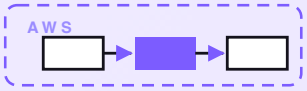
7-PART SERIES · FREE COMPANION



Compliance tracker

A serverless tracker that keeps a small business on top of its recurring compliance tasks — the checks that have to happen on a rhythm, like a monthly safety check, a quarterly data review, or an annual policy sign-off; reminds the right owner when each one is due; collects a quick note or file as proof it was done; and shows a simple done-and-overdue board. The owner can mark done, attach evidence, or snooze right from the reminder. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Compliance tracker

PART 1 OF 7

MAY 25, 2026 PART 1 OF 7 · COMPLIANCE TRACKER SERIES ~5 MIN READ

A compliance tracker on AWS for a few dollars a month

A small business has more recurring checks than anyone keeps in their head. The monthly fire-safety walk-through that has to be logged. The quarterly data-access review that nobody enjoys doing. The annual policy sign-off every staff member needs to complete. The weekly backup test, the cleaning-log sign-off, the equipment inspection that the insurer asks about every renewal. None of these is a one-off deadline — they come around again and again, and each one needs a quick note or photo to prove it actually happened. This post walks through the design of a small tracker that holds all of them, reminds the right owner when each is due, collects the proof, and shows what's done and what's overdue.

KEY TAKEAWAYS

- Three sources for tracked tasks: a Drive task list, a starter-pack lane, and an inbox forwarding lane.
- Every task ends in one of four moves on each tick: on-track, due now, overdue, or escalate.
- Per-task chains: a monthly check gets a 5-days-before / on-the-day / 2-days-after nudge; an annual sign-off gets 30/14/3 before.
- Reminders respect quiet hours and your holiday calendar. A task marked done goes quiet until next time.
- Designed on AWS for about $1.80/month at typical small-business volume.

The whole system on one page

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

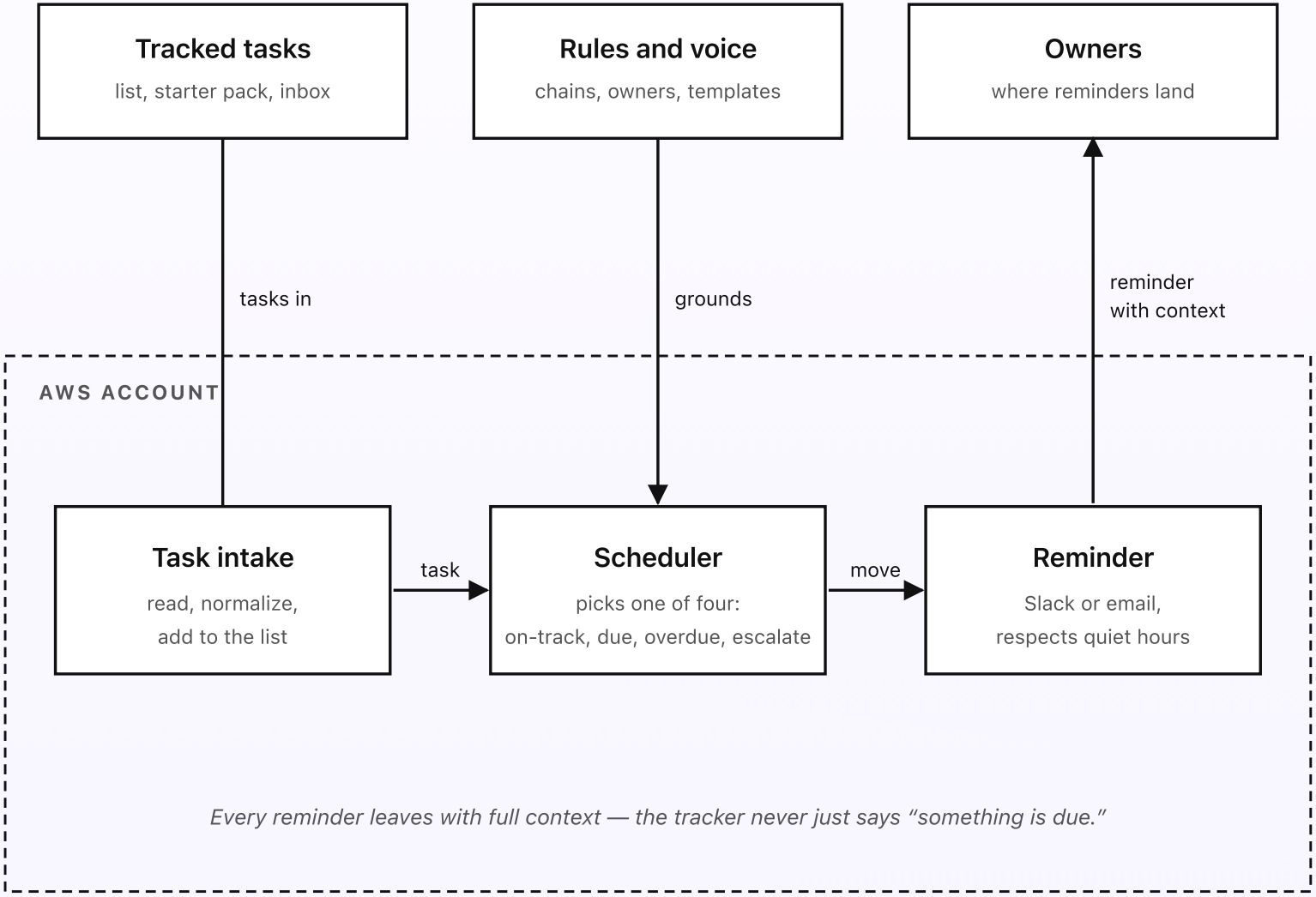

Tracked tasks
list, starter pack, inbox
Rules and voice
chains, owners, templates
Owners
where reminders land
tasks in
grounds
reminder
with context
AWS ACCOUNT
Task intake
read, normalize,
add to the list
task
Scheduler
picks one of four:
on-track, due, overdue, escalate
move
Reminder
Slack or email,
respects quiet hours
Every reminder leaves with full context — the tracker never just says “something is due.”

Fig 1. Three sources outside, three pieces inside AWS. Tasks flow in from a Drive list, a starter-pack lane, and an inbox forwarding lane. The Scheduler runs daily and picks one of four moves. The Reminder sends the right message to the right person at the right time.

What you set up once (the outside)

- **Tracked tasks.** A Google Sheet in a Drive folder, one row per recurring task: name, control area (safety, data, finance, HR, IT, facilities), how often it repeats (weekly, monthly, quarterly, yearly), owner email, what proof to keep (a note, a photo, a signed form), and the date it was last done. You can fill it in once and forget it; new tasks can also enter via two other lanes covered in Part 2 — a starter-pack lane (load a ready-made set of common controls for your industry and edit from there) and an inbox-forwarding lane (forward a policy or checklist to a dedicated address and the tracker proposes a task for one-tap approval).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the reminder chain for each task — how many days before the due date the tracker should nudge, and how many times. A monthly safety check typically gets a nudge 5 days before, on the day, and 2 days after; an annual policy sign-off gets 30, 14, and 3 days before. The doc also lists the owner per control area (or per individual task, if it overrides), the escalation target if the owner doesn't act, the quiet hours, and any holiday calendars to skip. The *voice* doc holds one reminder message template per control area — what the Slack DM or email actually says.
- **Owners.** The people responsible for each control. Each owner has a Slack member ID (so the reminder is a DM, not a public ping) or, if Slack isn't set up for them, an email address. Reminders land with the task name, the due date,

what proof to keep, a link to the task row, and a “Done” button that records the task complete and stops further nudges for this cycle.

