. The model prompt is short: “Pull out who said this and a one-line summary. Return JSON only. Do not invent an email that isn’t in the message.” The output goes to a small Slack interactive message that pings the person who forwarded the email: the proposed row and three buttons — *approve*, *edit*, *discard*. On *approve*, a Lambda writes the row to the Drive sheet via the Sheets API. On *edit*, they get a fillable modal pre-populated with the proposal. On *discard*, the message is logged and the email moved to a discarded prefix in S3 for audit.

The reason every parsed row goes to a human first is simple: asking the wrong person for a testimonial — or getting their name wrong — is a worse outcome than the email never making the list at all.

Lane 3: ratings webhook

Many teams already collect star ratings — on Google, on a booking tool, in a post-job survey. When a 5-star rating comes in, there’s no reason to retype it. Lane 3 is a small Lambda Function URL that your review tool calls on each new rating. The Lambda checks the score; a 5-star (or 4-and-above, if you set it that way) rating with a contact attached is written straight onto the list as a candidate, with the rating and the date already filled in. A lower rating is ignored — this system only ever asks the people who were genuinely happy.

The webhook is the most hands-off of the three lanes. Once it's wired up, happy customers flow onto the list without anyone lifting a finger, and the never-nag cool-down (covered below) makes sure nobody gets asked twice in a short window.

How the ask actually goes out

Once a candidate is on the list and has cleared the cool-down, the daily tick decides on a move (the full move logic is the next post's territory; here's the short version). On the *first ask*, the collector reads the matching template from the voice doc, fills in the customer's name and the moment, and sends one short, warm email through SES with a single link to a reply form. If there's no reply after a set gap (default 5 days), it sends exactly *one* gentle reminder. If that also gets no reply, the candidate is closed quietly — no reply is treated as a polite no, and a decline buys a full year of silence. Every send respects quiet hours and the holiday calendar, so an ask never lands at 2am or on a public holiday.

The whole point of capping it at one reminder is trust. A business that asks once, kindly, and then stops is a business people are happy to hear from. A business that asks five times is one people mute.

Next post: how a customer's reply gets tidied into a short, clean quote — without changing a word of what they meant.

