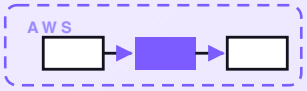
7-PART SERIES · FREE COMPANION



Onboarding guide

A serverless guide that walks every new customer through the right first steps automatically. When someone signs up, it starts a friendly sequence of timed messages and tasks — welcome, setup help, a check-in — watches whether each step gets done, nudges gently if someone stalls, and tells the owner who is stuck so a human can step in. It adapts the pace to each customer and never spams. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Onboarding guide

PART 1 OF 7

MAY 16, 2026 PART 1 OF 7 · ONBOARDING GUIDE SERIES ~5 MIN READ

An onboarding guide on AWS for a few dollars a month

A small business signs up more customers than anyone can hand-hold one at a time. The new user who tried the product once and never came back. The customer who got stuck on the very first setup step and quietly gave up. The trial that ran out before anyone ever showed them the feature they actually needed. Walking each new person through their first steps is the work that pays off most and slips first when the week gets busy. This post walks through the design of a small guide that greets every new customer, walks them through the right first steps, nudges gently when someone stalls, and tells the owner who is stuck — and never spams anyone.

KEY TAKEAWAYS

- Three sources for new customers: a signup webhook, a Drive sheet, and an inbox forwarding lane.
- Every customer ends in one of four moves on each tick: on track, next step due, gentle nudge, or flag to owner.
- Per-plan step plans: a starter plan gets welcome at day 0, setup at day 2, a check-in at day 6.
- Messages adapt to pace, respect quiet hours and weekends, and carry a one-click done link.
- Designed on AWS for about $2.10/month at typical small-business volume.

The whole system on one page

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

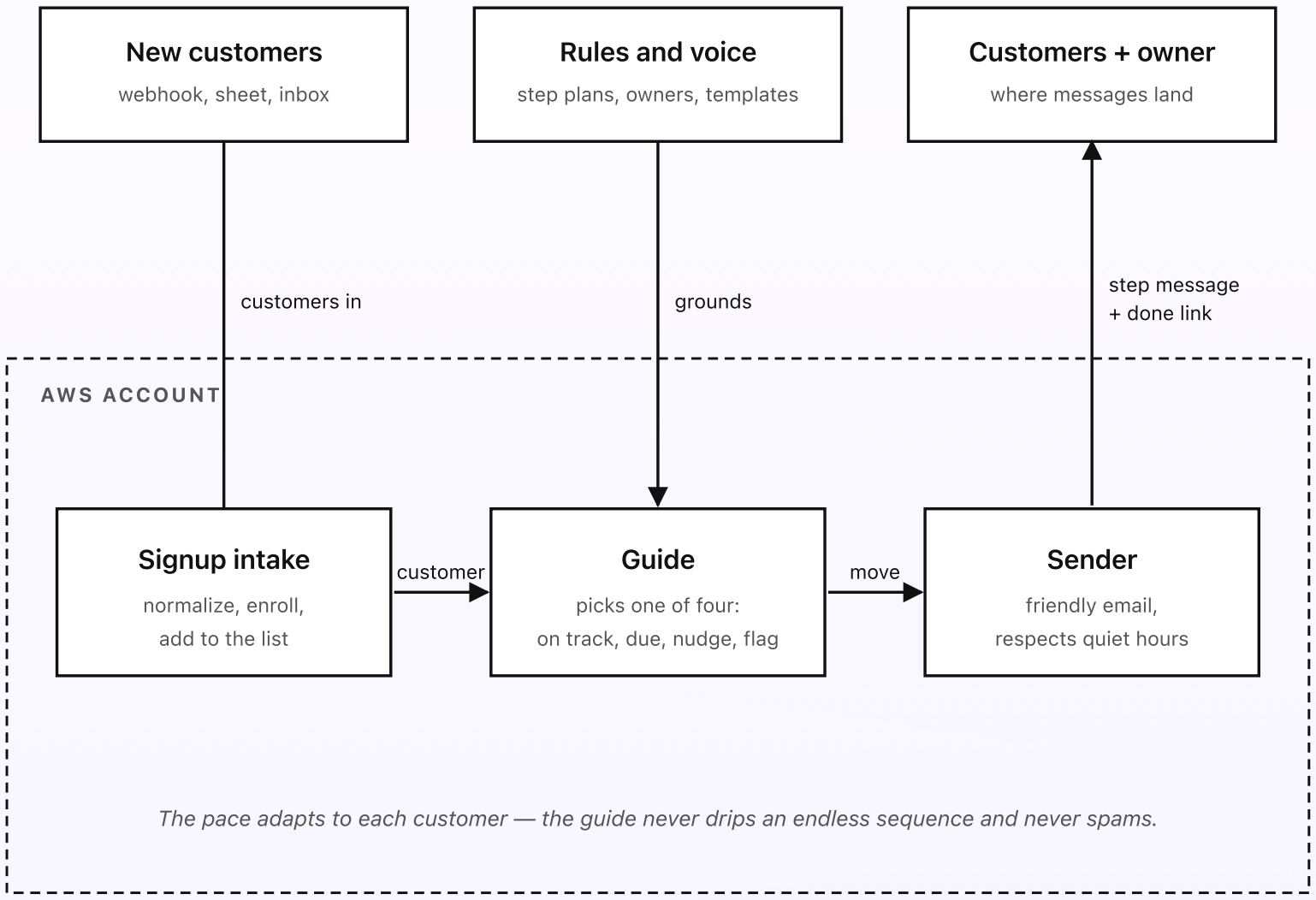

New customers
webhook, sheet, inbox
Rules and voice
step plans, owners, templates
Customers + owner
where messages land
customers in
grounds
step message
+ done link
AWS ACCOUNT
Signup intake
normalize, enroll,
add to the list
customer
Guide
picks one of four:
on track, due, nudge, flag
move
Sender
friendly email,
respects quiet hours
The pace adapts to each customer — the guide never drips an endless sequence and never spams.

Fig 1. Three sources outside, three pieces inside AWS. Customers flow in from a signup webhook, a Drive sheet, and an inbox forwarding lane. The Guide runs daily and picks one of four moves. The Sender emails the right step to the right person at the right time.

What you set up once (the outside)

- **New customers.** A Google Sheet in a Drive folder, one row per customer: name, email, plan (starter, pro, team), signup date, the current step, which steps are done, and a paused flag. New customers usually flow in straight from your app via two of the lanes covered in Part 2 — a signup webhook (your app calls a small endpoint the moment someone signs up) and an inbox-forwarding lane (forward the welcome email to a dedicated address and the guide proposes a row for one-tap approval). The sheet itself is the third lane, for the customers you add by hand.
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the step plan for each plan — which steps there are and how many days after signup each one should land. A starter plan typically gets welcome at day 0, setup at day 2, a check-in at day 6; a pro plan gets a longer plan with an extra feature tour. The doc also lists the owner who gets flagged when a customer is stuck, the quiet hours, and any holiday days to skip. The *voice* doc holds one message template per step — what the welcome, the setup help, the check-in, and the gentle nudge actually say.
- **Customers and owner.** The new customer receives the step messages and nudges. The internal owner receives a flag when a customer gets stuck. Messages land with the next step, a short how-to, a link into the product, and a

one-click “I’ve done this” link so the customer can mark a step done without logging a ticket.

What runs on every tick (the inside)

- **The signup intake.** Three sources feed the list. The signup webhook is the main one — your app posts a new customer to a small endpoint the moment they sign up, and the row is enrolled at once. The Drive sheet is the canonical store you can also edit by hand. And the inbox forwarding lane lets anyone forward a welcome email to `welcome@your-company.com`; the guide uses Bedrock Haiku 4.5 to read the name, email, and plan, then drops a one-tap approval card in Slack before the row is added.
- **The guide.** Runs once a day at 9am local. Reads the onboarding list. For each customer, computes days-since-signup and looks at which steps are already done. Compares against the per-plan step plan in the rules doc. Picks one of four moves. *On track*: nothing is due, or the customer is moving fast — do nothing. *Next step due*: a step’s day has arrived and it isn’t done — send the step message. *Gentle nudge*: a step passed its window with no progress — send one friendly reminder. *Flag to owner*: the final window passed and the step is still undone — tell the owner named in the rules doc so a human can reach out. The guide itself doesn’t call a model on the daily tick — the move logic is plain Python.
- **The sender.** Reads the voice doc, formats the message for the chosen step, and sends it. Email goes through SES outbound. It honors quiet hours (no sends between 6pm and 8am local by default) and skips weekends and holidays. Every send writes a row in DynamoDB so the next day’s tick can tell which steps already went out. A weekly digest summarizes who finished and who’s stuck. A

monthly summary writes a one-paragraph activation narrative: how many finished, where people drop off, the longest-stuck customers.