What runs on every tick (the inside)

- **The task intake.** Three sources feed the list. The Drive sheet itself is the canonical store. New tasks can also be added via the starter-pack lane (pick a ready-made set of common controls for your industry and the tracker drops them into the sheet for review) and the inbox forwarding lane (forward a policy PDF to `controls@your-company.com`, the tracker uses Textract to read it and Bedrock Haiku 4.5 to suggest a task name, control area, and repeat rule, then drops a one-tap approval card in the owner’s Slack to confirm before the row is added).
- **The scheduler.** Runs once a day at 8am local. Reads the task list. For each task, computes the next due date from its repeat rule and the last-done date. Compares against the per-task reminder chain in the rules doc. Picks one of four moves. *On-track*: more than the first nudge away — do nothing. *Due now*: just crossed the first nudge threshold — remind the owner with full context. *Overdue*: the due date has passed with no completion — re-nudge, mention when the previous reminder went out. *Escalate*: stayed overdue past the final nudge — tell the escalation target named in the rules doc; log it. The scheduler itself doesn’t call a model on the daily tick — the move logic is plain Python.
- **The reminder.** Reads the voice doc, formats the reminder message for the chosen move and control area, and sends it. Slack DMs go through the Slack Web API. Email goes through SES outbound. Both honor quiet hours (no reminders between 6pm and 8am local by default) and the holiday calendar (no reminders on configured days). Every reminder writes a row in DynamoDB so

the next day's tick can tell whether the owner marked it done. A weekly digest summarizes everything done that week, plus what's coming up. A monthly summary writes a board-ready paragraph: count by control area, tasks done on time, longest-overdue items.

In plain words

Your monthly fire-safety walk-through is due on the 1st. The owner is your office manager Maria, and the proof to keep is a photo of the signed checklist. On May 27 (5 days before) the tracker pings her in Slack: "Monthly fire-safety walk-through — due June 1, please keep a photo of the signed sheet. *[link to the task]*" with a Done button. Maria's busy that week and doesn't open it. On June 1 she gets the on-the-day reminder. She does the walk-through, taps Done, and attaches a photo of the checklist. The tracker stamps the date, files the photo as proof, and computes the next due date — July 1. It stops nudging her. Next month the cycle starts again, and the board summary now shows the May check was done on time with evidence on file.

The cost of running this is about $1.80 a month at SMB volume. The cost of *not* running it is the missed safety check the insurer asks about after an incident, or the data review nobody did before the audit, or the policy sign-off half the team forgot.

DESIGN RULES THAT SHAPED EVERY DECISION

- Every reminder ships with full context — task, due date, what proof to keep, link to the row. The owner never has to dig.
- Four moves, always. On-track, due now, overdue, escalate. There is no fifth.
- Quiet hours and holidays are respected. Reminders are a finite resource; bad timing burns them.
- Done stops further nudges for this cycle and stamps the date. The next due date is computed from the repeat rule.
- The task list lives in Drive. Adding a task, changing an owner, or shifting a repeat rule doesn't need a deploy.
- Every reminder and every completion is logged with its proof. Audit a control next year and you can see it was done.

Why this shape

Most teams track recurring compliance in one of three places: a spreadsheet that nobody opens, a calendar invite that gets dismissed, or somebody's head. The spreadsheet works until it doesn't — one missed month and the whole thing goes stale. The calendar invite is the worst kind of false comfort: it pings on the day, with no context and no place to record that you did it, when there's no longer time to plan. And the head, of course, fails the moment the person who held it goes on holiday or leaves the company — and leaves no proof behind that anything was ever done.

The setup above moves the source of truth into a list the team already edits, but adds a small system that *looks at* that list every day and acts only when something needs doing. Reminders come early enough to plan around. They include what proof to keep so the owner doesn't have to ask. They escalate cleanly when the owner is out. And they collect the note or photo right there, so the audit trail builds itself. The tracker is invisible most days; visible only on the days a control actually needs attention.

The next four posts walk through each piece in turn: how a compliance task gets set up, how a control comes due, how a reminder reaches the right person, and how evidence gets captured and the cycle restarts. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 25, 2026 PART 2 OF 7 · COMPLIANCE TRACKER SERIES ~4 MIN READ

