7-PART SERIES · FREE COMPANION

Review responder

A serverless responder that watches every new review across Google, Facebook, and Yelp, drafts replies in your voice from your own policies, posts the safe ones automatically, and quietly hands you the rest with everything you need to respond in one sitting. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Review responder

01 A review responder on AWS for a few dollars a month

02 How a review reaches the responder

03 How the responder reads a review

04 How the responder picks a move

05 How a reply stays in your voice

06 What the review responder costs

07 Engineering reference: the review responder architecture

PART 1 OF 7

MAY 2, 2026 PART 1 OF 7 · REVIEW RESPONDER SERIES ~5 MIN READ

A review responder on AWS for a few dollars a month

Your business has thirty new reviews this month. Twenty-two are positive, four are mixed, three are angry, and one mentions a name and a date you don't remember. You meant to reply to all of them. You replied to two. Here's how to design a small responder that watches every new review across your platforms, drafts replies in your voice from your own policies, posts the safe ones automatically, and hands you the rest with everything you need to respond in one sitting.

KEY TAKEAWAYS

- Three review sources, three AWS pieces. Google Business Profile, Facebook, and Yelp fold into one internal queue.
- Every review ends in one of four moves: auto-reply, draft, escalate, ignore. There is no fifth.
- The composer writes only from your voice file and your policies file. It never invents a refund window, a phone number, or a promise.
- Anything 1- or 2-star, safety-flagged, or that names a staff member skips auto-reply and pings a human with a draft and the matched policy excerpt attached.
- Runs on AWS for about $3/month at typical small-business review volume.

The whole system on one page

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

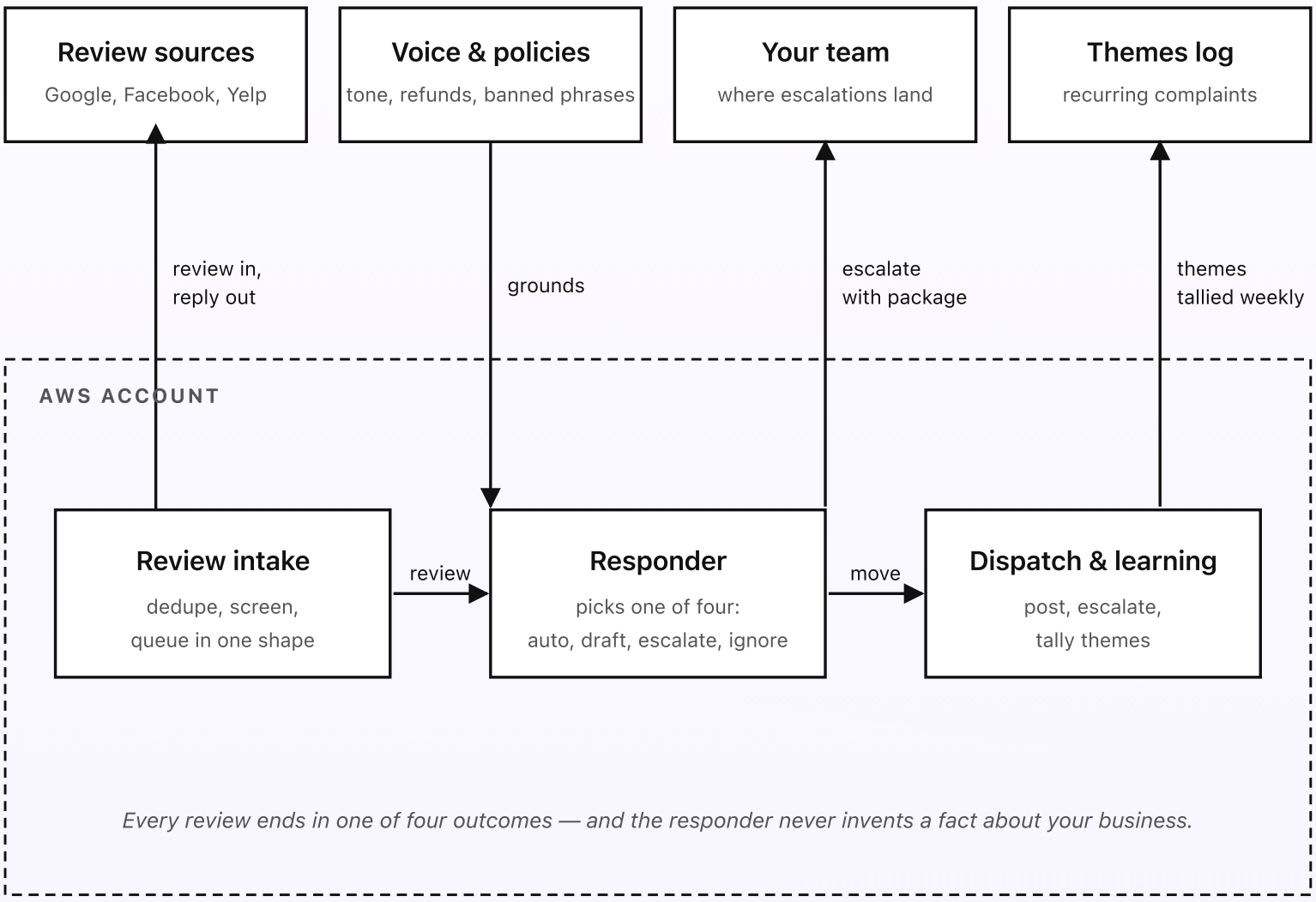

Review sources
Google, Facebook, Yelp
Voice & policies
tone, refunds, banned phrases
Your team
where escalations land
Themes log
recurring complaints
review in,
reply out
grounds
escalate
with package
themes
tallied weekly
AWS ACCOUNT
Review intake
dedupe, screen,
queue in one shape
review
Responder
picks one of four:
auto, draft, escalate, ignore
move
Dispatch & learning
post, escalate,
tally themes
Every review ends in one of four outcomes — and the responder never invents a fact about your business.

Fig 1. Four outside surfaces, three pieces inside AWS. Reviews flow in from your platforms, the Responder picks one of four moves, and replies go back out under the same accounts they were left under.

What you set up once (the outside)

- **Your review sources** — Google Business Profile, your Facebook page, your Yelp listing, connected through each platform's official integration. The responder polls or receives push events, never scrapes. You stay one click away from disconnecting any source if you don't like a reply that went out.
- **A voice-and-policies folder** — three short Google Docs in a Drive folder. *How you sound* (warm, brief, never defensive; example openers; signature line). *What you can promise* (refund window, replacement policy, escalation phone number, hours). *What you must never say* (specific phrases, comparative claims, anything legal or medical, any commitment beyond the policies file). Edit a doc, the responder picks up the change on the next refresh; no deploy.
- **A team destination** — the inbox, Slack channel, or shared queue you already use. Anything 1- or 2-star, anything that mentions a staff name, anything that hits a safety or legal keyword, anything the responder isn't confident about, lands here with the original review, the proposed draft, and the matching policies excerpt attached.
- **A themes log** — a small Drive sheet where recurring topics accumulate. "Wait time at the front desk" mentioned in seven reviews this quarter is one row in your operations meeting, not seven separate replies that don't fix anything.

What runs on every review (the inside)

- **The review intake** — receives or polls each platform, deduplicates by review ID so the same review is never processed twice, screens out obvious junk (profanity floods, off-topic links, banned-words spam), and writes the cleaned review to a single internal queue. By the time the responder sees a review, it's in one shape regardless of which platform it came from.
- **The responder** — for every queued review, reads the rating and the body, extracts what the customer is praising or complaining about, and picks one of four moves: *auto-reply* (the obvious thanks-yous), *draft for review* (anything with a specific complaint or named staff), *escalate* (anything 1-star, safety, or legal), or *ignore* (already replied, off-topic, junk that slipped past intake). When it composes, it pulls only from your voice file and your policies file — never invents a refund window, a phone number, or a promise.
- **Dispatch and learning** — on *auto-reply*, posts through the platform's API and marks the review answered. On *draft*, drops a package in your inbox or Slack: original review, proposed reply, matched policy excerpt, and a one-line reason it's a draft and not auto. On *escalate*, the same package, marked urgent. And on every review, regardless of move, extracts the themes the customer mentioned and increments them in the monthly log.

In plain words

A new review lands. The cloud reads it within a minute, decides "this is a 5-star thanks-for-the-coffee, my voice file says 'warm and brief' on these and the policies file doesn't apply," and posts the reply automatically. Another review lands. This one is 3-star and mentions a specific dish that disappointed; the cloud writes a draft that acknowledges the complaint and offers a return-visit credit

(which is in your policies file), and drops the draft in your inbox. You glance at it, change one word, hit send. A third review lands and it's 1-star with a phrase like "food poisoning"; the cloud doesn't try to compose a reply at all. It marks the review urgent, drafts a single-sentence acknowledgement — "We're sorry, we'd like to look into this; please reply with your visit details or call the number on your receipt" — and sends the whole package to your manager. You see all three, in one place, at the time of day you choose to deal with reviews.

Total cost runs in coffee-money territory at typical small-business volume — pennies per review, dominated by the few model calls per draft.

DESIGN RULES THAT SHAPED EVERY DECISION

- The responder composes from your voice file and your policies file only — never invents a refund window, a phone number, or a promise you don't offer.
- Four moves, always. Auto-reply, draft, escalate, ignore. There is no fifth.
- Negative or specific reviews never auto-post. Drafts are for you; auto is for the obvious thanks-yous.
- Safety-and-legal keywords (food poisoning, injury, lawyer, refund-stalled, named-staff complaint) bypass everything and escalate immediately.
- Configuration lives in Drive. Tweaking your tone or your refund window doesn't need a deploy.
- Recurring themes come back as a list, not as N replies. Use the list to fix the underlying problem.

