7-PART SERIES · FREE COMPANION

Lead intake bot

A serverless intake bot that catches every new lead from your website forms, Meta Lead Ads, Google Ads, and the shared sales inbox, qualifies each one against your ICP, instantly routes the hot ones to the right person with full context attached, and quietly drops the cold ones into a nurture queue. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Lead intake bot

01 A lead intake bot on AWS for a few dollars a month

02 How a lead reaches the intake bot

03 How the bot reads a lead

04 How the bot scores and routes a lead

05 How a reply stays on policy

06 What the lead intake bot costs

07 Engineering reference: the lead intake bot architecture

PART 1 OF 7

MAY 3, 2026 PART 1 OF 7 · LEAD INTAKE BOT SERIES ~5 MIN READ

A lead intake bot on AWS for a few dollars a month

A “Contact us” form gets twelve submissions this week. Three are real buyers. One of those wants a quote by Friday. Four are tire-kickers. Three are competitors fishing. One is a vendor pitching themselves because nobody answered their cold email. One is spam. By Tuesday afternoon you’ve replied to two of them and forgotten the rest. This post walks through the design of a small intake bot that catches every new lead the moment it arrives, scores it against your ideal-customer profile, pings the right rep on the hot ones with full context, and quietly parks the cold ones for nurture.

KEY TAKEAWAYS

- Four outside surfaces, three AWS pieces. The bot turns four lead sources into one queue.
- Every lead ends in one of four moves: hot route, warm follow-up, nurture, reject.
- The qualifier writes only from your voice and pricing files. It never invents a discount, an SLA, or a feature you don’t offer.
- Hot leads ping the right rep in Slack within seconds, with a draft reply already written.
- Runs on AWS for about $3/month at typical small-business volume.

The whole system on one page

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

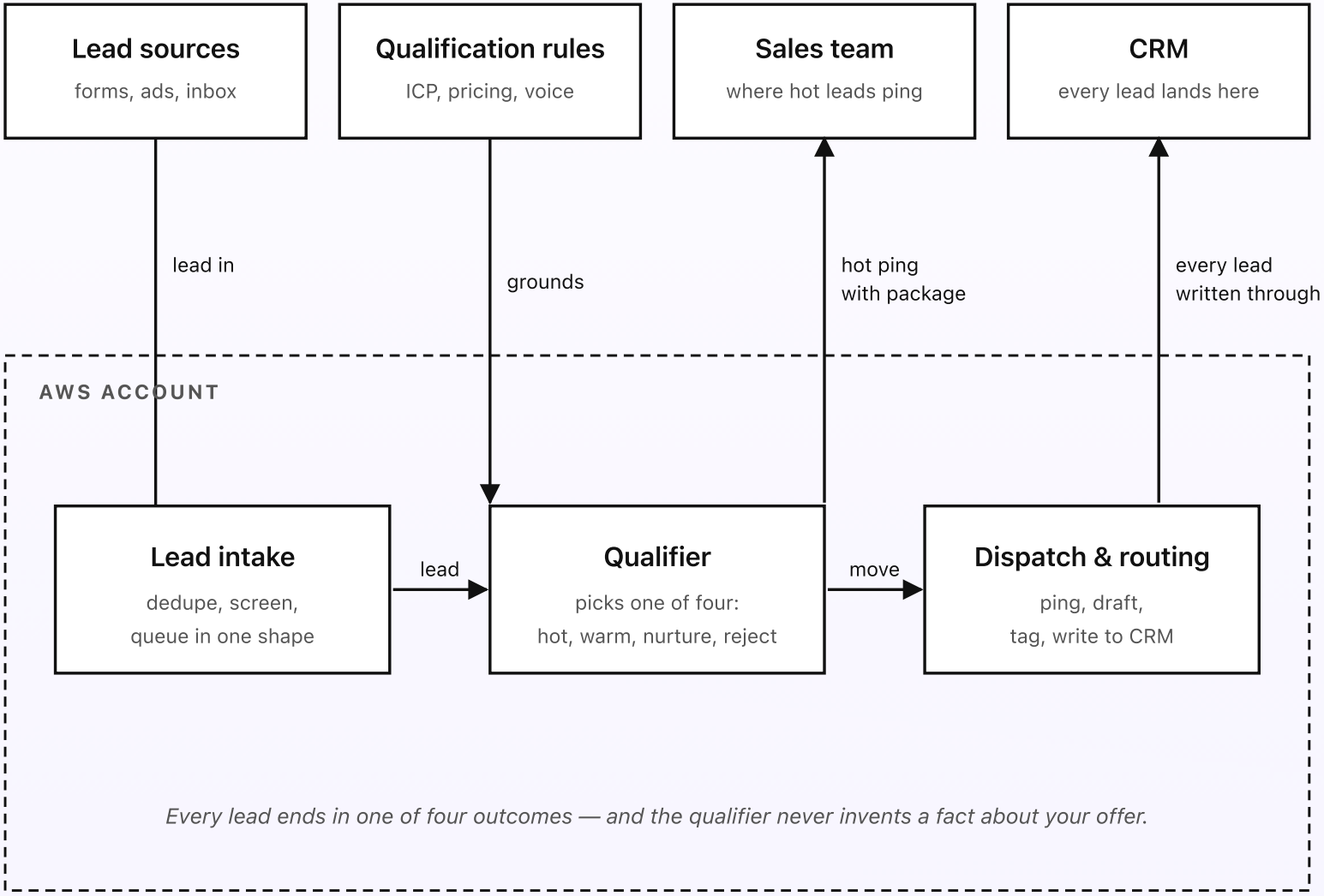

Lead sources
forms, ads, inbox
Qualification rules
ICP, pricing, voice
Sales team
where hot leads ping
CRM
every lead lands here
lead in
grounds
hot ping
with package
every lead
written through
AWS ACCOUNT
Lead intake
dedupe, screen,
queue in one shape
lead
Qualifier
picks one of four:
hot, warm, nurture, reject
move
Dispatch & routing
ping, draft,
tag, write to CRM
Every lead ends in one of four outcomes — and the qualifier never invents a fact about your offer.

Fig 1. Four outside surfaces, three pieces inside AWS. Leads flow in from your forms and platforms, the Qualifier picks one of four moves, and hot leads land in front of a human within seconds — with the original lead, a draft reply, and the matched ICP excerpt attached.

What you set up once (the outside)

- **Your lead sources.** The "Contact us" form on your website, your Meta Lead Ads (Facebook and Instagram lead forms), your Google Ads lead form assets, and the shared sales inbox. The bot takes webhooks from the platforms that push and polls the ones that don't. It reads inbound email through SES. You can disconnect any source in one click if a campaign goes sideways.
- **A rules folder.** Three short Google Docs in a Drive folder. *ICP and disqualifiers*: the buyers you sell to and the ones you don't, with the reasons for each (geography, company size, industry, role, budget). *Pricing and promos*: your tiers, what each one includes, the active campaign codes and their end dates, and what you can and can't commit to in a first reply. *Voice*: warm, brief, never pushy. The signature line and a few opening templates per source. Edit a doc and the bot picks up the change on the next refresh. No deploy.
- **A sales-team destination.** The Slack channel or shared queue your team already watches. Hot leads ping right away with the original message, the score and reasons, a draft first reply, the matched rules passage, and the suggested owner. Warm and nurture leads don't ping. They show up in the morning digest, where they belong.
- **Your CRM.** HubSpot, Salesforce, Pipedrive, or a Drive sheet for the smallest setups. Every lead lands here, no matter the move. Each row has the source,

the score, the parsed payload, and a link to the full audit row. Nothing falls through the cracks — not even the leads the bot decides not to escalate.

What runs on every lead (the inside)

- **The lead intake.** Receives or polls each source. Drops duplicates by email and phone, so the same person filling out three forms in five minutes doesn't become three rows. Screens out the obvious junk: banned-domain spam, competitor email domains, banned-phrase floods, submissions missing required fields. Writes the cleaned lead to a single queue. By the time the qualifier sees a lead, it's in one shape regardless of source.
- **The qualifier.** For every queued lead, reads the message and the form fields. Pulls out three things in parallel: *intent* (what they're asking for), *urgency* cues (deadlines, "ASAP," named events), and *fit* signals (industry, company size, budget mentions). Looks up the email domain for free enrichment. Then it scores against the ICP doc and picks one of four moves. *Hot route*: matches the ICP and shows real intent or urgency. *Warm follow-up*: plausible fit, no rush; deserves a thoughtful reply. *Nurture*: long-tail fit, no signal; tag and park. *Reject*: off-ICP, competitor, vendor pitch, or spam. When it writes a first reply, it uses only the voice file and the pricing file. It never invents a discount, a deadline, or an availability promise.
- **Dispatch and routing.** On *hot route*: pings the on-call sales owner in Slack with the full package, assigns ownership in the CRM, and starts a 15-minute timer if nobody acknowledges. On *warm follow-up*: drops the proposed reply in the team's draft queue with the matched pricing and ICP excerpts. A human glances at it and hits send. On *nurture*: tags the contact in the CRM with the right campaign and adds them to the slow-drip list. On *reject*: archives the lead

with a reason. On every lead, regardless of move, the bot writes the full row to the CRM and appends to the 8am digest the team reads in the morning.

In plain words

A new submission lands on your “Contact us” form. The cloud reads it within a second. The email is from a real-looking domain in your ICP. The company size matches. The message says “we’re replacing our current vendor by end of June, can we get a demo this week.” The cloud pings the on-call rep in Slack with four things: the original message, a one-line summary (“hot — mid-market, urgency: end of June, demo requested”), a draft reply pulled from the voice and pricing files, and a button to claim ownership in the CRM. The rep sees it on their phone. They tap claim, glance at the draft, change one sentence, hit send. The whole loop takes under two minutes. The lead arrived at 11:47pm Saturday and would otherwise have sat in the inbox until Tuesday morning.