In plain words

Priya signs up for the starter plan on Monday. The same day, the guide emails her a warm welcome with the one first step: connect your data source. Here's how, and here's a button that says "I've done this." She connects it that afternoon and clicks done. On Wednesday (day 2) the guide sends the setup step: invite a teammate. She's busy and skips it. On Thursday the window passes, so the guide sends one gentle nudge that mentions the welcome. Still nothing by the weekend. On the following Monday the final window passes, so the guide flags Priya to the owner, Sam, with the full picture: starter plan, stuck on "invite a teammate," five days in. Sam emails her himself, learns she's a team of one, and pauses her sequence. Priya never gets nagged again about a step that doesn't apply to her.

The cost of running this is about $2.10 a month at SMB volume. The cost of *not* running it is the new customer who signs up, gets stuck on step one, and is gone before anyone notices — the most expensive kind of churn, because you already paid to win them.

DESIGN RULES THAT SHAPED EVERY DECISION

- Every message ships with full context — the next step, a short how-to, a link in, and a one-click done link. The customer never has to dig.
- Four moves, always. On track, next step due, gentle nudge, flag to owner. There is no fifth.
- The pace adapts. A customer moving fast skips ahead and gets fewer messages; a stuck one gets one nudge, then a human.
- Quiet hours, weekends, and holidays are respected. A message is a finite resource; bad timing burns it.
- A finished or paused customer stops getting messages. The guide never drips an endless sequence.
- Every send and every owner action is logged. Review an onboarding next quarter and you can see every message that went out.

Why this shape

Most teams onboard new customers in one of three ways: a person who reaches out when they have time, a generic drip sequence that emails everyone the same five things on the same five days, or nothing at all. The person works until they're busy — and the weeks they're busiest are exactly the weeks the most new customers are arriving. The generic drip is the worst kind of false comfort: it keeps emailing the customer who already finished, and keeps emailing the customer who's stuck the exact thing that didn't help the first time. And "nothing at all" is how a business quietly loses the customers it worked hardest to win.

The setup above keeps the customer list in a sheet the team already has, but adds a small system that *looks at* that list every day and acts only when a customer actually needs the next step or has stalled. Messages go out when they're useful, not on a fixed drip. The pace bends to the person: fast movers get out of the way, stuck movers get a hand. It escalates to a human cleanly when a nudge isn't enough. And it stops the moment someone finishes or the owner pauses them. The guide is invisible on the days a customer is moving along fine; it only shows up when there's a next step to take or someone is stuck.

The next four posts walk through each piece in turn: how a new customer gets enrolled, how an onboarding step comes due, how a nudge reaches a customer, and how an onboarding finishes. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 16, 2026 PART 2 OF 7 · ONBOARDING GUIDE SERIES ~4 MIN READ

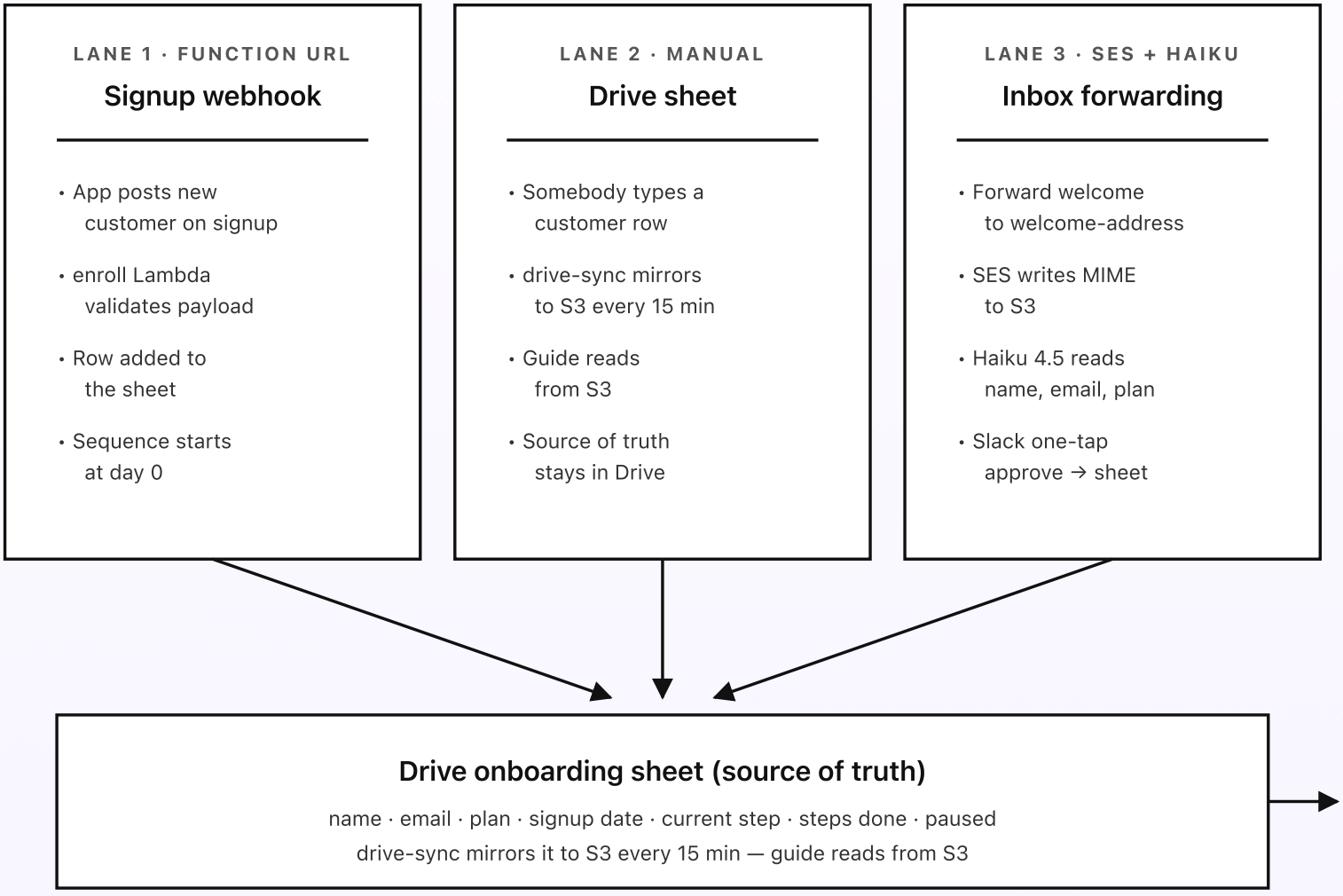
How a new customer gets enrolled

The guide only guides the customers who are on the list. So the first job is making sure a new customer lands on the list the moment they sign up — without anyone having to remember to add them. There are three ways a customer gets in: your app calls a webhook when someone signs up, somebody types a row in the Drive sheet, or somebody forwards the welcome email to a dedicated address. The first one carries almost everyone. The other two exist for the customers your app can't tell the guide about on its own.

KEY TAKEAWAYS

- Three enrollment lanes feed one list: a signup webhook, the Drive sheet, and an inbox-forwarding lane.
- The webhook is a Lambda Function URL — your app posts a new customer and the row is enrolled at once.
- Forwarded welcome emails are read by Bedrock Haiku 4.5, which proposes a row.
- Every webhook and parsed row is checked before the sequence starts at day 0.
- The Drive sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one list



The Drive sheet stays the source of truth — the webhook and inbox lanes propose rows for it.

Fig 2. Three lanes converge on one Drive sheet. The sheet is the source of truth; the signup webhook and the inbox lane write into it. The drive-sync Lambda mirrors the sheet to S3 so the guide can read it without hitting Drive on every tick.