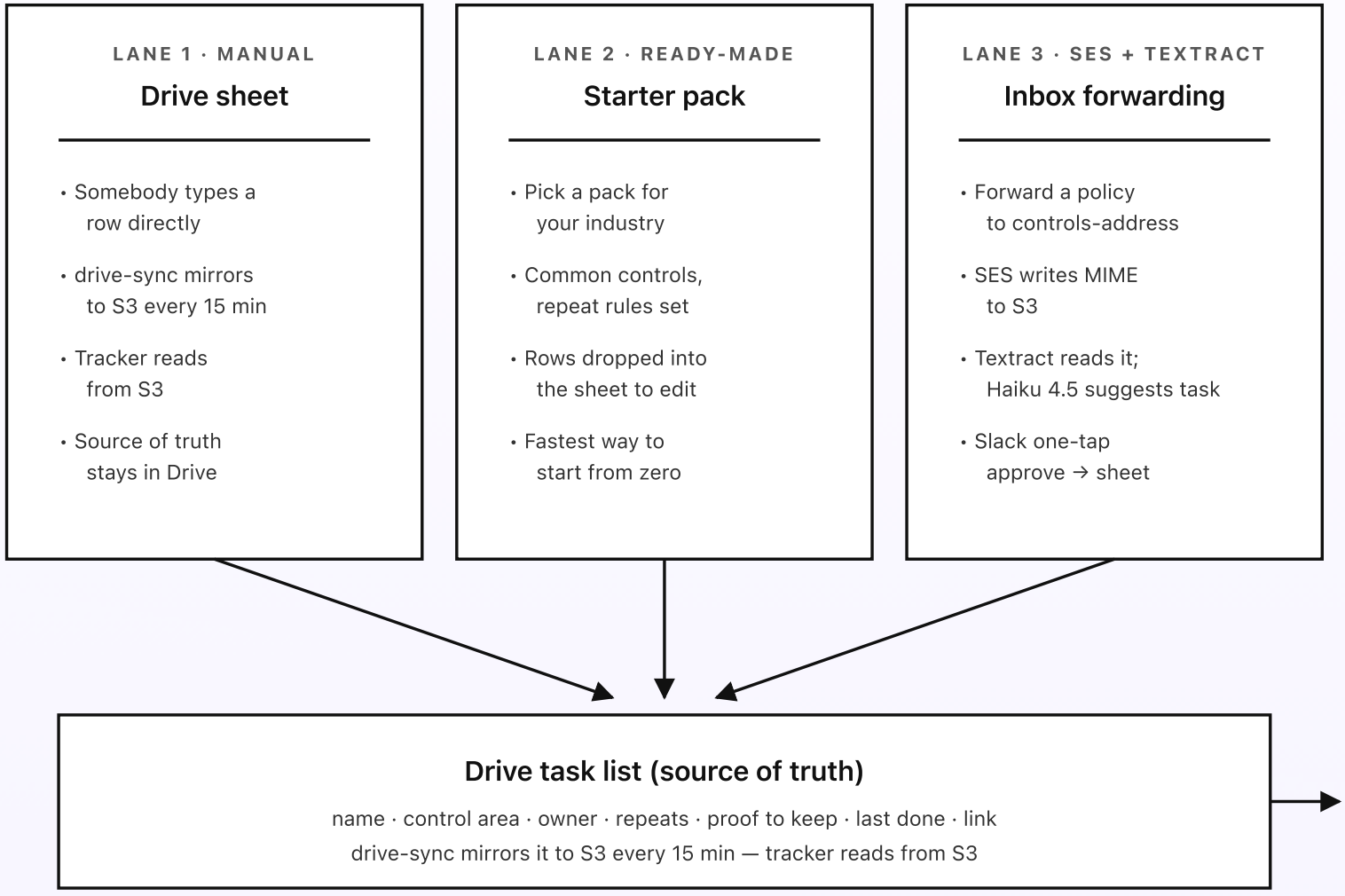
How a compliance task gets set up

The tracker only tracks what's in the list. So the first job is making sure the list actually reflects the recurring checks your business has to do. There are three ways a task gets in: somebody types a row in the Drive sheet, somebody loads a ready-made starter pack of common controls, or somebody forwards a policy document to a dedicated address. The first one is obvious. The other two exist because in real life nobody wants to type forty rows by hand on day one.

KEY TAKEAWAYS

- Three intake lanes feed one task list: the Drive sheet, a starter-pack lane, and an inbox-forwarding lane.
- The starter pack drops a ready-made set of common controls for your industry into the sheet for review.
- Forwarded policies are read by Textract; Bedrock Haiku 4.5 suggests a task name, control area, and repeat rule.
- Every proposed task goes to the owner's Slack for one-tap approval before it lands in the list.
- The Drive sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one task list



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

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

Lane 1: the Drive sheet itself

The simplest lane. Open the task list in Drive, add a row, save. The columns are short: name, control area, owner email, how often it repeats, what proof to keep, and the date it was last done. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://ct-tasklist-source/tasks.csv` if the sheet has changed since the last sync. The tracker reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so a bad bulk-edit can be rolled back in one click.

This lane covers the cases where you know exactly what a control is, how often it repeats, and you can spend thirty seconds typing it in. It's also where every later edit happens — changing an owner, adjusting a repeat rule, retiring a task.

Lane 2: the starter pack (the fastest way to start)

Typing forty rows by hand is the thing that stops a compliance tracker before it begins. So the second lane ships a set of ready-made packs — a short, sensible list of the controls a business in your industry usually has to run. A cafe gets food-safety checks, cleaning sign-offs, and fire-equipment inspections. A small software shop gets access reviews, backup tests, and an annual security-policy sign-off. A clinic gets sterilization logs, waste-disposal checks, and staff-training renewals.

Pick a pack and the tracker drops its rows straight into the Drive sheet, each pre-filled with a sensible repeat rule (monthly, quarterly, yearly), a proof type (note, photo, signed form), and a placeholder owner. Nothing is set in stone — the pack is a starting point you edit down to what actually applies. Delete the rows that don't fit, change the owners, adjust a couple of repeat rules, and you have a working list in minutes instead of an afternoon. The packs live as plain templates in the rules folder, so you can tweak them or add your own without touching code.

Lane 3: inbox forwarding

Some controls are buried in a policy document somebody already wrote. The data-retention policy says reviews happen quarterly. The health-and-safety manual lists a monthly walk-through. The IT handbook calls for an annual access review. Re-typing those into the task list is a fight you don't need to have when the document already says it.

Set up a dedicated inbound address — something like `controls@your-company.com` — via Amazon SES. Forward a policy or checklist PDF to it and the tracker takes it from there. SES writes the raw MIME to `s3://ct-raw-mime/`. The S3 PUT triggers a parser Lambda. The Lambda walks the MIME tree to the attachment, runs Amazon Textract on it (Textract reads PDF, PNG, JPEG, and TIFF natively; for a Word document the parser falls back to `python-docx`), and gets back the text. A Bedrock Haiku 4.5 call then reads the text and suggests a task: name, control area, a repeat rule, and a proof type, with a confidence score per field and a short note on where in the document it found each one. The prompt is short: "Suggest a recurring compliance task for the list. Return JSON only. Do not invent a frequency the text doesn't state." The output goes to a Slack message with *approve*, *edit*, and *discard* buttons. On *approve*, the row is written to the

Drive sheet. Every proposed task goes to a human first, because a control the model misread is worse than one that was never added — the misread one will quietly tell you everything is on track.

Why the task list stays the source of truth

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

Next post: how the scheduler actually reads the list, computes the next due date from each repeat rule, and picks one of four moves.