Why this shape

Most "AI review reply" tools fall into one of two traps. The first kind auto-posts everything in a generic upbeat voice, which is fine until it auto-replies "Thank you for the kind words!" to a 1-star review about a missed delivery. The second kind drafts everything for human review, which sounds responsible but quietly trains you to skip the stack — by the third week the drafts pile up and nothing gets posted, exactly the problem you started with. Neither is what you want for the

review of your dental office that arrived at 11pm Saturday from a patient who was upset about a wait time.

The setup above splits the difference. The obvious thanks-yous post themselves in your voice; you don't read those, you just see the count went up. Specific complaints, mixed reviews, anything that needs a real human eye, becomes a one-tap draft — the cloud has done the writing, you keep the editorial control. And anything serious is shoved up the chain to a real person with all the context attached, because a 1-star review with a safety complaint isn't a "review" anymore, it's a triage event.

The next four posts walk through each piece in turn — how a review reaches the responder, how the responder reads it, how it picks one of four moves, and how the reply stays in your voice. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 2, 2026 PART 2 OF 7 · REVIEW RESPONDER SERIES ~4 MIN READ

How a review reaches the responder

Reviews don't arrive at your business through one door. Google sends an instant notification. Facebook used to push too, but Meta deprecated that webhook in early 2025 — in 2026 you reach Facebook reviews through a third-party aggregator instead. Yelp doesn't tell you at all — you have to ask. The intake's job is to fold those three different mechanisms into one queue, drop duplicates, and screen out junk before any AI sees a single word.

KEY TAKEAWAYS

- Google Business Profile pushes via Pub/Sub the moment a review lands; the cloud receives the notification and fetches the full review by ID.
- Meta deprecated the Facebook recommendations webhook in January 2025; in 2026 the Facebook lane runs through a third-party aggregator (Birdeye, Yext, ReviewTrackers).
- Yelp has no push for SMBs, so the cloud polls the Fusion endpoint hourly — well inside the 500-call free daily tier.
- Reply APIs differ: Google supports public posting; Facebook and Yelp are draft-only in 2026.
- One Lambda per lane normalizes into a common shape, dedupes by review ID, and runs cheap in-Lambda screens for spam before any AI runs.

Three lanes at the door

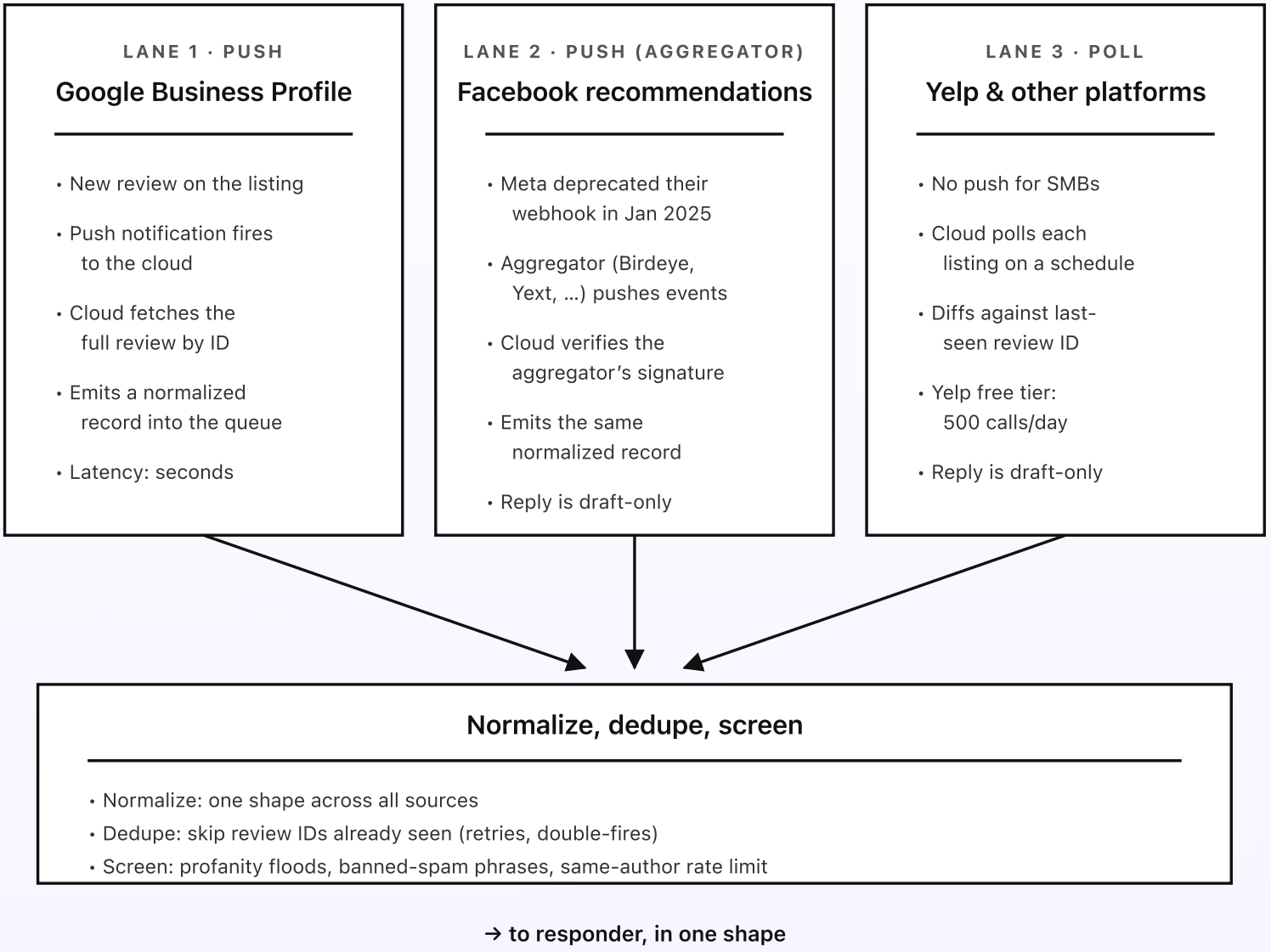

LANE 1 · PUSH
Google Business Profile
• New review on the listing
• Push notification fires to the cloud
• Cloud fetches the full review by ID
• Emits a normalized record into the queue
• Latency: seconds
LANE 2 · PUSH (AGGREGATOR)
Facebook recommendations
• Meta deprecated their webhook in Jan 2025
• Aggregator (Birdeye, Yext, …) pushes events
• Cloud verifies the aggregator's signature
• Emits the same normalized record
• Reply is draft-only
LANE 3 · POLL
Yelp & other platforms
• No push for SMBs
• Cloud polls each listing on a schedule
• Diffs against last-seen review ID
• Yelp free tier: 500 calls/day
• Reply is draft-only
Normalize, dedupe, screen
• Normalize: one shape across all sources
• Dedupe: skip review IDs already seen (retries, double-fires)
• Screen: profanity floods, banned-spam phrases, same-author rate limit
→ to responder, in one shape

Fig 2. Three lanes, one queue. Push when the platform offers it, poll when it doesn't. Cheap filters first — nothing reaches the AI until the platform-specific noise is stripped.

Why three different mechanisms

The three platforms make very different choices about how to tell a business their listing got a review, and the intake mirrors those choices honestly instead of pretending they're uniform.

Google pushes natively, and that's the easy lane. When a review lands on a Google Business Profile, Google fires a notification through Pub/Sub at a URL the cloud is listening on, and the responder can have the review in hand within seconds. Google is also the only one of the three that offers a public reply API to a regular small-business owner, so this lane is fully two-way: read in, reply out, no human in the middle for the safe cases.

Facebook used to push too, but stopped. Meta deprecated the Page recommendations webhook in early 2025 (Graph API v22.0); the old `ratings` field no longer fires, and there's no documented reply API for a recommendation. The realistic Facebook lane in 2026 goes through a third-party aggregator — Birdeye, Yext, ReviewTrackers, or whichever one the business already uses to monitor listings — that watches the page for you and forwards normalized events to a webhook URL you control. The shape of the lane is the same; the source of the push moved from Meta directly to whichever aggregator you connect. Replies on Facebook are draft-only: the responder writes the reply, the human pastes it into the Pages app.

Yelp doesn't push, and its reply API is enterprise-only. The honest workaround is a polling job that wakes up on a schedule, asks Yelp's public Fusion endpoint "anything new?", and queues whatever it didn't see last time. Yelp's free tier is 500 calls per day, so hourly polling per listing is well inside the budget. The same polling pattern works for niche platforms (Tripadvisor, Healthgrades, OpenTable, BBB, the platform every industry has its own one of). The rare review that lands at 2:47pm gets processed at 3pm, which is still inside the same business hour for any practical reply. Yelp's reply path is also draft-only for SMBs — the cloud writes the reply, the human pastes it into `biz.yelp.com`.

Push is faster, poll is more universal. Auto-reply works on Google today and is draft-only on Facebook and Yelp. Mixing all three in the same intake means the responder doesn't care which platform a given review came from once it's in the queue — the downstream code never branches on source, only on content. The auto-vs-draft difference is a single per-source flag the dispatch column reads at the very end.

What "normalize" actually means

Each platform hands the cloud a slightly different shape: Google gives a star rating from one to five and a free-text comment; Facebook offers a binary "recommends/doesn't recommend" and a comment; Yelp gives a star rating, a comment, and sometimes a photo. Normalization folds those into one common review object — a source name, a stable review ID, a numeric score on the same one-to-five scale (binary recommendations map to 5 or 2 with a flag), the comment text, the author display name, the timestamp, and a small bag of platform-specific extras for the engineering reference.