Lane 1: the signup webhook (the lane that carries everyone)

The main lane. Your app already knows the instant someone signs up — that's the moment to enroll them. Set up a Lambda Function URL (a plain HTTPS endpoint on a Lambda, no API Gateway needed) and have your app POST a small JSON body to it whenever a new customer is created: name, email, plan, and a signup timestamp. The `enroll` Lambda validates the body (checks the email looks real, the plan is one you know, the signup date isn't in the future), normalizes it into the sheet's column shape, and writes the row via the Sheets API. The customer is on the list within a second of signing up, and the sequence starts at day 0 on the next tick.

The endpoint is protected with a shared secret your app sends in a header, checked against a value in Secrets Manager — so a random POST from the internet can't enroll a fake customer. This is the lane that should carry almost every real signup; the other two exist for the edges.

Lane 2: the Drive sheet itself

The simplest lane. Open the onboarding sheet in Drive, add a row, save. The columns are short: name, email, plan, signup date, the current step, which steps are done, and a paused flag. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://og-list-source/onboarding.csv` if the sheet has changed since the last

sync. The guide reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so a bad bulk-edit can be rolled back in one click.

This lane covers the customers your app can't webhook for you — the enterprise account your sales team set up by hand, the customer you migrated from an old system, the one who signed a contract over email. Type the row and the guide picks them up on the next tick.

Lane 3: inbox forwarding

Some signups arrive as an email, not an app event. A reseller sends over a new account. A customer replies to your sales thread to confirm they want to start. The welcome message your billing tool sends lands in a shared inbox. Forcing someone to retype those into a sheet is a small chore that gets skipped exactly when things are busy.

Set up a dedicated inbound address — something like `welcome@your-company.com` — via Amazon SES. Anyone forwards the welcome email there. SES writes the raw MIME to `s3://og-raw-mime/`. The S3 PUT triggers a parser Lambda that calls Bedrock Haiku 4.5 to read the email and emit a structured row: name, email, plan (if it can tell), and a signup date. The prompt is short: "Extract a customer row. Return JSON only. Mark each field with a confidence score. Do not invent a plan that isn't in the text." The proposal goes to a small Slack message that pings the rep who forwarded it — the proposed row, the confidence per field, and three buttons: *approve*, *edit*, *discard*. On *approve*, the row is written to the sheet via the Sheets API.

The reason every parsed row goes to a human first is simple: a customer enrolled on the wrong plan gets the wrong step plan, and a customer enrolled with a typo in their email never gets a single message. Both are worse than a row that needed ten seconds of human confirmation.

Why the list stays the source of truth

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

Next post: how the guide actually reads the list, computes days-since-signup, looks at which steps are done, and picks one of four moves.

PART 3 OF 7

MAY 16, 2026 PART 3 OF 7 · ONBOARDING GUIDE SERIES ~5 MIN READ

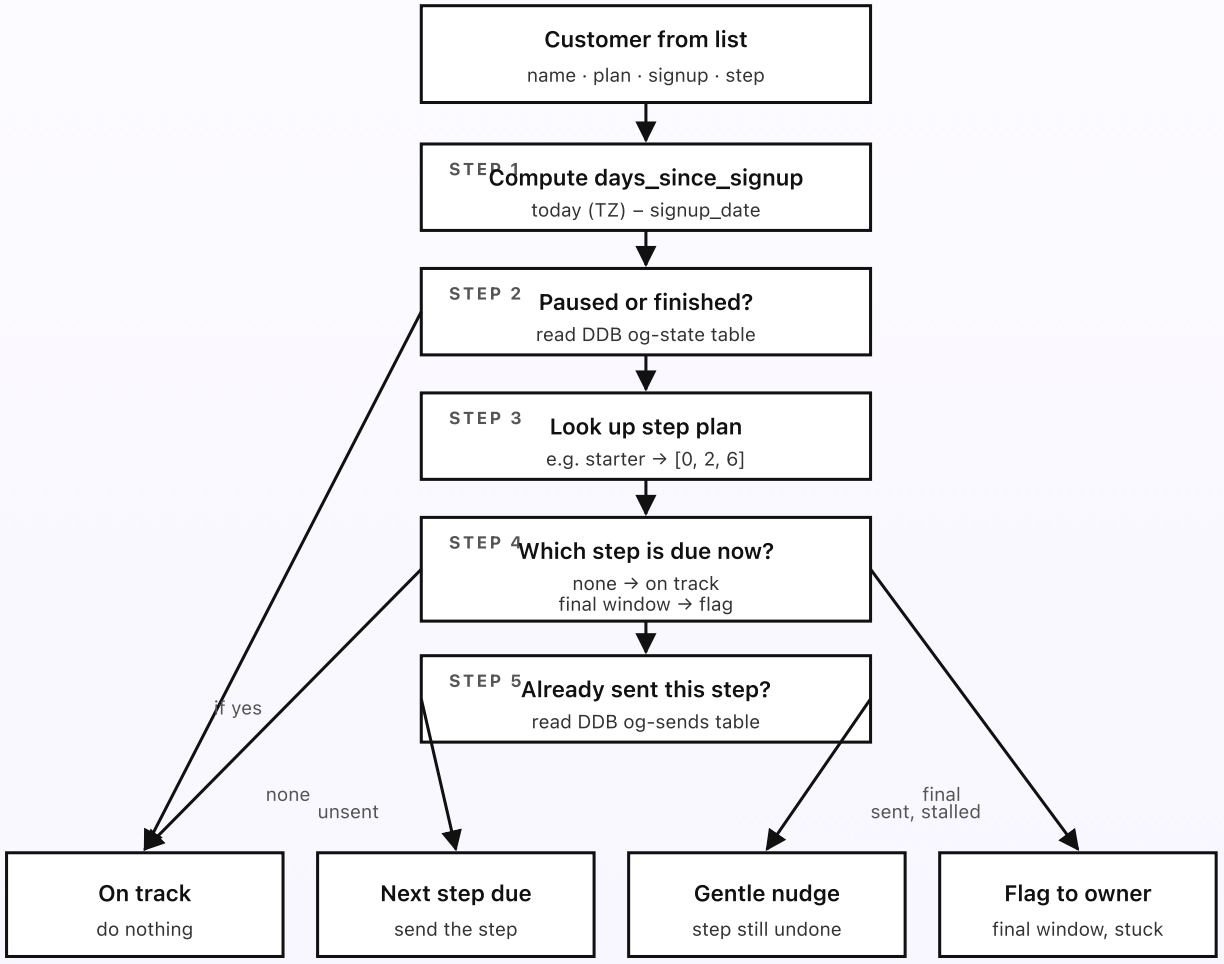
How an onboarding step comes due

Once a day, at 9am local time, an EventBridge Scheduler rule fires the guide Lambda. The Lambda reads the onboarding list, looks at one customer at a time, computes how many days it's been since they signed up, checks which steps they've already finished, and decides whether to do nothing or to send something — and if so, which kind. The whole decision is plain Python. No model. No vector retrieval. Every step and every window lives in the rules doc, where a rep can edit it without a deploy.

KEY TAKEAWAYS

- The guide runs once a day via EventBridge Scheduler at 9am local time.
- Per-plan step plans live in the rules doc — starter gets welcome at day 0, setup at day 2, check-in at day 6.
- Four moves per customer, every tick: on track, next step due, gentle nudge, flag to owner.
- DynamoDB tracks which steps were sent and which are done so messages aren't duplicate spam.
- The guide itself never calls a model. The decision is entirely deterministic.

The decision flow, per customer



The rules doc holds every step — change a window and tomorrow's tick uses the new value.

Fig 3. The guide's decision tree, per customer, per daily tick. Five steps decide which of four moves applies. The rules doc holds every step and window; the guide only enforces them.

Step plans: day 0/2/6 isn't magic, it's in the doc

The rules doc has one short section per plan. Each section names the steps in plain prose: "Starter plan: welcome on day 0, setup on day 2, check-in on day 6. Pro plan: welcome on day 0, connect data on day 1, feature tour on day 3, invite team on day 5, check-in on day 9." The numbers are days after signup when each step's message should land. Each step also has a window — how many extra days to wait, with no progress, before sending one gentle nudge, and then how many more before flagging the owner.

