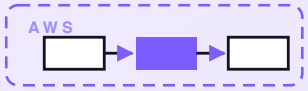
7-PART SERIES · FREE COMPANION



Quote follow-up

A serverless system that makes sure a sent quote never goes cold. After you send a quote, it follows up on a sensible cadence with a friendly nudge, watches for the customer's reply, and stops the moment they accept, decline, or ask a question — handing those straight to you. It flags quotes about to expire and tells you which deals are still open. Never pushy; a person handles every real reply. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Quote follow-up

PART 1 OF 7

JUNE 13, 2026 PART 1 OF 7 · QUOTE FOLLOW-UP SERIES ~5 MIN READ

A quote follow-up system on AWS for a few dollars a month

A small business sends more quotes than anyone keeps in their head. The kitchen remodel quote you emailed on a Tuesday and never heard back about. The catering proposal the customer loved on the call and then went quiet on. The annual service plan that's a yes if you just remind them once more. Most of those deals don't die because the price was wrong. They die because nobody followed up, or somebody followed up too hard. This post walks through the design of a small system that follows up on every sent quote on a sensible cadence, stops the second the customer replies, and hands every real reply to a person.

KEY TAKEAWAYS

- Three sources for tracked quotes: a Drive sheet, an inbox lane that reads sent-quote emails, and a webhook lane.
- Every quote ends in one of four moves on each check: resting, first nudge, follow-up nudge, or last call.
- A friendly cadence set in the rules doc — for example a nudge at 3 days, 7 days, and a few days before expiry.
- Nudges respect quiet hours and weekends. The moment a customer replies, the chain stops and a person takes over.
- Designed on AWS for about $2 a month at typical small-business volume.

The whole system on one page

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

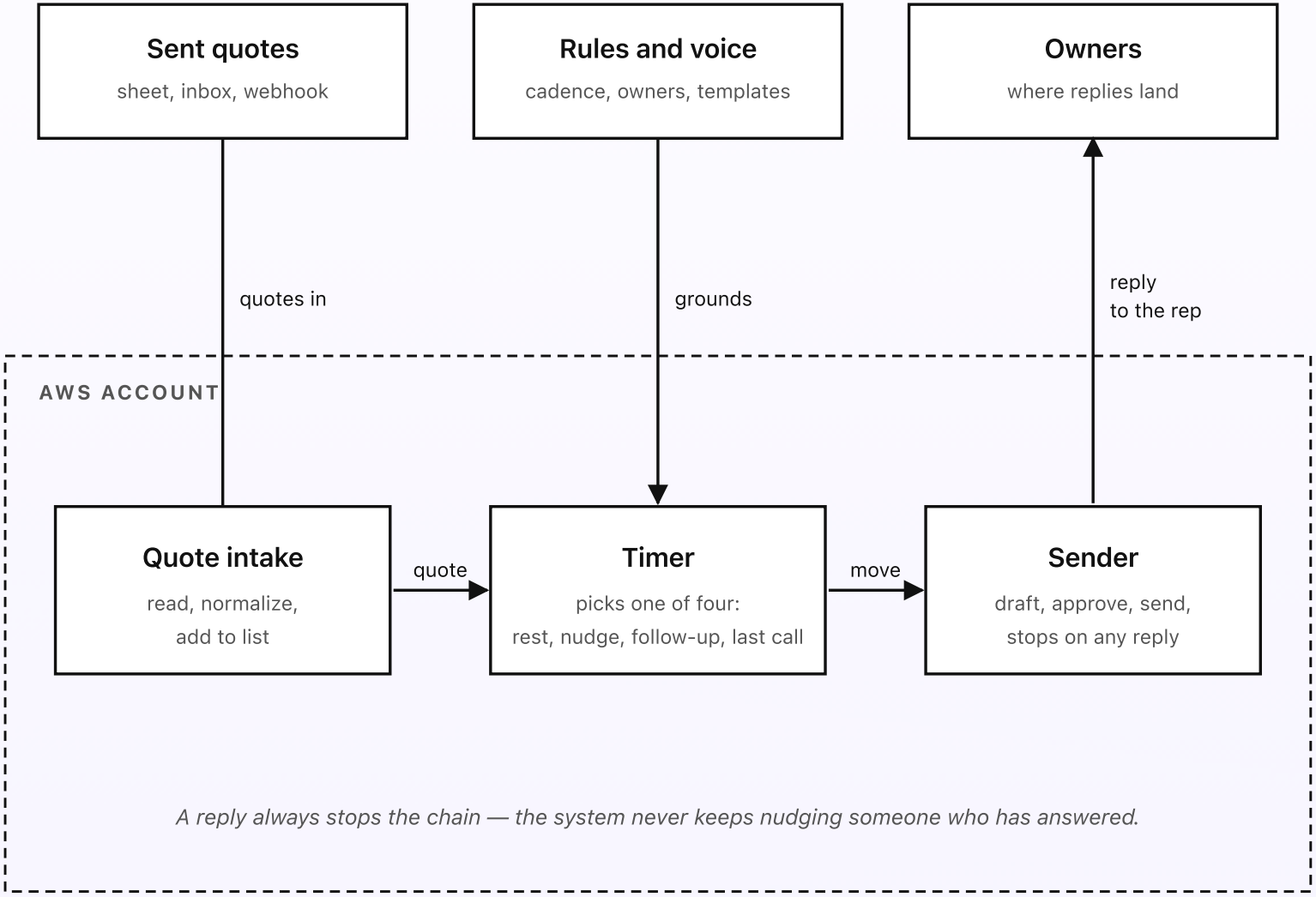

Sent quotes
sheet, inbox, webhook
Rules and voice
cadence, owners, templates
Owners
where replies land
quotes in
grounds
reply
to the rep
AWS ACCOUNT
Quote intake
read, normalize,
add to list
quote
Timer
picks one of four:
rest, nudge, follow-up, last call
move
Sender
draft, approve, send,
stops on any reply
A reply always stops the chain — the system never keeps nudging someone who has answered.

Fig 1. Three sources outside, three pieces inside AWS. Quotes flow in from a Drive sheet, an inbox lane, and a webhook lane. The Timer runs daily and picks one of four moves. The Sender drafts the right nudge, gets it approved, sends it, and stops the chain the moment the customer replies.

What you set up once (the outside)

- **Sent quotes.** A Google Sheet in a Drive folder, one row per quote: customer name, contact email, quote number, amount, the date you sent it, the date it expires, the owner (the rep who sent it), and a link to the quote PDF. You can fill it in once and forget it; new quotes can also enter via two other lanes covered in Part 2 — an inbox lane (forward or BCC the sent-quote email to a dedicated address and the system reads it and proposes a row for one-tap approval) and a webhook lane (your quoting tool posts a small message when you mark a quote as sent).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the follow-up cadence — how many days after sending the system should nudge, and how many times. A common cadence is a first nudge after 3 days, a second after 7, and a last call a few days before the quote expires. The doc also lists the owner per quote (or a default per rep), the quiet hours, whether to skip weekends, and the cap on how many nudges a single quote can get. The *voice* doc holds one nudge template per stage — the tone and wording the email actually uses.
- **Owners.** The rep who sent each quote. Each reply the customer sends, and each draft nudge waiting for approval, lands in that rep's inbox. Replies come with the quote name, days since you sent it, the amount, a link to the quote

PDF, and the customer's message in full, so the rep can pick it up without digging.

What runs on every check (the inside)

- **The quote intake.** Three sources feed the list. The Drive sheet itself is the canonical store. New quotes can also be added via the inbox lane (BCC the sent quote to `quotes@your-company.com`, the system reads the email with Bedrock Haiku 4.5 to pull out customer, amount, and quote number, then drops a one-tap approval in the rep's inbox before the row is added) and the webhook lane (your quoting tool posts to a Function URL when a quote is marked sent).
- **The timer.** Runs once a day at 9am local. Reads the list. For each quote, computes days since you sent it and days until it expires. Compares against the cadence in the rules doc. Picks one of four moves. *Resting*: not yet due for a nudge, or already replied — do nothing. *First nudge*: just crossed the first cadence step — draft a friendly check-in. *Follow-up nudge*: crossed a later step with no reply — draft a short, warmer follow-up. *Last call*: the quote is about to expire with no reply — draft a gentle "this is set to expire" note. The timer itself doesn't call a model — the move logic is plain Python.
- **The sender.** Reads the voice doc, drafts the nudge for the chosen move in your voice with Bedrock, and shows it to the rep for one-tap approval before anything goes out. On approve, it sends the email via SES outbound, honoring quiet hours and weekends. Then it watches for the customer's reply via SES inbound. Any reply — accept, decline, or a question — stops the chain and forwards the reply to the rep. A weekly digest lists every open quote and which ones are about to expire. A monthly summary writes a short paragraph: quotes still open, total amount in play, oldest open quote.