The reason for putting normalization here, before anything else, is so the rest of the system reads exactly one kind of message. The responder doesn't have to know what Yelp's rating field is called; it just reads `score` and `text`.

Dedupe and screen, before any AI runs

Two free filters sit between the lanes and the responder.

Dedupe is a one-line check against a small table of review IDs the cloud has already processed. Webhooks retry; platforms occasionally fire the same event twice if their backend gets unsure; and the polling job, by design, asks the same questions on a loop. Without dedupe you'd pay to draft the same reply twice and then have to figure out which copy actually got posted. With it, the second arrival is dropped silently and metrics-only.

Screen handles the obvious junk — banned-words spam (the same crypto-pump comment posted to a thousand businesses), profanity floods (the same review re-posted under different names by an angry ex-employee), or same-author rate-limit hits (one reviewer posting twelve reviews in an hour). The screen runs in plain Lambda code with a small banned-words list and a frequency check; no AI involved. Reviews that fail the screen don't reach the responder — they're logged for you to inspect (and report to the platform) but they don't cost a single model call.

The point of doing both before the AI runs is straightforward: token spend is the only line on the bill that grows fast, and most junk is identifiable without it. Free gates first, paid gates only when the message has already proved it's a real review.

What this hands to the next post

By the time a review leaves the intake, it’s in one shape, with a stable ID, no duplicates, and no obvious spam. The next post is about what the responder does with that — how it actually reads the review, extracts what the customer is praising or complaining about, and starts to decide which of the four moves applies.

PART 3 OF 7

MAY 2, 2026 PART 3 OF 7 · REVIEW RESPONDER SERIES ~5 MIN READ

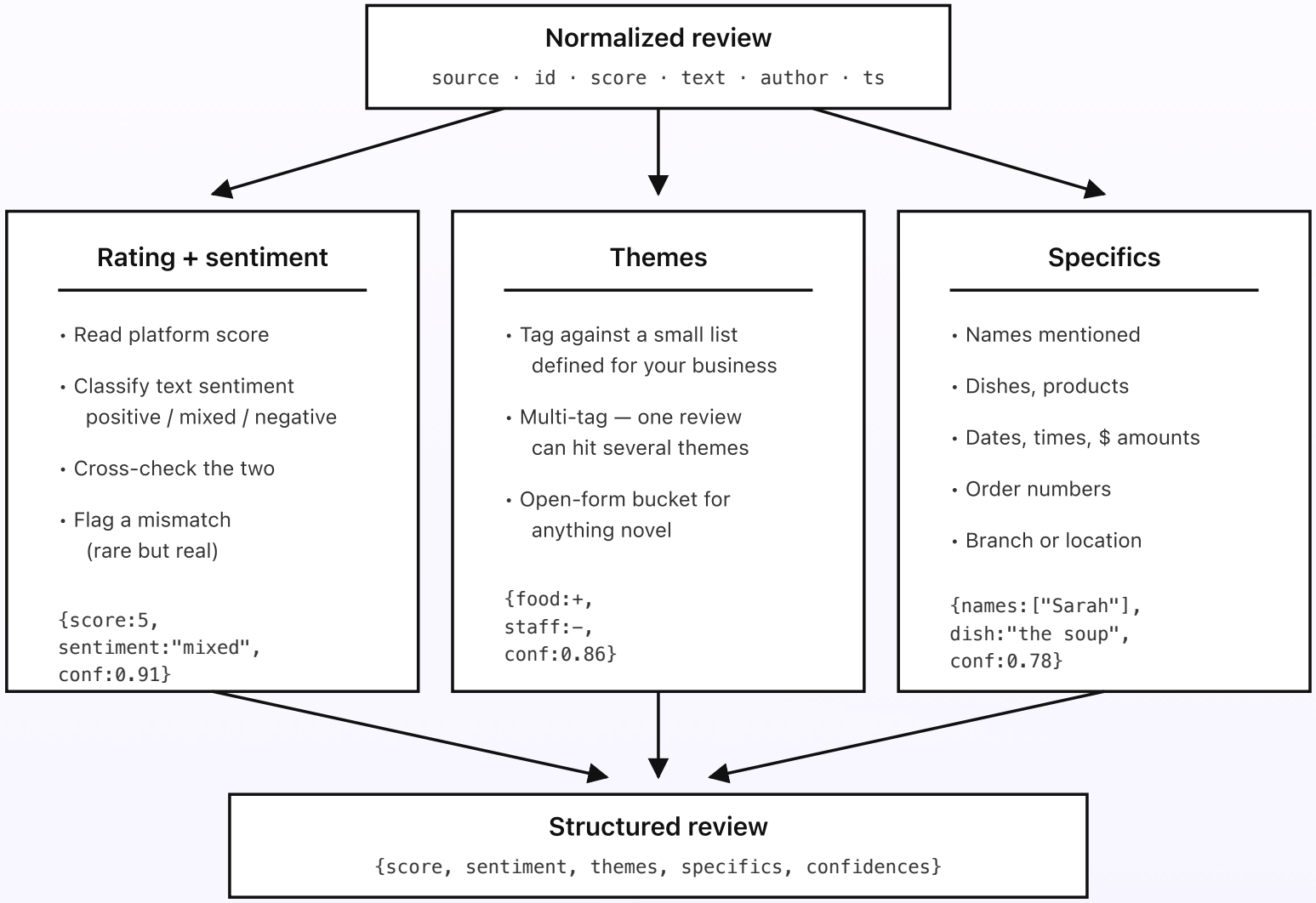
How the responder reads a review

A review is two words to a paragraph of free text, plus a star rating that may or may not match it. Before the responder can pick a move, it has to read the review three different ways: the score, the themes, and the specifics. Three small extractors run in parallel, each returning a confidence score the move-picker reads before it decides anything.

KEY TAKEAWAYS

- Three extractors run in parallel against every review: rating with sentiment cross-check, themes, and specifics.
- The rating extractor flags mismatches — a 5-star with strongly negative text, or a 1-star whose body is a typo of “love it”.
- The themes extractor tags against a small business-specific list (food, service, cleanliness, value, etc.) plus an open-form bucket for anything novel.
- The specifics extractor pulls names, dishes or products, dates, dollar amounts, order numbers, and which branch.
- Each extractor returns a 0–1 confidence score; high confidence on all three is the gate to auto-reply, anything borderline becomes a reason to draft.

Three extractors in parallel



→ to move-picker, with confidences attached

Fig 3. Three extractors run in parallel against the same review. Each one returns its piece with a confidence score; the move-picker reads all three before it decides anything.

Why sentiment, not just stars

The platform-given score is a useful signal but not a complete one. The clearest counterexample is a 5-star review whose body says “food was cold, server was rude, will not be back — the only good thing was the door wasn’t locked.” That happens. So does the inverse: a 1-star review whose body reads “Five stars, sorry hit the wrong button.” Auto-replying to either based on the score alone is a small disaster — a chirpy “thanks for the kind words!” under a complaint, or a confused apology under a happy customer.

The cross-check is cheap. Read the score, run the text through a small sentiment classifier, and if they don’t agree, raise a mismatch flag. The flag doesn’t decide anything by itself; it’s a signal the move-picker uses to push the review out of the auto-reply lane. A draft for your eyes is the right move when the rating and the text disagree, no matter which direction the disagreement runs.

Themes that come back as tags

Themes are the categories a review touches: for a restaurant, that’s typically food quality, service speed, value, cleanliness, atmosphere, and staff friendliness. The list lives in the same Drive folder as the voice file — small, business-specific, and editable without a deploy. The extractor multi-tags: a single review about “great

food, but we waited an hour" lights up food-quality (positive) and service-speed (negative), and the composer can address both in the reply.

An open-form bucket catches anything the pre-defined list doesn't cover — "the parking situation", "the new menu format", "your dog-friendly patio". Open-form themes don't drive auto-reply (the responder won't commit to addressing something it doesn't have a policy for), but they do feed the themes log. After a quarter, the open-form bucket tells you what your customers are talking about that you haven't named yet.

Specifics — names, dishes, dates, branches

The specifics extractor pulls the named bits a real human reply would acknowledge: a staff member's first name ("Sarah was wonderful", "Alex didn't apologize"), a dish or product ("the soup", "the new mug we ordered"), a numeric like an order number or a dollar figure ("they wouldn't honour the $50 promo"), a date or time of visit, and — for a multi-location business — which branch the review was about.

These have two distinct uses. First, the composer reads them, so a draft can say "thanks for mentioning Sarah" instead of "thanks for mentioning our staff." Second, the move-picker uses them as routing signals: a review that names a current staff member by first name is a draft (so a human reads it before it auto-posts under the business's account), and a review that mentions a specific dollar amount or order number is also a draft (because compose-with-numbers is exactly where the AI most wants to make up the wrong figure).

Names are filtered against a roster — a small list of current staff first names in the policies file. “Sarah” mentioned but not on the roster is treated as a regular noun; “Alex” on the roster is flagged. The roster lives in Drive next to the voice file; updating it when someone joins or leaves is one edit, no deploy.

Confidence is the gate

Each extractor returns a confidence score between zero and one. The move-picker won’t auto-reply unless all three are above a configured threshold — a single low score is enough to push the review into the draft lane.

That’s the entire role of confidence in this system: a vote on whether the responder is sure enough to act without you reading first. It is not a measure of correctness, and the responder doesn’t pretend it is. It’s a humility threshold, deliberately tuned so the cost of a low-confidence false positive (you having to glance at one more draft) is much lower than the cost of a high-confidence false positive (a tone-deaf reply posted publicly under your business’s name).

What this hands to the next post

The move-picker now has a structured review with a score, a sentiment, a list of themes, a small bag of specifics, and a confidence on each piece. The next post is what it does with that — how it picks one of four moves, and why the boundaries between them are deliberately conservative.