The plans exist for a reason. A starter customer wants to be useful fast and out of your inbox; three light steps over a week is plenty. A pro customer has more to set up, so the plan spaces things out and earns the right to send more. The shape of the plan reflects what the customer actually needs to do, not a fixed marketing drip.

Per-customer overrides exist too. The onboarding sheet has an optional column called `plan_override`. Name a different plan there and the guide uses that plan's steps for that one customer. This is the right escape hatch for the enterprise account that needs the long plan even though they bought the starter tier.

Four moves, always

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

- **On track.** Nothing is due, the customer is moving fast and finishing steps ahead of schedule, or they're paused or finished. Do nothing. Most customers, most days, are on track.
- **Next step due.** A step's day has arrived and the customer hasn't done it yet, and the message hasn't been sent. Send the step message with full context. Write a row to the `og-sends` DynamoDB table marking that this step has been sent.
- **Gentle nudge.** A step was sent, its window passed, and the customer still hasn't finished it. Send one friendly follow-up that names the original message so the customer doesn't feel like they're seeing it for the first time. Write the nudge to `og-sends`. There is exactly one nudge per step — never a chain of them.
- **Flag to owner.** The final window for a step passed and it's still undone. Don't message the customer again. Instead, tell the owner named in the rules doc — usually whoever handles onboarding — so a human can reach out. Mark the customer as flagged in DynamoDB; the guide won't flag the same step twice. Spamming a stuck customer with more automated mail is exactly the wrong move; a real person is the right one.

State that makes the decision deterministic

The guide reads two DynamoDB tables every tick. `og-sends` records every message that's gone out: `(customer_id, step_id, kind, sent_date)` where

kind is step or nudge. `og-state` records each customer's progress: `(customer_id, steps_done, paused, finished, flagged)`. With those two tables, the move-decision logic is a few dozen lines of Python and zero magic. A given customer with a given signup date, a given plan, and a given send/done history always produces the same move. Re-running the tick produces no extra messages (because the state in DDB shows what already fired).

When a customer marks a step done — via the one-click link in Part 4 — their row in `og-state` gets the step added to `steps_done`. The next tick sees it's done and moves on to the next step, or marks them finished if it was the last one.

Why the daily tick uses no model

The guide could call a model on the tick to write a smarter message, or to guess whether a customer is the type who needs more help. It doesn't. Two reasons. First, the daily tick should be the one part of the system that is utterly predictable — if the rules doc says the setup step lands on day 2 and it isn't done, the message goes out. A model in that loop introduces variance the team can't reason about. Second, model calls cost money, and most days most customers are on track, so the call would be wasted nine times out of ten.

Bedrock fires elsewhere — on the occasional personalized rewrite of a step message and on the monthly summary mentioned in Part 6. Not on the daily tick. The guide itself is plain Python that reads a doc and writes events.

Next post: how a message finds the right customer, how quiet hours and weekends are honored, and what the one-click done link actually does.

PART 4 OF 7

MAY 16, 2026 PART 4 OF 7 · ONBOARDING GUIDE SERIES ~5 MIN READ

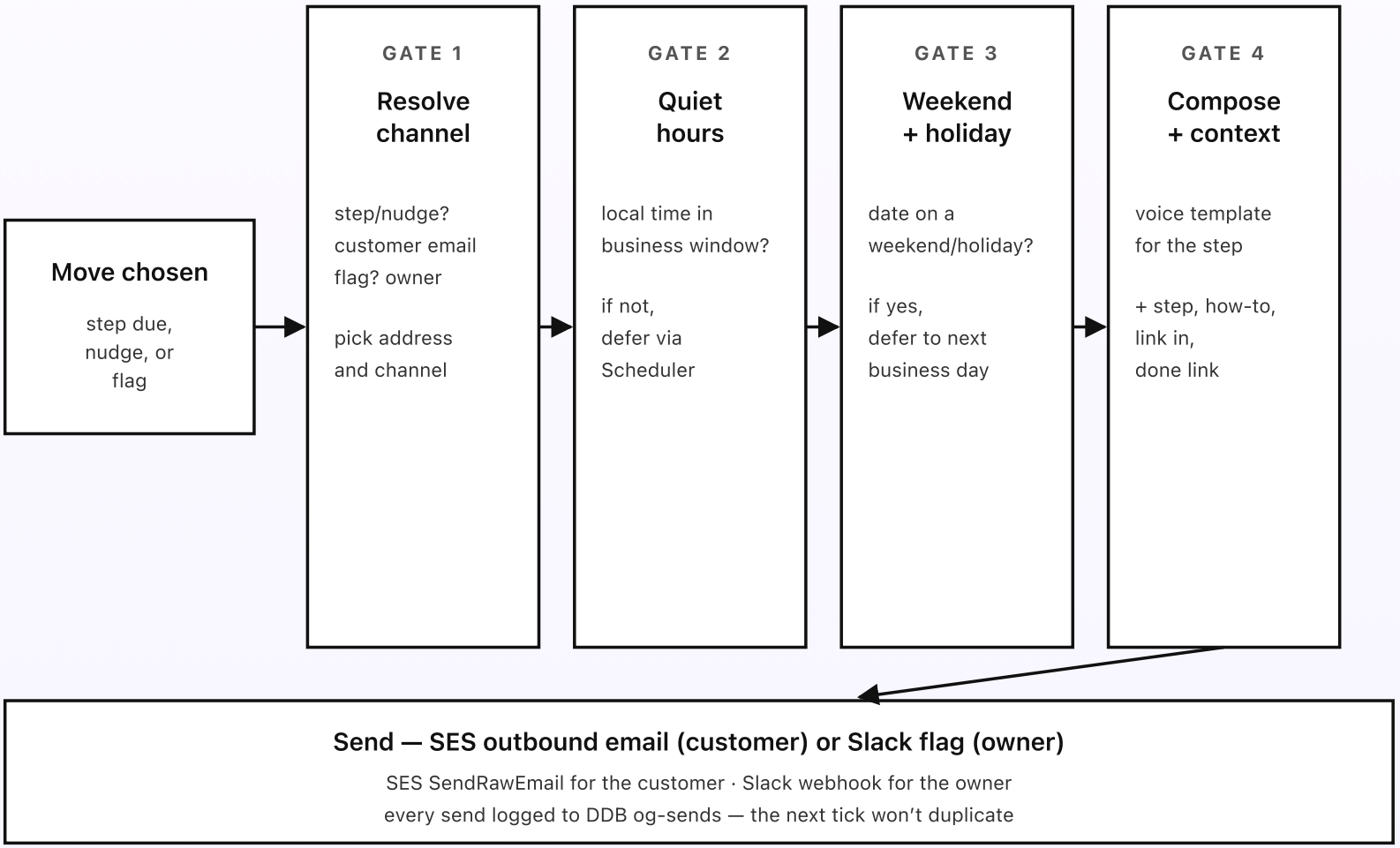
How a nudge reaches a customer

The guide picked a move — next step due, gentle nudge, or flag to owner. Now the sender Lambda has to figure out who to send it to, on what channel, at what time of day, and with what context attached. Get any of those wrong and the message is worse than no message: a 2am email, a generic “keep going,” a nudge to someone who already finished. Four small guardrails sit between the move and the message actually landing.

KEY TAKEAWAYS

- Channel resolution: a flag goes to the owner in Slack; a step or nudge goes to the customer by email.
- Email is the default for customers; the owner’s flag uses Slack.
- Quiet hours and weekends defer messages to the next available business hour.
- Every message ships with the next step, a short how-to, a link in, and a one-click done link.
- A flag goes to the owner instead of the customer; the customer is not messaged again automatically.

Four guardrails on every message



Every gate is a deterministic check — no model calls, no surprise message on a Sunday at midnight.

Fig 4. Four guardrails between the move and the sent message. Resolve the channel. Honor quiet hours. Skip weekends and holidays. Compose with full context. Then send via email or Slack and log it so the next tick doesn't duplicate.

Gate 1: resolve the channel

The first question is who this message is even for. A *next step due* or *gentle nudge* move is for the customer — it goes to their email address from the onboarding sheet. A *flag to owner* move is not for the customer at all; it goes to the onboarding owner named in the rules doc, delivered in Slack so it lands where the team already works. The sender resolves the right recipient and the right channel before it does anything else.