Total cost runs in coffee-money territory at small-business volume. Pennies per lead, dominated by a few model calls per qualified lead.

DESIGN RULES THAT SHAPED EVERY DECISION

- The qualifier writes only from your voice file and your pricing file. It never invents a discount, an SLA, or a feature you don't offer.
- Four moves, always. Hot, warm, nurture, reject. There is no fifth.
- Off-ICP and competitor leads never ping the team. They're tagged and archived with a reason. Pings are a finite resource.
- Hot leads always include the full context: original message, score reasons, draft reply, matched ICP excerpt. The rep never has to dig.
- Configuration lives in Drive. Editing your ICP or your pricing doesn't need a deploy.
- Every lead is written to the CRM, regardless of move. Nothing falls through.

Why this shape

Most "lead automation" tools fall into one of two traps. The first kind dumps every form submission into the team's shared inbox, or pings every one to Slack. The team mutes the channel within a week. The third "hot lead!" ping that turns out to be a vendor pitching their lead-automation product is the moment the alerting stops working. The second kind qualifies with a static rubric ("company size > 50 = +20 points") that can't read the actual message. So an obvious buyer with a small website and a one-line message gets scored as cold. A tire-kicker from a Fortune 500 IP gets scored as hot. Neither is what you want at 11pm on a Saturday

when a real buyer fills out your form because they finally got time to think about it after the kids went to bed.

The setup above splits the difference. Real buyers with real signal ping the on-call rep within seconds, with enough context to reply intelligently from a phone. Plausible buyers without signal become drafts the team reviews in batches; the expensive part (writing the reply) is already done. Long-tail fits get tagged and parked in nurture. Junk — vendor pitches, competitor fishing, spam — gets archived with a reason. The team never sees the junk, but you can audit it later if a real lead ever ends up there by accident.

The next four posts walk through each piece in turn: how a lead reaches the bot, how the bot reads it, how it scores and routes, and how the first-touch reply stays on policy. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 3, 2026 PART 2 OF 7 · LEAD INTAKE BOT SERIES ~4 MIN READ

How a lead reaches the intake bot

Leads don’t arrive at your business through one door. Your website form posts to an HTTPS endpoint. Meta Lead Ads pushes a webhook the moment a Facebook or Instagram user submits a lead form. Google Ads pushes through a lead form asset webhook (or a poll of the conversion-import API for older campaigns). And your shared sales inbox sees the rest. The intake’s job is to fold those different mechanisms into one queue, drop duplicates, and screen out junk before any AI sees a single field.

KEY TAKEAWAYS

- Three intake lanes feed one queue: website form, ad platforms (Meta + Google), and the shared sales inbox via SES.
- Push when the platform offers it; poll when it doesn’t. Each lane has its own current-2026 reality.
- Dedupe drops emails or phones already seen in the last 24 hours.
- Screen kills banned-domain spam, competitor emails, and missing-required-field junk.
- Both filters run in plain Lambda code — no AI, no token cost.

Three lanes at the door

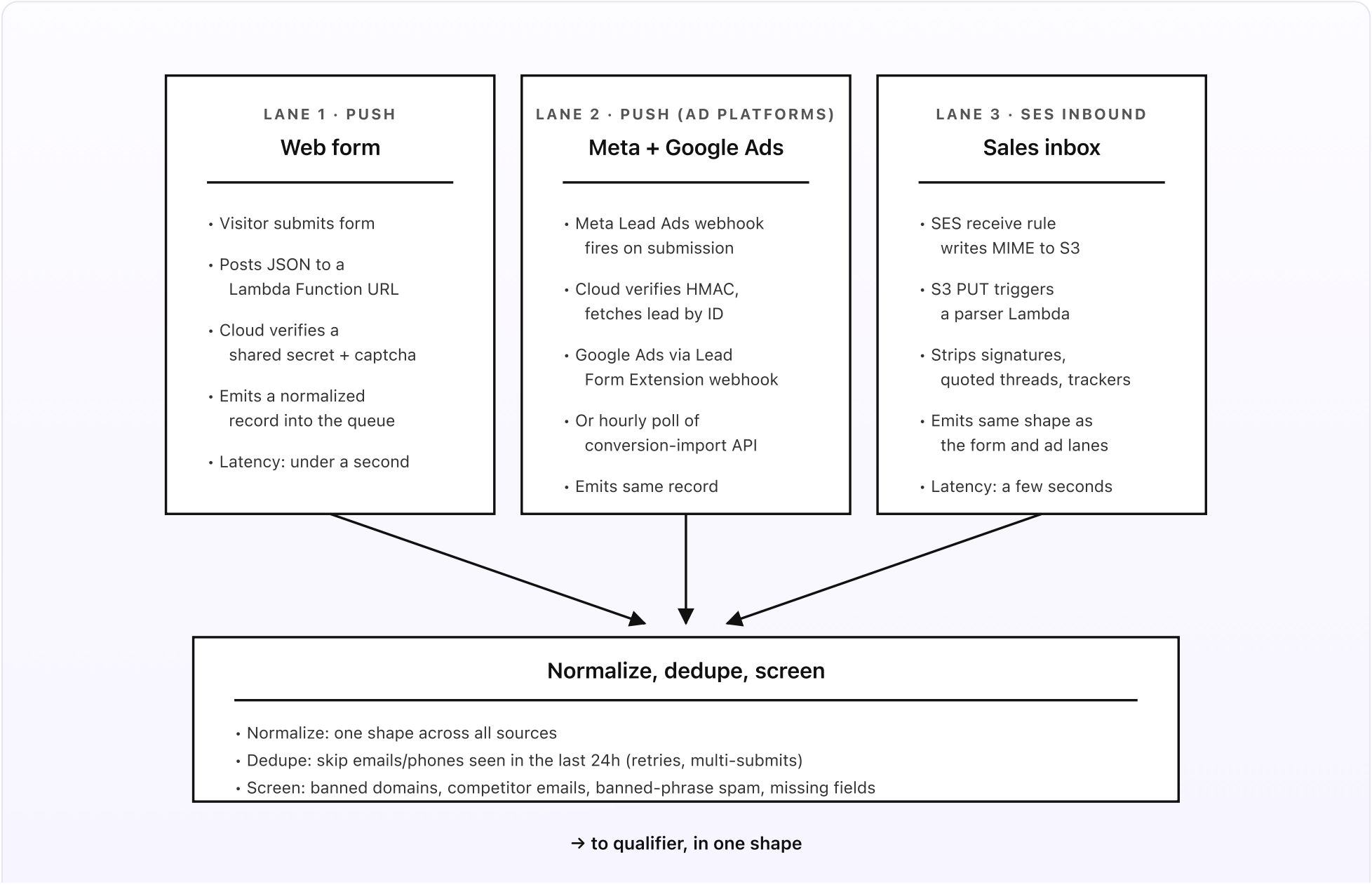

LANE 1 · PUSH
Web form
• Visitor submits form
• Posts JSON to a Lambda Function URL
• Cloud verifies a shared secret + captcha
• Emits a normalized record into the queue
• Latency: under a second
LANE 2 · PUSH (AD PLATFORMS)
Meta + Google Ads
• Meta Lead Ads webhook fires on submission
• Cloud verifies HMAC, fetches lead by ID
• Google Ads via Lead Form Extension webhook
• Or hourly poll of conversion-import API
• Emits same record
LANE 3 · SES INBOUND
Sales inbox
• SES receive rule writes MIME to S3
• S3 PUT triggers a parser Lambda
• Strips signatures, quoted threads, trackers
• Emits same shape as the form and ad lanes
• Latency: a few seconds
Normalize, dedupe, screen
• Normalize: one shape across all sources
• Dedupe: skip emails/phones seen in the last 24h (retries, multi-submits)
• Screen: banned domains, competitor emails, banned-phrase spam, missing fields
→ to qualifier, in one shape

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

Why three different mechanisms

The three lanes look different because the platforms behind them work differently. The intake mirrors those differences honestly instead of pretending they're uniform.

Your own form is the easiest lane. The form posts JSON over HTTPS to a Lambda Function URL with a shared secret in the body and a captcha token in the headers. The cloud verifies both before doing any work. Captcha alone catches most automated form spam. The shared secret stops anyone from POSTing to the URL directly even if they find it in your page source. You own this lane end-to-end. No platform deprecation can break it.

Meta Lead Ads is push, with a small twist. When a Facebook or Instagram user submits a lead form attached to an ad, Meta's webhook fires within seconds. But the payload is just the `lead_id` and the form ID, not the answers. The cloud verifies the `X-Hub-Signature-256` HMAC (App Secret as the key, computed over the raw body, constant-time comparison), then makes a separate call to the Graph API to fetch the field values. That second call is the only fragile bit. The page access token expires every 60 days, so a small refresh worker rotates it before it lapses. Pin to `v24.0` or `v25.0` on outgoing calls. `v18.0` sunset on 2026-01-26, `v19.0` sunsets on 2026-05-21, and `v20.0` sunsets on 2026-09-24.

Google Ads has two patterns. If your campaigns use a *lead form asset* (Google's current name for the surface formerly called the Lead Form Extension), you can configure a webhook URL right in Google Ads with a `google_key`. Google posts the form fields the moment a user submits, and you have the lead immediately. If your campaigns don't use a lead form asset (search ads with a different lead surface, older campaigns), the alternative is a small hourly poll of the conversion-import API to pick up anything new. Both patterns end at the same normalized record.