PART 4 OF 7

MAY 2, 2026 PART 4 OF 7 · REVIEW RESPONDER SERIES ~5 MIN READ

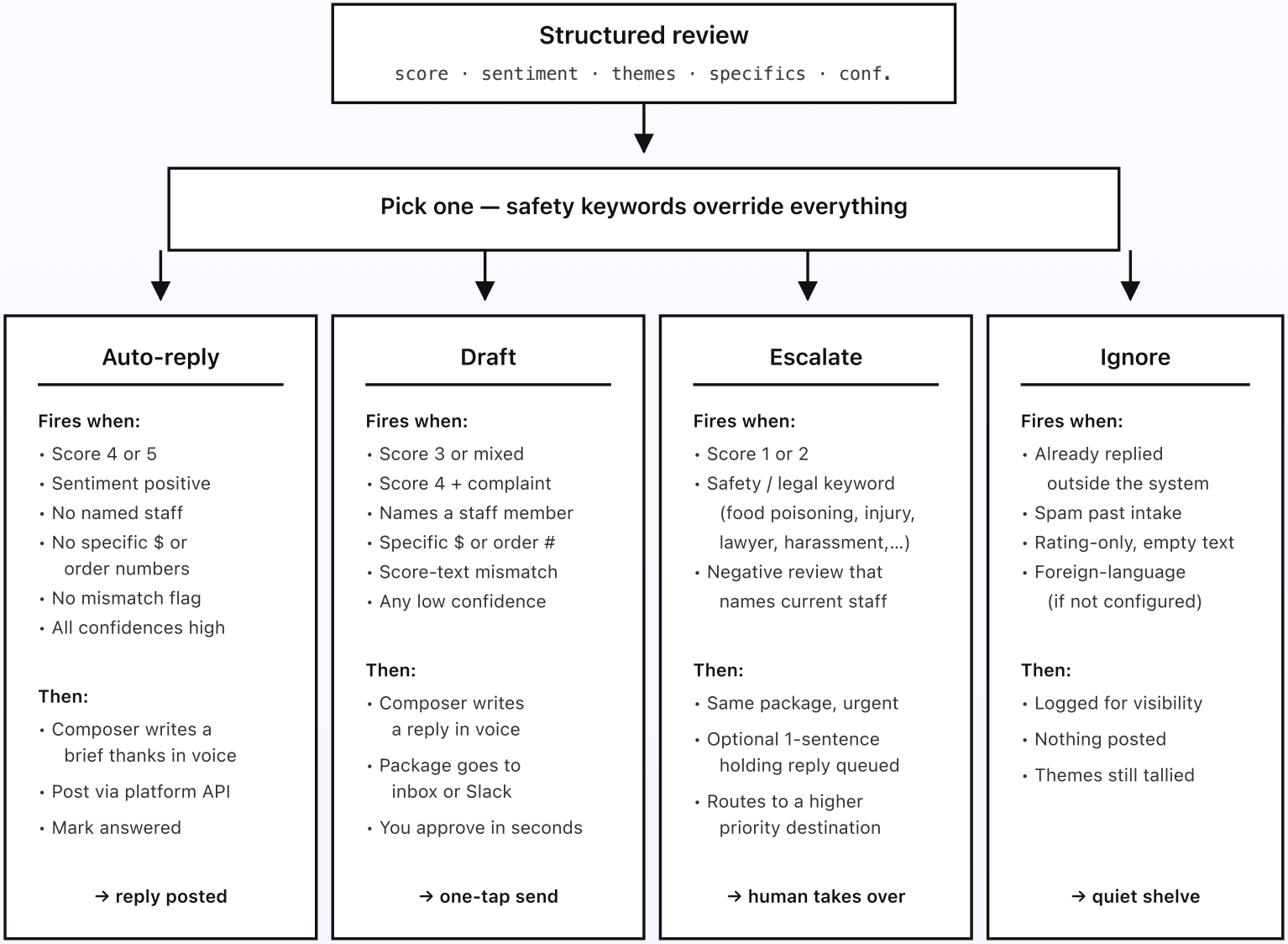
How the responder picks a move

Once the extractors have done their work, the structured review goes to a small decision step that picks one of four moves. The boundaries between them are deliberately conservative: when in doubt, draft. When safety or legal language shows up, escalate, regardless of anything else. Auto-reply is reserved for the cases that genuinely don't need you.

KEY TAKEAWAYS

- Four moves, always: auto-reply, draft, escalate, ignore. There is no fifth.
- Safety and legal keywords (food poisoning, injury, lawyer, refund-stalled) bypass scoring and escalate immediately, regardless of star rating.
- Auto-reply is reserved for 4– or 5-star reviews with no specifics, no named staff, no themes flagged for review, and high confidence on all three extractors.
- Anything that names a staff member or includes a specific complaint becomes a draft — the AI writes the reply, a human glances and sends.
- The pick is enforced via Bedrock `tool_use` with four named tools, so the model can't free-form a fifth option.

Four moves, one pick per review



Safety override is a hard gate — any safety/legal keyword goes to Escalate, no matter what else looks fine.

Fig 4. Four moves, one pick per review. Auto-reply is the smallest active group; Draft is the workhorse; Escalate is the safety lane; Ignore is the quiet cleanup.

Auto-reply: the smallest active group

Auto-reply is reserved for the cases that genuinely don't need you in the loop. Score four or five, sentiment positive, no named staff, no specific dollar amount or order number, no rating-text mismatch, and high confidence on every extractor. In a typical month at a typical small business, that's less than half the reviews, often a third — and that's on purpose. Most positive reviews still mention something specific ("Maria was wonderful", "loved the new menu"), and specifics route to draft so a human gets the chance to thank a person by name in their own voice.

What does end up here is the "Great service! Will be back!" class — warm, generic, no specifics, no risk. The composer writes a brief one- or two-sentence thanks in your voice (covered in the next post), posts it via the platform API, and marks the review answered. You see a count, not a stack.

Draft: the workhorse middle

Most reviews you actually want a human voice on land here. Three stars and mixed sentiment, four stars with a specific complaint about something concrete, any review that names a current staff member (positive or negative), any review that mentions a specific dollar amount or an order number, any rating-text mismatch, and any low extractor confidence.

The difference from auto-reply is editorial control: the cloud has done the writing, but you read it and hit send. Each draft package contains the original review, the proposed reply, the matching policy excerpt the composer used, and a one-line reason it's a draft ("mentions Sarah by name", "score-text mismatch", "refund amount"). The reason matters more than it sounds — reading "mentions a $ figure" before approving makes you look at the figure in the draft, instead of skim-approving.

The work this saves is the writing, not the deciding. A real reply takes a few minutes to compose; a draft takes a few seconds to glance over. The cumulative time across thirty reviews a month is the difference between "I'll do it on Saturday" and "done in five minutes after lunch."

Escalate: the safety lane

Escalate is the move that fires the moment the responder reads something serious. One- and two-star reviews always escalate, full stop. So does any review that hits a safety or legal keyword from the policies file — food poisoning, injury, lawyer, allergen, harassment, refund-stalled, threat, the list grows with experience. So does a negative review that names a current staff member, because that's a workplace conversation as much as it is a review reply.

The escalate package looks like a draft package, but with three changes: it's marked urgent, it routes to a higher-priority destination (a manager email or a dedicated "reviews-urgent" Slack channel), and it can optionally queue a one-sentence holding reply — "We're sorry; we'd like to look into this. Please reply with your visit details or call <the policies-file phone number>" — that posts

immediately so the customer doesn't see silence while a human takes over. Whether the holding reply is on by default is a setting in the policies file; some industries (medical, legal) want the holding reply off entirely until a real person looks.

Ignore: the quiet cleanup

Ignore exists for the cases where the right move is no move. A review that already has a reply on it (from before the responder was set up, or because someone on your team replied directly through the platform's native UI). A spam comment that slipped past the screen. A rating-only entry with no text the responder could meaningfully reply to. A review in a language you haven't configured the responder for.

Ignored reviews still get logged, and their themes still get counted — one of them is just as valid a data point about service speed or food quality as a review the responder replied to. They just don't produce an outbound message.

Why these four and not five

Every additional bucket is a new branch a human has to remember the meaning of. Four covers the meaningful behaviour: act now, help me act, bring me in, do nothing. The most common request for a fifth is "queue this for later," which on inspection is always either a draft (the human will get to it) or an ignore (they'll never get to it). Naming the difference forces a small honesty that helps the system.

The other rule that stays out of the move-picker on purpose is anything trying to optimise for SEO or platform algorithms. The responder's job is to handle the reply correctly, not to game the listing's ranking. If the platform rewards reply latency or reply length, that's a downstream effect — doing the right thing first earns it anyway.

One thing the four-move shape doesn't show: platform reach

Not every review platform lets a small-business owner post a reply through an API in 2026. Google Business Profile does, so the auto-reply move on Google reviews actually posts. Facebook deprecated their public reply path for page recommendations at the start of 2025, and Yelp's reply API is only available to enterprise partners. So for Facebook and Yelp, the auto-reply move is automatically demoted to a draft regardless of how confident the responder is — the cloud has done the writing, but a human pastes it into the platform's own app.

This is one per-source flag the dispatch column reads at the very end. The decision logic in this post stays exactly the same; what differs is the destination of the produced reply.

What this hands to the next post

Three of the four moves end with a reply being composed: auto-reply posts it, draft and escalate package it for human review with the reply already written. The next post is the composer itself — how a reply stays in your voice and only ever promises what the policies file actually allows.