PART 3 OF 7

MAY 12, 2026 PART 3 OF 7 · TESTIMONIAL COLLECTOR SERIES ~4 MIN READ

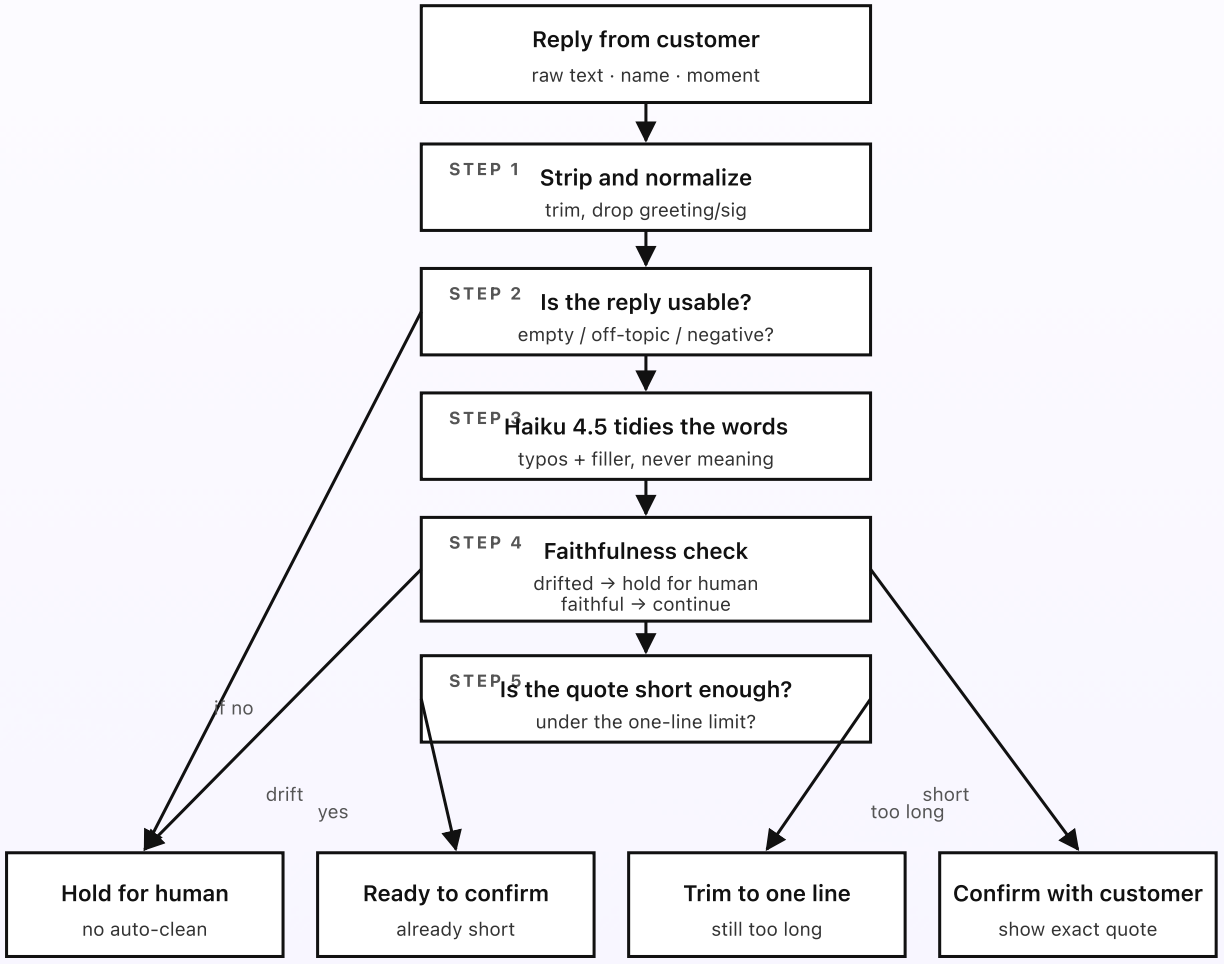
How a testimonial reply becomes a quote

A customer who fills in the reply form is doing you a favour, and they're doing it in thirty seconds from their phone. So their reply often has a typo, a run-on sentence, or a half-finished thought. A good testimonial shouldn't be lost to a missing apostrophe. This post walks through the one place the collector uses AI on purpose: a small, careful pass that tidies the customer's words into a short clean quote — and a second pass that proves it didn't change what they meant.

KEY TAKEAWAYS

- When a reply lands, Bedrock Haiku 4.5 trims it into a short clean quote — typos and filler only.
- A second model pass checks the clean quote against the original and flags anything invented.
- If the check fails, the raw reply is kept as-is and routed to a human instead of auto-cleaning.
- The customer always sees the exact final quote before it can be used — no surprises.
- The model never adds praise, never changes meaning, and never makes a lukewarm reply sound glowing.

From reply to quote, step by step



The model only cleans — the customer still sees and confirms the exact final quote.

Fig 3. One reply, turned into a clean quote. Haiku 4.5 tidies typos and filler, a faithfulness check proves nothing was added, and a quote that drifts is held for a human instead of auto-cleaned. The customer always confirms the exact final words.

What the tidy pass does — and doesn't

When a customer submits the reply form, the form's Function URL Lambda writes the raw text to the `tc-replies` table and kicks off the cleanup. The cleanup is one Bedrock Haiku 4.5 call with a deliberately narrow job. The prompt says, in effect: "Here is what a customer wrote about us. Fix spelling and punctuation, remove filler words, and make it read as one clean sentence or two. Do not add any praise. Do not change the meaning. Do not make a mild comment sound enthusiastic. Return only the cleaned text."

So "they turned our mess of a launch into something we're actually proud of, couldnt recommend more" becomes "They turned our mess of a launch into something we're actually proud of — couldn't recommend them more." Same meaning, same enthusiasm, just readable. What the pass will *not* do is invent a second sentence, swap "good" for "amazing," or paper over a complaint. If a customer's reply is actually lukewarm, the clean version stays lukewarm — and a lukewarm quote rarely makes it past the human in Part 4 anyway.

Why there's a second pass

Even a careful prompt can drift — a model might tighten a sentence and accidentally strengthen it. So a second Bedrock call does nothing but compare. It gets the original and the cleaned version and answers one question: "Does the cleaned text say anything the original didn't, or sound stronger than the original?"

If the answer is yes, the clean version is thrown away, the raw reply is kept exactly as the customer wrote it, and the whole thing is routed to a human to handle by hand. This is the same belt-and-braces idea used elsewhere in the series: the cheap automated path handles the easy 95%, and anything the check isn't sure about goes to a person rather than out the door.

Both calls are tiny — a few hundred input tokens and a few hundred output tokens each — so even at a couple thousand replies a month, this is a few cents. The daily tick that finds and asks customers still uses no model at all; the only AI in the whole system lives right here, on the reply.

The customer always sees the final words

The clean quote is never the end of the story. The next post covers the two gates a quote must clear, but the first one matters here: the customer is shown the exact final quote — the cleaned version, not their raw text — and asked to confirm it reads right before it can be used. If the tidy pass changed a word they don't like, they fix it on the spot. Nobody ever finds their words altered after the fact. The model proposes; the customer disposes.

Next post: how a tidied quote clears the two sign-off gates — the customer's permission and your approval — before it's allowed anywhere near your site.