The inbox lane catches everything else. Plenty of leads still come in as plain emails: a partner referral, a reply to a cold email, a sign-up from a directory site that doesn't do webhooks, a forwarded RFP. SES inbound rules accept email at your domain, write the raw MIME to an S3 bucket, and trigger a parser Lambda. The parser strips signatures, quoted threads, and tracking pixels (the same tools the email-assistant series uses), then emits the same normalized record as the other lanes. The bot doesn't care that the lead came through email; the downstream code reads `name`, `email`, and `message` just like a form submission.

Mixing all three in the same intake means the qualifier doesn't care which source a lead came from once it's in the queue. The downstream code never branches on source, only on content. Source is a tag on the record, not a control-flow split.

What "normalize" actually means

Each source hands the cloud a different shape. A web form gives you the field names you defined. Meta Lead Ads returns an array of question-answer pairs keyed by your form's field names (which aren't always what you'd call them).

Google Ads gives a flat key-value object with mostly stable names. SES gives raw MIME you have to parse. Normalization folds those into one common lead object: a source name, a stable lead ID, the contact (name, email, phone), the company (domain, name if provided), the free-text message, the timestamp, the campaign or page identifier, and a small bag of source-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 qualifier doesn't have to know what Meta calls the email field. It just reads `contact.email`.

Dedupe and screen, before any AI runs

Two free filters sit between the lanes and the qualifier.

Dedupe drops a lead whose email or phone has been seen in the last 24 hours. Real buyers occasionally submit your form twice in a minute (different tabs, slow connection). Webhooks retry on transient failures. Meta sometimes double-fires a lead webhook if its backend gets unsure. Without dedupe the team gets pinged twice on the same hot lead and one of them goes stale. With it, the second arrival is dropped quietly and the existing row is updated with whatever the second submission added.

Screen handles the obvious junk: banned-domain spam (the same crypto-pump message posted to a thousand contact forms), competitor email domains (a list the sales team maintains), banned-phrase floods ("buy our SEO service"), and submissions missing required fields (an email with no message). Vendor pitches with "I work for X and would love to discuss" phrasing get their own bucket.

They're archived, not deleted, so a real lead with that phrasing can be retrieved on appeal. The screen runs in plain Lambda code with a small banned-list and a regex. No AI involved.

The point of doing both before the AI runs is simple. 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 lead.

What this hands to the next post

By the time a lead leaves the intake, it's in one shape, with a stable ID, no duplicates, no obvious spam, and a source tag the downstream code uses for analytics but not for branching. The next post is about what the qualifier does with that — how it actually reads the lead, extracts intent and urgency and fit signals, and starts to decide which of the four moves applies.

PART 3 OF 7

MAY 3, 2026 PART 3 OF 7 · LEAD INTAKE BOT SERIES ~5 MIN READ

How the bot reads a lead

A lead reaches the qualifier as one clean object: a contact, a company, a free-text message, and a source tag. Reading it well is the difference between a hot ping that lands a meeting and a hot ping that wakes a rep at 2am for a tire-kicker. Three small extractors run in parallel against the message. Each looks at one thing only. Each returns a confidence score the move-picker reads before it decides anything.

KEY TAKEAWAYS

- Three small extractors run in parallel against the lead message.
- Intent (demo, quote, info, partnership, support, other), urgency (deadlines, switch language), fit (industry, size, role, budget).
- Plus a free passive enrichment from the email domain — free-mail vs business, partner allowlist, competitor list.
- Each extractor returns a 0–1 confidence score the move-picker reads before it decides anything.
- Splitting into narrow extractors makes failures small and easy to fix.

Three extractors, one consolidated read

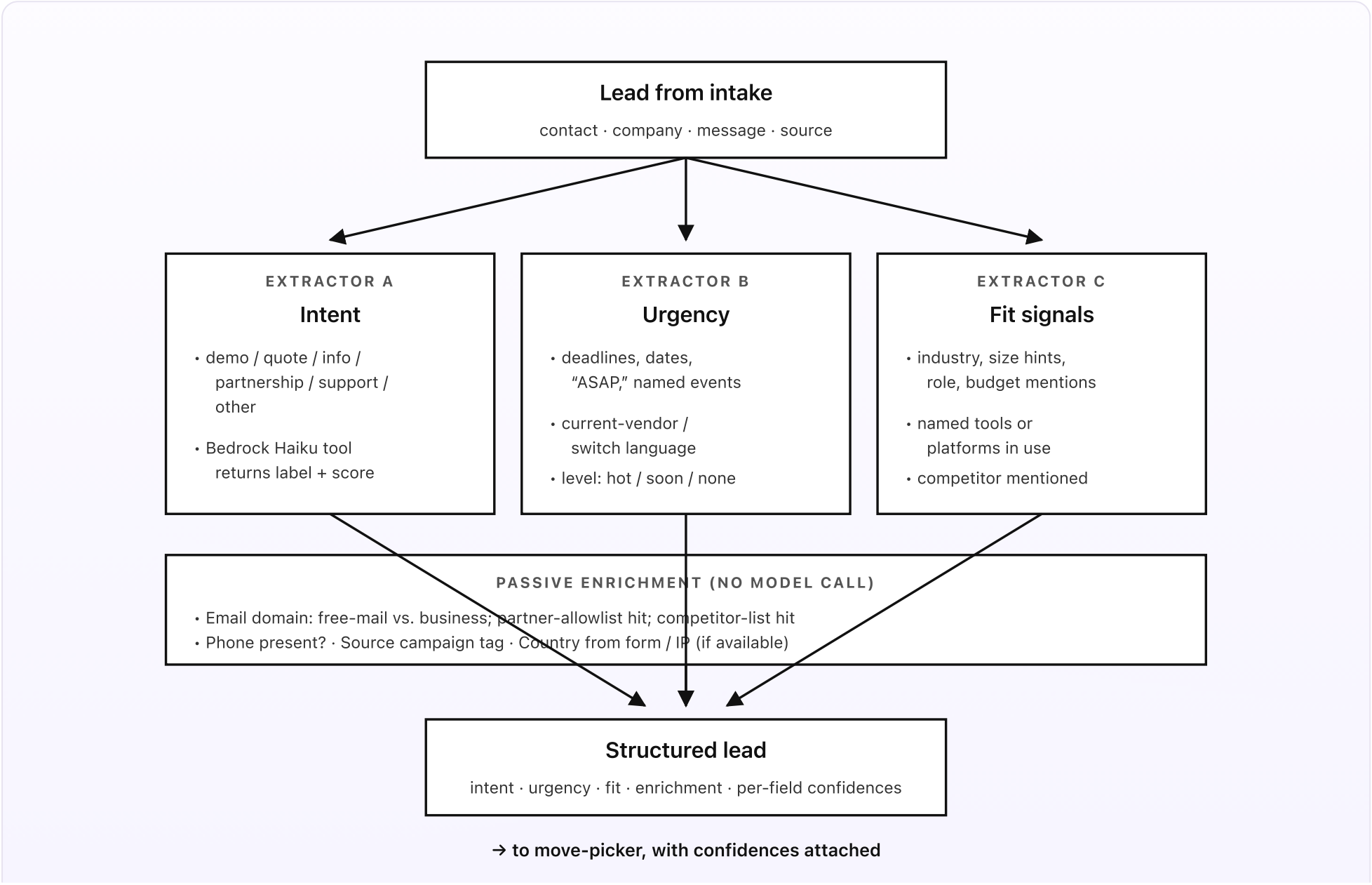

Lead from intake
contact · company · message · source
EXTRACTOR A
Intent
• demo / quote / info / partnership / support / other
• Bedrock Haiku tool returns label + score
EXTRACTOR B
Urgency
• deadlines, dates, "ASAP," named events
• current-vendor / switch language
• level: hot / soon / none
EXTRACTOR C
Fit signals
• industry, size hints, role, budget mentions
• named tools or platforms in use
• competitor mentioned
PASSIVE ENRICHMENT (NO MODEL CALL)
• Email domain: free-mail vs. business; partner-allowlist hit; competitor-list hit
• Phone present? · Source campaign tag · Country from form / IP (if available)
Structured lead
intent · urgency · fit · enrichment · per-field confidences
→ to move-picker, with confidences attached

Fig 3. Three small extractors plus a free enrichment, run in parallel. Each one knows one thing only — easier to tune, easier to debug, easier to swap. The move-picker reads the consolidated record next.

Why three extractors instead of one big prompt

You could ask one large prompt to read a lead and return everything — intent, urgency, fit, summary, suggested move — and it would mostly work. But "mostly" is the problem. When the consolidated answer is wrong, you don't know which sub-decision went wrong. The model entangles them and you end up rewriting a 600-line prompt to fix one symptom. Splitting into three narrow extractors gives you three small failures instead of one big one. When intent is wrong, you tune the intent prompt without breaking urgency. When urgency starts overweighting words like "today" (which often means "today I had a thought," not "I need this today"), you fix it with a one-line clarification in the urgency tool description. Fit and intent stay exactly the same.

The three extractors run in parallel against Bedrock Haiku. Each one has a tightly scoped tool definition that forces the model to return a typed object plus a 0-to-1 confidence score. That's three model calls per lead instead of one. But each call is small and fast, and the parallelism means wall-clock time is the slowest of the three, not their sum. Per-lead spend is still measured in tenths of a cent.