PART 5 OF 7

MAY 2, 2026 PART 5 OF 7 · REVIEW RESPONDER SERIES ~5 MIN READ

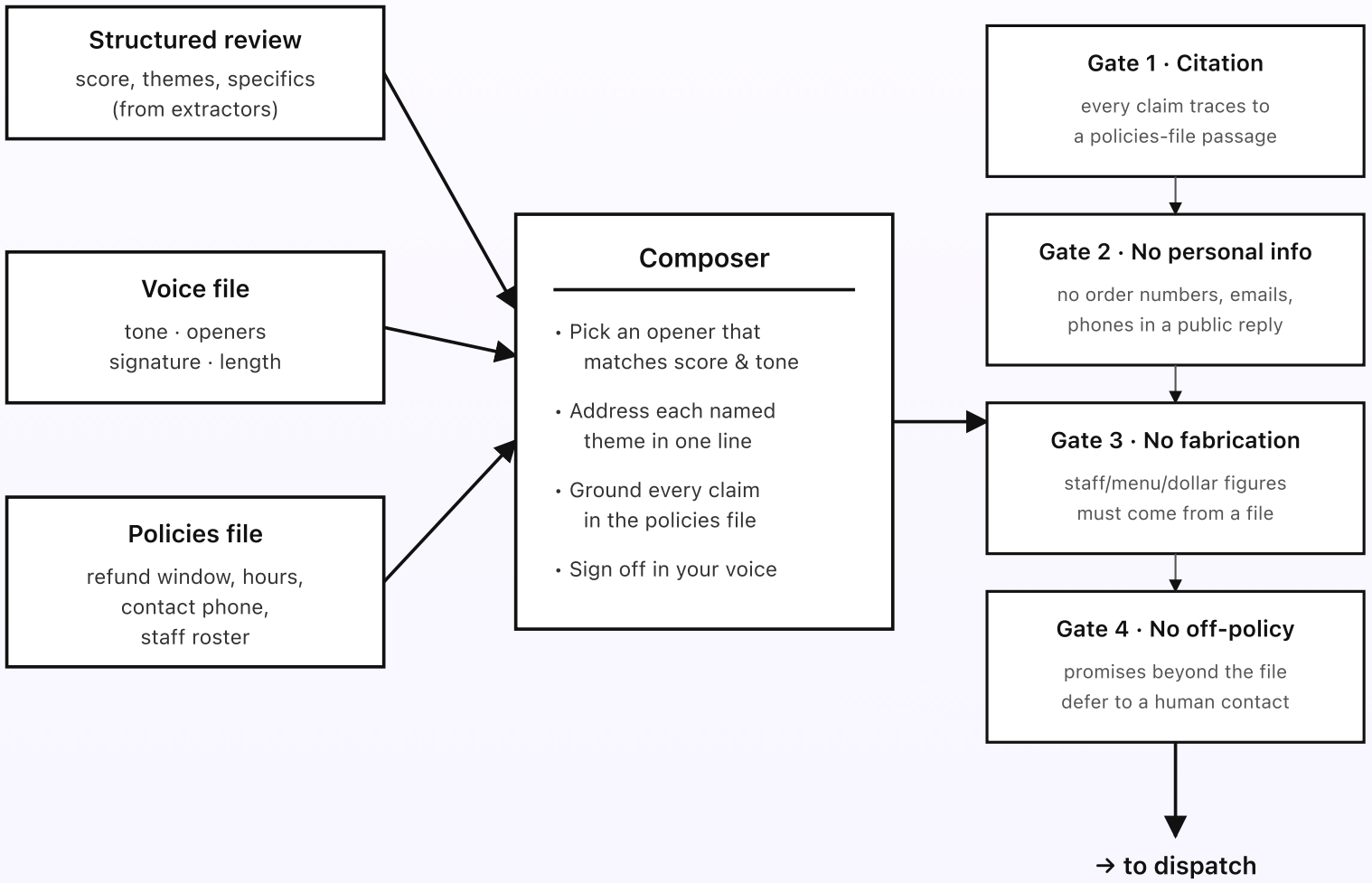
How a reply stays in your voice

A generic AI reply to a review is recognisable in the first sentence — the cadence, the “we appreciate your feedback,” the careful avoidance of any fact about the actual business. The composer here reads three short Drive docs that describe how you sound, what you can promise, and what you must never say, and writes from those only. Four guardrails sit between the model and the post button.

KEY TAKEAWAYS

- Three Drive docs ground every reply: a voice file (tone, openers, signature), a policies file (refund window, hours, staff roster), and a never-say file (banned phrases, comparative claims, legal/medical language).
- The composer pulls only from those three docs — it never invents a refund window, a phone number, or a staff name not on the roster.
- Four guardrails sit between the model and the post button: citation required, no personal-info echo, no fabricated specifics, no off-policy promises.
- Edit a Drive doc and the system picks up the change on the next refresh. No deploy, no code change.
- Any guardrail failure on an auto-reply demotes it to a draft — the human sees why on the package.

The composer and its guardrails



A guardrail failure on a draft is shown on the package; on an auto-reply, the move is demoted to a draft.

Fig 5. Three Drive files ground the composer; four guardrails sit between it and the post button. Anything that fails a gate is either shown on the draft or demoted out of the auto-reply lane.

The voice file: how you sound

The voice file is one short Google Doc, ideally a single page. It captures three things about how the business writes. Tone — warm and brief, or formal and detailed, or playful and short, or whatever fits. A few example openers for each rating band (a five-star opener is not a one-star opener), so the composer has a template to start from rather than inventing one. And a signature line if there is one ("— the team at Such-and-Such" works for some businesses; for others, the owner's first name).

The voice file does not contain facts about the business. It contains *how*, not *what*. The clean separation matters because tone changes rarely — you write it once and edit twice a year — while the policies-and-facts file changes regularly as hours, prices, and offers shift.

The policies file: what you can promise

The policies file is the fact base. It lists the things the responder is allowed to say to a customer in a reply: refund window ("up to 14 days from purchase"), replacement policy ("a return-visit credit equal to the order value if you let us know within a week"), contact information (the phone number and email a complaint can be redirected to), opening hours, and the current staff roster (first

names of people who work there, so a review that names a current employee can be flagged correctly).

If the customer is asking for something that isn't covered in the policies file, the responder cannot promise it on the business's behalf. It will defer to a human ("please reach out to <phone-from-policies-file> so we can look into this with you") instead of making up a refund window or compensation. The whole point of the file is to make "what we offer" a thing the system reads, not a thing the model invents.

The four guardrails between model and post button

The composer's output goes through four gates. Each is small and deliberate.

Citation. Every factual claim in the reply must trace to a passage in the policies file. "We're glad you enjoyed the soup" doesn't need a citation; "We offer a 14-day refund window" does. The composer attaches the policy passage it leaned on; the runtime checks that the passage actually exists in the current policies file. If a claim has no citation, that claim cannot appear in the reply. (This is the same shape as the chat assistant's citation rule from the previous series — same logic, different fact base.)

No personal info echo. If the reviewer wrote "order number 81234" or their email or phone number into the public review, the reply doesn't repeat it. The reviewer might be comfortable putting their order number in a review they wrote at 11pm; the business should not be amplifying that to every future visitor of the listing. The gate strips out personal info — emails, phone numbers, addresses, order IDs — from the proposed reply before it's sent.

No fabricated specifics. Staff names not in the roster, menu items not in the menu file (if you have one), dollar figures not in the policies file. The composer can refer to “your visit” or “the team that served you” without making up a name. If it tries to insert “Sarah” and Sarah isn’t on the roster, the gate replaces the specific with the generic phrasing.

No off-policy promises. Anything the reply commits the business to — a future refund, a special accommodation, an appointment, a free replacement, a callback — must trace to the policies file. If the customer is asking for something the policies file doesn’t allow, the reply pivots to deferral: “We’d like to look into this with you — please call <phone-from-file> or reply to this directly.” That’s a fine answer, and it’s the only honest one when the system genuinely can’t commit on the business’s behalf.

What “in your voice” actually means

The phrase gets used loosely, so it’s worth being specific. “In your voice” here means three concrete things.

First, the cadence and word choice come from the voice file. A business that uses contractions and short sentences does not suddenly start writing “We sincerely regret to inform you”; one that uses formal language doesn’t suddenly drop into “hey, sorry about that.”

Second, the facts come from the policies file. The reply doesn’t invent the business’s refund window or guess the manager’s name. If a fact isn’t in the file, it isn’t in the reply.

Third, the structure follows a small handful of templates the voice file provides. An opener that matches the rating, a sentence acknowledging the specific theme the customer raised, a sentence offering the appropriate next step (or a thank-you, on positive reviews), and a sign-off. That's the shape; the words inside it are filled in by the model, grounded in the two files. The result reads like the same business wrote a hundred replies, because in the only sense that matters, it did.

What this hands to the next post

The composer hands a reply (and its citations) to dispatch, which posts it, packages it for your inbox, or escalates it — depending on the move the previous post described. The next post is the cost picture: where the dollars go, and why a typical small business runs this for coffee money.