PART 4 OF 7

MAY 12, 2026 PART 4 OF 7 · TESTIMONIAL COLLECTOR SERIES ~4 MIN READ

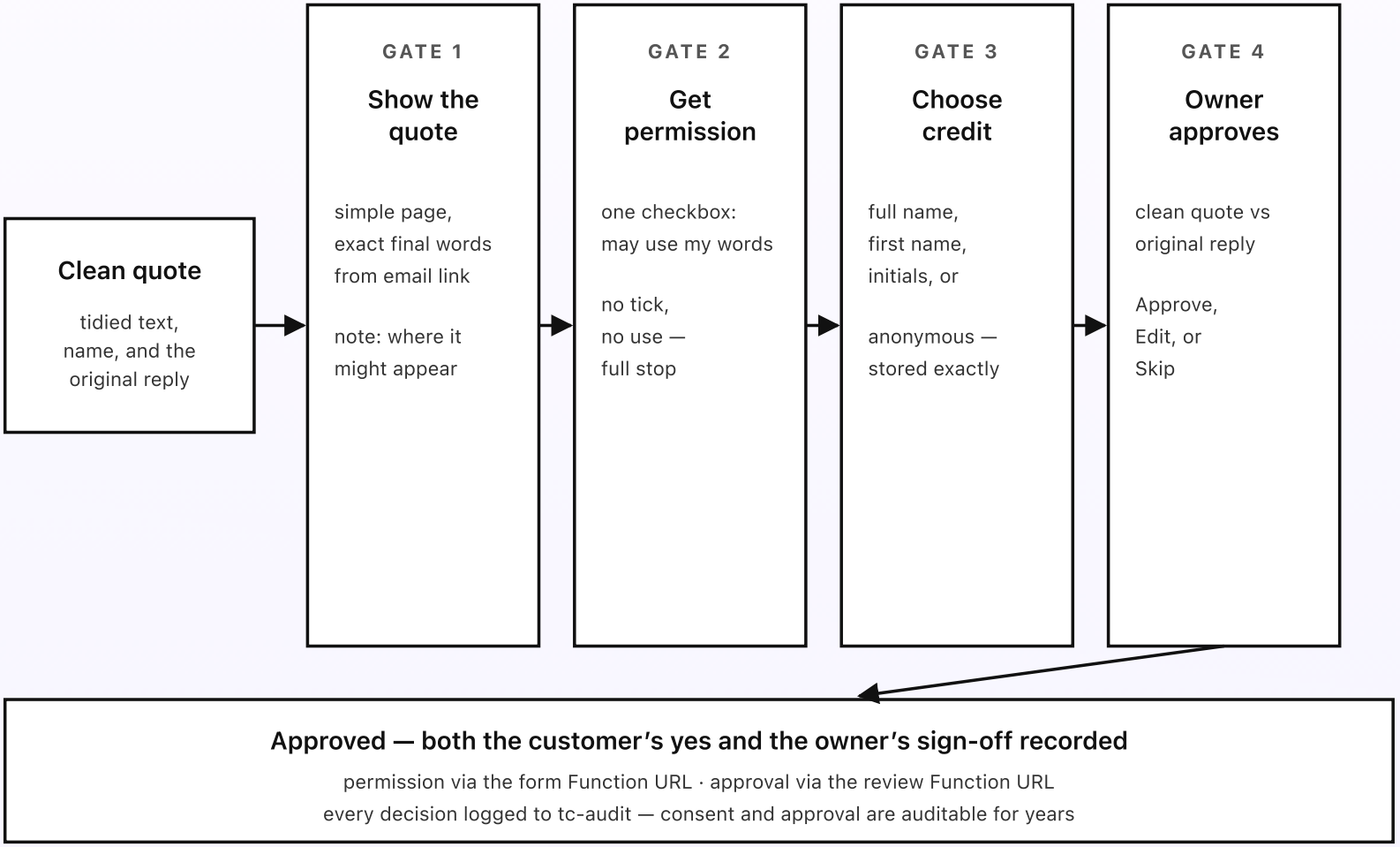
How a testimonial gets sign-off

This is the part that keeps the whole system honest. A clean quote is not a publishable quote. Before a single word reaches your site, it has to clear two gates that both say yes: the customer's permission, given on a simple page that shows them the exact quote, and your own approval, given from a card that puts the clean quote next to what they actually wrote. Miss either gate and nothing happens. There is no override.

KEY TAKEAWAYS

- Two gates, both required, in order: the customer's permission, then your approval.
- The customer sees the exact final quote and picks how to be credited before they say yes.
- Crediting options: full name, first name only, initials, or fully anonymous.
- You approve from a card with the clean quote beside the original — Approve, Edit, or Skip.
- Every gate is a deterministic check, and every decision is written to the audit trail.

Two gates before anything can publish



Both yeses are required — the customer's permission and the owner's approval. No override.

Fig 4. Four gates between a clean quote and a publishable one. The customer sees the exact quote, ticks permission, and picks their credit; then a teammate approves it against the original. Only a quote with both yeses ever reaches the approved state.

Gates 1 and 2: the customer's permission

The link in the ask email opens a small page served by a Lambda Function URL — no login, no account. The page shows the customer the exact final quote (the cleaned version from Part 3, not their raw text) and a one-line note about where it might appear: "We'd love to show this on our website and in marketing." Below it is a single permission checkbox. This is the hard gate of the whole system. If the customer does not tick that box, the quote is marked *declined* and can never be used — there is no path in the code that publishes an unticked quote. A decline is also logged so the never-nag policy from Part 2 leaves them alone for a year.

Keeping permission on the same page as the quote is deliberate. The customer agrees to the *specific words* they can see, not to some vague future use. That's both kinder and safer: nobody is ever surprised by a quote they don't remember approving.

Gate 3: how they want to be credited

Right next to the permission box, the customer picks how they want to be named: full name, first name only, initials, or fully anonymous ("a happy customer"). Whatever they choose is stored on the row and is *exactly* how the quote will ever appear — the publishing step in the next post reads this field and nothing else. A customer who wants to share their words but not their name can do that in one

tap, which means more people say yes. Their choice is theirs to change later, too; updating the credit re-runs the publish step with the new setting.

Gate 4: your approval

A customer's yes is necessary but not sufficient. Once permission is in, a review card lands for a teammate — in Slack via the bot, or in a simple web view from another Function URL. The card shows the clean quote right next to the customer's original reply, so the reviewer can see at a glance that the tidy pass stayed faithful. Three buttons: *Approve* moves the quote to the approved state; *Edit* opens a box to fix a small thing (a stray comma, a name spelling) before approving; *Skip* drops it quietly without ever telling the customer it was dropped. This second gate catches the rare quote that's technically fine but not quite right for the brand — and it means a human always stands between a customer's words and the public site.

Every action across all four gates — permission given or declined, credit chosen, approve, edit, skip — is written to the `tc-audit` table with a timestamp and the before-and-after. Pull up any published quote a year later and you can show exactly when the customer consented and who approved it.

Next post: how an approved testimonial actually gets published — written to a clean file your site reads, credited exactly how the customer chose, and removable the moment anyone changes their mind.