Extractor A — intent

What is the lead actually asking for? The intent extractor returns one of a small fixed set: *demo*, *quote*, *info*, *partnership*, *support*, *other*. The label matters because the four moves downstream branch heavily on it. A *support* intent is

almost never a hot route — it belongs in your help system, not a sales rep's phone. A *partnership* intent goes to a different owner than a *quote*. A clear *demo* request from the right ICP is the easiest hot ping you'll ever send. The confidence score lets the move-picker downgrade an uncertain "is this a quote or just info?" lead from hot to warm, even if everything else lines up.

Extractor B — urgency

Hot leads have time pressure. The urgency extractor reads the message for explicit deadlines ("by end of June"), implicit ones ("our current contract expires next month"), named events ("before our board meeting"), and switch language ("we're replacing X," "we're evaluating alternatives to Y"). It returns a level — *hot*, *soon*, or *none* — with a confidence score and the specific phrases it picked up. The phrases matter. When a rep gets the hot ping, the message reads "urgency: hot — mentioned 'contract expires June 30'," not just "urgency: hot." They can check the AI read the message correctly before they pick up the phone.

This extractor is the one most prone to false positives. Words like "quickly" and "soon" show up in friendly throwaway sentences as often as in real urgency. The tool definition includes explicit examples of the difference, and the confidence score reflects when the model is unsure. The move-picker is calibrated to require either a high-confidence hot urgency or a clear deadline phrase before it pings the team. An enthusiastic but vague "hoping to hear back soon" doesn't qualify.

Extractor C — fit signals

The third extractor reads the message for everything that helps decide whether the lead matches the ICP. Industry mentions. Company-size hints ("our team of 30," "we're a small agency"). Role mentions ("I'm the head of operations," "I run

procurement"). Budget signals ("we have around $5K/month allocated"). Named tools or platforms in use ("we're currently on HubSpot"). Competitor mentions. It also flags the absence of those signals. A one-line message with no fit signals at all is a different kind of lead than one with three.

The fit extractor doesn't score the lead against your ICP. That's the move-picker's job in the next post, with the ICP doc as input. The extractor just pulls out the raw signals so the scoring step has something concrete to compare against.

Passive enrichment, free of charge

One more piece of context arrives without a model call. The email domain is checked against three small lists. A free-mail providers list (gmail.com, yahoo.com, outlook.com, etc.) versus business domains. A partner allowlist (companies you have an existing relationship with, where their leads should always route hot to a specific owner). A competitor list (companies whose leads should be archived with a reason). A phone-present flag, the source campaign tag, and a country code (when the form or ad platform supplied one) round out the enrichment. None of this needs Bedrock. All of it costs essentially nothing.

The free-mail-versus-business signal is surprisingly load-bearing for B2B leads. A lead from `jane@somecompany.com` is a different lead than the same words from `jane@gmail.com`. Not always, but often enough that the move-picker should know.

What the move-picker sees next

By the end of this step, the qualifier has built a structured lead: `{ contact, company, message, source, intent: { label, confidence }, urgency: { level, confidence, phrases }, fit: { signals[], absent[] }, enrichment: { domain_class, partner, competitor, phone_present, country } }`. The next post is about how the move-picker reads this object, scores it against your ICP file, and picks one of four moves. Two override gates handle the cases that bypass scoring entirely.

PART 4 OF 7

MAY 3, 2026 PART 4 OF 7 · LEAD INTAKE BOT SERIES ~5 MIN READ

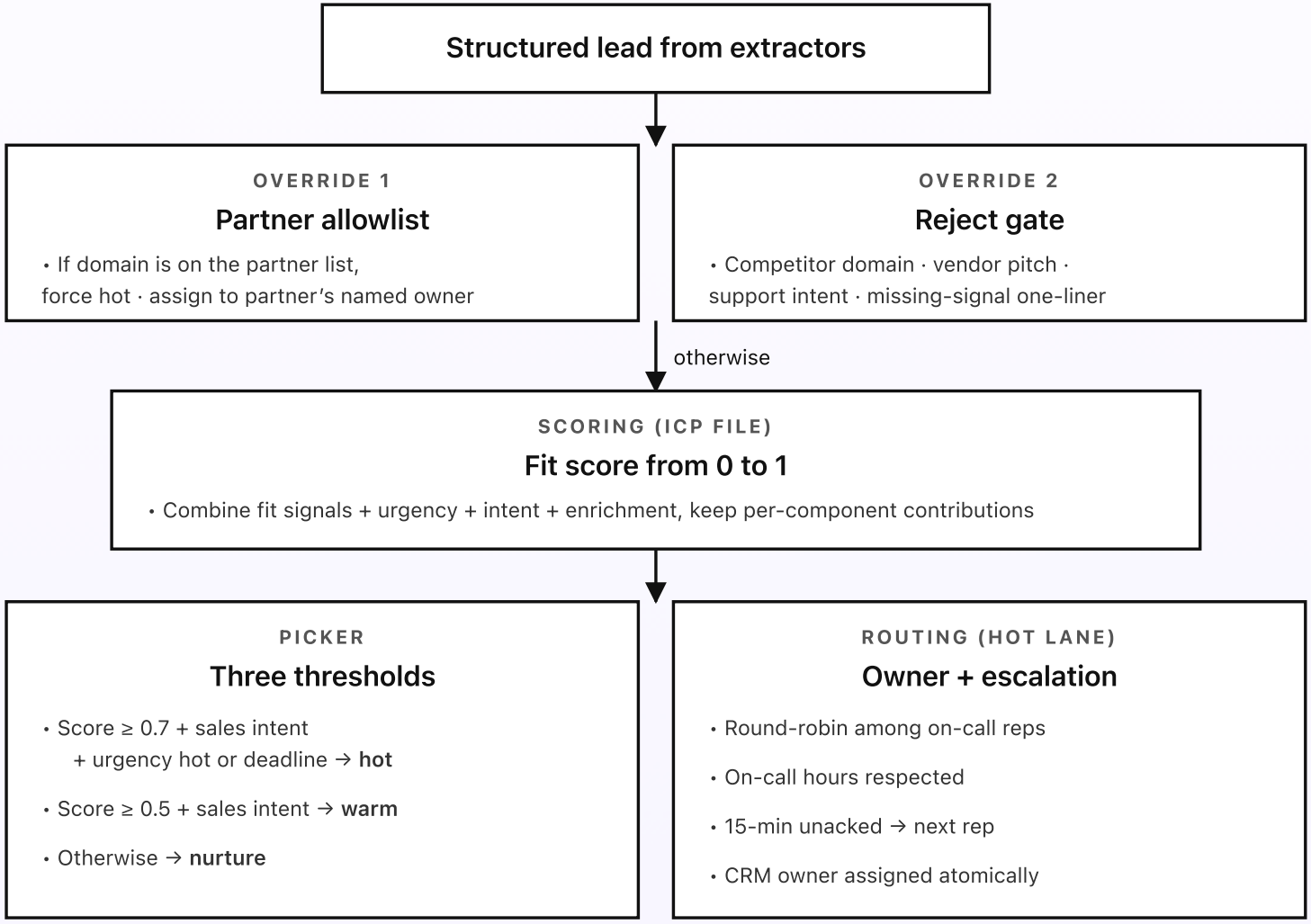
How the bot scores and routes a lead

A structured lead arrives at the move-picker with a clean read of intent, urgency, fit signals, and free enrichment. Now the bot has to commit. Four moves, one pick per lead. Two overrides that bypass scoring entirely. A round-robin owner assignment that respects who's on call. The whole step is small. The hard work was the reading.

KEY TAKEAWAYS

- Four moves, one pick per lead: hot route, warm follow-up, nurture, reject.
- Two override gates fire before scoring: partner allowlist forces hot; reject gate kills off-ICP, competitors, support, and one-line junk.
- The linear scorer reads the ICP file and combines fit signals, urgency, intent, and enrichment into a single 0–1 score.
- Hot needs three things at once: score ≥ 0.7, sales-shaped intent, and either hot urgency or a clear deadline phrase.
- Round-robin assignment respects on-call hours; a 15-minute escalation timer re-routes if the first rep doesn't acknowledge.

Four moves, one pick



Cold leads never ping the team — pings are a finite resource. Every lead still lands in the CRM.

Fig 4. Two override gates fire first — partner forces hot, the reject gate kills off-ICP and support. Then a single fit score against the ICP file. Three thresholds pick the move; the routing column assigns an owner only on hot.

The two overrides come first

Before any scoring, two override gates fire on every lead.

The partner allowlist forces hot route. Companies you already have a relationship with — named partners, current customers asking about an expansion, the four agencies you trade referrals with — should never be slowed down by score thresholds. If the email domain is on the partner allowlist (a small list in the policies file), the move is hot. The owner is the named partner-account contact. The ping carries a tag that tells the rep this isn't a cold lead. An existing customer asking about an upgrade should not be scored by the same rubric as a stranger filling out a form for the first time.

The reject gate kills off-ICP fast. Four signals drop a lead straight into reject without ever hitting the scoring step. The email domain matches a competitor on the competitor list. The intent extractor returned *support* — that's a help-system question, not a sales lead, and it routes to your support inbox instead. The screen earlier flagged a vendor pitch. Or the message is a one-liner with no fit signals, no urgency phrases, and no specific question. (These are almost always typos, tests, or soft "just looking" messages that belong in nurture, not in front of a rep.) Each rejection records its reason in the audit row so you can spot a misfire on review.

Override gates exist for the same reason guardrail keywords exist on the review responder. Some categories of leads have a right move so obvious that running

them through scoring is just expensive noise. Catch them at the front of the pipeline. The scoring step has fewer things to balance, and the team never sees the rejected ones.