In plain words

You email a $6,400 landscaping quote to a customer named Dave on a Monday. The quote expires in 30 days. The owner is you. The system rests for three days, then on Thursday it drafts a short, friendly check-in in your voice: “Hi Dave — just making sure the landscaping quote reached you. Happy to walk through any of it. *[link to quote PDF]*” You glance at it, tap approve, and it sends. Dave’s busy, doesn’t reply. A week later the system drafts a one-line follow-up; you approve that too. The morning after, Dave replies: “Looks good — can we start the week of the 20th?” The system reads that as an accept, stops every future nudge, and drops the reply straight in your inbox marked “Dave replied — looks like a yes.” You take it from there. Dave never got a third message, and you never had to remember to send the first two.

The cost of running this is about $2 a month at SMB volume. The cost of *not* running it is the quote that sat unread for a week, the deal that went to a competitor who simply followed up once, or the customer who would have said yes if anyone had asked again.

DESIGN RULES THAT SHAPED EVERY DECISION

- A reply always stops the chain. The customer never gets nudged after they've answered — in any way.
- Four moves, always. Resting, first nudge, follow-up nudge, last call. There is no fifth.
- Nudges are capped and gentle. Quiet hours and weekends are respected; nobody gets a fourth message.
- A person approves every nudge before it sends, and a person handles every real reply. Nothing irreversible auto-sends.
- The quote list lives in Drive. Adding a quote, changing an owner, or shifting an expiry doesn't need a deploy.
- Every send and every reply is logged. Review a deal next quarter and you can see every message that went out.

Why this shape

Most teams follow up on quotes in one of three places: a spreadsheet that nobody opens, a calendar reminder that gets dismissed, or somebody's memory. The spreadsheet works until it doesn't — one busy week and three quotes go cold. The calendar reminder is the worst kind of false comfort: it pings on the day, with no context, and you still have to write the email yourself. And memory, of course, fails the moment a rep is on the road or juggling ten live deals.

The setup above keeps the source of truth in a sheet the team already edits, but adds a small system that *looks at* that sheet every day and acts only when a quote is going quiet. Nudges are drafted for you, in your voice, so approving one takes a second. They're gentle and capped, so you never become the company that pesters. They stop the instant the customer says anything. And every real reply lands with a person, because closing a deal is not something you want a robot doing on its own.

The next four posts walk through each piece in turn: how a sent quote gets tracked, how a follow-up gets timed, how a quote nudge reaches the buyer, and how a reply stops the follow-ups and hands the deal to a person. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

JUNE 13, 2026 PART 2 OF 7 · QUOTE FOLLOW-UP SERIES ~4 MIN READ

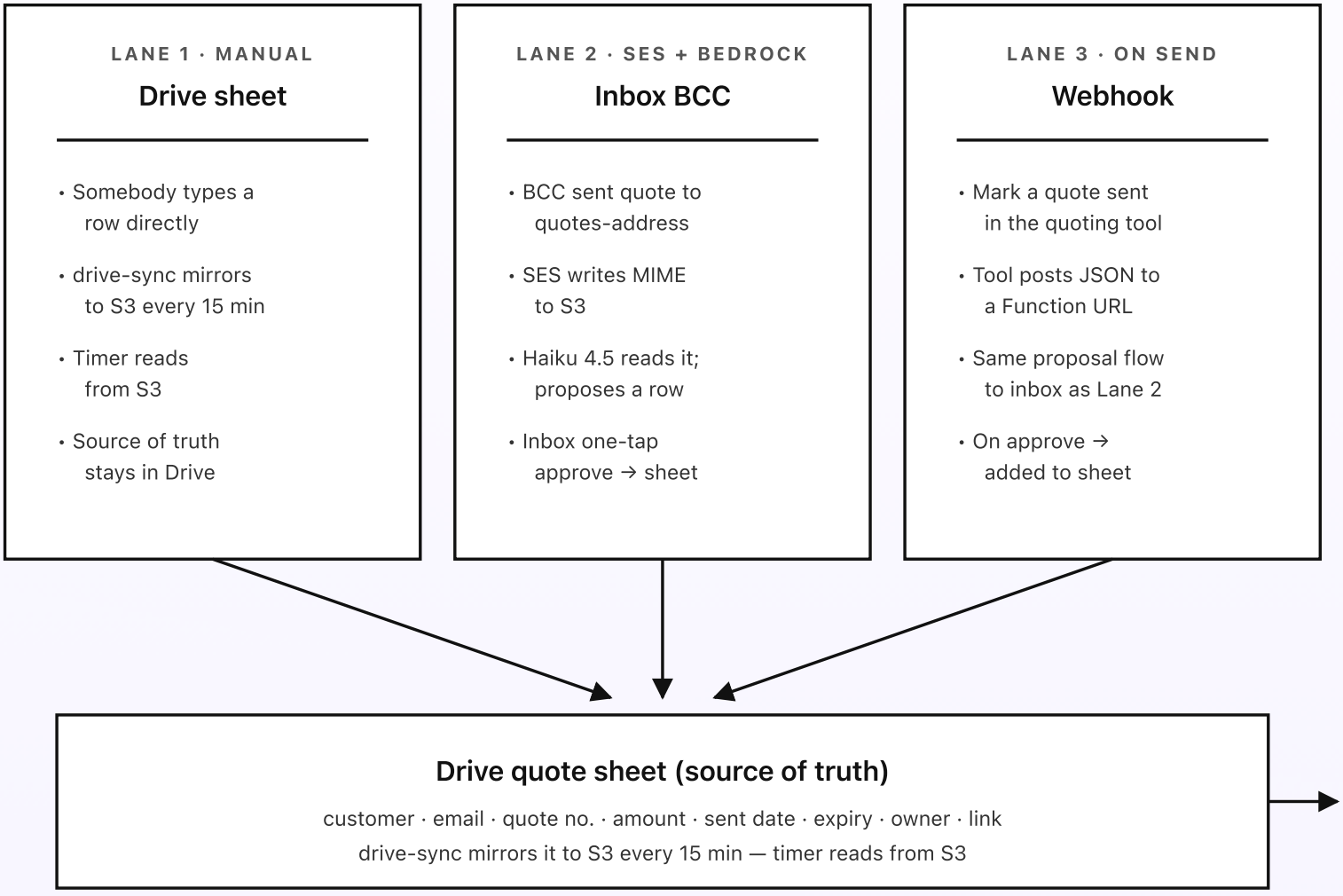
How a sent quote gets tracked

The system only follows up on what's in the list. So the first job is making sure the list actually reflects every quote you've sent. There are three ways a quote gets in: somebody types a row in the Drive sheet, somebody BCCs the sent-quote email to a dedicated address, or your quoting tool posts a small message when a quote is marked sent. The first one is obvious. The other two exist because in real life nobody types a row in a sheet for the quote they just emailed thirty seconds ago.

KEY TAKEAWAYS

- Three intake lanes feed one list: the Drive sheet, an inbox lane, and a webhook lane.
- Inbox emails are read by Bedrock Haiku 4.5, which proposes a row from the sent-quote message.
- Every proposed row goes to the rep’s inbox for one-tap approval before it lands in the list.
- Quoting tools that support webhooks can post a quote the moment it’s marked sent.
- The Drive sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one list



The Drive sheet stays the source of truth — the other lanes are conveniences that propose rows for it.

Fig 2. Three lanes converge on one Drive sheet. The sheet is the source of truth; the inbox lane and the webhook lane are conveniences that propose rows for human approval. The drive-sync Lambda mirrors the sheet to S3 so the timer can read it without hitting Drive on every check.