PART 5 OF 7

MAY 12, 2026 PART 5 OF 7 · TESTIMONIAL COLLECTOR SERIES ~4 MIN READ

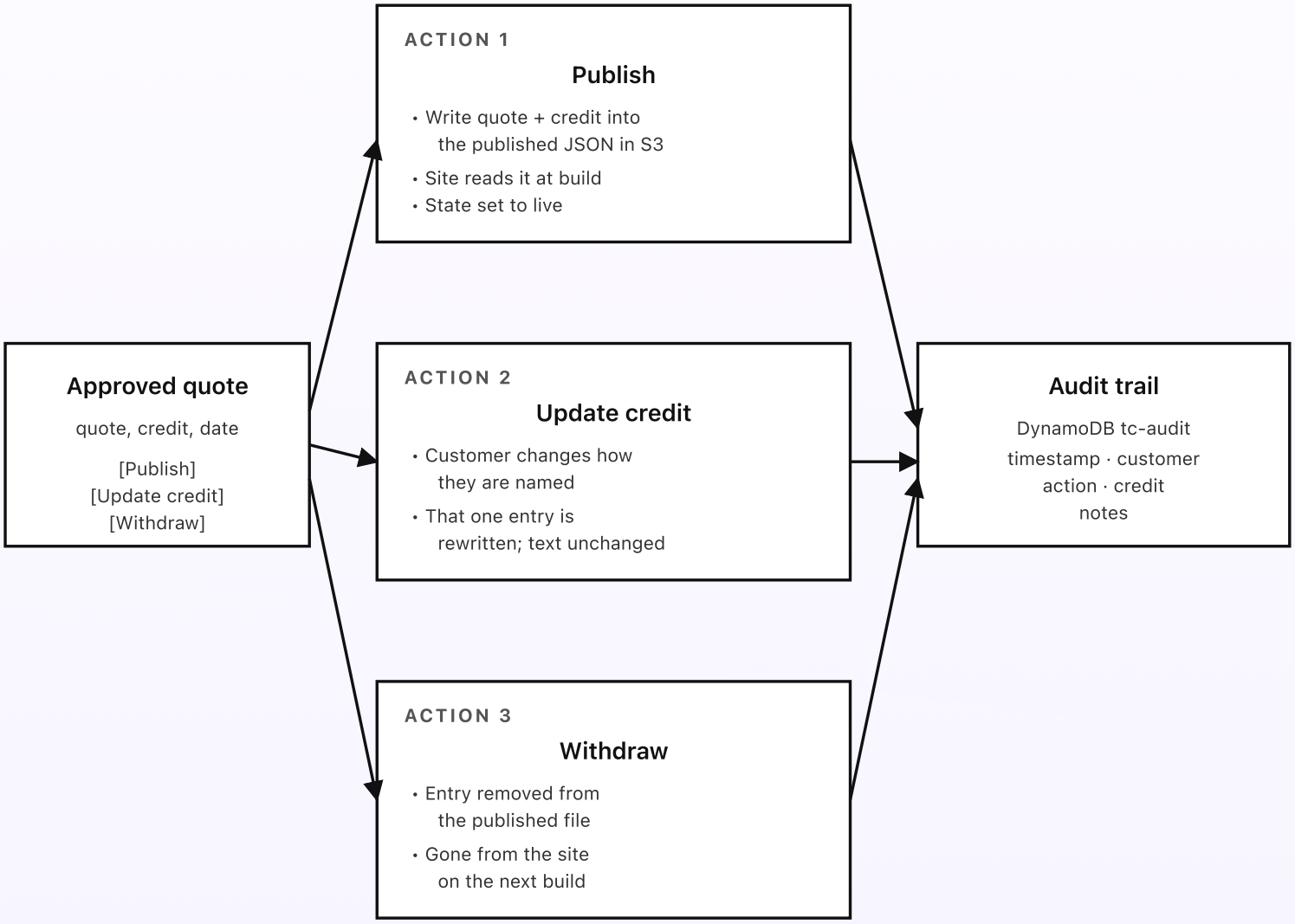
How a testimonial gets published

A quote that has cleared both gates is approved, but approved isn't the same as live. The last step turns an approved row into something your website can actually show — a small, clean file with the quote and the credit exactly as the customer chose. Just as important, this step works in reverse: a customer who changes their mind can update their credit or pull their quote entirely, and the site reflects it on the next build. Consent here is a thing you can take back.

KEY TAKEAWAYS

- An approved quote is written to a clean JSON file in S3 that your static site reads at build time.
- The credit is rendered exactly as the customer chose — name, first name, initials, or anonymous.
- A customer can update their credit later; the next build picks up the change.
- A customer can withdraw consent; the quote drops from the file and the site on the next build.
- Every publish, update, and withdrawal is logged to the audit trail.

Three things that happen to an approved quote



Publishing is reversible — a withdrawal drops the quote on the next build; the consent record stays.

Fig 5. Three things that happen to an approved quote. Publish writes it into the file your site reads; Update credit rewrites just the credit; Withdraw removes it on the next build. Every action is logged to the audit trail.

Action 1: publish to a file your site reads

When a quote is approved, a Lambda writes it into a single JSON file — `s3://tc-published/testimonials.json` — that holds every live quote. Each entry is small and clean: the quote text, the credit (rendered exactly per the customer's choice), the moment type, and the date. Your website reads this one file at build time and renders the quotes wherever you want them — a homepage strip, a wall of love, a proposal template. Because the site is static and reads a plain file, there's no database call on every page view and nothing to keep running. The published file is the clean boundary between the collector and your site: the collector owns the file, the site just reads it.

Keeping everything in one versioned file (S3 versioning is on) has a nice side effect: if a bad edit ever slips in, you roll the file back in one click and the next build is clean again.

Action 2: update the credit

People's preferences change. Someone who said "first name only" in March might be happy to use their full name in June, or the other way around. The link in their original email keeps working, so a customer can reopen their page any time and change how they're credited. The Lambda rewrites just that one entry in the published file with the new credit — the quote text itself never changes — and the next build shows it the new way. Honoring this in one tap, with no email to

support, is the kind of small respect that keeps customers comfortable being quoted at all.