The scoring step

For everything that survives the overrides, the move-picker computes a single fit score from 0 to 1. The score combines four inputs from the structured lead:

- **Fit signals against the ICP doc.** The ICP doc lists the industries, company sizes, geographies, and roles you sell to, plus disqualifiers. The picker matches the extracted fit signals against that list and produces a sub-score.
- **Urgency level.** Hot urgency adds a fixed bump. A deadline phrase adds another. None contributes nothing.
- **Intent shape.** Sales-shaped intents (demo, quote, partnership) carry weight. *Info* contributes a smaller amount. *Other* contributes nothing on its own.
- **Enrichment.** A business-domain email adds a small bump over a free-mail address. The absence of a phone number doesn't penalize; the presence of one slightly helps.

The combination is a small linear formula in code, not a model call. It runs in microseconds. It costs nothing. It's fully auditable — every score in the audit row breaks down into its four sub-scores, so you can see why a given lead came out at 0.62 instead of 0.75. When you change your ICP, the formula doesn't change. The ICP doc does, and the fit-signals sub-score reads the new doc on the next refresh.

The three thresholds

From the score, the move-picker chooses between hot, warm, and nurture. Reject was already handled.

Hot needs three things at once: score at least 0.7, intent that's sales-shaped, and either hot urgency or a clear deadline phrase. All three is the bar because hot pings are expensive — not in dollars, in attention. The rep's phone goes off. A wrong hot ping costs you trust in the system. A score of 0.71 with no urgency is a warm lead, not a hot one. The rep gets to it Monday, not at midnight.

Warm is a score of at least 0.5 with a sales-shaped intent. These don't ping. They show up in the team's draft queue with the proposed first reply already written. The rep glances at the draft, edits a sentence, sends it. The follow-up SLA is "within one business day," not "within fifteen minutes." That's the right SLA for these.

Nurture is everything else: low score, info-only intent, or no urgency on a borderline fit. The bot tags the contact in the CRM with the right campaign, adds them to the slow-drip list, and writes a row to the 8am digest the team reads in the morning. Long-tail fits that show up later as real leads (because their need finally became urgent six months after first contact) get re-evaluated automatically when they re-engage. The dedupe step in intake checks for prior leads on the same email and surfaces the history to the qualifier.

Routing the hot move — owner and escalation

Hot leads need an owner. The routing step picks one round-robin from the list of on-call sales reps in the policies file, with two refinements that matter on a real team. *On-call hours are respected.* A lead that lands at 3am gets queued for the next on-call rep waking into their shift — not pinged into the previous shift's now-asleep phone. *Acknowledgement is timed.* If the assigned rep doesn't click the acknowledge button within fifteen minutes, the lead re-routes to the next rep on the list with a quick "reassigned because the first one didn't pick up" tag. The CRM owner field is set atomically when the rep clicks acknowledge, so two reps can't both think they own the same lead.

For warm leads, the owner field stays unassigned until a rep picks it up from the draft queue. For nurture leads, it stays unassigned indefinitely (campaigns don't need owners). For partner-override hot leads, the owner is forced to the named partner-account contact, ignoring round-robin entirely.

What the next post handles

By the end of this step every lead has a move, a score breakdown, and (for hot moves) an assigned owner with an escalation timer. What it doesn't yet have is the actual auto-acknowledgement reply — the warm draft and the optional hot first-touch. The next post is about how that reply gets composed without the bot inventing a discount, an SLA, or an availability promise you don't offer.

PART 5 OF 7

MAY 3, 2026 PART 5 OF 7 · LEAD INTAKE BOT SERIES ~5 MIN READ

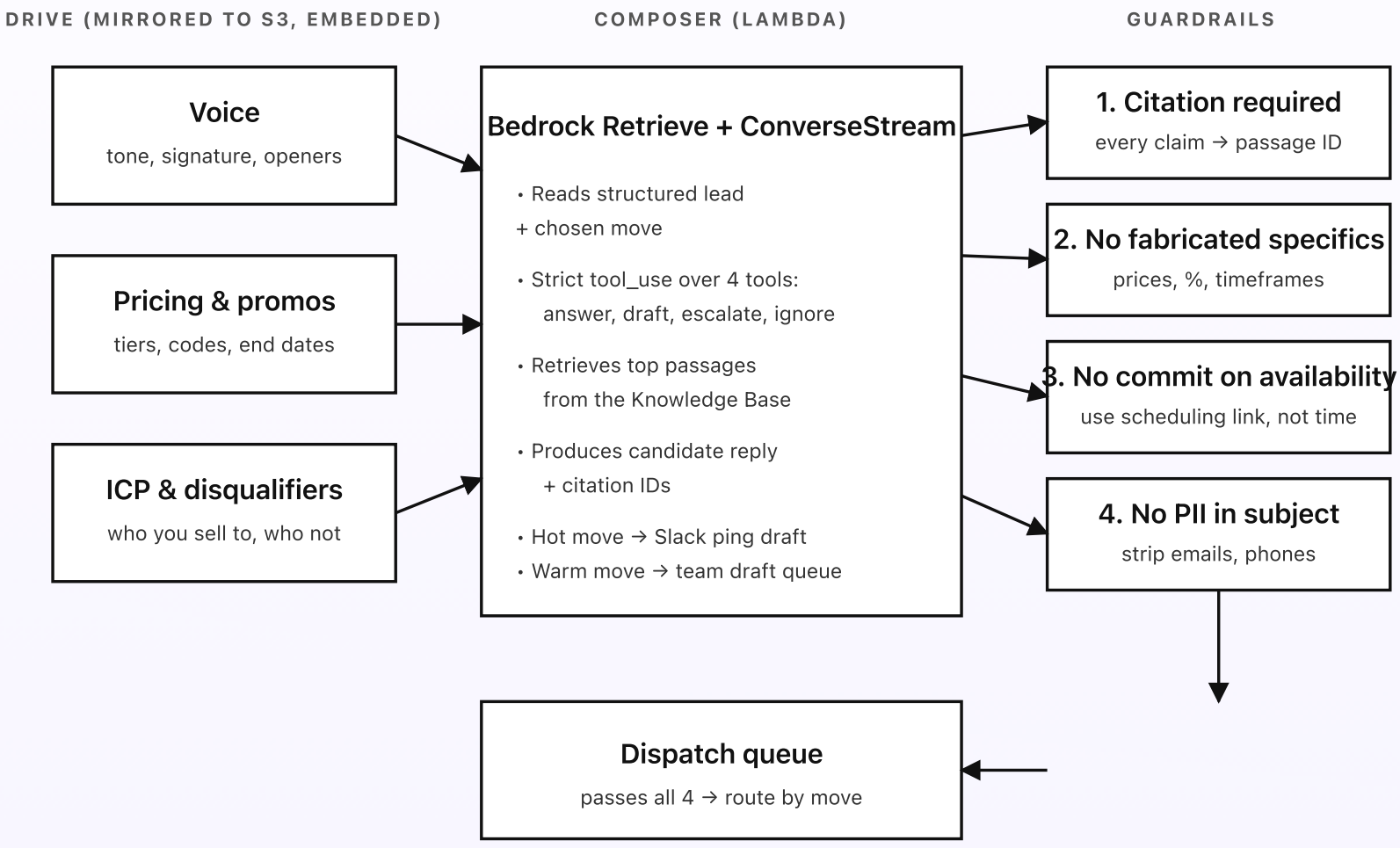
How a reply stays on policy

A first-touch reply to a real lead is one of the easiest places for an AI to make an expensive mistake. A discount that doesn't exist. An SLA the team can't meet. A "yes, we offer that" on a feature you don't. The composer in this system can't do any of those things, by design. Three Drive docs ground every reply. Four guardrails sit between the model and the send button.

KEY TAKEAWAYS

- Three Drive docs ground every reply: voice, pricing-and-promos, and ICP-and-disqualifiers.
- Four guardrails sit between the model and the send button.
- Citation required: every factual claim must point to a retrieved passage.
- No fabricated specifics, no commit on availability, no PII in subject lines.
- Hot leads never auto-send. Warm follow-ups can — but only when explicitly enabled per source-and-intent.

Three docs in, four guardrails out



A reply that fails any guardrail is downgraded, never sent, and audited.

Fig 5. Three docs ground the composer; four guardrails gate the send button. The model is required to cite a retrieved passage for every factual claim — if it can't, the reply downgrades to a draft for human review.

Three Drive docs ground every reply

Voice is the smallest of the three. A page of how you sound: warm, brief, never pushy. The sentence templates you use to open and close. A signature line per source (a Meta Lead Ads response opens differently than a website-form response). The voice doc is the part most likely to change month to month as you tune what works. Keeping it small means edits are easy, and the model has nowhere to hide if a phrase slipped past you.

Pricing and promos is where the real risk lives. Your tier names. What each one includes. What it doesn't. The active campaign codes and their end dates (so a January discount doesn't get quoted in May). And an explicit list of things you can and can't commit to in a first reply. "We can offer a free 30-day trial" goes here. "We can match any competitor's price" never goes here, even if you sometimes do, because the bot can't make case-by-case judgement calls. Edit this doc the moment a campaign ends. The composer reads the update on the next refresh. No deploy.

ICP and disqualifiers is the same doc the move-picker reads. Including it in the composer's grounding lets the reply gracefully de-escalate when the lead isn't a fit. Instead of pushing on a sale that won't happen, the bot can write "we focus on companies in [these industries]; we may not be the best fit, but here's a partner

who might be" from a passage you wrote. Disqualifying a lead politely is a feature the disqualifiers doc enables.