For customer messages, email is the right surface: it's where a welcome and a how-to belong, it survives if the person isn't logged in, and the one-click done link works from any inbox. For owner flags, Slack is the right surface: it's a work-context ping the team will actually see, and it can carry buttons to pause or hand off, which Part 5 covers. The split keeps customer-facing messages calm and team-facing messages actionable.

Gate 2: quiet hours

The guide itself runs at 9am local time, so the first time a move fires it's already in business hours. But deferred sends — a message held from a weekend, or a one-off computed send — can come due later in the day or outside the configured window.

Gate 2 reads the rules doc's quiet-hours setting (default 6pm to 8am, configurable per business). If the current local time is in the quiet window, the sender creates a one-off EventBridge Scheduler rule that fires at the next business-hour minute and exits without sending. The Scheduler invokes the same sender Lambda with the same payload at the deferred time, where Gate 2 will let it through. A welcome email landing at 2am reads as spam; the same email at 9am reads as care.

Gate 3: weekend and holiday

The rules doc lists the holidays you observe — either a static list ("Christmas Day, New Year's Day, Independence Day…") or a reference to a Google Calendar that holds them — and a setting for whether weekends count as business days for onboarding messages (off by default; most B2B onboarding shouldn't email on a Saturday). Gate 3 checks the current local date and, if it's a weekend or a configured holiday, defers the send to the next business day.

The list is explicit on purpose — the guide won't auto-detect a country's public holidays for you. A holiday you forgot to add sends a message that lands on a closed laptop; a holiday in the list that's no longer observed just delays a message by one business day, which is fine. The trade-off favors keeping the list explicit.

Gate 4: compose with full context, then ship

The voice doc has one email template per step: a short message with placeholders for the customer name, the step name, a one-line how-to, and a link into the product where the step actually happens. The sender fills the

placeholders, attaches a one-click “I’ve done this” link, and sends the email via SES outbound. The done link is a Function URL that, when clicked, records the step as finished for that customer — no login, no support ticket, one tap. The signed token in the link ties it to the exact customer and step so it can’t be reused or guessed.

For an owner flag, the same context is wrapped in a Slack message instead: the customer name, plan, the step they’re stuck on, how many days it’s been, and buttons to pause or hand off. The owner gets the whole picture without opening the sheet.

A gentle nudge is composed from a softer template than the first step message — it references the original (“a couple of days ago I sent over how to connect your data”) so it reads as a friendly follow-up, not a fresh demand. There is only ever one nudge per step; after that the move becomes a flag, and the customer is left alone.

Every send — customer email or owner flag — writes a row to `og-sends` in DynamoDB. The next day’s tick reads that row and knows not to send the same step again.

Why the guardrails exist

None of these gates are exotic. They’re the kind of small care a thoughtful human would take if they were sending the messages themselves — check who this is actually for, don’t email at 11pm, skip the weekend, include enough context that the customer can act without asking a follow-up question. Putting them in code as

four small sequential gates makes them part of the design, not a feature you're trusting the writer of any one message to remember.

Next post: how an onboarding finishes once a customer has worked through their steps — how the guide sends a wrap-up, how the owner can pause a sequence, and how a hand-off to a human works.

PART 5 OF 7

MAY 16, 2026 PART 5 OF 7 · ONBOARDING GUIDE SERIES ~5 MIN READ

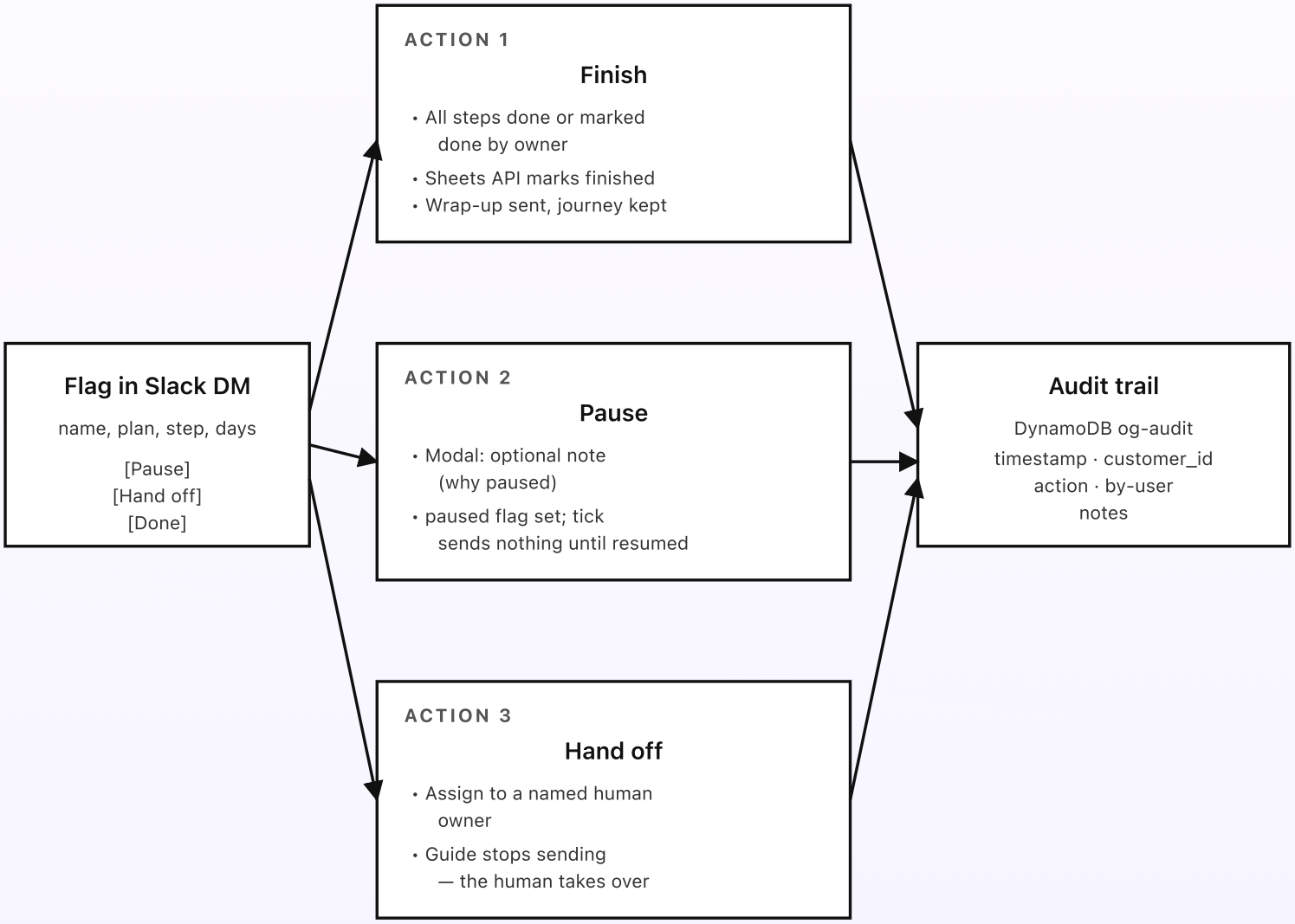
How an onboarding finishes

A flag lands in Sam's Slack at 9:04am. Priya is stuck on the "invite a teammate" step, five days in. There are buttons on the flag. What happens when he taps one? The honest answer is "it depends on what's actually going on." This post walks through the three ways a sequence ends — finish, pause, hand off — and how the onboarding list, the step state, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three ways a sequence ends: *finish* (all steps done, send a wrap-up), *pause* (hold without dropping), *hand off* (a human takes over).
- Each action updates the onboarding sheet via the Sheets API and writes an audit row.
- A finish archives the journey to a separate sheet for history.
- Pause is open-ended but visible — paused customers show up in the weekly digest so none are forgotten.
- The flag buttons are a Slack interactive message backed by a Function URL.

Three ways a sequence ends



Pause doesn't drop the customer — it holds the sequence. A paused customer can always be resumed.

Fig 5. Three ways a sequence ends, three different effects. Finish marks the customer done and sends a wrap-up. Pause holds without dropping. Hand off lets a human take over. Every action writes to the audit trail.