Lane 1: the Drive sheet itself

The simplest lane. Open the quote sheet in Drive, add a row, save. The columns are short: customer name, contact email, quote number, amount, the date you sent it, the date it expires, the owner, and a link to the quote PDF. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://qf-quotes-source/quotes.csv` if the sheet has changed since the last sync. The timer 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 sent the quote, you know the amount and the expiry, and you can spend thirty seconds typing it in. Most quotes go in this way if your quoting workflow is mostly email.

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

Set up a dedicated inbound address — something like `quotes@your-company.com` — via Amazon SES. When you email a quote to a customer, add that address as a BCC. The system takes it from there. SES writes the raw MIME to `s3://qf-raw-mime/`. The S3 PUT triggers a parser Lambda. The Lambda walks the MIME tree, pulls the email body and any attached quote PDF, and gets the text it needs.

Then a Bedrock Haiku 4.5 call reads the text and emits a structured row: customer name, contact email (the "To" line of your original email), quote number, amount, sent date (the message date), and an expiry if the quote states one. The model prompt is short: "Extract a row for the quote list. Return JSON only. Mark each field with a confidence score. Do not invent an amount or a date that isn't in the text." The output goes to a small email back to the rep who sent the quote: the proposed row, the confidence per field, and three links — *approve*, *edit*, *discard*. On *approve*, a Lambda writes the row to the Drive sheet via the Sheets API. On *edit*, the rep gets a short form pre-filled with the proposal. On *discard*, the message is logged and the email moved to a discarded prefix in S3 for audit.

The reason every proposed row goes to a person first is simple: a quote the model misread is worse than a quote that never made it into the list at all. The misread one will quietly nudge the wrong customer about the wrong amount.

Lane 3: webhook from your quoting tool

Some teams send quotes from a dedicated tool — an invoicing app, a proposal builder, a CRM. Many of those can post a small message (a webhook) when a quote is marked sent. Forcing those teams to also type rows in a sheet is a fight you don't need to have on day one.

Lane 3 accepts that message at a Lambda Function URL. The payload usually already has the customer, the amount, and the quote number, so the parser does very little — it normalizes the fields and runs the same proposal flow as Lane 2, so a person still confirms before the row lands. The Function URL checks a shared secret on every request so a stray post can't inject a fake quote. Once approved, the row is in the list and the timer picks it up on the next check.

Webhook import is the most opt-in of the three lanes. A team that doesn't use it loses nothing; a team that does avoids retyping things their quoting tool already knows.

Why the list stays the source of truth

Three lanes in, but only one place where the timer actually looks. That's a deliberate constraint. If two lanes both wrote directly to the timer's state, every "why did this nudge go out?" question would mean checking three places. Funneling everything through the Drive sheet means there is exactly one row per quote, and any rep can read or edit any of it without learning a new tool. The convenience lanes are first-class for getting quotes in, but they always pass through the sheet on the way.

Next post: how the timer actually reads the list, computes days since sent and days to expiry, and picks one of four moves.

PART 3 OF 7

JUNE 13, 2026 PART 3 OF 7 · QUOTE FOLLOW-UP SERIES ~5 MIN READ

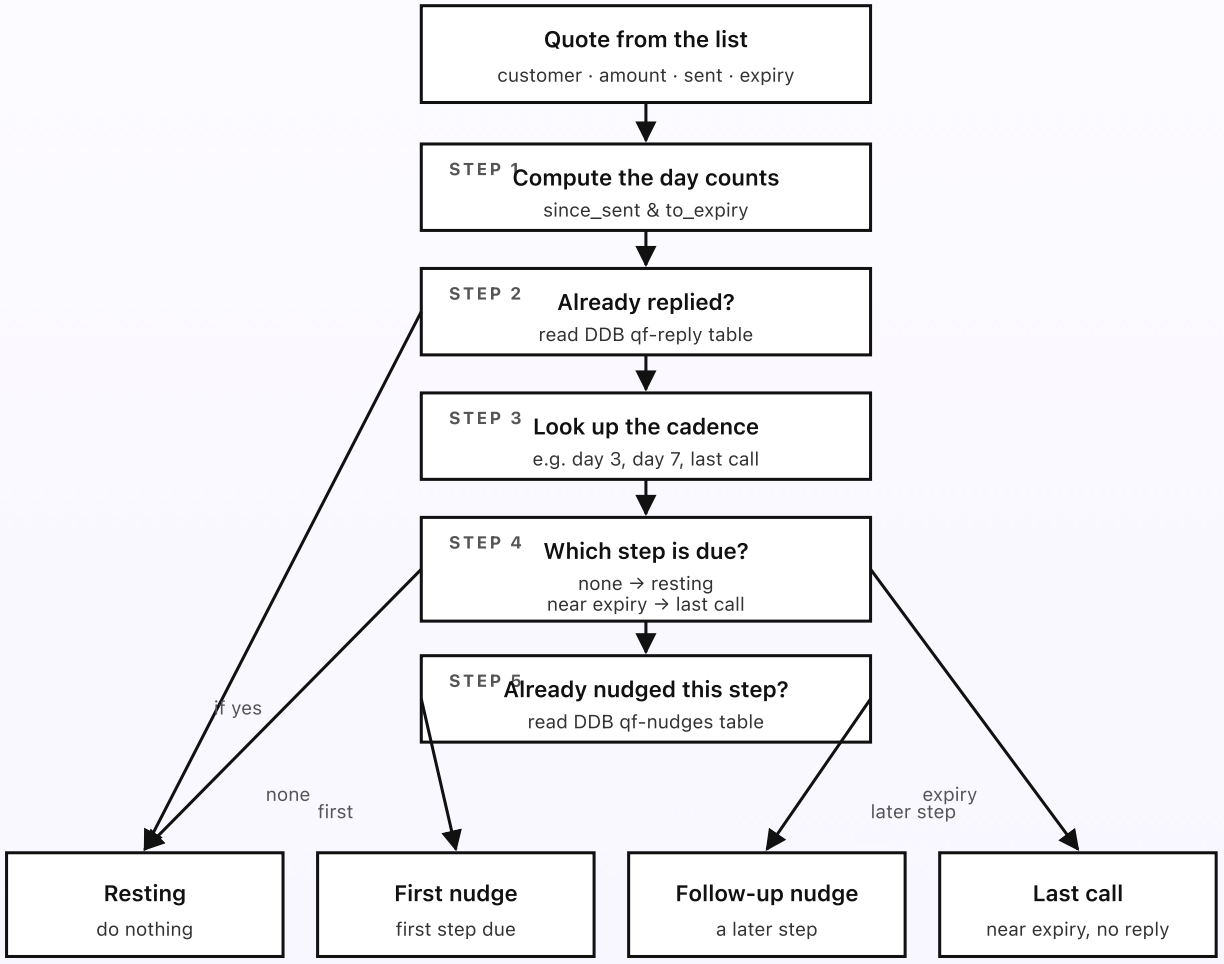
How a follow-up gets timed

Once a day, at 9am local time, an EventBridge Scheduler rule fires the timer Lambda. The Lambda reads the quote list, looks at one row at a time, computes how many days since you sent it and how many days until it expires, and decides whether to do nothing or to draft a nudge — and if so, which kind. The whole decision is plain Python. No model. No vector retrieval. Every threshold lives in the rules doc, where a rep can edit it without a deploy.

KEY TAKEAWAYS

- The timer runs once a day via EventBridge Scheduler at 9am local time.
- The cadence lives in the rules doc — for example a first nudge at 3 days, a second at 7, and a last call a few days before expiry.
- Four moves per quote, every check: resting, first nudge, follow-up nudge, last call.
- DynamoDB tracks last-nudge and reply state per quote so the same step never fires twice.
- The timer itself never calls a model. The decision is entirely deterministic.

The decision flow, per quote



The rules doc holds every threshold — change the cadence and tomorrow's check uses the new value.

Fig 3. The timer's decision tree, per quote, per daily check. Five steps decide which of four moves applies. The rules doc holds every threshold; the timer only enforces them.