All three docs are mirrored from Drive to an S3 prefix every five minutes by a small sync Lambda. Bedrock Knowledge Bases don't have a native Drive connector, so the mirror is the bridge. The Knowledge Base reads from S3, chunks each doc by heading, embeds each chunk with Titan Embeddings, and stores the vectors in the managed S3 Vectors index. At composer time, the model retrieves the top passages relevant to the lead before it writes a single word.

Four guardrails between the model and the send button

Guardrail 1 — citation required. The composer runs with strict tool_use. The `answer` tool requires a `citation_passages` array referencing one or more retrieved passages by ID. Every factual claim about pricing, availability, or features must point to a passage. If the model emits an answer with a citation that wasn't in the retrieved set (a hallucinated reference), the runtime downgrades the move from auto-send to draft. That's the safer-by-default failure mode, not a silent error. The rep gets the proposed reply with a flag "citation verification failed" and decides what to do.

Guardrail 2 — no fabricated specifics. A regex sweep over the candidate reply looks for prices, percentages, and timeframes: "$X," "X%," "within X days," "by tomorrow," specific dates. Every match is checked against the cited passages. The price has to actually appear in pricing.gdoc. The "within 30 days" SLA has to actually appear in a passage. Anything specific that isn't in the source downgrades the reply. This catches a class of failures that pure citation

enforcement misses, because models occasionally cite a passage and then write a number that isn't in it.

Guardrail 3 — no commit on availability. A small block-list of phrases like "we can have someone on a call tomorrow at 2pm" or "Friday at noon works" is rejected outright. The bot doesn't know the team's actual availability and shouldn't pretend to. The composer is required to use scheduling-link language instead: "here's a link to book a time that works for you," not "does Friday at 2pm work for you?" This guardrail is the one most likely to false-positive on a friendly "happy to chat soon." The block-list is tuned for specificity, not vagueness — it rejects committed times, not warm intent.

Guardrail 4 — no PII in subject lines. Outbound email subjects get a final pass that strips any token looking like an email address or a phone number. Subject lines often end up logged in cleartext by mail providers, monitoring tools, and Slack notifications. A subject like "Re: lead from `jane@example.com`" is a quiet PII leak across all of those. The bot uses opaque references ("Re: your inquiry," "Re: about your demo request") and the lead's email address only inside the body.

What the composer doesn't do

The composer never sends an auto-reply for a hot lead. Hot moves always go to the human first, with the proposed first reply as a draft inside the Slack ping. The bot writes; the human chooses to send. Auto-send is reserved for warm follow-ups, and only where the team has explicitly enabled it for a specific source-and-intent combination. A typical setup might auto-send for warm *info*-intent leads on the website-form lane — where the message is "send me your pricing sheet" and

the right answer is “here’s the pricing page link, and a person will follow up.” Warm *quote*-intent leads never auto-send. They wait for a human glance.

The composer also never invents a feature, a partnership, a customer logo, or a case study. If the lead asks “do you integrate with HubSpot?” and the pricing doc doesn’t mention HubSpot in the integrations passage, the reply has two options: “great question, let me get the right person to confirm,” or escalate to a human. There is no third option that involves the bot guessing.

What the next post handles

The composer is the last AI step in the pipeline. The next post is a cost breakdown — what all of this actually adds up to per month at the volumes a small business sees, and where the dollars go when the volume grows.

PART 6 OF 7

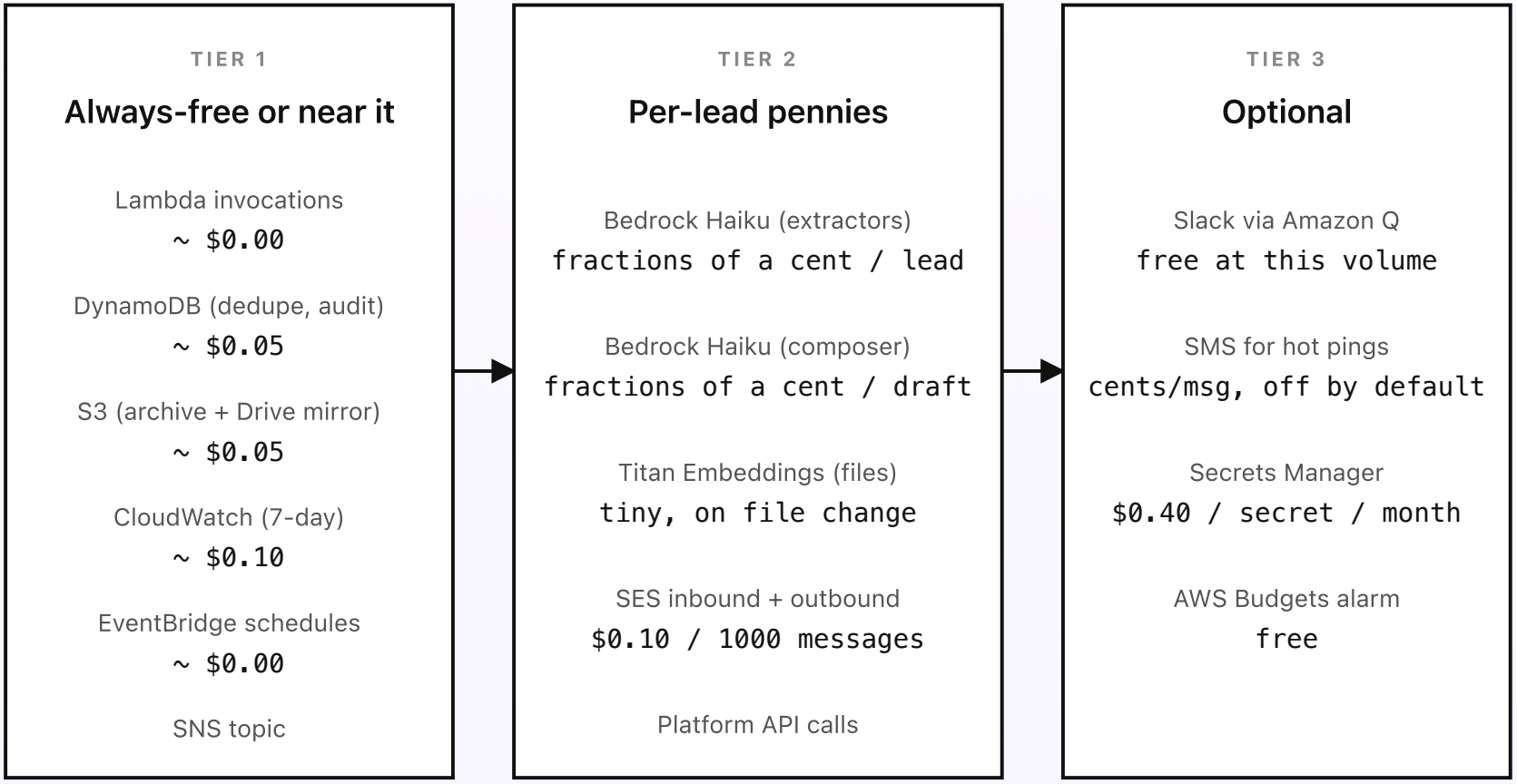
MAY 3, 2026 PART 6 OF 7 · LEAD INTAKE BOT SERIES ~3 MIN READ

What the lead intake bot costs

A coffee a month at SMB lead volume. The fixed cost is essentially zero. If your campaigns 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 · VERIFIED MAY 2026

- Fixed cost is essentially zero. Quiet weeks bill nothing.
- Variable cost is pennies per lead, dominated by Bedrock tokens for the extractors and the composer.
- ~100 leads/month → under three dollars total.
- ~1000 leads/month → under ten dollars total.
- A $10 AWS Budgets alarm catches anything strange before it surprises you.



~ 100 leads/month → under three dollars total. ~ 1000 leads/month → under ten.

Fig 6. Three tiers of cost. The bill scales with how many leads land; the floor is nearly zero.

The fixed cost is essentially zero

There is no per-lead licence, no minimum monthly fee, no “starter plan.” If your campaigns are quiet for a 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 Drive-sync schedule every five minutes and the Google Ads poll every hour, all month long, costs a vanishingly small amount. EventBridge scheduled invocations are essentially free at this volume, and each fire is a single Lambda call that does a small amount of work.

The biggest single line item in tier one is usually CloudWatch log storage. 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-lead pennies

Three things scale with lead volume:

- **Bedrock Haiku tokens for the extractors.** Each lead pays for three small model calls (intent, urgency, fit) running against the message text. At small-model rates and the short input lengths most lead messages have, the all-in cost per lead for extraction lands at fractions of a cent.

- **Bedrock Haiku tokens for the composer.** Hot-route drafts and warm follow-up drafts each pay for one composer call. The input is the structured lead plus the relevant pricing, voice, and ICP excerpts retrieved from the Knowledge Base. The output is a short reply. Even on the most expensive moves this is well under a cent per lead. The composer doesn't fire on nurture or reject, so cold leads cost only the extractor tier.
- **Titan Embeddings for the policy docs.** One-off per file change. If you edit your pricing doc twice a month and your voice doc once, that's three embeddings runs totalling tiny fractions of a cent. The embeddings power the citation gate — finding the right policy passage to ground a reply.

Add it up at typical small-business lead volumes (fifty to a few hundred leads a month across all sources). Tier two lands in the under-three-dollars range. Tier one floors are under fifty cents. The all-in monthly bill is a coffee a month. At higher volumes — a busy B2B campaign at a thousand leads a month — tier two becomes the headline number. Even there it stays in the "phone bill, not Netflix subscription" range.