PART 6 OF 7

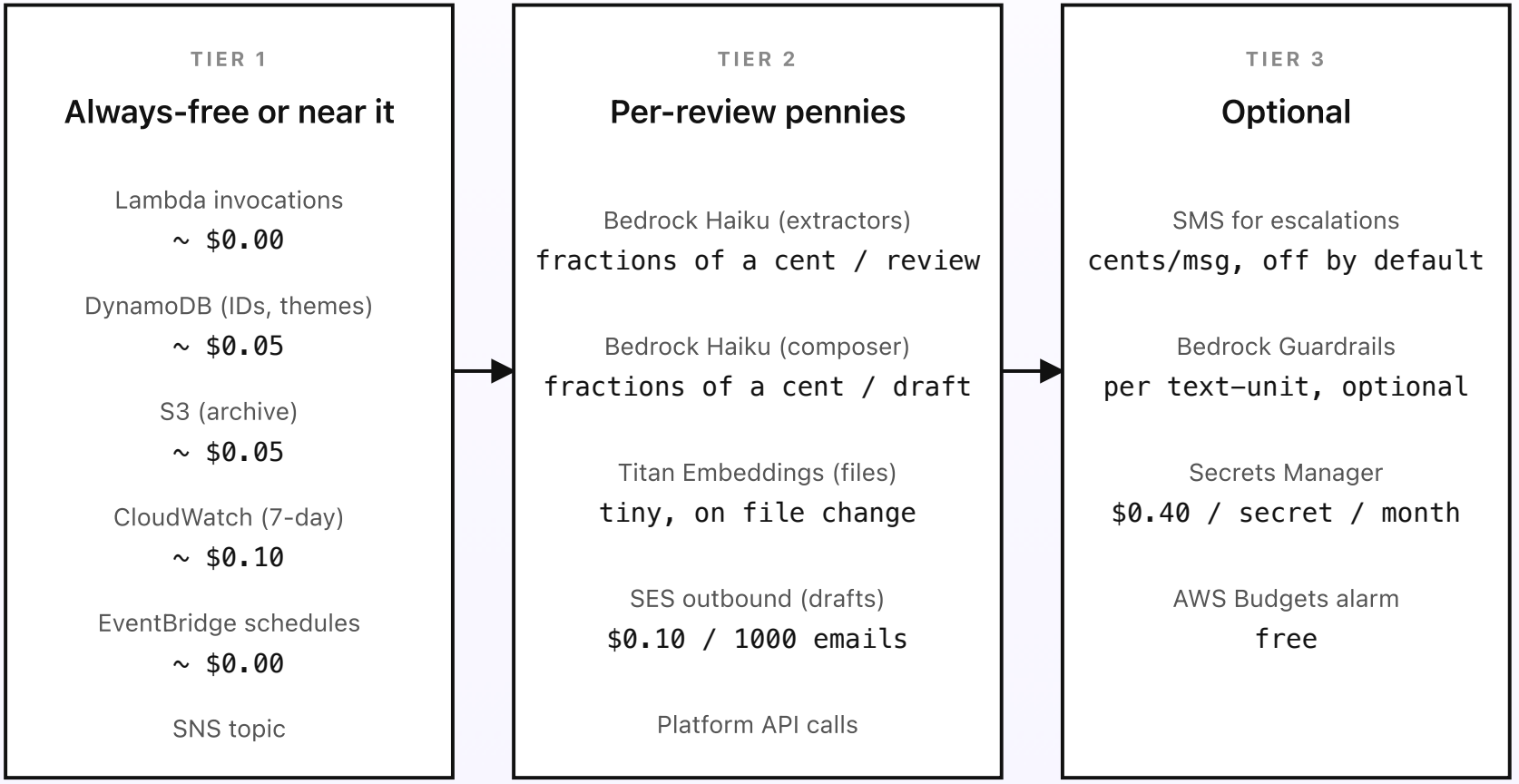
MAY 2, 2026 PART 6 OF 7 · REVIEW RESPONDER SERIES ~3 MIN READ

What the review responder costs

A coffee a month at SMB review volume. The fixed cost is essentially zero — if your listings are quiet for a week, the bill matches. The variable cost is dominated by Bedrock tokens for the extractors and the composer; everything else rounds to pennies.

KEY TAKEAWAYS

- Three cost tiers: always-free or near it (Lambda, DynamoDB, S3, CloudWatch, EventBridge, SNS), per-review pennies (Bedrock tokens for the three extractors and the composer), and optional add-ons.
- Fixed cost is essentially zero. A quiet week bills nothing because every component is on-demand.
- Variable cost is dominated by Bedrock Haiku tokens; Titan embeddings for the policies and voice files are a tiny one-off when the docs change.
- Platform API calls (Google Business Profile, Meta Graph, Yelp polling) are free within rate limits; the Yelp poller fits inside the 500-call free daily tier.
- ~50 reviews/month lands under $3 total; ~500 reviews/month lands under $15.



~ 50 reviews/month → under three dollars total. ~ 500 reviews/month → under fifteen.

Fig 6. Three tiers of cost. The bill scales with how often new reviews land; the floor is nearly zero.

The fixed cost is essentially zero

There is no per-listing licence, no minimum monthly fee, no “starter plan.” If your listings have a quiet week, the AWS bill matches. Lambda, DynamoDB pay-per-request, S3, CloudWatch, EventBridge, and SNS all sit in or near the always-free tier at the volumes a small business sees. Even running the polling job for Yelp every hour, twenty-four hours a day, all month, costs a vanishingly small amount — EventBridge scheduled invocations are essentially free at this volume, and each poll is a single Lambda call that reads from one platform endpoint.

The biggest single line item in tier one is usually CloudWatch log storage, and you control that by retaining seven days instead of forever. After day eight, the logs are gone and the bill stops compounding.

The variable cost is per-review pennies

Three things scale with review volume:

- **Bedrock Haiku tokens for the extractors.** Each review pays for three small model calls (rating-with-sentiment, themes, specifics) running against the review text. At small-model rates and the short input lengths reviews tend to be, the all-in cost per review for extraction lands at fractions of a cent.
- **Bedrock Haiku tokens for the composer.** Auto-replies and drafts each pay for one composer call. The input is the structured review plus the relevant policies

excerpt; the output is a short reply. Even on the most expensive moves, this is well under a cent per review. The composer doesn't fire on Ignore, so junk reviews cost only the screening tier.

- **Titan Embeddings for the policies and voice files.** One-off per file change; if you edit your policies file twice a month, that's two embeddings calls totalling tiny fractions of a cent. The embeddings power the citation gate — finding the right policy passage to ground a claim.

Add it up at typical small-business review volumes (twenty to a hundred new reviews a month across all platforms): tier two is in the under-three-dollars range, tier one floors are under fifty cents, and the all-in monthly bill lands at a coffee a month. At higher volumes — a chain with several hundred reviews a month — tier two is the headline number, and even there it stays in the "phone bill, not Netflix subscription" range.

Three traps you're avoiding

- **Per-listing reputation tools.** Most off-the-shelf reputation managers charge $40–$200 per listing per month, regardless of volume. You're trading a flat per-listing bill for pay-per-use that mostly comes in pennies, with the model running on the same shape regardless of how many listings you connect.
- **Always-on infrastructure for polling.** A naive setup might run a small server that polls Yelp once a minute. That's a $30+/month idle bill before it processes a single review. EventBridge schedules trigger Lambda — pay only when the schedule fires, scale to zero between fires.

- **Compose-on-every-review at long context.** A larger model would spend ten to twenty times more per review for what is, at this scale, a short-input short-output task. Haiku-class models are correctly sized for review composition; reaching for a bigger one because it sounds smarter is the most common way to triple the bill.

When this stops being cheap

The math changes if you connect dozens of listings (a multi-location franchise) and the review traffic goes into the thousands per month. At that point the tier-two number becomes the headline, and there are levers: caching policy embeddings (already cached after the first lookup), batching the extractors into a single multi-output call, or moving to a smaller dedicated extraction model for the rating-with-sentiment step. Most small businesses, including small chains, never need any of those levers.

For everyone below that — and that's most small businesses — a $10 monthly AWS Budgets alarm catches anything strange before it becomes a surprise.

In plain words

The fixed bill is nearly zero. The variable bill is pennies per review, dominated by tokens. A typical setup runs at coffee-money for the whole month. Set a budget alarm that fits your expected volume and the bill can't surprise you.

PART 7 OF 7

MAY 2, 2026 PART 7 OF 7 · REVIEW RESPONDER SERIES ~6 MIN READ

Engineering reference: the review responder architecture

Same system as the rest of the series, drawn purely for engineers. Service names, resource identifiers, region, Bedrock model IDs, Knowledge Base wiring, Google Business Profile and Meta Graph API specifics, and the actual flow operations — everything you'd need to recreate this in your own AWS account.

KEY TAKEAWAYS · VERIFIED MAY 2026

- Single AWS account in `ap-southeast-1` (Singapore); Bedrock via Global cross-Region inference.
- Five subsystems: Build & Deploy, Knowledge Sync, Intake (3 lanes → SQS), Responder (parallel extractors + decision + composer), Dispatch & learning.
- Models: `global.anthropic.claude-haiku-4-5-20251001-v1:0` + `amazon.titan-embed-text-v2:0`; vector store is S3 Vectors (GA Dec 2, 2025).
- Review sources: Google Business Profile via Pub/Sub push, Facebook via third-party aggregator (Meta deprecated their own webhook in Graph API v22.0, January 2025), Yelp via hourly Fusion API poll.
- Day-one paperwork: Google Business Profile API Basic Access (~14-day approval), Facebook aggregator credentials, Drive service account.

Posts 1–6 walk through the system in plain language. This page is the dense version — nothing softened, just the architecture as you'd sketch it on a whiteboard during a design review.