The cadence: day 3, day 7, last call isn't magic, it's in the doc

The rules doc has one short section that names the cadence in plain prose: "Standard quotes: nudge at 3 days and 7 days after sending, then a last call 3 days before the quote expires. High-value quotes over $20,000: a gentler 5/12 cadence so it never feels like pressure. Small quotes under $500: a single nudge at 4 days, then stop." The numbers are days since you sent the quote. The first number is the first nudge. The last-call step is keyed to the expiry date instead — it fires a few days before the quote lapses, whatever the sent date was.

The cadence exists for a reason. A nudge at day 3 catches the quote that simply got buried in an inbox. A nudge at day 7 catches the customer who meant to reply and forgot. A last call before expiry is the honest "this price is about to change" reminder that often turns a maybe into a yes. Different deal sizes deserve different tempos; the cadence reflects that.

Per-quote overrides exist too. The quote sheet has an optional column called `cadence_override`. Type a comma-separated list of days there and the timer uses your numbers instead of the default for that one row. This is the right escape hatch for the customer who told you on the call "give me two weeks."

Four moves, always

Every quote, every check, lands in exactly one of four buckets. The names are simple on purpose.

- **Resting.** The quote isn't due for a nudge yet, or the customer has already replied. Do nothing. Most quotes, most days, are resting.
- **First nudge.** The first cadence step just came due and there's no reply yet. Draft a friendly check-in. Write a row to the `qf-nudges` DynamoDB table marking that the first step has fired.
- **Follow-up nudge.** A later cadence step came due without a reply. Draft a short, warm follow-up that gently references the earlier message so the customer doesn't feel cold-emailed twice. Write the new nudge to `qf-nudges`.
- **Last call.** The quote is inside the last-call window before its expiry, still with no reply. Draft a gentle "this quote is set to expire on the 30th — happy to extend it if you need more time" note. Mark it in DynamoDB. This is the final message in the chain; after it, the quote rests until it expires and gets closed out.

State that makes the decision deterministic

The timer reads two DynamoDB tables every check. `qf-nudges` records every nudge that's gone out: `(quote_id, step_index, nudge_date, sent_via)`. `qf-reply` records every customer reply: `(quote_id, reply_date, outcome)`. With those two tables, the move-decision logic is a few dozen lines of Python and zero magic. A given quote with a given sent date, a given cadence, and a given reply/nudge history always produces the same move. Re-running the check produces no extra nudges (because the state in DDB shows what already fired).

When a customer replies, that's an explicit stop: a row in `qf-reply` moves the quote to resting forever. Part 5 covers exactly what each kind of reply does.

Why the daily check uses no model

The timer could call a model on the check to decide whether to nudge at all, or to judge the tone of the moment. It doesn't. Two reasons. First, the daily check should be the one part of the system that is utterly predictable — if the rules doc says nudge at day 7 and there's no reply, the nudge is queued. A model in that loop introduces variance the team can't reason about. Second, model calls cost money, and most days most quotes are resting, so the call would be wasted nine days out of ten.

Bedrock fires elsewhere — on the inbox lane in Part 2, on drafting the nudge wording in Part 4, and on reading the reply in Part 5. Not on the daily check. The timer itself is plain Python that reads a doc and writes events.

Next post: how a quote nudge reaches the buyer, how a draft gets written in your voice and approved, and how quiet hours and weekends are honored.

PART 4 OF 7

JUNE 13, 2026 PART 4 OF 7 · QUOTE FOLLOW-UP SERIES ~5 MIN READ

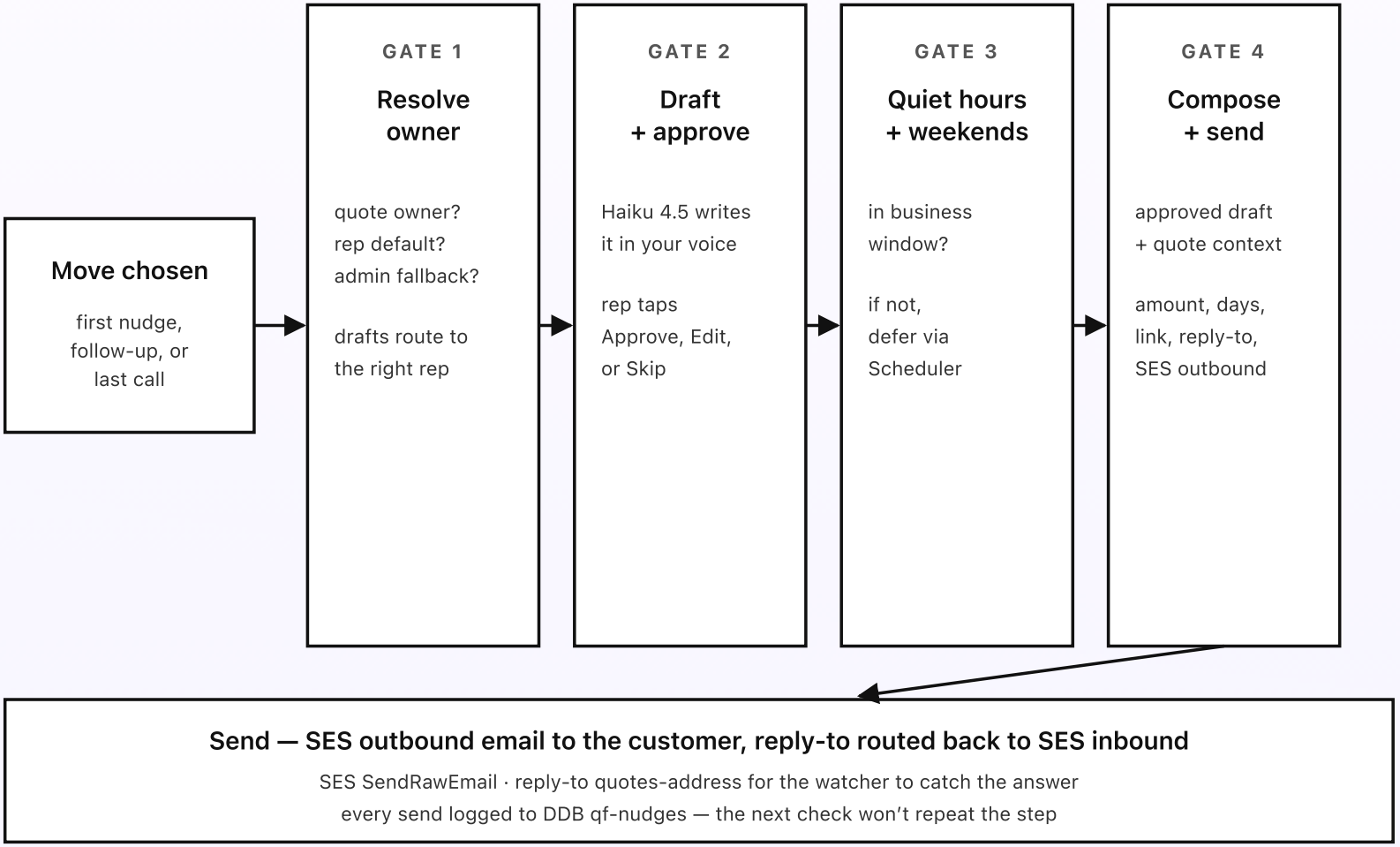
How a quote nudge reaches the buyer

The timer picked a move — first nudge, follow-up, or last call. Now the sender Lambda has to figure out who owns the quote, write a draft in your voice, get a person to approve it, and send it at a decent time of day. Get any of those wrong and the nudge is worse than no nudge: a generic “just checking in” with the wrong amount, an email at 11pm on a Saturday, a third message to someone who already said no. Four small guardrails sit between the move and the email landing.

KEY TAKEAWAYS

- Owner resolution: per-quote owner beats per-rep default beats fallback to the configured admin.
- Bedrock Haiku 4.5 drafts the nudge in your voice; the rep approves it with one tap before anything sends.
- Quiet hours and weekends defer the send to the next reasonable business hour.
- Every nudge ships with the quote name, amount, days since sent, a link to the quote PDF, and a clear reply path.
- Nothing auto-sends. A person approves every draft, so the customer never gets a robotic message.

Four guardrails on every nudge



A person approves every draft — the customer never gets a message nobody chose to send.

Fig 4. Four guardrails between the move and the sent nudge. Resolve the owner. Draft in your voice and get it approved. Honor quiet hours and weekends. Compose with full context and send. Then log the send so the next check doesn't repeat the step.