PART 3 OF 7

MAY 25, 2026 PART 3 OF 7 · COMPLIANCE TRACKER SERIES ~5 MIN READ

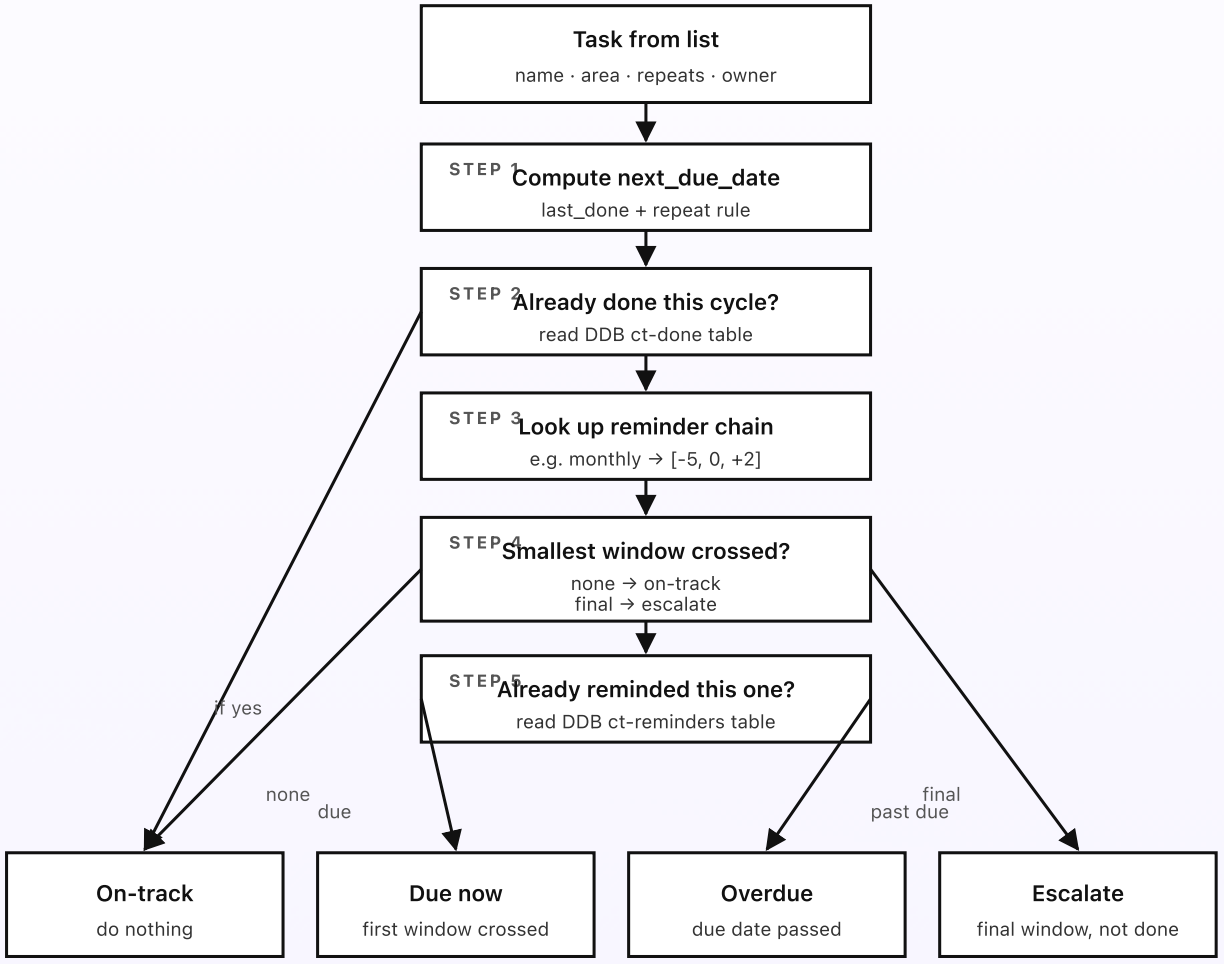
How a control comes due

Once a day, at 8am local time, an EventBridge Scheduler rule fires the scheduler Lambda. The Lambda reads the task list, looks at one row at a time, works out the next time that task is due, and decides whether to do nothing or to send a reminder — and if so, which kind. The whole decision is plain Python. No model. No vector retrieval. Every threshold lives in the rules doc, where an owner can edit it without a deploy.

KEY TAKEAWAYS

- The scheduler runs once a day via EventBridge Scheduler at 8am local time.
- Per-task reminder chains live in the rules doc — a monthly check gets 5-before / on-day / 2-after, an annual sign-off gets 30/14/3 before.
- Four moves per task, every tick: on-track, due now, overdue, escalate.
- DynamoDB tracks last-reminder and last-done per task so reminders aren't duplicate spam.
- The scheduler itself never calls a model. The decision is entirely deterministic.

The decision flow, per task



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

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

Reminder chains: the timing isn't magic, it's in the doc

The rules doc has one short section per task or control area. Each section names the chain in plain prose: "Monthly safety check: remind 5 days before, on the day, and 2 days after. Quarterly data review: 14 and 3 days before, then on the day. Annual policy sign-off: 30, 14, and 3 days before." The numbers are days relative to the due date when the reminder fires — negative means before, zero means on the day, positive means after. The first reminder is the early nudge. The last one is the escalation point — if the task still isn't done by then, the escalation target gets reminded too.

The chains exist for a reason. A 30-day annual-sign-off nudge gives the whole team time to read and acknowledge a policy. A 5-day monthly-check nudge is enough to plan the walk-through into the week without being so early it gets ignored. A 2-day-after reminder catches the check that slipped past its date. Different rhythms need different chains; the doc reflects that.

Per-task overrides exist too. The task list has an optional column called `chain_override`. Type a comma-separated list of day offsets there and the scheduler uses your numbers instead of the control-area default for that one row. This is the right escape hatch for the audit-prep review you want to start chasing two months out.

Four moves, always

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

- **On-track.** The due date is more than the first window away, or the task has already been done for the current cycle. Do nothing. Most tasks, most days, are on-track.
- **Due now.** The due date just crossed the first window threshold and the task isn't done yet. Send a fresh reminder with full context. Write a row to the `ct-reminders` DynamoDB table marking that the first window has fired.
- **Overdue.** The due date has passed without the task being done. Send a follow-up that names the previous reminder's date so the owner doesn't feel like they're seeing it for the first time. Write the new reminder to `ct-reminders`.
- **Escalate.** The final window in the chain passed without the task being done. Send to the escalation target named in the rules doc — usually the owner's manager — in addition to the owner. Mark the task as escalated in DynamoDB; the next tick keeps escalating daily until somebody marks it done. Bad timing burns reminders; an escalated control is one of the few cases where daily noise is the right answer.

State that makes the decision deterministic

The scheduler reads two DynamoDB tables every tick. `ct-reminders` records every reminder that's gone out: `(task_id, chain_index, reminded_date, dispatched_via)`. `ct-done` records every completion: `(task_id, done_date, by_user, cycle_due_date)`. With those two tables, the move-decision logic is a

few dozen lines of Python and zero magic. A given task with a given due date, a given reminder chain, and a given done/reminder history always produces the same move. Re-running the tick produces no extra reminders, because the state in DynamoDB shows what already fired.

Marking a task done is an explicit roll-forward: rows for the old cycle are kept for audit, the last-done date is stamped, and the next due date is computed fresh from the repeat rule. Part 5 covers the done-and-evidence flow in detail.

Why the daily tick uses no model

The scheduler could call a model on the tick to write a smarter reminder, or to decide whether to remind at all. 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 nudge 5 days before and the task isn't done, the nudge fires. A model in that loop introduces variance the team can't reason about. Second, model calls cost money, and most days most tasks are on-track, so the call would be wasted nine days out of ten.

Bedrock fires elsewhere — on the evidence-reading lane in Part 5, and on the monthly summary mentioned in Part 6. Not on the daily tick. The scheduler itself is plain Python that reads a doc and writes events.

Next post: how a reminder finds the right person, how quiet hours and holidays are honored, and what marking a task done actually does to the cycle.