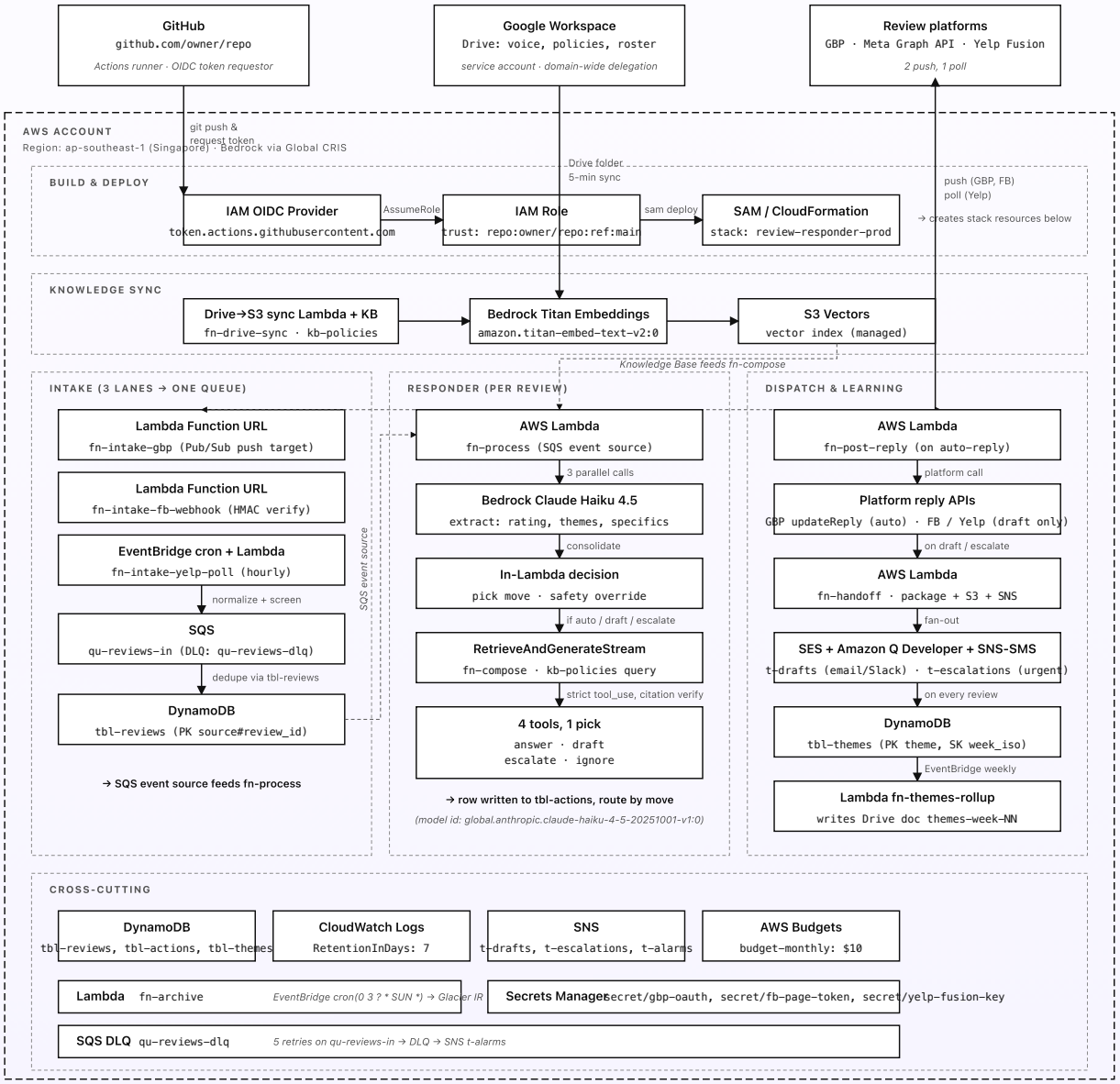

GitHub
github.com/owner/repo
Actions runner · OIDC token requestor
Google Workspace
Drive: voice, policies, roster
service account · domain-wide delegation
Review platforms
GBP · Meta Graph API · Yelp Fusion
2 push, 1 poll
AWS ACCOUNT
Region: ap-southeast-1 (Singapore) · Bedrock via Global CRIS
git push & request token
Drive folder 5-min sync
push (GBP, FB) poll (Yelp)
BUILD & DEPLOY
IAM OIDC Provider
token.actions.githubusercontent.com
AssumeRole
IAM Role
trust: repo:owner/repo:ref:main
sam deploy
SAM / CloudFormation
stack: review-responder-prod
→ creates stack resources below
KNOWLEDGE SYNC
Drive→S3 sync Lambda + KB
fn-drive-sync · kb-policies
Bedrock Titan Embeddings
amazon.titan-embed-text-v2:0
S3 Vectors
vector index (managed)
Knowledge Base feeds fn-compose
INTAKE (3 LANES → ONE QUEUE)
Lambda Function URL
fn-intake-gbp (Pub/Sub push target)
Lambda Function URL
fn-intake-fb-webhook (HMAC verify)
EventBridge cron + Lambda
fn-intake-yelp-poll (hourly)
normalize + screen
SQS
qu-reviews-in (DLQ: qu-reviews-dlq)
dedupe via tbl-reviews
DynamoDB
tbl-reviews (PK source#review_id)
→ SQS event source feeds fn-process
SQS event source
RESPONDER (PER REVIEW)
AWS Lambda
fn-process (SQS event source)
3 parallel calls
Bedrock Claude Haiku 4.5
extract: rating, themes, specifics
consolidate
In-Lambda decision
pick move · safety override
if auto / draft / escalate
RetrieveAndGenerateStream
fn-compose · kb-policies query
strict tool_use, citation verify
4 tools, 1 pick
answer · draft
escalate · ignore
→ row written to tbl-actions, route by move
(model id: global.anthropic.claude-haiku-4-5-20251001-v1:0)
DISPATCH & LEARNING
AWS Lambda
fn-post-reply (on auto-reply)
platform call
Platform reply APIs
GBP updateReply (auto) · FB / Yelp (draft only)
on draft / escalate
AWS Lambda
fn-handoff · package + S3 + SNS
fan-out
SES + Amazon Q Developer + SNS-SMS
t-drafts (email/Slack) · t-escalations (urgent)
on every review
DynamoDB
tbl-themes (PK theme, SK week_iso)
EventBridge weekly
Lambda fn-themes-rollup
writes Drive doc themes-week-NN
CROSS-CUTTING
DynamoDB
tbl-reviews, tbl-actions, tbl-themes
CloudWatch Logs
RetentionInDays: 7
SNS
t-drafts, t-escalations, t-alarms
AWS Budgets
budget-monthly: $10
Lambda fn-archive
EventBridge cron(0 3 ? * SUN *) → Glacier IR
Secrets Manager
secret/gbp-oauth, secret/fb-page-token, secret/yelp-fusion-key
SQS DLQ qu-reviews-dlq
5 retries on qu-reviews-in → DLQ → SNS t-alarms

Fig 7. Full architecture, ap-southeast-1. White boxes = AWS resources; dashed AWS container; dashed grey boxes = subsystem groupings; dashed grey arrows = config feed and side branches.

Read this top-down, then column-by-column

Top row is the three external surfaces. Below it, the AWS account contains five subsystems: Build & Deploy across the top, then Knowledge Sync, then three runtime columns (Intake, Responder, Dispatch & learning), with a Cross-cutting strip at the bottom. Reviews enter through three intake paths (two webhooks behind Lambda Function URLs, one cron-driven poller) and all three write into a single SQS queue `qu-reviews-in` after deduplicating against `tbl-reviews`. The SQS event source invokes `fn-process`, which runs the three extractors in parallel against Bedrock Claude Haiku, picks one of four moves with safety-keyword override, and on auto-reply / draft / escalate calls `fn-compose`. The composer issues a Bedrock `RetrieveAndGenerateStream` against `kb-policies` with strict tool_use over four tools (`answer`, `draft`, `escalate`, `ignore`), verifies citations, strips PII, and writes the chosen action to `tbl-actions`. Dispatch routes by move: auto-reply via `fn-post-reply` to the originating platform's reply API, draft and escalate via `fn-handoff` to S3 plus SNS fan-out. Themes are tallied on every review into `tbl-themes` and rolled up weekly by `fn-themes-rollup`.

Naming conventions used in the diagram

- **Lambda functions:** `fn-<purpose>` — `fn-intake-gbp`, `fn-intake-fb-webhook`, `fn-intake-yelp-poll`, `fn-process`, `fn-compose`, `fn-post-`

`reply`, `fn-handoff`, `fn-themes-rollup`, `fn-drive-sync`, `fn-archive`.

- **Lambda runtimes:** Python 3.13 for the responder, composer, themes rollup, drive sync, and archive functions (the Bedrock SDK is more ergonomic in Python). Python 3.14 has been available on Lambda since November 2025 and is fully supported; 3.13 is the safe production default in May 2026. Node.js 22.x is fine for `fn-intake-fb-webhook` if you prefer JS for HMAC verification; Node.js 24.x is also available since 2025 and either is current.
- **DynamoDB tables:** `tbl-reviews` (partition key `source#review_id`, attribute set: `seen_at`, `raw_payload`, `screen_verdict`; used for dedupe and audit), `tbl-actions` (partition key `review_id`, sort key `action_ts`, with `move`, `reply_text`, `cited_passages`, `guardrail_flags`), `tbl-themes` (partition key `theme`, sort key `week_iso`, with rolling counts; `theme` values come from the policies file's themes list).
- **SQS queues:** `qu-reviews-in` (standard queue with 5-minute visibility timeout), `qu-reviews-dlq` (5 retries before failure goes to DLQ; CloudWatch alarm on DLQ depth > 0 fires `t-alarms`).
- **SNS topics:** `t-drafts` for normal-priority human review fan-out (email, optional Slack), `t-escalations` for urgent fan-out (email + optional SMS), `t-alarms` for general failures.
- **S3 layout:** single bucket `review-responder-data` with prefixes `kb-source/` (Drive mirror), `drafts/{date}/` (full draft packages), `archive/`.
- **Knowledge Base:** `kb-policies`, a Bedrock managed Knowledge Base with an **S3 connector** pointed at the synced policies/voice/menu prefix. Bedrock KBs do not have a native Drive connector as of 2026-05 (current native connectors: S3, Confluence, SharePoint, Salesforce, Web Crawler, plus a custom-API

option), so a small `fn-drive-sync` Lambda mirrors the Drive folder to S3 on a 5-minute schedule. Embeddings model is `amazon.titan-embed-text-v2:0`; vector store is **Amazon S3 Vectors** (GA December 2025 — cheapest quick-create option for small/medium KBs: no provisioned capacity, no monthly minimum, ~$0.06/GB-month for stored vectors plus per-query and per-PUT charges — provisioned and managed by Bedrock when you create the KB). OpenSearch Serverless and Aurora pgvector remain valid alternatives for higher query throughput.