Gate 1: resolve the owner

Three places the sender Lambda looks for the owner of a quote, in order. First, the quote sheet's per-quote `owner` column — if a row names a specific rep, that rep owns it. Second, the per-rep default in the rules doc ("quotes from the catering team default to Priya"). Third, the configured admin fallback — the person who set up the system and gets every unowned quote. The fallback should never fire in steady state; if it does, the weekly digest names every quote that hit the fallback so the sheet can be corrected.

The owner matters because the draft goes to them for approval, and because the customer's eventual reply needs to land with the right person. Getting this wrong sends the approval to someone who doesn't know the deal, which is how a quote ends up with no follow-up at all.

Gate 2: draft in your voice, then approve

This is the gate that keeps the system from feeling robotic. The voice doc holds one short template per move — first nudge, follow-up, last call — with the tone you want and placeholders for the customer name, amount, and quote link. The sender calls Bedrock Haiku 4.5 with that template plus the quote details and asks for a short, warm draft that doesn't repeat itself across the chain. The prompt is strict: "Keep it under five sentences. Reference the quote by name and amount.

Do not invent terms, discounts, or dates. Sound like a helpful person, not a sales bot."

The draft is then emailed to the rep with three one-tap options: *Approve* sends it (subject to Gate 3), *Edit* opens it in a short form to tweak the wording, and *Skip* drops this nudge for this quote without affecting the rest of the chain. Nothing reaches the customer until the rep approves. Because the draft is already written, approving takes a couple of seconds — the rep is reviewing, not writing.

Gate 3: quiet hours and weekends

The timer runs at 9am local, but approvals can come back at any hour. A nudge approved at 8pm shouldn't land in a customer's inbox at 8pm, and a nudge approved Saturday morning shouldn't go out on the weekend if the rules doc says to skip weekends.

Gate 3 reads the rules doc's quiet-hours setting (default 6pm to 8am) and the skip-weekends flag. If the approved nudge falls in a quiet window or on a skipped day, the sender creates a one-off EventBridge Scheduler rule that fires at the next business-hour minute and exits without sending. The Scheduler re-invokes the sender with the same approved draft at the deferred time, where Gate 3 lets it through. Good timing is part of not being pushy: the same message lands very differently at 9am Tuesday than at 9pm Sunday.

Gate 4: compose with full context, then send

The sender fills the approved draft with the final quote context — customer name, amount, days since you sent it, and a link to the quote PDF — and wraps it in a clean email. Crucially, it sets the reply-to address to the dedicated `quotes@your-company.com` inbox, so when the customer answers, the reply comes straight back through SES inbound where the watcher can catch it (that's the whole of Part 5). The email itself sends via SES `SendRawEmail` from the rep's verified sender identity, so to the customer it looks like it came from the person they've been dealing with.

A last-call nudge is slightly different in wording — it names the expiry date and offers to extend — but it goes through the exact same four gates. There is no "urgent" path that skips approval.

Every send — first nudge, follow-up, or last call — writes a row to `qf-nudges` in DynamoDB. The next day's check reads that row and knows not to fire the same step again.

Why the guardrails exist

None of these gates are exotic. They're the kind of small care a thoughtful salesperson would take if they were sending the nudges themselves — check who actually owns this deal, write something human, don't email at midnight, include enough context that the customer doesn't have to ask "which quote?" Putting them in code as four small sequential gates makes them part of the design, not something you're trusting a busy rep to remember on a Friday afternoon.

Next post: how a reply stops the follow-ups — how the system reads the customer's answer, sorts it into accept, decline, question, or out-of-office, and

hands every real reply to a person.

PART 5 OF 7

JUNE 13, 2026 PART 5 OF 7 · QUOTE FOLLOW-UP SERIES ~5 MIN READ

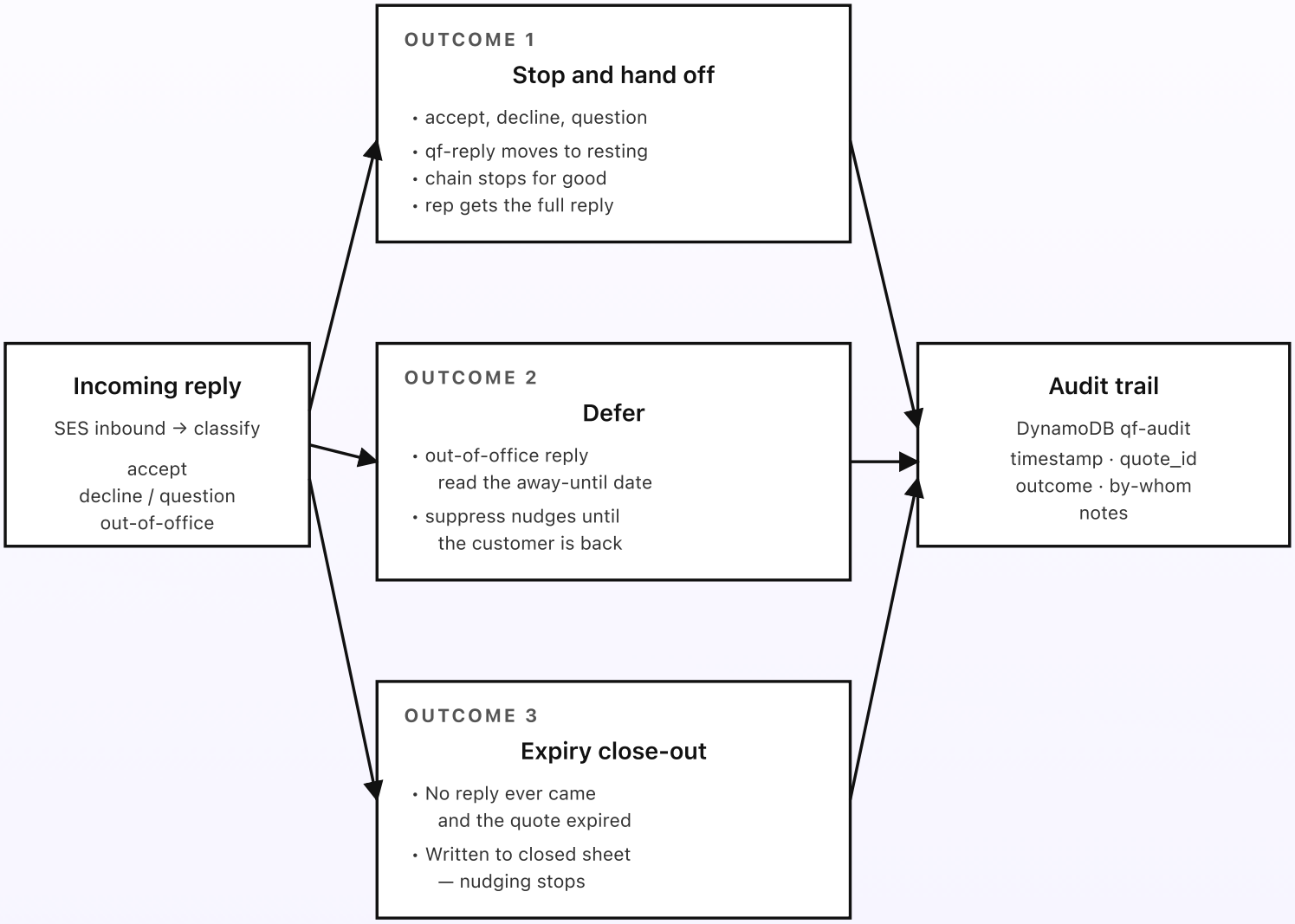
How a reply stops the follow-ups

A reply from Dave lands in the `quotes@` inbox at 8:03am: “Looks good — can we start the week of the 20th?” What happens next? The honest answer is “it depends on what the customer actually said.” This post walks through the three things the system can do when a reply comes in — stop and hand off, defer, or, when no reply ever comes, close the quote out at expiry — and how the list, the chain state, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three outcomes per reply: *stop & hand off* (accept, decline, or question), *defer* (out-of-office), and the *expiry close-out*.
- Bedrock Haiku 4.5 reads the reply and sorts it; a person always handles the real conversation.
- Accept, decline, and question all stop the chain and notify the rep with the reply attached.
- Out-of-office is the only reply that doesn’t stop the chain — it just pushes the next nudge.
- Every reply and every close-out writes a row to the audit trail.