PART 4 OF 7

MAY 25, 2026 PART 4 OF 7 · COMPLIANCE TRACKER SERIES ~5 MIN READ

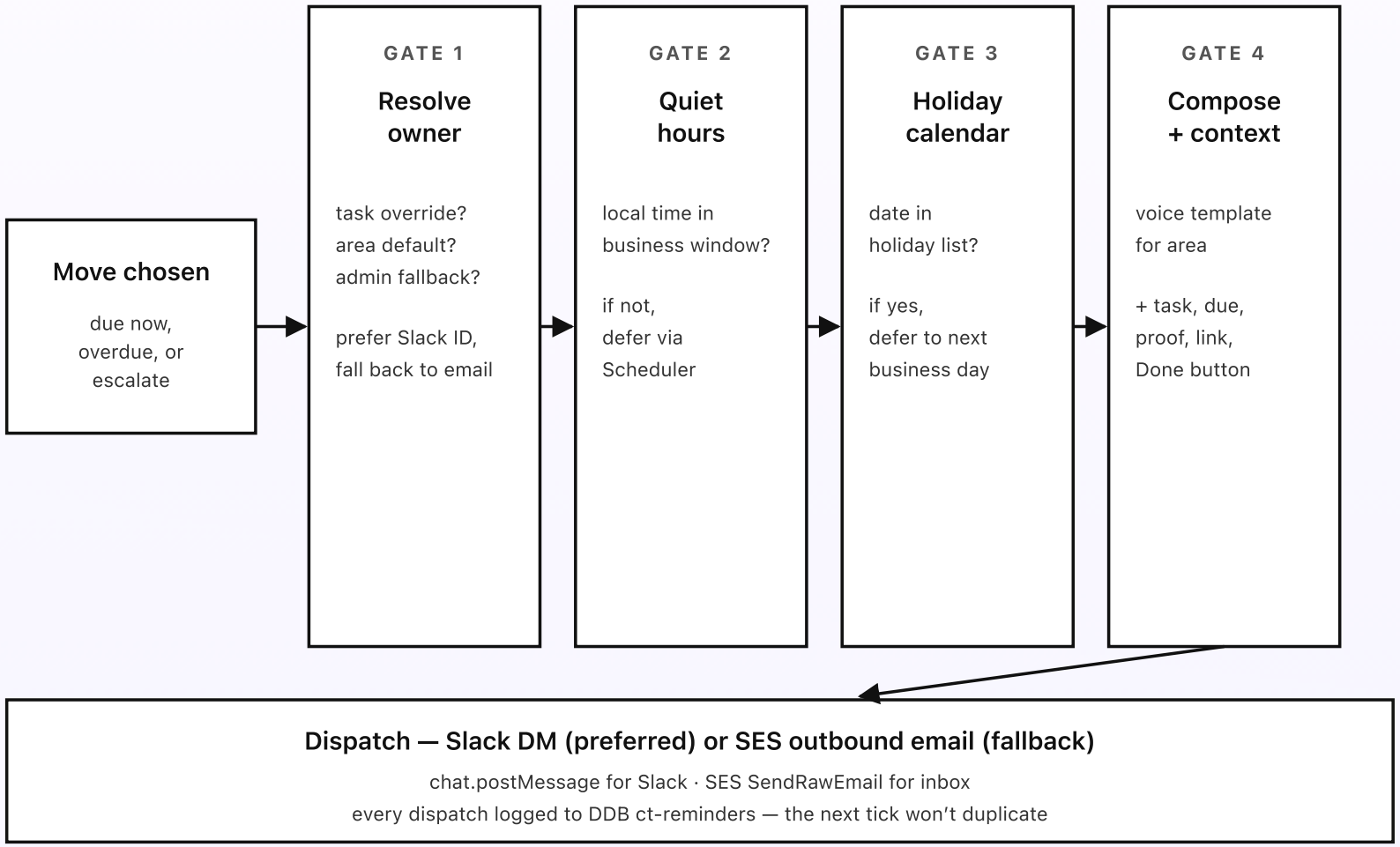
How a compliance reminder reaches its owner

The scheduler picked a move — due now, overdue, or escalate. Now the reminder 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 reminder is worse than no reminder: a 2am Slack ping, a generic “something is due,” a notification to somebody who left the company three months ago. Four small guardrails sit between the move and the actual reminder.

KEY TAKEAWAYS

- Owner resolution: per-task override beats per-control-area default beats fallback to the configured admin.
- Slack DMs are the default; email is the fallback if no Slack ID is configured.
- Quiet hours and holiday calendars defer reminders to the next available business hour.
- Every reminder ships with the task, due date, what proof to keep, a link to the row, and a Done button.
- Escalation reminds the named target instead of (or alongside) the owner; the owner stays in the loop.

Four guardrails on every dispatch



Every gate is a deterministic check — no model calls, no surprise behavior on a Tuesday in April.

Fig 4. Four guardrails between the move and the dispatched reminder. Resolve the owner. Honor quiet hours. Skip holidays. Compose with full context. Then ship via Slack or email and log the dispatch so the next tick doesn't duplicate.

Gate 1: resolve the owner

Three places the reminder Lambda looks for the owner of a task, in order. First, the task list's per-task `owner_email` column — if a row has a specific person assigned, that person owns it regardless of the control-area default. Second, the per-control-area default in the rules doc ("all safety tasks default to the office manager"). Third, the configured admin fallback — the person who set up the tracker and gets every unowned reminder. The fallback should never fire in steady state; if it does, the weekly digest names every task that hit the fallback so the rules doc can be updated.

Once the dispatch knows which person to remind, it looks up their delivery preference. The voice doc maps each owner to a Slack member ID if one is set, otherwise to an email address. Slack is preferred because reminders feel like work-context messages, and a Slack DM with a Done button is more useful than an email link. Email is the fallback so nobody falls through the cracks.

Gate 2: quiet hours

The scheduler itself runs at 8am local time, so the first time a move fires it's already in business hours. But escalations and overdue reminders that result from a tick can fire later in the day. And one-off computed dispatches (a second-

window reminder that takes effect the same day) can land 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 dispatch creates a one-off EventBridge Scheduler rule that fires at the next business-hour minute and exits without sending. The Scheduler invokes the same dispatch Lambda with the same payload at the deferred time, where Gate 2 will let it through.

Gate 3: holiday calendar

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. Gate 3 checks the current local date against that list and, if it's a configured holiday, defers the dispatch to the next non-holiday business day.

The list is on purpose — the tracker won't auto-detect a country's public holidays for you. The failure modes are very different. A holiday you forgot to add fires a reminder that lands on a closed laptop. A holiday in the list that's no longer observed just delays a reminder 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 Slack message template per control area: a short message with placeholders for the task name, due date, what proof to keep, and a link to

the task row. The dispatch Lambda fills the placeholders, attaches a “Done” button, and ships the message via the Slack Web API `chat.postMessage` call. The bot token itself lives in Secrets Manager.

For email fallback, the same template is wrapped in a small HTML email with the same fields and a link that, when clicked, hits a Function URL that records the completion — the email equivalent of the Slack button.

An escalate move adds a second recipient: the escalation target named in the rules doc for that control area. The owner is still reminded (the escalation isn’t a substitute for the original owner’s reminder — both go out), but the manager now sees it too. The escalate template is slightly different: it includes the previous reminder dates and how many days the task has been overdue, so the manager has the audit trail at hand.

Every dispatch — Slack or email, owner or escalate — writes a row to `ct-reminders` in DynamoDB. The next day’s tick reads that row and knows not to remind the same window 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 reminders themselves — check who actually owns this, don’t ping at 11pm, skip the day everyone’s off, include enough context that the recipient doesn’t have to ask 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 reminder to remember.

Next post: how evidence gets captured once an owner has acted on the reminder — how the tracker records the task done, files the note or photo as proof, and rolls the cycle forward to the next due date.