Action 1: finish (the goal)

The happy path. Priya works through every step in her plan — she connects her data, invites a teammate, books the check-in — clicking the one-click done link on each as she goes. When the daily tick sees that the last step in her plan is done, it lands her at *finish*. Or, if a human knows she's set even though a step is technically unmarked, the owner can tap *Done* on the flag to finish her manually.

Either way, the finish runs through a Function URL Lambda. Three things happen, in order. First, the Sheets API updates her row in the onboarding sheet: the `finished` flag is set and the `finished_date` is stamped with today and the user who acted. Second, her step rows in `og-sends` are copied to `og-sends-archive` with a journey id, and the live state is cleared. Third, a `action: finished` row is written to `og-audit` with the user, timestamp, and how long the whole onboarding took. One warm wrap-up email goes out — "you're all set, here's where to go next" — and then the guide goes quiet for her.

Tomorrow's tick reads the list, sees she's finished, and lands at *on track* with nothing to do. She won't hear from the guide again unless someone re-enrolls her for a new plan.

Action 2: pause (the hold)

Some onboardings need to stop without the customer being done. The customer asked to be left alone for a few weeks while they finish a migration. They're a team of one and the "invite a teammate" step will never apply. A deal is being renegotiated and nobody wants automated mail going out in the middle of it. Sam isn't finishing Priya and isn't handing her off — he just needs the guide to be quiet for now.

Pause opens a small modal with an optional note ("team of one, revisit after Q3"). On save, the `paused` flag is set on her row and a row is written to `og-state`. The next day's tick reads that flag in the "paused or finished?" check from Part 3 and treats her as on track — sends nothing. Pause is open-ended; there's no timer. But it's not a black hole: every paused customer appears in the weekly digest with their note and how long they've been paused, so nobody is quietly forgotten. Resuming is one click in the sheet (clear the flag) or a button in the digest, and the guide picks the sequence back up where it left off.

Action 3: hand off (the human)

Sometimes another automated email is exactly the wrong answer and a real person is the right one. The customer is clearly confused and needs a call. They're a big account that deserves a dedicated contact. They replied to a step email with a question the guide can't answer. Whatever the reason, it's time for a human to take the wheel.

Hand off assigns the customer to a named human owner — picked from a short list in the modal — and writes a row to `og-state` with `(customer_id, handed_off_to)`. The guide stops sending automated messages for that

customer entirely; from here the human owns the relationship. The assigned person gets a Slack ping with the full journey so far so they're not starting cold. If the human later decides the guide should resume (the call went well, the customer just needs the rest of the sequence), they can clear the hand-off and the guide picks back up.

Hand off is the escape hatch that keeps the whole system honest: the guide is for the common path, and the moment a customer needs more than the common path, a person steps in cleanly instead of the customer getting more of the same.

Every action is logged, every action is reversible

The `og-audit` table records every finish, pause, and hand-off with the user who took the action, the timestamp, and a snapshot of the row before and after. If a customer gets paused by mistake, or finished a week early, a rep can run an "undo last action" through a small admin command that reads the previous-state snapshot and restores the row. The undo is itself an audit row, so the trail of edits stays clean.

This reversibility matters because onboarding is the first impression. The next time anyone looks at how a customer's first week went — a support rep picking up a ticket, an account manager prepping a renewal — the audit trail is the memory of exactly what the customer saw and when.

Next post: the cost breakdown. The whole pipeline above runs in coffee-money territory at SMB volume; Part 6 explains exactly where the dollars go and why it's so cheap.

PART 6 OF 7

MAY 16, 2026 PART 6 OF 7 · ONBOARDING GUIDE SERIES ~3 MIN READ

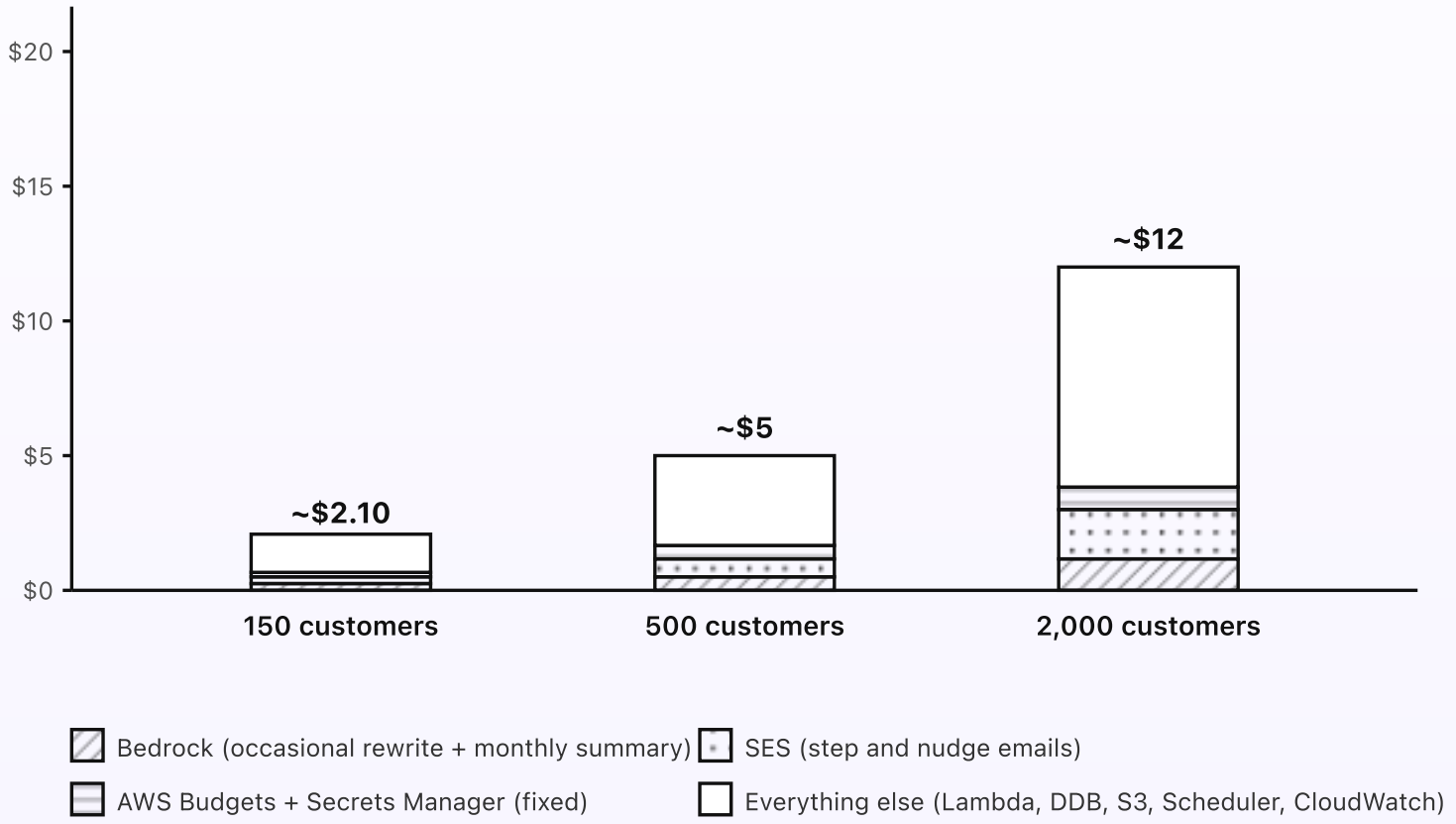
What the onboarding guide costs

The guide is one of the cheapest systems in this whole series. The daily tick reads a CSV from S3, does some date arithmetic, writes a few rows to DynamoDB, and sends a handful of emails. It calls no models on the tick. Bedrock fires only when a step message needs a small personalized rewrite and once a month for the owner summary. At typical SMB volume, the bill is a couple of dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2.10/month at typical SMB volume (around 150 customers onboarding at once).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily tick costs pennies — no model calls.
- Bedrock fires only on the occasional rewrite (a few a month) and the monthly summary.
- At 500 active onboardings the bill is around $5. At 2,000 it’s around $12.