Three outcomes on a reply



Only out-of-office keeps the chain alive — every real reply stops it and goes to a person.

Fig 5. Three outcomes per reply, three different effects. Stop and hand off ends the chain and sends the reply to the rep. Defer pauses the chain until the customer is back. The expiry close-out files a quote that never got an answer. Every outcome writes to the audit trail.

Reading the reply (a small, careful classify)

When the customer replies, the email lands in the `quotes@your-company.com` inbox — because Part 4 set that as the reply-to on every nudge. SES inbound writes the raw message to S3, which triggers a reply Lambda. The Lambda pulls the message text and the quote it belongs to (the subject line and a hidden reference tag both carry the quote id) and calls Bedrock Haiku 4.5 with one job: tag the reply as *accept*, *decline*, *question*, or *out-of-office*. The prompt is narrow: "Read this reply to a quote. Return one label only. If it's an automatic away message, say out-of-office. Do not summarize, do not reply."

The model only sorts the reply. It never writes back to the customer and never decides the deal. That's the line: AI is allowed to read and route, but a person handles every real conversation.

Outcome 1: stop and hand off (accept, decline, or question)

Three of the four labels — accept, decline, and question — all do the same structural thing: they stop the chain and hand the deal to a person. A Function URL Lambda writes a row to `qf-reply` with `(quote_id, reply_date,`

`outcome)`, which the timer reads in the “already replied?” check from Part 3 and treats as resting forever. No more nudges fire on that quote.

Then the rep gets an email with the full customer reply and a clear label at the top: *looks like a yes*, *looks like a no*, or *has a question*. A “yes” is the rep’s cue to send paperwork or schedule the work. A “no” is a chance to ask what changed, or just to file it. A “question” is a real conversation the system deliberately doesn’t try to have. The point of the label is speed of triage, not automation — the rep still reads the actual message before doing anything.

Outcome 2: defer (out-of-office)

Out-of-office is the one reply that *shouldn’t* stop the chain. The customer didn’t answer the quote; their mail server answered for them. Stopping the chain here would let a hot quote go cold just because the buyer was on holiday.

So the reply Lambda reads the away-until date from the auto-response (most include one) and writes a defer row to `qf-reply` that suppresses nudges until a couple of days after that date. If no date is present, it defers a sensible default — say five business days. The chain doesn’t reset; it simply pauses. When the customer is back, the next due nudge picks up where the cadence left off. The rep isn’t pulled in for an out-of-office, because there’s nothing for them to do yet.

Outcome 3: the expiry close-out (no reply ever comes)

Some quotes simply never get an answer. The last-call nudge went out, the chain finished, and the quote reached its expiry date with silence. The system does one

tidy thing here: the daily timer writes the quote to a separate *closed* sheet in the same Drive folder, with the date it expired and its last-known status, and stops considering it. The closed sheet is what the monthly summary in Part 6 reports on, so "quotes that went nowhere this month" is a number the owner can actually see and learn from.

A quote that was deferred and then expired without the customer ever truly replying ends up here too, with a note — "out-of-office on 2026-06-02, never resumed" — so the trail isn't a mystery later.

Every outcome is logged, every outcome is reversible

The `qf-audit` table records every reply outcome and every close-out with who (or what) caused it, the timestamp, and a snapshot of the quote's state before and after. If the classify gets a reply wrong — a sarcastic "sure, whatever" read as an accept — the rep can run a small "reopen quote" command that restores the chain from the snapshot and lets the cadence continue. The reopen is itself an audit row, so the trail of changes stays clean.

This kind of reversibility matters because the reply is where money is on the line. A mis-sorted reply that silently kills a live deal is the one failure mode worth guarding against hardest, which is exactly why a person sees every accept, decline, and question before anything happens on 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.

PART 6 OF 7

JUNE 13, 2026 PART 6 OF 7 · QUOTE FOLLOW-UP SERIES ~3 MIN READ

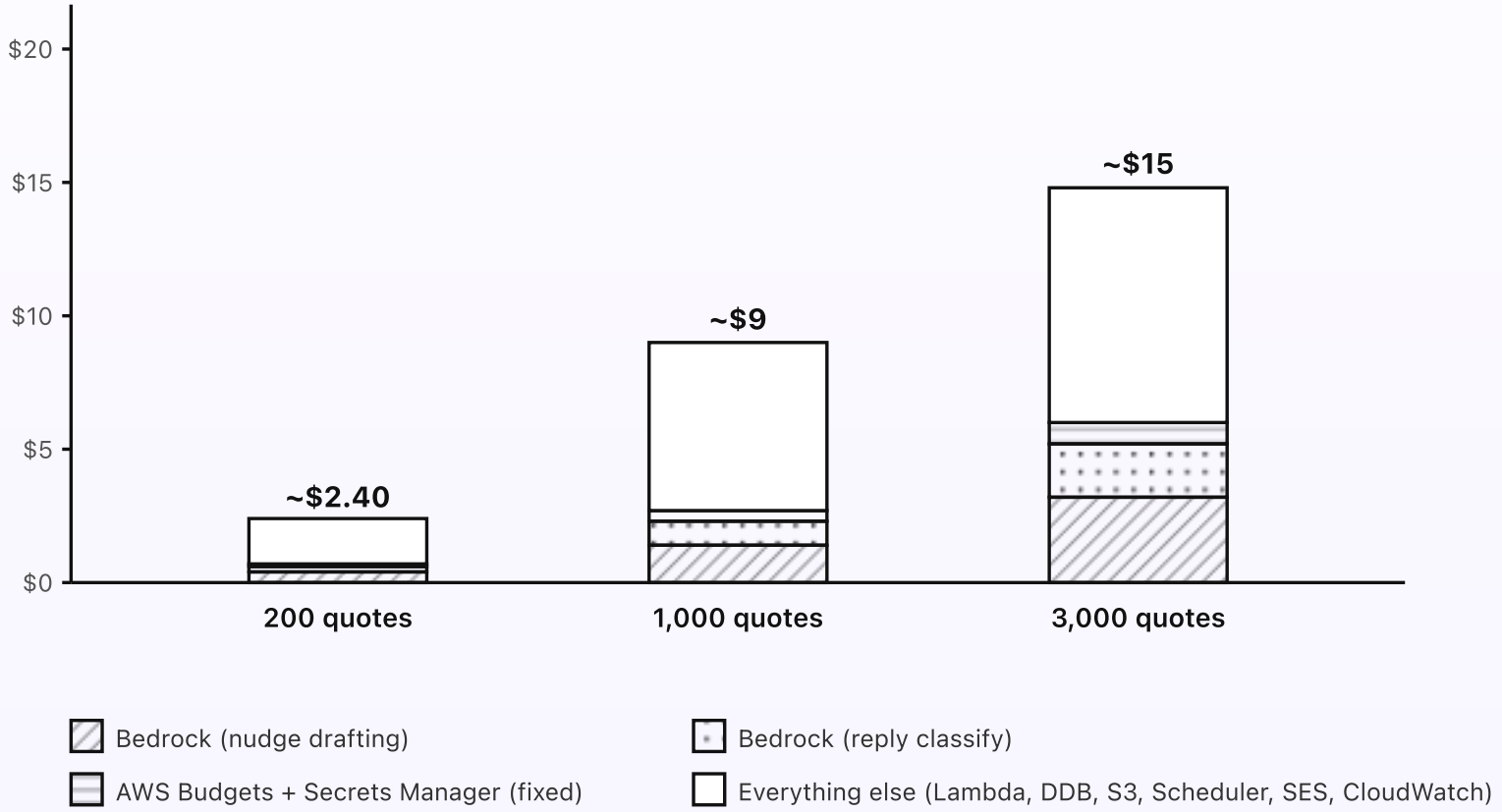
What the quote follow-up costs

The quote follow-up is one of the cheapest systems in this whole series. The daily check reads a CSV from S3, does some date arithmetic, writes a few rows to DynamoDB, and queues a handful of drafts. It calls no model on the check. Bedrock fires only to read each incoming reply and to draft each nudge — both small, both occasional. At typical SMB volume, the bill is a couple of dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2.40/month at typical SMB volume (around 200 open quotes).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily check costs pennies — no model calls.
- Bedrock fires only to read a reply and draft a nudge — a few cents at this volume.
- At 1,000 open quotes the bill is around $9. At 3,000 it's around $15.