Action 3: withdraw

The most important action is the one you hope is rarely used: withdrawing. Every published quote carries a withdraw link. One tap removes the entry from `testimonials.json`, flips the row's state to *withdrawn*, and the quote is gone from the site on the next build — no questions, no friction. Crucially, the consent and withdrawal are both kept in the `tc-audit` trail. So even after a quote comes down, you can still show that the customer once consented and later chose to withdraw, with timestamps for both. Consent you can't walk back isn't really consent; this step makes sure it's a door that opens both ways.

That completes the loop: a happy moment is spotted, one ask goes out, a reply is tidied into a clean quote, two gates sign it off, and an approved quote is published — reversibly. The next post adds up what all of this costs to run, and the last one is the engineering reference.

PART 6 OF 7

MAY 12, 2026 PART 6 OF 7 · TESTIMONIAL COLLECTOR SERIES ~3 MIN READ

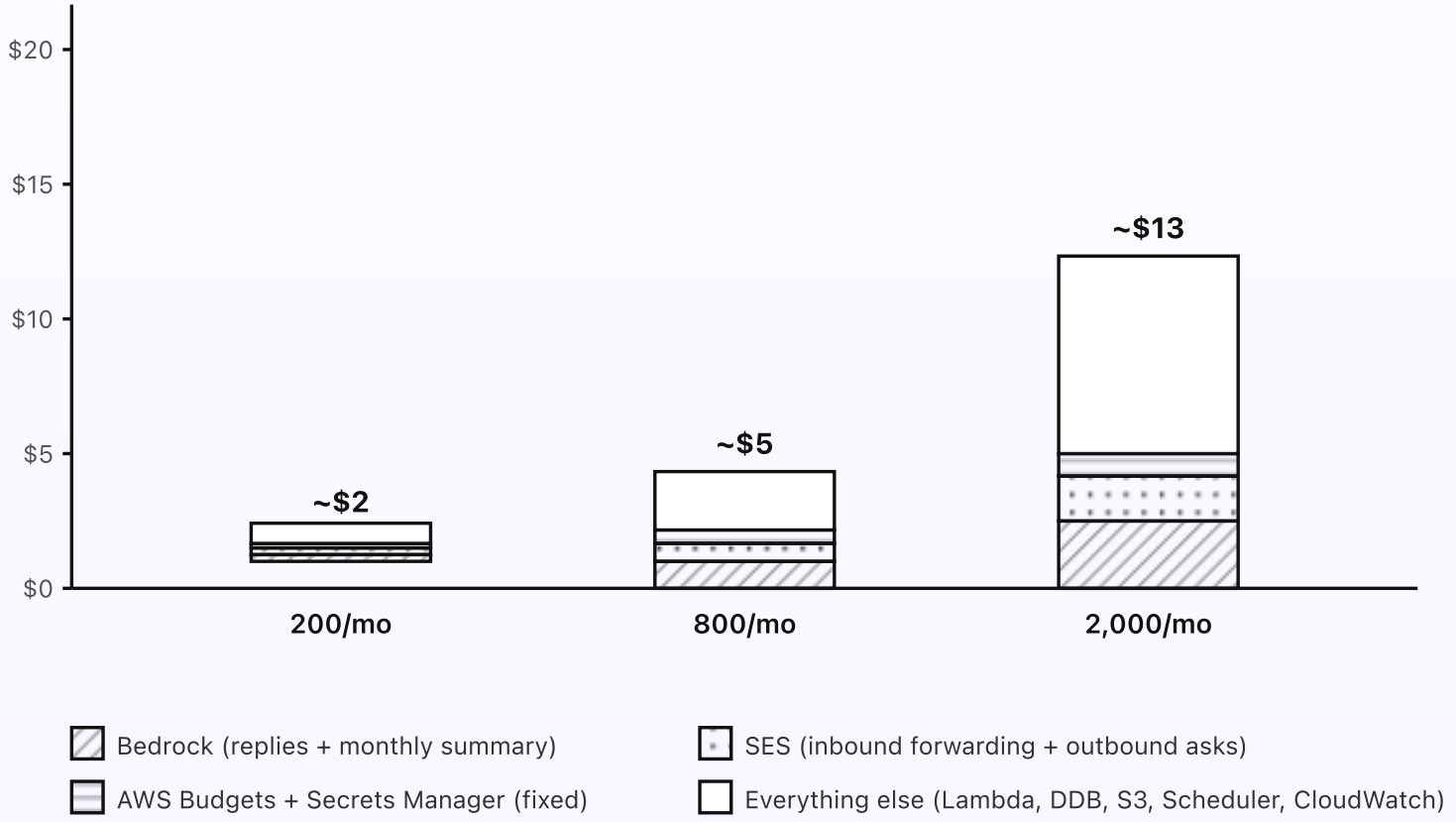
What the testimonial collector costs

The collector is one of the cheapest systems in this whole series. The daily tick reads a CSV from S3, does a little date arithmetic, writes a few rows to DynamoDB, and sends a handful of emails. It calls no models on the tick. Bedrock fires only when a customer replies (to clean up their words) and once a month for a short summary. At typical SMB volume, the bill is a couple of dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2/month at typical SMB volume (around 200 happy moments a month).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily tick costs pennies — no model calls.
- Bedrock fires only on a customer reply (a clean-up call) and the monthly summary.
- At 800 moments a month the bill is around $5. At 2,000 it's around $13.

Cost at three volumes



The daily tick is cheap — and each reply costs a fraction of a cent to clean up.

Fig 6. Monthly cost at three happy-moment volumes. Bedrock and SES are small slivers because Bedrock only fires on replies and the monthly summary, and SES email is fractions of a cent each. The dominant cost is the everything-else bucket: the daily tick and the dispatch of asks.