Cost at three volumes



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

Fig 6. Monthly cost at three onboarding volumes. Bedrock is a small sliver because it only fires on the occasional rewrite and the monthly summary. The dominant cost is the everything-else bucket: the daily tick reading every active onboarding.

Where the dollars actually go

Lambda runtime (the bulk). The guide runs once a day. Each tick reads the onboarding CSV from S3, iterates the rows, computes `days_since_signup` for each, checks step state, and decides on a move. At 150 customers, that's a few hundred milliseconds. At 2,000 it's a couple of seconds. Either way it's pennies a month. Add the sender Lambda firing for each message, the Function URL Lambda for the signup webhook and the done links, and the drive-sync Lambda every fifteen minutes — the Lambda total still lands under a dollar at all three volumes.

DynamoDB on-demand. Three small tables: `og-sends`, `og-state`, `og-audit`. Reads are dominant during the daily tick (a read or two per customer per tick). Writes are send events and audit rows. Pennies a month at any of these volumes.

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

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

SES. Outbound for the step and nudge emails: $0.10 per thousand sent. A starter customer gets three or four emails over their onboarding; even at 2,000 active customers that's a few thousand emails a month, so a few cents. Inbound for the forwarding lane is negligible. This is the one slice that grows with how many customers you onboard, and it's still tiny.

Bedrock (only when something fires it). The daily tick uses no Bedrock. A personalized rewrite fires Haiku 4.5 only when the rules doc asks for it on a particular step — a few hundred input and output tokens, a fraction of a cent per rewrite. At a handful a month, Bedrock costs cents. The monthly summary is one larger call: write a paragraph that summarizes the month's sign-ups, finishes, and drop-offs; a couple of cents.

The fixed services. AWS Budgets and Secrets Manager together cost well under a dollar a month and don't move with volume. They're the flat floor under the whole bill.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the signup webhook and the done-link endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The guide sleeps 23.99 hours a day.
- **A Knowledge Base.** The onboarding list is structured rows, not free text — deterministic lookup beats vector search here. No embeddings, no Knowledge Base, no S3 Vectors needed.
- **Models on the tick.** The daily decision is plain Python. Bedrock fires only on the occasional rewrite and the monthly summary.

How the cost scales

Lambda runtime grows roughly linearly with customer count, because every active onboarding is evaluated on every tick. DynamoDB grows linearly too. SES grows with the number of messages sent, which tracks customer count but stays cheap. Bedrock is uncorrelated with count — it only fires on a flagged rewrite or the first of the month. So the bill at 5,000 active onboardings is around $28; at 10,000 it's around $55. Past those volumes the daily-tick model probably stops being right (you'd switch to a partial-tick that only evaluates customers with a step coming due), but those are optimizations for very large lists — not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. The guide's normal-volume bill stays well under that ceiling.

Last post in the series: the engineering reference. Same system, drawn for engineers — service names, Lambda inventory, IAM scopes, DynamoDB schemas, SES rule set, and EventBridge Scheduler config.

PART 7 OF 7

MAY 16, 2026 PART 7 OF 7 · ONBOARDING GUIDE SERIES ~8 MIN READ

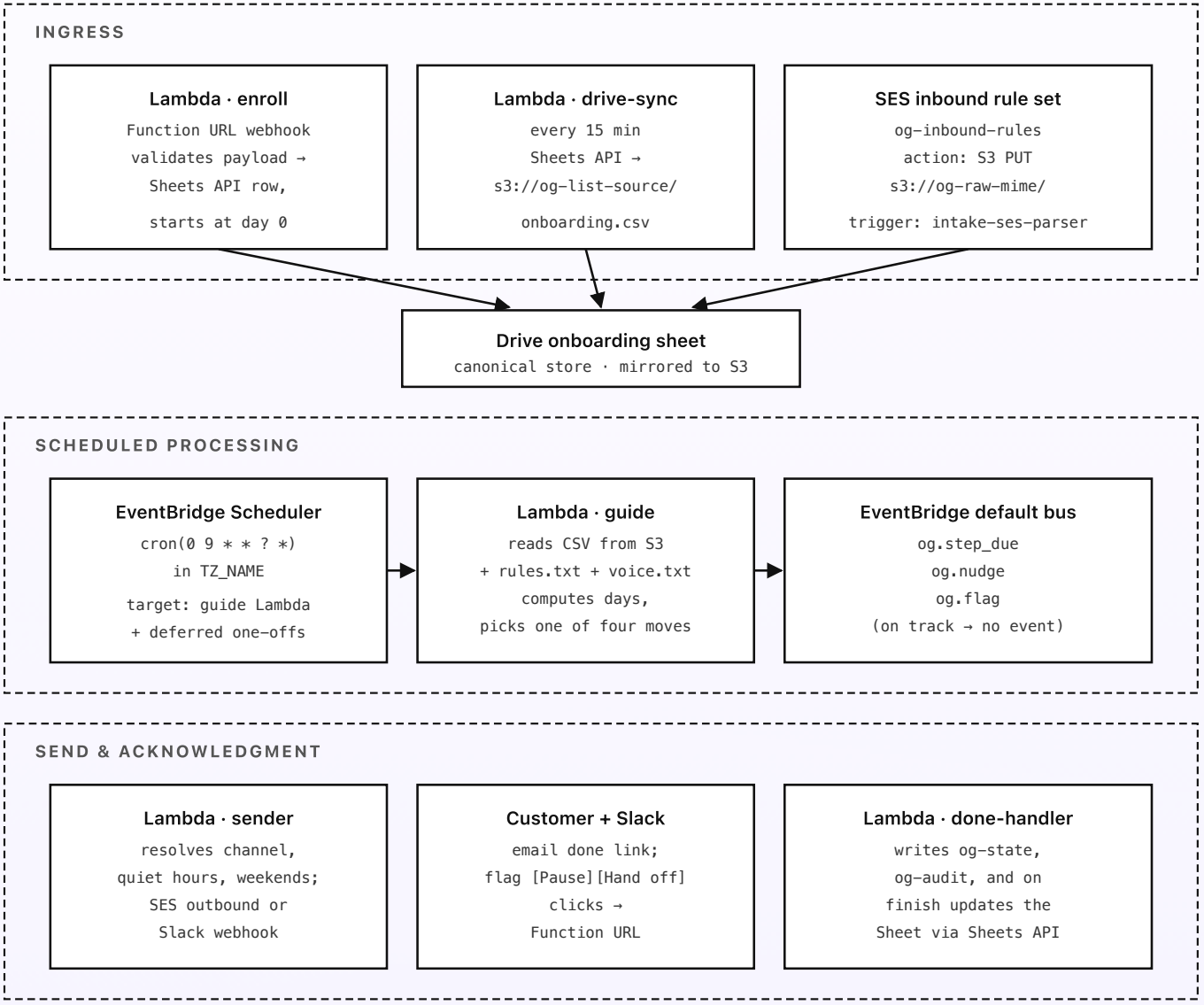
Engineering reference: the onboarding guide architecture

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

Region and account shape

Default region: **ap-southeast-1** (Singapore). SES inbound, Bedrock Global cross-Region inference, and EventBridge Scheduler are all in good shape there. A second region for multi-region resilience isn't worth the extra setup work at SMB volume — the failure mode for an SMB is a new customer missing a welcome email, not a regional outage. One AWS account dedicated to the guide (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Every message leaves with full context — and every interaction is logged to og-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the list), scheduled processing (the daily guide tick emitting events), send and acknowledgment (the message ships and the customer's or owner's response is recorded). Every Lambda is event- or schedule-driven; nothing is synchronous-chained.

Lambda functions

All Lambdas use the `arm64` architecture, the smallest memory size that meets latency targets (typically 256 MB), Python 3.14 runtime, and CloudWatch Logs at 7-day retention. Each function has its own least-privilege IAM role. None run inside a VPC.