Three traps you're avoiding

- **Per-seat sales-engagement tools.** Many off-the-shelf lead-routing tools charge $50–$300 per seat per month, regardless of lead volume. They assume your SDR team is permanent and full-time. You're trading a flat per-seat bill for pay-per-use that mostly comes in pennies. The bot runs on the same shape regardless of how many reps are on call.

- **Always-on infrastructure for polling.** A naive setup might run a small server that polls Google Ads or Meta every minute. That's a $30+/month idle bill before it processes a single lead. EventBridge schedules trigger Lambda — you pay only when the schedule fires, and Lambda scales to zero between fires.
- **One large model on every read.** A frontier-class model would spend ten to twenty times more per lead for what is, at this scale, a short-input short-output extraction task. Haiku-class models are correctly sized. Reaching for a bigger one because it sounds smarter is the most common way to triple the bill on a system where the per-call output is a small structured object.

When this stops being cheap

The math changes if the campaigns scale up dramatically (a paid acquisition push, a product launch that surfaces in many directories at once) and lead traffic goes into the tens of thousands per month. At that point the tier-two number becomes the headline, and there are levers: cache policy embeddings (already cached after the first lookup), batch the extractors into a single multi-output call, or move the urgency extractor to a smaller dedicated model. Most small businesses, including small teams running paid ads, 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 lead, 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 3, 2026 PART 7 OF 7 · LEAD INTAKE BOT SERIES ~6 MIN READ

Engineering reference: the lead intake bot architecture

Same system as the rest of the series, drawn purely for engineers. Service names, resource identifiers, region, Bedrock model IDs, Knowledge Base wiring, Meta Lead Ads and Google Ads API specifics, SES inbound, CRM destinations, 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 (4 lanes → SQS), Qualifier (parallel extractors + scorer + composer), Dispatch & routing.
- 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).
- Lead sources: Meta Lead Ads webhook (Graph API v24/v25), Google Ads lead form asset webhook, Google Ads conversion-import poll, SES inbound for email.
- Day-one paperwork: Meta App with `leads_retrieval`, Google Ads developer token under an MCC, SES domain MX verification.

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.

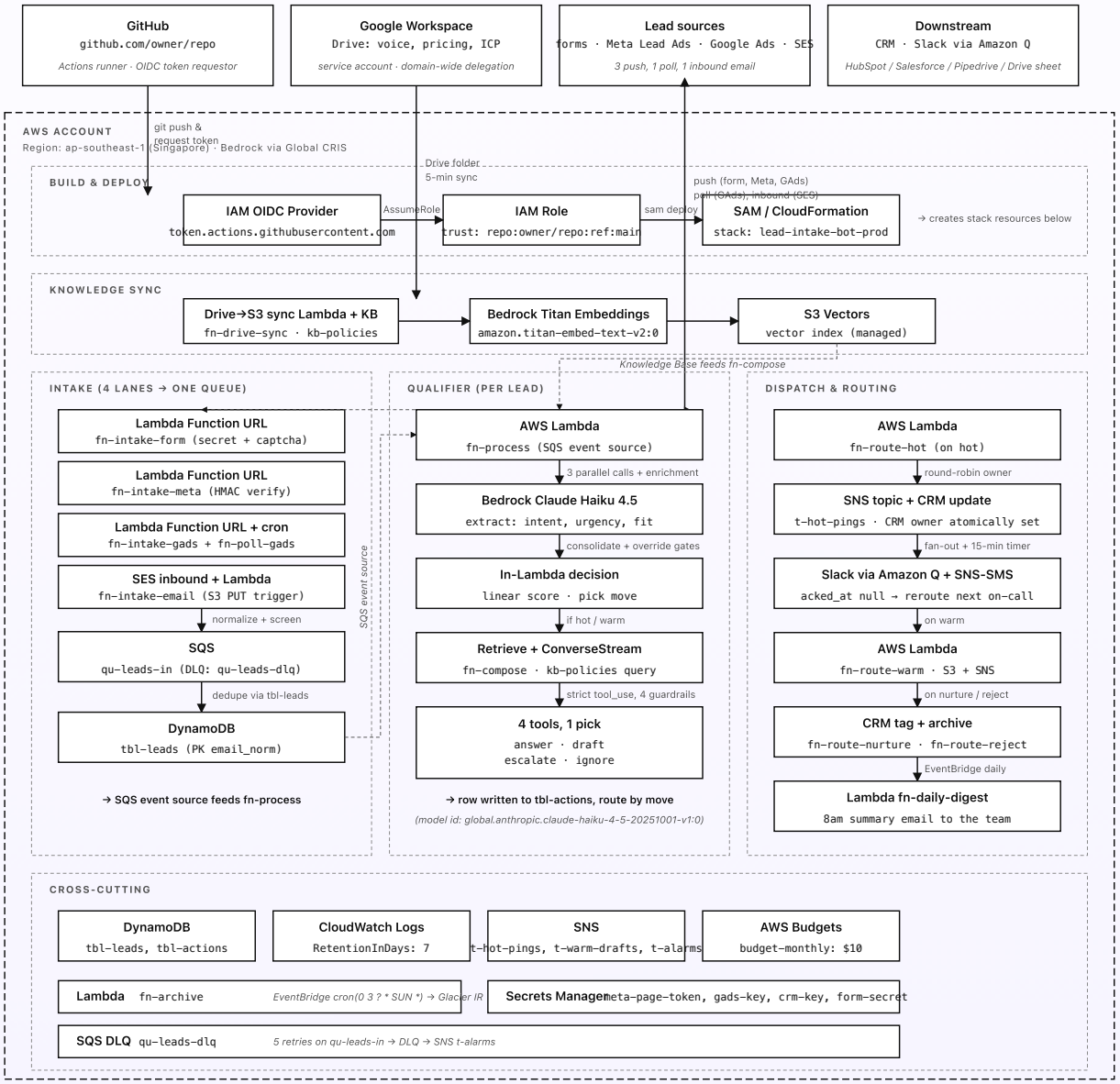

GitHub
github.com/owner/repo
Actions runner · OIDC token requestor
Google Workspace
Drive: voice, pricing, ICP
service account · domain-wide delegation
Lead sources
forms · Meta Lead Ads · Google Ads · SES
3 push, 1 poll, 1 inbound email
Downstream
CRM · Slack via Amazon Q
HubSpot / Salesforce / Pipedrive / Drive sheet
AWS ACCOUNT
Region: ap-southeast-1 (Singapore) · Bedrock via Global CRIS
git push & request token
Drive folder 5-min sync
push (form, Meta, GAds) pull (GAds), inbound (SES)
BUILD & DEPLOY
IAM OIDC Provider
token.actions.githubusercontent.com
AssumeRole
IAM Role
trust: repo:owner/repo:ref:main
sam deploy
SAM / CloudFormation
stack: lead-intake-bot-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 (4 LANES → ONE QUEUE)
Lambda Function URL
fn-intake-form (secret + captcha)
Lambda Function URL
fn-intake-meta (HMAC verify)
Lambda Function URL + cron
fn-intake-gads + fn-poll-gads
SES inbound + Lambda
fn-intake-email (S3 PUT trigger)
normalize + screen
SQS
qu-leads-in (DLQ: qu-leads-dlq)
dedupe via tbl-leads
DynamoDB
tbl-leads (PK email_norm)
→ SQS event source feeds fn-process
SQS event source
QUALIFIER (PER LEAD)
AWS Lambda
fn-process (SQS event source)
3 parallel calls + enrichment
Bedrock Claude Haiku 4.5
extract: intent, urgency, fit
consolidate + override gates
In-Lambda decision
linear score · pick move
if hot / warm
Retrieve + ConverseStream
fn-compose · kb-policies query
strict tool_use, 4 guardrails
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 & ROUTING
AWS Lambda
fn-route-hot (on hot)
round-robin owner
SNS topic + CRM update
t-hot-pings · CRM owner atomically set
fan-out + 15-min timer
Slack via Amazon Q + SNS-SMS
acked_at null → reroute next on-call
on warm
AWS Lambda
fn-route-warm · S3 + SNS
on nurture / reject
CRM tag + archive
fn-route-nurture · fn-route-reject
EventBridge daily
Lambda fn-daily-digest
8am summary email to the team
CROSS-CUTTING
DynamoDB
tbl-leads, tbl-actions
CloudWatch Logs
RetentionInDays: 7
SNS
t-hot-pings, t-warm-drafts, t-alarms
AWS Budgets
budget-monthly: $10
Lambda fn-archive
EventBridge cron(0 3 ? * SUN *) → Glacier IR
Secrets Manager
meta-page-token, gads-key, crm-key, form-secret
SQS DLQ qu-leads-dlq
5 retries on qu-leads-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 four external surfaces. Below it, the AWS account contains five subsystems: Build & Deploy across the top, then Knowledge Sync, then three runtime columns (Intake, Qualifier, Dispatch & routing), with a Cross-cutting strip at the bottom. Leads enter through four intake paths (three webhooks behind Lambda Function URLs plus an SES-inbound parser) and all four write into a single SQS queue `qu-leads-in` after deduplicating against `tbl-leads` on a normalized email key within a 24-hour window. The SQS event source invokes `fn-process`, which runs the three extractors in parallel against Bedrock Claude Haiku, runs the passive enrichment, applies the two override gates (partner-allowlist hot, reject), computes the linear fit score against the ICP doc, picks one of four moves, and on hot or warm calls `fn-compose`. The composer calls Bedrock `Retrieve` against `kb-policies` to fetch the top grounded chunks, then calls Bedrock `ConverseStream` with those chunks plus strict tool_use over four tools (`answer`, `draft`, `escalate`, `ignore`) — the streaming RAG endpoint `RetrieveAndGenerateStream` doesn't accept client-side tool definitions, so we pair the two APIs explicitly. The composer then runs the four guardrails (citation, no fabricated specifics, no commit on availability, no PII in subject) and writes the chosen action to `tbl-actions`. Dispatch routes by move: `fn-route-hot` picks an owner round-robin, sets the CRM owner field atomically, fans out via SNS to Slack and optional SMS, and arms a 15-minute escalation timer; `fn-route-warm`

packages the draft and fans out to email; `fn-route-nurture` tags in the CRM and enrolls in a campaign; `fn-route-reject` archives with a reason. `fn-daily-digest` emails the team summary at 8am.