PART 5 OF 7

MAY 25, 2026 PART 5 OF 7 · COMPLIANCE TRACKER SERIES ~5 MIN READ

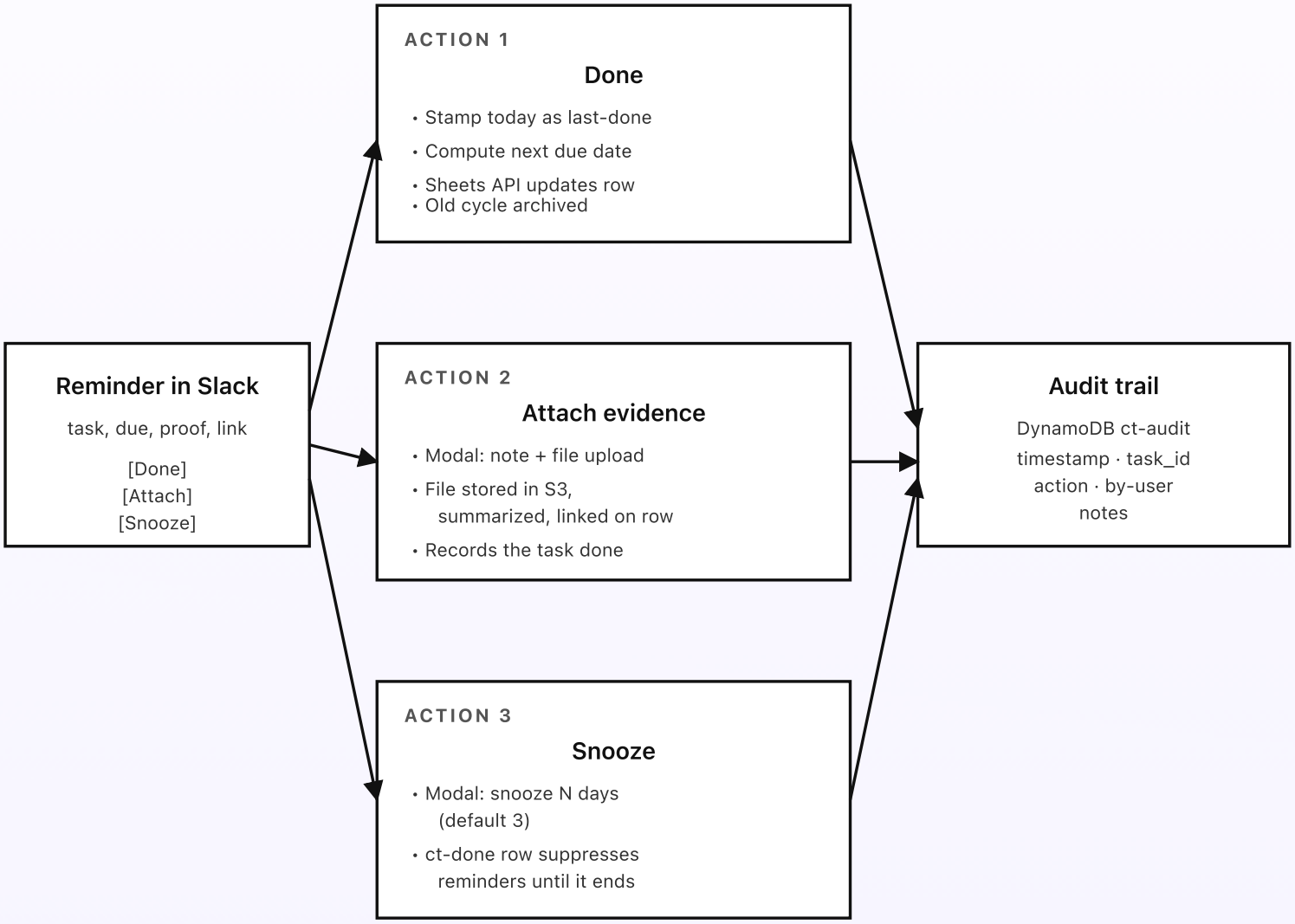
How evidence gets captured

A reminder lands in Maria's Slack DM at 8:03am. The monthly fire-safety walk-through is due today. There's a Done button. What happens when she taps it? The honest answer is "it depends on what she actually did." This post walks through the three things the tracker can do on a reminder — mark done, attach evidence, snooze — and how the task list, the cycle state, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three actions per reminder: *done* (stamp the date, compute the next due date), *attach evidence* (a note or file as proof), *snooze* (delay).
- Each action updates the task list via the Sheets API and writes an audit row.
- Marking done archives the old cycle to a separate sheet for history.
- Snooze is bounded — you can only snooze a few times before the tracker escalates anyway.
- The Done button is a Slack interactive message backed by a Function URL.

Three actions on Done



Done stamps the proof and rolls the cycle forward — the next due date is computed and the task goes quiet.

Fig 5. Three actions per reminder, three different effects. Done stamps the date and computes the next due date. Attach evidence keeps the proof. Snooze delays without dismissing. Every action writes to the audit trail.

Action 1: done (the most common)

Maria did the walk-through and signed the checklist. She taps *Done*. That's a one-tap action — no modal — for the common case where the proof is the fact that a trusted owner says they did it. The button submits to a Function URL Lambda, and three things happen, in order. First, the Sheets API updates the row in the task list: `last_done` set to today, and a small note in the `done_by` column with the user who acted. Second, the existing cycle's rows in `ct-reminders` are copied to `ct-reminders-archive` with a cycle id, and the live cycle state is cleared. Third, a `action: done` row is written to `ct-audit` with the user, timestamp, the cycle that was due, and the computed next due date.

Tomorrow's tick reads the task list, sees the last-done date is recent and the next due date is a month out, and lands at *on-track*. The owner won't hear from the tracker about this task again until the next cycle's first window crosses.

Action 2: attach evidence (when proof has to be on file)

Some controls aren't done until there's a record of them. The insurer wants a photo of the signed fire-safety sheet. The auditor wants the access-review export. The board wants the signed-off policy. For those, "trust me, I did it" isn't enough — the proof is the point.

Attach evidence opens a small Slack modal with a note field and a file upload. Maria types a one-line note and drops in the photo. On save, a Function URL Lambda stores the file in S3 under that task's evidence prefix (`s3://ct-evidence/<task_id>/<cycle>/`), runs Textract and a short Bedrock Haiku 4.5 call to pull a one-line summary of what the proof shows ("signed fire-safety checklist, all items ticked, dated June 1"), links the file on the task row, and records the task done in the same step. The summary is a convenience for the monthly board view — the file itself is the real evidence, kept versioned in S3 for years.

This is the one owner-facing place Bedrock touches in the whole system, and even here it's optional: if the model is unavailable, the file is still stored and the task is still recorded done — you just don't get the auto-summary that day. The proof never depends on the model.

Action 3: snooze (the deferral)

Some checks slip for reasons the reminder chain doesn't plan for. The contractor who runs the equipment inspection is out this week. The data review is stuck behind a holiday. The only person who can sign the policy is travelling. Maria isn't ready to mark it done, but she's also handling it — she just needs the tracker to be quiet for a couple of days.

Snooze opens a small modal asking for the number of days, with a 3-day default and a max of 7. On save, a row is written to `ct-done` with `(task_id, snooze_until)` . The next day's tick reads that row in the "already done this cycle?" check from Part 3 and treats the task as on-track until the snooze ends. When the snooze ends, the tracker re-evaluates the cycle from where it was — if

the task is now past its due date, the next reminder is an overdue one or an escalation.

Snooze is bounded. The rules doc has a configurable `max_snoozes_per_cycle` setting (default three). After that many snoozes on the same cycle, further snooze attempts are rejected with a "You've hit the snooze cap on this task; please mark it done or escalate" reply, and the next tick reminds normally regardless. This is a soft constraint that exists because the most dangerous failure mode is repeatedly snoozing a control to nowhere.

Every action is logged, every action is reversible

The `ct-audit` table records every done, attach, and snooze with the user who took the action, the timestamp, and a snapshot of the row before and after. If a wrong done date gets entered, a manager 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 kind of reversibility matters most for the controls you'll only think about once a quarter or once a year. The next time the annual policy sign-off comes up, it might be Maria again or it might be the person who took her job after she got promoted. Either way, the audit trail — and the evidence files in S3 — are the only memory the next person has.

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 one of the cheapest systems in the series.