Region, model access, platform APIs, and Drive auth

Everything runs in `ap-southeast-1` (Singapore). Bedrock model invocations use the **Global cross-Region inference** profile (`global.` prefix on model IDs) — data at rest stays in Singapore; inference may route to other regions for capacity, billed at on-demand Singapore rates.

The intake Lambdas run as Lambda Function URLs to keep webhook ingress free of API Gateway. Each lane has its own current-2026 reality and the design accounts for the differences honestly.

Google Business Profile (lane 1, fully automated). Push notifications go through the **My Business Notifications API v1** at `mybusinessnotifications.googleapis.com/v1/accounts/{accountId}/notificationSetting`; you create a Pub/Sub topic in your own GCP project, grant `pubsub.topics.publish` on that topic to `mybusiness-api-pubsub@system.gserviceaccount.com`, and PATCH the notification setting with `pubsubTopic` and `notificationTypes: ["NEW_REVIEW", "UPDATED_REVIEW"]`.

The Pub/Sub subscription pushes to `fn-intake-gbp`, which verifies Google's OIDC JWT before accepting the payload. Reading and replying to reviews stayed on the legacy v4 surface even after the broader v4 deprecation in 2024 — the canonical endpoints are still `GET mybusiness.googleapis.com/v4/accounts/{accountId}/locations/{locationId}/reviews` for list and the `accounts.locations.reviews.updateReply` method (PUT to `{name}/reply`) for the reply. Single OAuth scope: `https://www.googleapis.com/auth/business.manage`.

GBP API access is allowlist-gated, not partner-gated. A new GCP project starts at **0 queries-per-minute** — every API call returns `quotaExceeded` — until you submit the GBP "Application for Basic API Access" form (free) and Google approves, typically within ~14 days. Approved projects are bumped to 300 QPM with a hard cap of **10 edits-per-minute per profile** (the reply call counts as an edit). Prerequisites: a verified Business Profile that's been active 60+ days, a website on the profile, and an applicant email ideally on the website's domain. A regular single-location owner can apply directly; you don't need Partner status. The 0-QPM trap is the #1 first-time gotcha and worth surfacing in the SAM template README so future-you doesn't debug a half-day before realising the project is correctly enabled and just not allowlisted yet.

Facebook (lane 2, draft-only in 2026). Meta deprecated the Page recommendations webhook in **Graph API v22.0 (January 21, 2025)**: the `ratings` field on the Page object no longer fires, and reading a recommendation returns `error code 12`. There is no v23+ replacement and no documented reply-to-recommendation API (the Recommendation node reference now states the endpoint "cannot be queried directly"). The realistic Facebook path in 2026 is

therefore one of two patterns: a third-party aggregator (Birdeye, Yext, ReviewTrackers) that watches the page on your behalf and pushes normalized events to your webhook URL, or a periodic page-scraping fallback if you accept the fragility. Either way the Facebook lane is *read-only* from the platform's perspective, which means the move-picker downgrades all Facebook reviews to `draft` regardless of confidence: `fn-compose` still produces the reply text, but `fn-handoff` drops the package in your Pages-app paste-in queue rather than calling a non-existent reply API. HMAC-SHA256 signature verification (`X-Hub-Signature-256` , App Secret as the key, computed over the raw request body, constant-time comparison) is still the right pattern for any Meta webhook you do subscribe to. Pin to v23.0 or v24.0 on outgoing calls; v22.0 is the youngest version that received the recommendations deprecation enforcement, and v18.0 already sunset on 2026-01-26.

Yelp (lane 3, draft-only for SMBs). The reply API exists — the **Respond to Reviews API v2** at `partner-api.yelp.com/reviews/v1/{review_id}` — but it's partner-gated and effectively enterprise-only: access requires either a Yelp + Listing Management subscription or a chain of 10+ Branded/Enhanced Profile locations. For a single-location SMB, the only programmatic surface is the public Fusion API `GET /v3/businesses/{id}/reviews` , which returns up to **3 truncated review excerpts** per call, on Enhanced or Premium pricing tiers (the free tier was cut to **500 calls/day** total in May 2023). `fn-intake-yelp-poll` runs on EventBridge cron `cron(0 * * * ? *)` (hourly), reads the truncated endpoint per listing, diffs against the latest `review_id` seen in `tbl-reviews` , and queues only-new IDs. Like the Facebook lane, the dispatch column treats Yelp as draft-only: the responder produces the reply, the human pastes it into `biz.yelp.com` .

Auth: API key bearer (`Authorization: Bearer <API_KEY>`) on the public Fusion surface; OAuth on the partner surface if you ever upgrade.

Architecturally, a single per-source `auto_reply_supported` boolean flag in the lane config is enough to handle this: dispatch reads the flag and routes `auto-reply` moves through to either `fn-post-reply` (when supported) or `fn-handoff` as a draft (when not). The decision logic in the move-picker doesn't change shape; only the destination of the produced reply does.

Google Drive authentication uses a service account with **domain-wide delegation** over a single scope: `https://www.googleapis.com/auth/drive.readonly` on the policies-and-voice folder only. The credential lives in AWS Secrets Manager. The `fn-drive-sync` Lambda runs on a 5-minute EventBridge schedule, pulls any changed docs from Drive, writes them to `review-responder-data/kb-source/`, and lets the Bedrock KB's S3 connector index from there. Editing a doc and saving propagates within ~10 minutes (5 to sync + 5 to index); manual re-sync is one CLI call to `StartIngestionJob`.

The composer uses **strict tool_use**: four tool definitions (`answer`, `draft`, `escalate`, `ignore`) with required parameter schemas. The `answer` and `draft` tools require a `citation_passages` array referencing one or more retrieved passages by id; the runtime validates each citation against the retrieved set before allowing dispatch. If the model emits an `answer` with a citation that wasn't in the retrieved set, the runtime downgrades to `draft` — the safer-by-default failure mode. The PII strip and the staff-roster check both run after the model returns and before the reply is dispatched anywhere.

What's deliberately not on the diagram

- IAM policy details — per-Lambda execution roles are minimal (one bucket prefix, one or two tables, a single Bedrock KB ID, `InvokeModel` on one model, the relevant platform-API outbound permissions via Secrets Manager).
- Per-business policies layout — a flat Drive folder is fine for the first few months; subdivide by topic (`refunds/`, `hours/`, `roster/`) once the file count grows past a couple of dozen.
- X-Ray tracing — on for `fn-process` and `fn-compose`, sampling 100% during tuning, 10% in steady state.
- **Bedrock Guardrails** — managed contextual grounding (numeric grounding + relevance scores), PII redaction, prompt-attack/jailbreak filters, and the newer **Automated Reasoning checks** (formal-logic policy validation, GA in 2025). The custom citation-verify, PII-strip, and roster-check steps in `fn-compose` are roughly the contextual-grounding and PII ideas hand-rolled; turning on Guardrails moves the threshold into console configuration and adds prompt-attack defence on every model call. Worth enabling once thresholds are stable.
- **Multi-language replies** — the composer reads the language of the inbound review and falls back to `ignore` if the language isn't in the configured set. Adding a language is a config edit and a translated voice-file section, not a code change.
- **Multi-tenant variant** — if running this on behalf of multiple SMBs, namespace the KB and tables per tenant and inject `tenant_id` into every record. The architecture doesn't change shape; the IDs do.
- **Step Functions vs in-Lambda orchestration** — the per-review pipeline (extract → pick → compose → dispatch) fits comfortably inside a single Lambda

invocation under the 15-minute limit. Step Functions becomes worth it only if you need long-poll waits between human approval and post; for the synchronous draft package pattern shown here, in-Lambda is simpler and cheaper.

- **Retroactive backfill** — on day one the system is empty of historical reviews. A one-shot backfill script can populate `tbl-reviews` with existing review IDs (so they're marked "seen, not actioned") without triggering a flood of belated drafts. Off the diagram because it runs once.

IF YOU'RE RECREATING THIS

Day-one paperwork: submit the Google Business Profile API "Application for Basic API Access" on day one of the project — approval takes ~14 days, your GCP project sits at 0 QPM until then, and there's nothing technical you can do to skip the wait. If you're planning to use a Facebook aggregator (Birdeye, Yext, ReviewTrackers), get the credential / webhook URL from them on day one too; their onboarding can be a few business days.

Start with Build & Deploy alone (a single Lambda, no triggers). Once `git push` reliably updates an empty stack, wire up `fn-drive-sync` with one short policies doc and confirm the doc lands in S3 within five minutes. Create the Bedrock Knowledge Base over that S3 prefix and confirm a one-shot `RetrieveAndGenerateStream` call returns a passage. Then one intake lane — the Yelp poller is the easiest, since it's a pure cron and doesn't require a webhook URL to be reachable from the public internet. Then the SQS-driven `fn-process` with the three extractors and the in-Lambda decision step. Then `fn-compose` with strict tool_use and citation verification (this is the part most worth integration-testing — intentionally try to make the model cite a passage outside the retrieved set and confirm the runtime downgrades to `draft`). Then `fn-handoff` for drafts. Add the GBP intake lane (assuming your allowlisting came through) and the Facebook aggregator lane once the offline path works. Cross-cutting (audit, logs, alarms, budget, archive) goes in from day one.