Cost at three volumes



The daily check is the dominant cost — and even that is fractions of a cent per quote per day.

Fig 6. Monthly cost at three open-quote volumes. The Bedrock slices stay small because the model only reads replies and drafts nudges. The dominant cost is the everything-else bucket: the daily check reading every quote.

Where the dollars actually go

Lambda runtime (the bulk). The timer runs once a day. Each check reads the quote CSV from S3, iterates the rows, computes the day counts for each, and decides on a move. At 200 quotes, that's a few hundred milliseconds. At 3,000 quotes it's a couple of seconds. Either way it's pennies a month. Add the sender Lambda firing for each nudge (a handful a day), the reply Lambda for each incoming answer, the Function URL Lambda for webhook intake and approvals, 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: `qf-nudges`, `qf-reply`, `qf-audit`. Reads are dominant during the daily check (one read per quote per check). Writes are nudge sends, reply outcomes, and audit rows. Pennies a month at any of these volumes.

S3 + storage. The mirrored quote CSV plus the archived MIME from forwarded quotes and incoming replies. A few hundred KB total at SMB volume. Effectively free.

EventBridge Scheduler. The daily check rule plus deferred send rules from quiet-hours and weekend gates. A few invocations a day. Pennies.

SES. Inbound for the BCC lane and the customer replies: $0.10 per thousand received messages (so a couple of cents a year for an SMB). Outbound for the nudges: $0.10 per thousand sent. Both are negligible at this scale.

Bedrock (nudge drafting). The bigger of the two model lines. Each nudge is one short Haiku 4.5 call — a few hundred input tokens (the template plus the quote

details) and a few hundred output tokens (the draft) — so a fraction of a cent per draft. At a few hundred nudges a month, this is cents to a dollar or two.

Bedrock (reply classify). The smaller line. Each incoming reply is one tiny Haiku 4.5 call that returns a single label — a handful of tokens. Even at thousands of replies a month, this stays in the low single dollars.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the webhook and approval endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The timer sleeps 23.99 hours a day.
- **A Knowledge Base.** The quote list is structured rows, not free text — deterministic lookup beats vector search here. No embeddings, no Knowledge Base, no S3 Vectors needed.
- **Models on the check.** The daily timing decision is plain Python. Bedrock fires only to draft a nudge and read a reply.

How the cost scales

Lambda runtime grows roughly linearly with quote count, because every open quote is evaluated on every check. DynamoDB grows linearly too. The Bedrock lines grow with activity — more quotes mean more nudges drafted and more replies read — but each call is so cheap that they stay slivers. So the bill at 5,000 open quotes is around $25; at 10,000 it's around $50. Past those volumes the

daily-full-scan model probably stops being right (you'd switch to a partial check that only evaluates quotes inside an active cadence window), but those are optimizations for very large pipelines — not redesigns.

Set an AWS Budgets alarm at $25/month so anything unusual pages you before the bill matters. The system'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

JUNE 13, 2026 PART 7 OF 7 · QUOTE FOLLOW-UP SERIES ~8 MIN READ

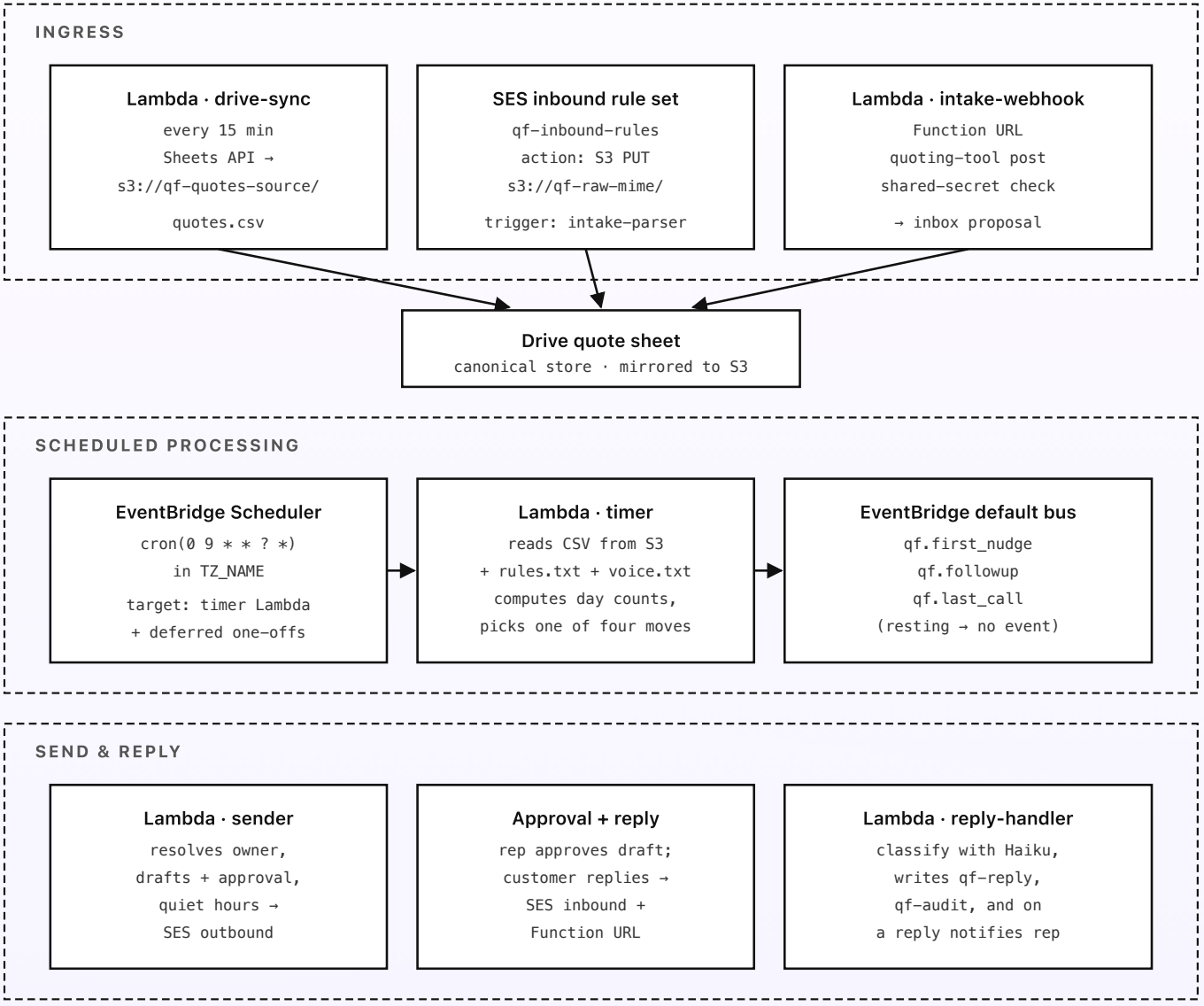
Engineering reference: the quote follow-up architecture

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 approval 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 quote going cold, not a regional outage. One AWS account dedicated to the system (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole thing.

Topology



A reply always stops the chain — and every interaction is logged to qf-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the quote list), scheduled processing (the daily timer emitting events), send and reply (the nudge is drafted, approved, and sent; the customer's reply is classified and 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 `qf/drive/sa`) to export the quote sheet as CSV and write to `s3://qf-quotes-source/quotes.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://qf-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`intake-parser`** — S3 PUT trigger on `s3://qf-raw-mime/`. Parses MIME, pulls the email body and any quote PDF, and calls Bedrock Haiku 4.5 (`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`) to propose a quote row: customer, contact email, quote number, amount, sent date, expiry. Posts the proposal to the rep's inbox via SES with Approve/Edit/Discard links backed by the `intake-webhook` Function URL. Memory: 512 MB. Timeout: 60 s.