- **`enroll`** — Lambda Function URL, `AuthType: NONE`; verifies a shared-secret header against `og/webhook/secret` in Secrets Manager. Triggered by your app on every signup. Validates the JSON body (email format, known plan, signup timestamp not in the future), normalizes it, and writes the row to the Drive sheet via the Sheets API. Idempotent on a client-supplied `signup_id` so a retried POST doesn't enroll twice. Memory: 256 MB. Timeout: 15 s.
- **`drive-sync`** — EventBridge Scheduler target, fires every 15 minutes. Uses the Google Drive API + Sheets API (service-account credentials in Secrets Manager under `og/drive/sa`) to export the onboarding sheet as CSV and write to `s3://og-list-source/onboarding.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://og-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`intake-ses-parser`** — S3 PUT trigger on `s3://og-raw-mime/`. Parses MIME, extracts the text body of the forwarded welcome email, and calls Bedrock Haiku

4.5 (`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`) to propose a customer row (name, email, plan, signup date). Posts the proposal to Slack via the incoming webhook with Approve/Edit/Discard buttons. No Textract is needed — welcome emails are plain text, not scanned PDFs. Memory: 256 MB. Timeout: 30 s.

- **`guide`** — EventBridge Scheduler target, daily at 9am local time (the schedule expression runs in `TZ_NAME` set to the SMB's timezone, e.g. `Asia/Singapore`). Reads `s3://og-list-source/onboarding.csv` and the rules and voice docs. For each row, computes `days_since_signup`, reads step state from `og-sends` and `og-state`, decides on a move. Emits one event per row that needs action: `og.step_due`, `og.nudge`, or `og.flag`, with the customer context as the event payload. On-track customers emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*
- **`sender`** — EventBridge rule on the three move events. Resolves channel, checks quiet hours and the weekend/holiday calendar, formats the message from the voice template, and ships via SES `SendRawEmail` (customer step/nudge) or the Slack incoming webhook (`og/slack/webhook` in Secrets Manager) for an owner flag. On quiet-hours or weekend defer, creates a one-off EventBridge Scheduler rule that re-invokes `sender` at the next available business minute. For steps the rules doc marks as personalized, calls Bedrock Haiku 4.5 to lightly rewrite the template body before sending. Writes a row to `og-sends` after a successful send. Memory: 256 MB. Timeout: 30 s.
- **`done-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a signed token on the done link and a Slack signature on the flag-button requests. Triggered by customer done-link clicks and by Slack interactive

button clicks (Pause/Hand off/Done). Writes to `og-state` and `og-audit`; on finish, updates the Drive sheet via the Sheets API and archives the old journey in `og-sends-archive`. Memory: 256 MB. Timeout: 15 s.

- **`digest`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads `og-sends` and `og-state` for the past week and the list; sends a digest message to a configured Slack channel summarizing who finished, who's stuck, and who's paused. No Bedrock; the message is a plain summary table. Memory: 256 MB.
- **`summary`** — EventBridge Scheduler target, monthly on the first Monday at 9am. Reads the past month's `og-sends`, `og-state`, and `og-audit`; calls Bedrock Haiku 4.5 to write a one-paragraph activation narrative (sign-ups, finishes, where people drop off); emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `og-sends`** — one row per message sent. PK `(customer_id, step_id)`; attributes: `sent_date`, `kind` (step/nudge), `channel` (email/slack), `recipient`. On-demand. No TTL.
- **DynamoDB · `og-state`** — one row per customer's live progress. PK `customer_id`; attributes: `steps_done` (set), `paused`, `finished`, `flagged`, `handed_off_to`, `paused_note`. On-demand.
- **DynamoDB · `og-audit`** — one row per write action of any kind. PK `(customer_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.

- **DynamoDB** · `og-sends-archive` — archived journeys after a finish. Same shape as `og-sends`; PK `(customer_id, journey_id, step_id)`. On-demand.
- **S3** · `og-list-source` — mirrored CSV from the Drive onboarding sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `og-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3** · `og-raw-mime` — raw inbound MIME from forwarded welcome emails. Lifecycle to Glacier at 30 days; expiry at 7 years.
- **S3** · `og-static` — the small HTML for the done-link landing page and any images referenced in the email templates.

Bedrock

- **Foundation model.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0`. Three callsites: `intake-ses-parser` for the welcome-email parsing, `sender` for the occasional personalized rewrite, and `summary` for the monthly narrative. `anthropic.claude-sonnet-4-6-20250930-v1:0` is wired but unused at this volume — reserved for a future richer rewrite path if one is justified.
- **Embeddings.** Not used. The onboarding list is structured rows; deterministic lookup beats vector retrieval here. No Knowledge Base, no S3 Vectors.
- **Quotas.** Default account quotas are more than enough at SMB volume. The guide itself doesn't call Bedrock; the parsing and rewrite lanes fire a few times a day at most.

EventBridge Scheduler config

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

SES inbound and outbound

- Set the MX record on a dedicated subdomain (e.g. `welcome.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `og-inbound-rules`: one rule with recipient `welcome@your-company.com` → spam scan → S3 PUT to `s3://og-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-ses-parser`.
- SES outbound for the step, nudge, and wrap-up emails: verify a sender identity at `hello@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request. Configure a configuration set with open/click tracking off and a bounce/complaint SNS topic so a hard-bounced customer email flips the row to `paused` automatically.

IAM (least privilege per Lambda)

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

- **guide role:** `s3:GetObject` on the list, rules, and voice keys; `dynamodb:Query` + `GetItem` on `og-sends`, `og-state`; `events:PutEvents` on the default bus. *No* `bedrock:*`.
- **sender role:** `events:ListSchedules` + `CreateSchedule` for the deferred-send one-offs; `secretsmanager:GetSecretValue` on the Slack webhook secret; `ses:SendRawEmail` from the verified sender identity; `bedrock:InvokeModel` on the Haiku ARN (for the rewrite path); `dynamodb:PutItem` on `og-sends`; outbound network access to `hooks.slack.com`.
- **done-handler role:** `dynamodb:PutItem` on `og-state` and `og-audit`; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret; outbound network access to `sheets.googleapis.com`; `dynamodb:Query` for state lookup; on finish, `dynamodb:BatchWriteItem` for archiving the journey to `og-sends-archive`.
- **enroll role:** `secretsmanager:GetSecretValue` on the webhook secret and the Sheets service-account secret; outbound network to `sheets.googleapis.com`.
- **intake-ses-parser role:** `s3:GetObject` on `og-raw-mime`; `bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the Slack webhook.
- **drive-sync role:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the list and rules buckets; outbound network to `www.googleapis.com`.

Slack interactive flow

The Slack incoming webhook is the simplest delivery surface but doesn't support interactive button responses. So the owner flag messages are posted via the `chat.postMessage` Web API instead, with Block Kit blocks containing the action buttons. Button clicks are sent by Slack to the configured Interactivity request URL, which is the `done-handler` Function URL. `done-handler` verifies the Slack signing secret on the inbound request, parses the action_id (`pause`, `hand_off`, `done`), opens a modal if needed (Pause and Hand off open modals; Done is one-tap), and processes the response when the modal is submitted.

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

Observability and cost gates

- **CloudWatch Logs:** all Lambdas, 7-day retention, structured JSON. Subscription filter on `"error"` + `"throttle"` + `"timeout"` to a CloudWatch metric for alerting.
- **Alarms:** guide Lambda failures > 0 in a day (the daily tick is the one piece that has to run); sender failure rate > 1% in 24h; done-handler signature-verification failures > 5/hour (might mean the Slack secret rotated); SES bounce rate above the SES reputation threshold.
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $15/month threshold, alarm at 80% and 100%, posts to SNS topic `og-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for Drive and Sheets APIs live in Secrets Manager under `og/drive/sa` (one service account with scopes for both APIs). The signup-webhook shared secret is `og/webhook/secret`. Slack bot token, signing secret, and webhook URL all under `og/slack/*`. SES sender identity lives in IAM and the verified-domain config. The configured timezone, holiday list reference, quiet-hours window, weekend setting, and onboarding-owner Slack ID all live in Parameter Store under `/og/config/`. Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC into a deploy role (no long-lived keys) running AWS SAM. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning for both `og-list-source` and `og-rules-source` so a bad Drive edit can be rolled back in one click, and version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily tick in UTC after a CI rotation. SAM fits cleanly here; CDK with a Python stack file also works. Total deployable surface: around eight Lambdas, four DDB tables, four S3 buckets, one EventBridge rule on the default bus (plus the Scheduler rules), one SES rule set, and one Budgets alarm.

That's the full system. Six narrative posts and this engineering reference. If you want to talk about adapting it for your business, see Work with me.