Naming conventions used in the diagram

- **Lambda functions:** `fn-<purpose>` — `fn-intake-form`, `fn-intake-meta`, `fn-intake-gads`, `fn-poll-gads`, `fn-intake-email`, `fn-process`, `fn-compose`, `fn-route-hot`, `fn-route-warm`, `fn-route-nurture`, `fn-route-reject`, `fn-daily-digest`, `fn-drive-sync`, `fn-archive`.
- **Lambda runtimes:** Python 3.13 for the qualifier, composer, daily digest, drive sync, archive, and routing functions (the Bedrock SDK is more ergonomic in Python). Python 3.14 has been available on Lambda since November 2025; 3.13 is the safe production default in May 2026. Node.js 22.x is fine for `fn-intake-meta` and `fn-intake-gads` if you prefer JS for HMAC verification; Node.js 24.x is also available since 2025 and either is current.
- **DynamoDB tables:** `tbl-leads` (partition key `email_norm` — lowercased, plus-tag-stripped — sort key `seen_at`, with TTL of 30 days; used for dedupe and 24-hour-window resubmission collapse), `tbl-actions` (partition key `lead_id`, sort key `action_ts`, with `move`, `score`, `score_breakdown`, `owner`, `acked_at`, `reply_text`, `cited_passages`, `guardrail_flags`; queried by the escalation timer to detect unacked hot leads).
- **SQS queues:** `qu-leads-in` (standard queue with 5-minute visibility timeout), `qu-leads-dlq` (5 retries before failure goes to DLQ; CloudWatch alarm on DLQ depth > 0 fires `t-alarms`).

- **SNS topics:** `t-hot-pings` for urgent fan-out (Slack via Amazon Q Developer + optional SMS via SNS), `t-warm-drafts` for normal-priority human review fan-out (SES email + optional Slack), `t-alarms` for general failures.
- **S3 layout:** single bucket `lead-intake-bot-data` with prefixes `kb-source/` (Drive mirror), `inbound-mime/` (raw SES messages), `drafts/{date}/` (full warm draft packages), `archive/`.
- **Knowledge Base:** `kb-policies`, a Bedrock managed Knowledge Base with an **S3 connector** pointed at the synced policies 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 2, 2025 — cheapest quick-create option for small/medium KBs: no provisioned capacity, no monthly minimum, $0.06/GB-month for stored vectors plus tiered query charges and ~$2.50 per million API calls — 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, lead-source 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.

Website forms (lane 1). Your form posts JSON over HTTPS to `fn-intake-form`'s Function URL. Two checks before any real work: a per-form shared secret in the body (random string per origin, stored in Secrets Manager and embedded in the form via a server-rendered include), and a captcha token verified against the captcha provider's siteverify endpoint. **Cloudflare Turnstile** is fully free with no monthly cap. **hCaptcha**'s publisher tier is free up to 10K verifications per month. **Google reCAPTCHA Enterprise** on Google Cloud has a free tier of 10K assessments per month, with paid pricing above that. Classic free reCAPTCHA v2/v3 no longer accepts new keys — new sites must use reCAPTCHA Enterprise on Google Cloud, and existing v2/v3 keys created outside Google Cloud are being migrated through Q1 2026. **CORS** on the Function URL is restricted to your own domain(s). Preflight with `Access-Control-Allow-Origin: https://yourdomain.com`; no wildcards.

Meta Lead Ads (lane 2). Webhooks fire on the **Page** object with the `leadgen` field. You subscribe in Meta Business Manager and configure the callback URL plus a verify token. The webhook payload contains `leadgen_id` and `form_id` only, not the field answers. `fn-intake-meta` verifies the `X-Hub-Signature-256` header (App Secret as the HMAC key, computed over the raw request body, constant-time comparison), then makes a separate call to `GET /v25.0/{leadgen_id}?fields=field_data,form_id,created_time` with the page access token to retrieve the answers. Page tokens last ~60 days. A small refresh worker runs weekly to exchange the current token for a fresh long-lived one

before it expires. Pin to `v24.0` or `v25.0` on outgoing calls. `v18.0` sunset on 2026-01-26, `v19.0` sunsets on 2026-05-21, and `v20.0` sunsets on 2026-09-24. v25.0 is the current latest as of May 2026.

Google Ads (lane 3). Two patterns supported. The **lead form asset webhook** (the surface formerly called the Lead Form Extension before Google's Extensions-to-Assets rename) fires on submission to a URL you configure in the Google Ads campaign settings, with a `google_key` in the request body that you verify against the value you set for that lead form. For accounts using **conversion-import-only** lead capture (older campaigns, search ads with no lead form asset), `fn-poll-gads` runs on EventBridge cron `cron(0 * * * ? *)` hourly and queries the Google Ads API for new conversion events using a developer token plus an OAuth 2.0 refresh token (installed-app flow under your manager account). Service-account auth is supported only via Google Workspace domain-wide delegation, which is a narrow path most advertisers can't use; the OAuth refresh-token flow is the standard production setup. Note that as of April 21, 2026, Google Ads API enforces MFA on new OAuth refresh tokens (service accounts via DwD are exempt) — plan for re-auth as part of operations. The Google Ads API requires an approved developer token under a Manager Account (MCC) — the application takes a few business days; do this on day one.

SES inbound (lane 4). SES inbound rules accept email at your domain. Verify the domain's MX record points to SES inbound; some regions require Easy DKIM and SPF. Each rule has an action chain — for this system, write to S3 first, then trigger a Lambda. The Lambda receives an S3 PUT event with the raw MIME, parses it with the Python `email` module (or `mailparser` for richer extraction), strips signatures and quoted threads with the same heuristics the email-assistant

series uses, and writes the cleaned record to `qu-leads-in`. Inbound mail is also a frequent vector for auto-replies and list confirmations. The parser drops anything with `Auto-Submitted:` headers or known list-management subject patterns before queueing. SES inbound is not available in every region. In May 2026, `ap-southeast-1` supports inbound — but double-check before you commit.

CRM destinations. `fn-route-hot`, `fn-route-warm`, `fn-route-nurture`, and `fn-route-reject` each have a small CRM client that knows how to `upsert_contact`, `set_owner`, `add_tag`, and `archive`. A single CRM-adapter module switches on the configured CRM (`HUBSPOT`, `SALESFORCE`, `PIPEDRIVE`, or `DRIVE_SHEET` for tiny SMBs); each adapter handles the API specifics. The Drive Sheet adapter is the smallest fallback — `fn-route-*` functions append rows to a configured Google Sheet via the Sheets API. CRM API keys live in Secrets Manager.

Google Drive authentication uses a service account with **domain-wide delegation** over two scopes: `https://www.googleapis.com/auth/drive.readonly` on the policies folder, and `https://www.googleapis.com/auth/spreadsheets` if the Drive Sheet adapter is enabled. 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 `lead-intake-bot-data/kb-source/`, and lets the Bedrock KB's S3 connector index from there. Editing a doc and saving propagates within about 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 four guardrails (citation, no fabricated specifics, no commit on availability, no PII in subject) all 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 outbound permissions via Secrets Manager).
- Per-business policies layout — a flat Drive folder is fine for the first few months; subdivide by topic (`pricing/`, `integrations/`, `icp/`) 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 August 2025). The custom citation-verify, no-fabricated-specifics, no-commit-on-availability, and no-PII-in-subject 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 message and falls back to `escalate` 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-lead pipeline (extract → score → pick → compose → route) 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 send; for the synchronous draft package pattern shown here, in-Lambda is simpler and cheaper.
- **Backfill** — on day one the system is empty of historical leads. A one-shot backfill script can populate `tbl_leads` with existing email keys (so re-engagements get re-evaluated as "known contact" instead of cold leads) without triggering a flood of belated drafts. Off the diagram because it runs once.

IF YOU'RE RECREATING THIS

Day-one paperwork. If you're going to use Meta Lead Ads, register a Meta App in Business Manager and submit it for the `leads_retrieval` permission. Review takes a few business days. If you're going to use Google Ads, request a **Google Ads API developer token** on day one. It requires a Manager Account (MCC) and the application takes a few business days. SES inbound requires verifying your domain's MX record points to SES, which can take several hours to propagate.

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 ICP 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 `Retrieve` call returns a passage. Then add one intake lane. `fn-intake-form` is the easiest — you control the form, and you can iterate on the shared secret and captcha pattern without waiting on platform approvals. Then add the SQS-driven `fn-process` with the three extractors, the override gates, the linear scorer, and the four-move picker. Then `fn-compose` with strict tool_use and the four guardrails. (This is the part most worth integration-testing. Intentionally try to make the model cite a passage outside the retrieved set, fabricate a discount, or commit to a specific time. Confirm each guardrail downgrades the reply correctly.) Then `fn-route-warm` and `fn-route-hot`. Then add the Meta and Google

> Ads intake lanes once the offline path works end-to-end. Cross-cutting (audit, logs, alarms, budget, archive) goes in from day one.