- **`intake-webhook`** — Lambda Function URL, `AuthType: NONE` with a shared-secret check on every request. Accepts a quoting-tool post when a quote is marked sent, normalizes the payload, and runs the same proposal-and-approval flow as `intake-parser`. Also serves the Approve/Edit/Discard links from the inbox proposals. Memory: 256 MB. Timeout: 15 s.
- **`timer`** — 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://qf-quotes-source/quotes.csv` and the rules and voice docs. For each row, computes `days_since_sent` and `days_to_expiry`, reads chain state from `qf-nudges` and `qf-reply`, decides on a move. Emits one event per quote that needs a nudge: `qf.first_nudge`, `qf.followup`, or `qf.last_call`, with the quote context as the event payload. Resting quotes emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*
- **`sender`** — EventBridge rule on the three move events. Resolves owner, calls Bedrock Haiku 4.5 to draft the nudge from the voice template, emails the draft to the rep for one-tap approval, checks quiet hours and weekends, and ships the approved nudge via SES `SendRawEmail` with reply-to set to the quotes inbox. On a quiet-hours or weekend defer, creates a one-off EventBridge Scheduler rule that re-invokes `sender` at the next available business minute. Writes a row to `qf-nudges` after a successful send. Memory: 512 MB. Timeout: 30 s.
- **`reply-handler`** — triggered by SES inbound (customer replies to the quotes address) and by the email-link approval clicks via Function URL. Calls Bedrock Haiku 4.5 to classify the reply as accept/decline/question/out-of-office. Writes to `qf-reply` and `qf-audit`; on accept, decline, or question, notifies the rep

with the full reply; on out-of-office, writes a defer row instead. Verifies the quote reference (subject tag) before acting. Memory: 256 MB. Timeout: 30 s.

- `digest` — EventBridge Scheduler target, weekly Monday 8am. Reads `qf-nudges` and the quote list; emails the owner a digest of every open quote, which ones got a nudge that week, and which are about to expire. 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 `qf-nudges`, `qf-reply`, and the closed sheet; calls Bedrock Haiku 4.5 to write a one-paragraph narrative of quotes still open, total amount in play, and quotes that went nowhere; emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `qf-nudges`** — one row per nudge sent. PK `(quote_id, step_index)`; attributes: `nudge_date`, `sent_via` (email), `recipient`, `move` (first_nudge/followup/last_call). On-demand. No TTL.
- **DynamoDB · `qf-reply`** — one row per reply or defer. PK `quote_id`; sort key `reply_date`; attributes: `outcome` (accept/decline/question/out-of-office), `defer_until` (if out-of-office), `by_user` (the rep, once they pick it up). On-demand.
- **DynamoDB · `qf-audit`** — one row per write action of any kind. PK `(quote_id, ts)`; attributes: `action`, `by_user` (or `by_system`), `before`, `after`. On-demand. No TTL — this is the long-term audit trail.

- **DynamoDB** · `qf-approvals` — pending draft approvals keyed by a one-time token. PK `token`; attributes: `quote_id`, `move`, `draft_body`, `created_at`. TTL 7 days so stale drafts self-clean. On-demand.
- **S3** · `qf-quotes-source` — mirrored CSV from the Drive quote sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `qf-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3** · `qf-raw-mime` — raw inbound MIME from BCC'd quotes and customer replies. Lifecycle to Glacier at 30 days; expiry at 7 years.
- **S3** · `qf-source-pdfs` — the parsed quote PDFs after the inbox parser handles them, kept for reference if the quote row links to one.

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-parser` for reading a sent-quote email, `sender` for drafting each nudge, and `reply-handler` for classifying each reply. If a heavier reasoning task ever appears (e.g. summarizing a long multi-thread negotiation for the monthly narrative), `sender` / `summary` can escalate that one call to `anthropic.claude-sonnet-4-6-20250930-v1:0` via its Global profile — but the hot paths stay on Haiku.
- **Embeddings.** Not used. The quote list is structured rows; deterministic lookup beats vector retrieval here. No Knowledge Base, no S3 Vectors. (If a future version needs to retrieve past quote language for tone matching, Amazon Titan

Text Embeddings V2 at 1024-dim into Amazon S3 Vectors is the path — not needed today.)

- **Quotas.** Default account quotas are more than enough at SMB volume. The timer itself doesn't call Bedrock; drafting and classify are short calls.

EventBridge Scheduler config

- `qf-daily-check` — `cron(0 9 * * ? *)` in the SMB's timezone. Target: `timer` Lambda.
- `qf-drive-sync` — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- `qf-weekly-digest` — `cron(0 8 ? * MON *)` in TZ. Target: `digest` Lambda.
- `qf-monthly-summary` — `cron(0 9 ? * 2#1 *)` (first Monday at 9am) in TZ. Target: `summary` Lambda.
- **One-off rules** — created on the fly by `sender` when a quiet-hours or weekend 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. `quotes.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `qf-inbound-rules`: one rule with recipient `quotes@your-company.com` → spam scan → S3 PUT to `s3://qf-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-parser` for BCC'd quotes and `reply-handler` for customer replies (the message subject tag distinguishes them).

- SES outbound for the nudges and the rep notifications: verify each rep's sender identity (e.g. `priya@your-company.com`) with DKIM and SPF on the parent domain. Out of sandbox by request. Nudges send from the owning rep's identity so the customer sees a familiar name.

IAM (least privilege per Lambda)

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

- **timer role:** `s3:GetObject` on the quotes, rules, and voice keys; `dynamodb:Query` + `GetItem` on `qf-nudges`, `qf-reply`; `events:PutEvents` on the default bus. *No* `bedrock:*`.
- **sender role:** `events:ListSchedules` + `CreateSchedule` for the deferred-send one-offs; `bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the rep sender-identity config; `ses:SendRawEmail`; `dynamodb:PutItem` on `qf-nudges` and `qf-approvals`.
- **reply-handler role:** `s3:GetObject` on `qf-raw-mime`; `bedrock:InvokeModel` on the Haiku ARN; `dynamodb:PutItem` on `qf-reply` and `qf-audit`; `ses:SendRawEmail` for the rep notification; `dynamodb:Query` for chain-state lookup.
- **intake-parser role:** `s3:GetObject` on `qf-raw-mime`; `bedrock:InvokeModel` on the Haiku ARN; `ses:SendRawEmail` for the inbox proposal; `dynamodb:PutItem` on `qf-approvals`.
- **intake-webhook role:** `dynamodb:PutItem` on `qf-approvals`; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret

and the webhook shared secret; outbound network to `sheets.googleapis.com` for the on-approve write.

- **drive-sync role:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the quotes and rules buckets; outbound network to `www.googleapis.com`.

Approval and reply flow

Every nudge is draft-then-approve. `sender` writes the draft to `qf-approvals` keyed by a one-time token and emails the rep a short message with three signed links (Approve/Edit/Skip) that resolve to the `intake-webhook` Function URL. Approve flips the row and triggers the actual send (subject to the quiet-hours and weekend gates); Edit returns a pre-filled form; Skip records the skip in `qf-audit` without affecting the rest of the chain. Tokens are single-use and the `qf-approvals` TTL clears stale drafts after 7 days.

On the reply side, every outbound nudge carries a hidden quote reference (a tag in the subject and a header). When the customer replies to the quotes address, `reply-handler` reads that reference, classifies the body, and acts. The reference is what lets a single inbound address fan replies back to the exact quote without guessing.

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:** timer Lambda failures > 0 in a day (the daily check is the one piece that has to run); sender failure rate > 1% in 24h; reply-handler classify failures > 5/hour (might mean a malformed inbound or a Bedrock throttle).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $25/month threshold, alarm at 80% and 100%, posts to SNS topic `qf-cost-alarm` subscribed to the on-call admin's email.

Config and secrets

Service-account credentials for Drive and Sheets APIs live in Secrets Manager under `qf/drive/sa` (one service account with scopes for both APIs). The quoting-tool webhook shared secret lives under `qf/webhook/secret`. SES sender identities live in IAM and the verified-domain config. The configured timezone, quiet-hours window, skip-weekends flag, default cadence, nudge cap, and admin fallback owner all live in Parameter Store under `/qf/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 both `qf-quotes-source` and `qf-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 check in UTC after a CI rotation. Total deployable surface: around eight 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.