Where the dollars actually go

Lambda runtime (the bulk). The collector runs once a day. Each tick reads the candidate CSV from S3, iterates the rows, computes `days_since_moment` for each, and decides on a move. At 200 candidates that's a few hundred milliseconds. At a few thousand it's a couple of seconds. Either way it's pennies a month. Add the dispatch Lambda firing for each ask (a few hundred a month at 200 moments), the Function URL Lambdas for the reply form and the review buttons, and the drive-sync Lambda every fifteen minutes — the Lambda total still lands under a couple of dollars at all three volumes.

DynamoDB on-demand. Three small tables: `tc-asks`, `tc-state`, `tc-audit`. Reads are dominant during the daily tick (one read per candidate per tick). Writes are ask events, replies, and audit rows. Pennies a month at any of these volumes.

S3 + Storage. The mirrored candidate CSV plus the archived MIME from any forwarded praise. A few hundred KB total at SMB volume. Effectively free.

EventBridge Scheduler. The daily tick rule plus deferred dispatch rules from the quiet-hours gate. A few invocations a day. Pennies.

SES. Inbound for the forwarding lane: $0.10 per thousand received messages (so a couple of cents a year for an SMB). Outbound for the asks and reminders: $0.10 per thousand sent. At a few hundred asks a month, both are negligible.

Bedrock (only when something fires it). The daily tick uses no Bedrock. Each reply fires Haiku 4.5 twice — once to clean up the words and once for the faithfulness check — both small calls: a few hundred input tokens and a few hundred output tokens, so a fraction of a cent per reply. Only a fraction of asks

turn into replies, so at 200 moments a month this is cents. The monthly summary is one larger call: write a paragraph that summarizes the month's asks, replies, and approvals; a couple of cents.

Forwarded praise. The inbox lane fires one Haiku call per forwarded email to propose a row. A few forwards a month at most, so cents. There's no document parsing here — praise arrives as plain email text, so no Textract and no per-page charges.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the reply form, the ratings webhook, and the review buttons.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The collector sleeps 23.99 hours a day.
- **A Knowledge Base.** The list is structured rows and each reply is cleaned on its own — there's nothing to search across. No embeddings, no Knowledge Base, no S3 Vectors needed.
- **Models on the tick.** The daily decision is plain Python. Bedrock fires only on replies, forwarded praise, and the monthly summary.

How the cost scales

Lambda runtime grows roughly linearly with the number of candidates, because every candidate is evaluated on every tick. DynamoDB grows linearly too. Bedrock

grows with the number of *replies*, not moments — and only a fraction of asks get a reply — so it stays modest. SES grows with the number of asks. So the bill at 5,000 moments a month is around $30; at 10,000 it's around $60. Past those volumes the daily-tick model probably stops being right (you'd switch to a partial-tick that only evaluates candidates inside an active ask window), but those are optimizations for very large lists — not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. The collector'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, SES rule set, and EventBridge Scheduler config.