PART 6 OF 7

MAY 25, 2026 PART 6 OF 7 · COMPLIANCE TRACKER SERIES ~3 MIN READ

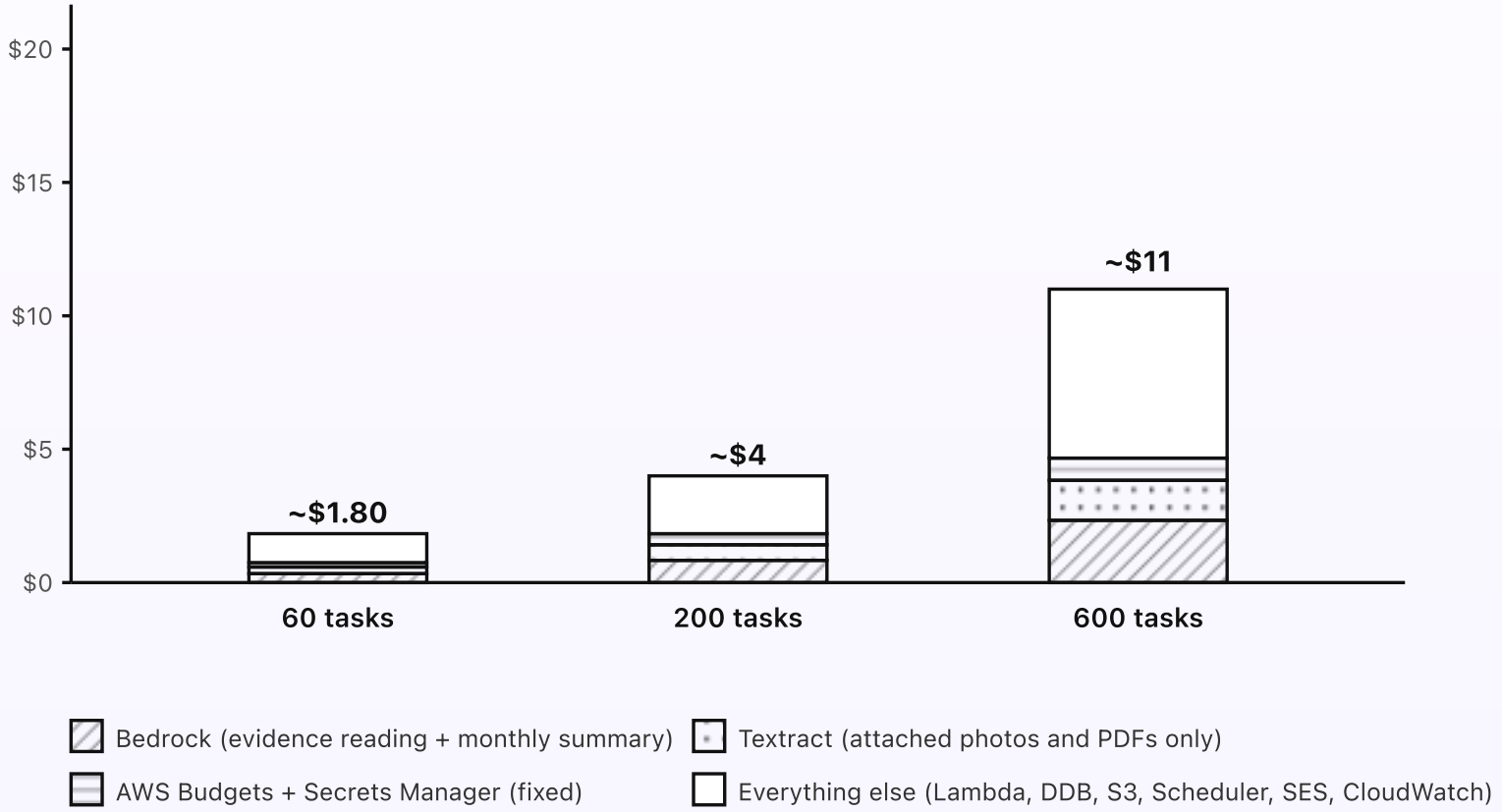
What the compliance tracker costs

The tracker 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 posts a handful of messages to Slack. It calls no models on the tick. Bedrock fires only when somebody attaches a photo or PDF as evidence and once a month for the board summary. At typical SMB volume, the bill is under two dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $1.80/month at typical SMB volume (around 60 recurring tasks).
- 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 evidence reading (a few times a month) and the monthly summary.
- At 200 tasks the bill is around $4. At 600 tasks it's around $11.

Cost at three volumes



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

Fig 6. Monthly cost at three task volumes. Bedrock and Textract are small slivers because they only fire on the evidence-reading lane and the monthly summary. The dominant cost is the everything-else bucket: the daily tick reading every task.

Where the dollars actually go

Lambda runtime (the bulk). The scheduler runs once a day. Each tick reads the task CSV from S3, iterates the rows, computes the next due date for each, and decides on a move. At 60 tasks, that's a few hundred milliseconds. At 600 tasks it's under a second. Either way it's pennies a month. Add the dispatch Lambda firing for each reminder (around five to twenty reminders a month at 60 tasks, fifty to a couple hundred at 600), the Function URL Lambda for done-and-evidence, the calendar-sync and drive-sync Lambdas — the Lambda total still lands under a dollar at all three volumes.

DynamoDB on-demand. Three small tables: `ct-reminders`, `ct-done`, `ct-audit`. Reads are dominant during the daily tick (one read per task per tick, plus cycle history). Writes are reminder events and audit rows. Pennies a month at any of these volumes.

S3 + Storage. The mirrored task CSV plus the evidence files (photos, signed forms, PDFs). A few megabytes total at SMB volume, growing slowly as proof accumulates. Cents a month.

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

SES. Inbound for the forwarding lane: $0.10 per thousand received messages (so a couple of cents a year for an SMB). Outbound for email-fallback reminders: $0.10 per thousand sent. Both are negligible at this scale.

Bedrock (only when something fires it). The daily tick uses no Bedrock. The evidence-reading lane fires Haiku 4.5 once per attached photo or PDF: a few

thousand input tokens (the Textract output) and a few hundred output tokens (the one-line summary), so a fraction of a cent per read. At a few attachments a month, Bedrock costs cents. The monthly summary is one larger call: write a paragraph that summarizes the month's reminders, completions, and overdue items; a couple of cents.

Textract (only on attached photos and PDFs). Per-page pricing; a typical attachment is one to a few pages. A few cents per read. At a few attachments a month, Textract is a few cents. At 600 tasks with twenty attachments per month, it lands around a dollar.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the done-and-evidence endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The tracker sleeps 23.99 hours a day.
- **A Knowledge Base.** The task 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 evidence-reading lane and the monthly summary.

How the cost scales

Lambda runtime grows roughly linearly with task count, because every task is evaluated on every tick. DynamoDB grows linearly too. Bedrock and Textract are uncorrelated with task count — they only fire when somebody attaches evidence or it's the first of the month. So the bill at 1,500 tasks is around $25; at 3,000 it's around $50. Past those volumes the daily-tick model probably stops being right (you'd switch to a partial-tick that only evaluates tasks inside the union of all reminder windows), but those are optimizations for very large task lists — not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. The tracker'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 25, 2026 PART 7 OF 7 · COMPLIANCE TRACKER SERIES ~8 MIN READ