PART 7 OF 7

MAY 12, 2026 PART 7 OF 7 · TESTIMONIAL COLLECTOR SERIES ~8 MIN READ

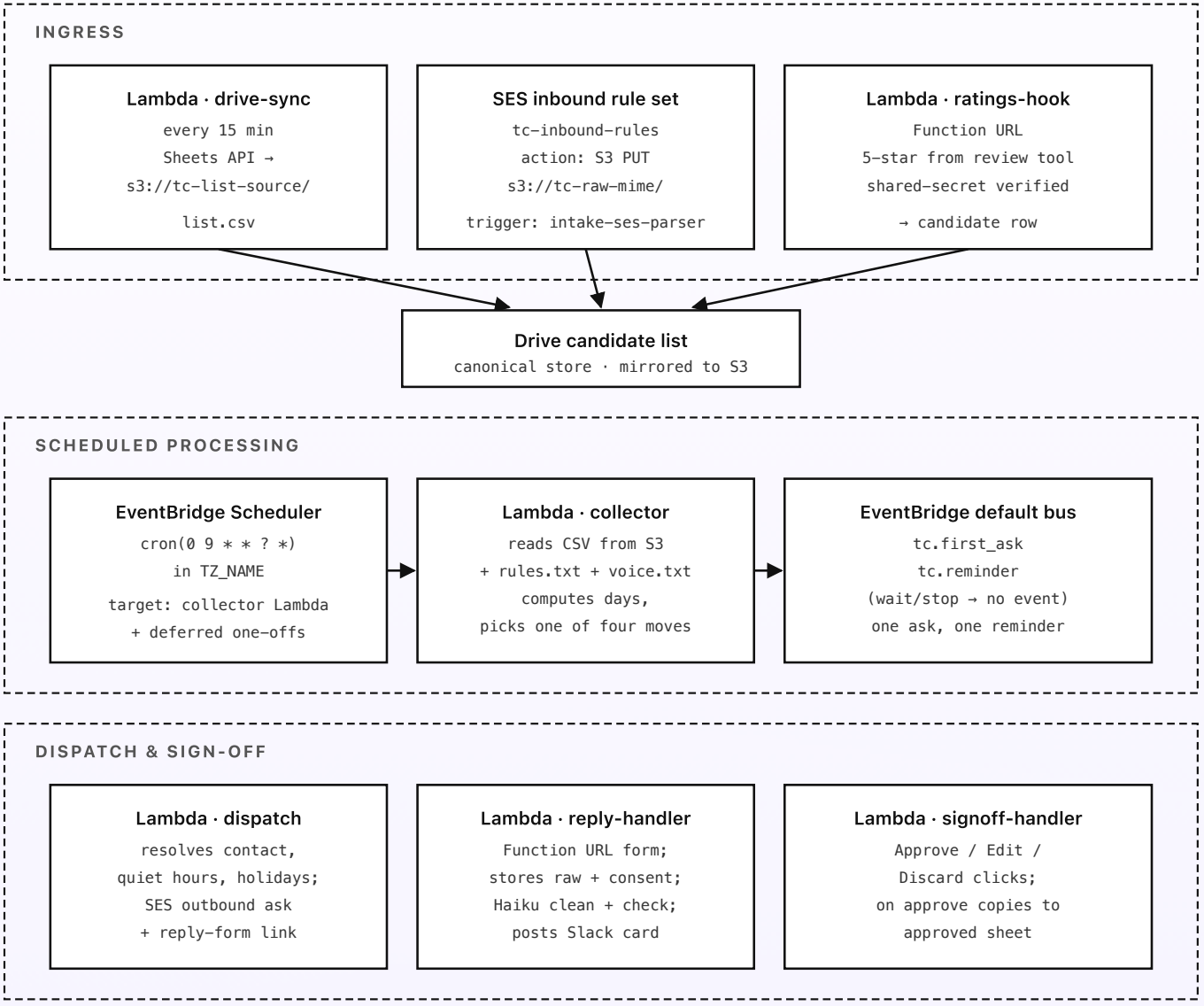
Engineering reference: the testimonial collector architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, Lambda inventory, IAM scopes, the SES inbound rule set, 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, Bedrock cross-Region inference, and EventBridge Scheduler are all in good shape 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 missed testimonial, not a regional outage. One AWS account dedicated to the collector (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Nothing is published without permission and approval — every interaction is logged to tc-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the list), scheduled processing (the daily collector tick emitting events), dispatch and sign-off (the ask ships, the reply is cleaned, and the human decision is recorded). 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 + Sheets API (service-account credentials in Secrets Manager under `tc/drive/sa`) to export the candidate sheet as CSV and write to `s3://tc-list-source/list.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://tc-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`ratings-hook`** — Lambda Function URL, public with `AuthType: NONE`; verifies a shared secret (in Secrets Manager under `tc/ratings/secret`) on each inbound request from the review tool. Reads the score; if it clears the threshold in the rules doc, writes a candidate row to the Drive sheet via the Sheets API with the moment set to `rating`. Low scores are dropped. Memory: 256 MB. Timeout: 15 s.
- **`intake-ses-parser`** — S3 PUT trigger on `s3://tc-raw-mime/`. Parses MIME, extracts the email body and the original sender. Calls Bedrock Haiku 4.5

(`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`) to decide whether the message is genuine praise and, if so, propose a candidate row (name, email, one-line summary, confidence). Posts the proposal to Slack via `chat.postMessage` with Approve/Edit/Discard buttons. Praise arrives as plain email text, so there is no document parsing on this path — no Textract. Memory: 512 MB. Timeout: 30 s.

- **`collector`** — EventBridge Scheduler target, daily at 9am local time (the schedule expression runs in `TZ_NAME` set to the SMB's timezone, e.g. `Asia/Singapore`). Reads `s3://tc-list-source/list.csv` and the rules and voice docs. For each row, computes `days_since_moment`, reads state from `tc-asks` and `tc-state`, applies the never-nag cool-down, and decides on a move. Emits one event per row that needs action: `tc.first_ask` or `tc.reminder`, with the candidate context as the event payload. Wait/stop emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*
- **`dispatch`** — EventBridge rule on the two ask events. Resolves the contact, checks quiet hours and holiday calendar, formats the ask from the voice template, and ships via SES `SendRawEmail` with a signed reply-form link. On a quiet-hours or holiday defer, creates a one-off EventBridge Scheduler rule that re-invokes `dispatch` at the next available business minute. Writes a row to `tc-asks` after a successful send. Memory: 256 MB. Timeout: 30 s.
- **`reply-handler`** — Lambda Function URL, public with `AuthType: NONE`; serves the reply form (GET, with a signed token tying it to the candidate) and accepts the submission (POST). On submit: writes the raw reply and the permission choice to `tc-state` and the list, untouched, first. If permission is granted, calls Bedrock Haiku 4.5 once to clean the text into a quote and once for the

faithfulness check, then posts a Slack review card via `chat.postMessage`. If permission is declined, marks the candidate declined and writes the year-long do-not-ask entry. Memory: 512 MB. Timeout: 30 s.

- **`signoff-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a Slack signature on the request body. Triggered by Slack interactive button clicks (Approve/Edit/Discard). Writes to `tc-state` and `tc-audit`; on approve (or a small edit), copies the quote to the approved sheet via the Sheets API; on a large edit, flags for re-consent rather than auto-approving. Memory: 256 MB. Timeout: 15 s.
- **`digest`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads `tc-asks` and `tc-state` for the past week; sends a digest message to a configured Slack channel summarizing asks sent, replies received, and quotes awaiting sign-off. No Bedrock; the message is a plain summary table. Memory: 256 MB.
- **`summary`** — EventBridge Scheduler target, monthly on the first Monday at 9am. Reads the past month's `tc-asks`, `tc-state`, and `tc-audit`; calls Bedrock Haiku 4.5 to write a one-paragraph narrative (asks, reply rate, approvals, best new quotes); emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `tc-asks`** — one row per email sent. PK `(customer_id, step)`; attributes: `ask_date`, `sent_via` (email), `step` (first_ask/reminder), `moment`. On-demand. No TTL.
- **DynamoDB · `tc-state`** — one row per state change. PK `customer_id`; sort key `state_date`; attributes: `state` (replied/declined/approved/discarded),

`permission` (bool), `raw_reply`, `clean_quote`, `do_not_ask_until` (if declined). On-demand.

- **DynamoDB** · `tc-audit` — one row per write action of any kind. PK `(customer_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term consent and approval trail.
- **DynamoDB** · `tc-published` — mirror of the approved sheet for fast lookup of what's live. PK `customer_id`; attributes: `quote`, `approved_by`, `approved_at`, `permission_ref`. On-demand.
- **S3** · `tc-list-source` — mirrored CSV from the Drive candidate sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `tc-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3** · `tc-raw-mime` — raw inbound MIME from forwarded praise. Lifecycle to Glacier at 30 days; expiry at 7 years.
- **S3** · `tc-replies` — archived raw reply text and the permission record per submission, kept for the consent trail.

Bedrock

- **Foundation 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`. Three callsites: `intake-ses-parser` (praise classification), `reply-handler` (clean-up + faithfulness check), and `summary` (monthly narrative). Heavier reasoning isn't needed anywhere, so `anthropic.claude-`

`sonnet-4-6` is left out; if quote clean-up ever needed more nuance, the reply-handler is the one callsite that would justify it.

- **Embeddings.** Not used. Each reply is cleaned on its own and the list is structured rows; there's nothing to retrieve. No Knowledge Base, no S3 Vectors, no Titan Text Embeddings V2.
- **Quotas.** Default account quotas are more than enough at SMB volume. The collector itself doesn't call Bedrock; the model fires only on replies, forwarded praise, and the monthly summary.

EventBridge Scheduler config

- **`tc-daily-tick`** — `cron(0 9 * * ? *)` in the SMB's timezone. Target: `collector` Lambda.
- **`tc-drive-sync`** — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- **`tc-weekly-digest`** — `cron(0 18 ? * SUN *)` in TZ. Target: `digest` Lambda.
- **`tc-monthly-summary`** — `cron(0 9 ? * 2#1 *)` (first Monday at 9am) in TZ. Target: `summary` Lambda.
- **One-off rules** — created on the fly by `dispatch` when a quiet-hours or holiday defer is needed. Use `at(YYYY-MM-DDTHH:MM:SS)` expressions with `--action-after-completion DELETE` so the rule self-cleans.

SES inbound and outbound

- Set the MX record on a dedicated subdomain (e.g. `kudos.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.

- SES inbound rule set `tc-inbound-rules`: one rule with recipient `kudos@your-company.com` → spam scan → S3 PUT to `s3://tc-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-ses-parser`.
- SES outbound for the asks, reminders, and the monthly summary: verify a sender identity at `hello@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request. Keep a dedicated configuration set so bounce and complaint rates on the ask emails are tracked separately from transactional mail.

IAM (least privilege per Lambda)

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

- **collector role:** `s3:GetObject` on the list, rules, and voice keys; `dynamodb:Query` + `GetItem` on `tc-asks`, `tc-state`; `events:PutEvents` on the default bus. *No* `bedrock:*`.
- **dispatch role:** `events:ListSchedules` + `CreateSchedule` for the deferred one-offs; `secretsmanager:GetSecretValue` on the reply-form signing secret; `ses:SendRawEmail` from the verified sender identity; `dynamodb:PutItem` on `tc-asks`.
- **reply-handler role:** `dynamodb:PutItem` on `tc-state` and `tc-audit`; `s3:PutObject` on `tc-replies`; `bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the Slack bot token and the reply-form signing secret; outbound network to `slack.com`.
- **signoff-handler role:** `dynamodb:PutItem` on `tc-state`, `tc-audit`, `tc-published`; `secretsmanager:GetSecretValue` on the Sheets-API service-

account secret and the Slack signing secret; outbound network to `sheets.googleapis.com`.

- **intake-ses-parser role:** `s3:GetObject` on `tc-raw-mime`; `bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the Slack bot token.
- **drive-sync and ratings-hook roles:** `secretsmanager:GetSecretValue` on the Google service-account secret (and the ratings shared secret); `s3:PutObject` on the list and rules buckets; outbound network to `www.googleapis.com`.

Slack interactive flow

The alert and review messages are posted via the `chat.postMessage` Web API with Block Kit blocks containing the action buttons. Button clicks are sent by Slack to the configured Interactivity request URL, which is the `signoff-handler` Function URL (the praise-approval buttons from `intake-ses-parser` share the same handler, keyed by action_id). `signoff-handler` verifies the Slack signing secret on the inbound request, parses the action_id (`approve`, `edit`, `discard`, `praise_approve`, `praise_edit`, `praise_discard`), opens a modal when needed (Edit opens a modal; Approve and Discard are one-tap), and processes the response when the modal is submitted.

The Slack app needs `chat:write` and the Interactivity URL configured. The bot token lives in Secrets Manager under `tc/slack/bot-token`. The signing secret is `tc/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:** collector Lambda failures > 0 in a day (the daily tick is the one piece that has to run); SES bounce or complaint rate above the SES threshold (a noisy ask list is a real risk); signoff-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 `tc-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for the Drive and Sheets APIs live in Secrets Manager under `tc/drive/sa` (one service account with scopes for both). Slack bot token and signing secret under `tc/slack/*`. The ratings shared secret under `tc/ratings/secret` and the reply-form signing key under `tc/reply/signing`. SES sender identity lives in IAM and the verified-domain config. The configured timezone, holiday list reference, quiet-hours window, per-moment timings, never-nag windows, rating threshold, and `max_edit_distance` all live in Parameter Store under `/tc/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) running AWS SAM. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning for both `tc-list-source` and `tc-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 tick in UTC after a CI rotation. Total deployable surface: around nine Lambdas, four DDB tables, four S3 buckets, one EventBridge rule on the default bus (plus 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.