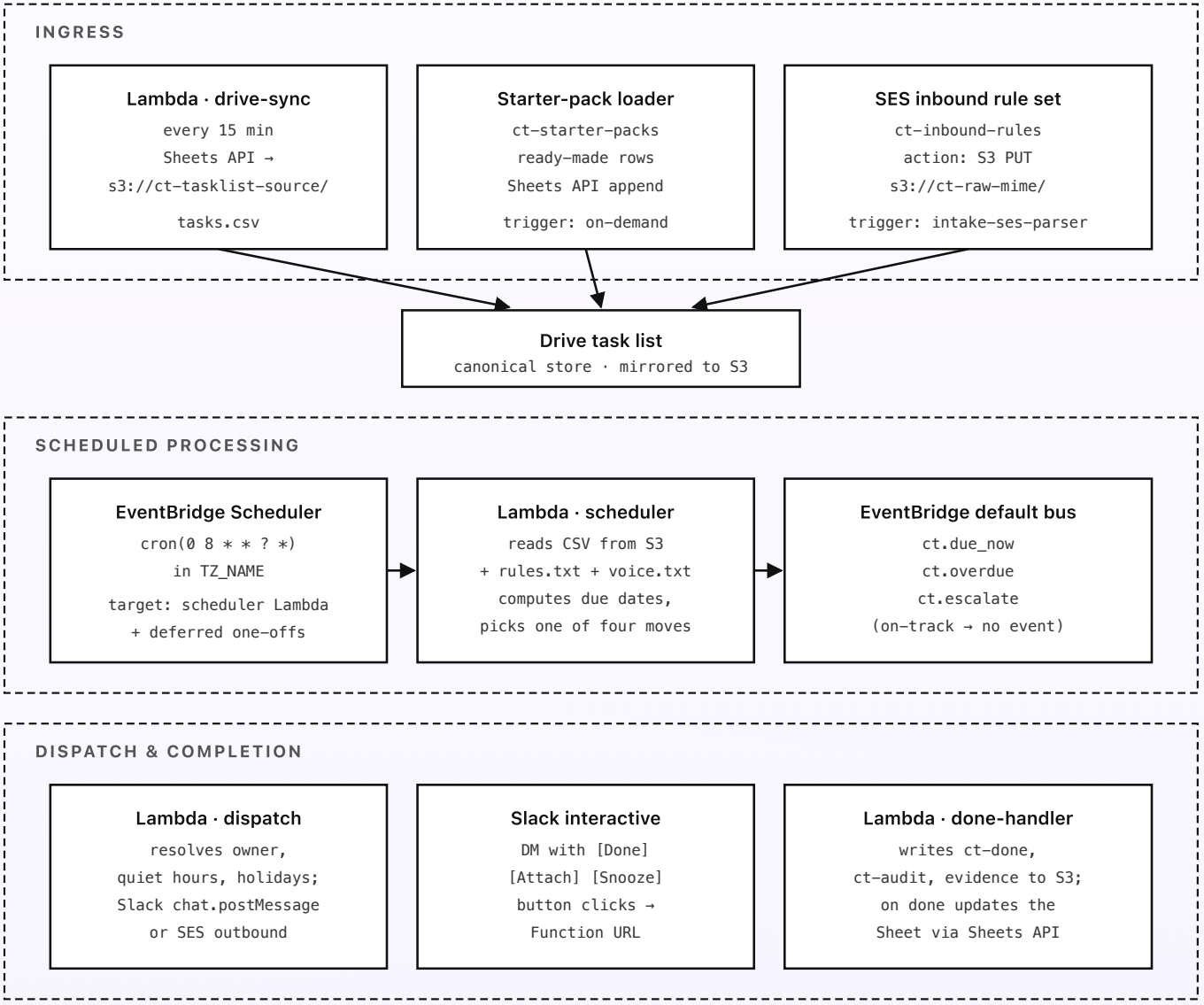
Engineering reference: the compliance tracker architecture

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

Region and account shape

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

Topology



Every reminder leaves with full context — and every interaction is logged to ct-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the task list), scheduled processing (the daily scheduler tick emitting events), dispatch and completion (the reminder ships and the 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.

- **`drive-sync`** — EventBridge Scheduler target, fires every 15 minutes. Uses the Google Drive API + Sheets API (service-account credentials in Secrets Manager under `ct/drive/sa`) to export the task sheet as CSV and write to `s3://ct-tasklist-source/tasks.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://ct-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`starter-pack-loader`** — Function URL (admin-only, IAM-authenticated) invoked from a small internal admin page. Reads a chosen pack template from `s3://ct-rules-source/packs/<pack>.json` and appends its rows to the Drive task sheet via the Sheets API, pre-filling repeat rules, proof types, and a placeholder owner. Idempotent per pack (skips rows already present by name). Memory: 256 MB. Timeout: 30 s.
- **`intake-ses-parser`** — S3 PUT trigger on `s3://ct-raw-mime/`. Parses MIME, extracts the attachment, runs Textract via `StartDocumentTextDetection` +

`StartDocumentAnalysis` (asynchronously to handle multi-page policies). On Textract completion (via SNS notification), reads the structured text 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 task row (name, control area, repeat rule, proof type). Posts the proposal to Slack via `chat.postMessage` with Approve/Edit/Discard buttons. For DOCX attachments (Textract doesn't accept them), falls back to `python-docx`; XLSX uses `openpyxl`. Both packages are stable and widely used in 2026, though their maintenance velocity is light — for a policy-parsing path that only runs a few times a month, that's acceptable. If extraction precision becomes a concern, the active community fork `python-docx-oss` is a drop-in alternative. Memory: 512 MB. Timeout: 60 s.

- **`scheduler`** — EventBridge Scheduler target, daily at 8am local time (the schedule expression runs in `TZ_NAME` set to the SMB's timezone, e.g. `Asia/Singapore`). Reads `s3://ct-tasklist-source/tasks.csv` and the rules and voice docs. For each row, computes `next_due_date` from the repeat rule and last-done date, reads cycle state from `ct-reminders` and `ct-done`, decides on a move. Emits one event per row that needs action: `ct.due_now`, `ct.overdue`, or `ct.escalate`, with the task context as the event payload. On-track tasks emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*
- **`dispatch`** — EventBridge rule on the three move events. Resolves owner, checks quiet hours and holiday calendar, formats the reminder from the voice template, and ships via Slack `chat.postMessage` (bot token `ct/slack/bot-token` in Secrets Manager) or SES `SendRawEmail`. On quiet-hours or holiday defer, creates a one-off EventBridge Scheduler rule that re-invokes `dispatch`

at the next available business minute. Writes a row to `ct-reminders` after a successful send. Memory: 256 MB. Timeout: 30 s.

- **`done-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a Slack signature on the request body. Triggered by Slack interactive button clicks (Done/Attach/Snooze) and by email-link clicks. Writes to `ct-done` and `ct-audit`; on done, updates the Drive sheet via the Sheets API and archives the old cycle in `ct-reminders-archive`. On attach, stores the uploaded file in `s3://ct-evidence/` and calls Textract + Bedrock Haiku 4.5 for a one-line summary. Memory: 256 MB. Timeout: 15 s.
- **`digest`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads `ct-reminders` and `ct-done` for the past week and the task list; sends a digest message to a configured Slack channel summarizing tasks done and items coming up. 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 `ct-reminders`, `ct-done`, and `ct-audit`; calls Bedrock Haiku 4.5 to write a one-paragraph board narrative; emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `ct-reminders`** — one row per dispatch. PK `(task_id, chain_index)`; attributes: `reminded_date`, `dispatched_via` (slack/email), `recipient`, `move` (due_now/overdue/escalate). On-demand. No TTL.
- **DynamoDB · `ct-done`** — one row per completion. PK `task_id`; sort key `done_date`; attributes: `action` (done/attach/snooze), `by_user`,

`snooze_until` (if action = snooze), `cycle_due_date`, `next_due_date`, `evidence_key` (if action = attach). On-demand.

- **DynamoDB** · **`ct-audit`** — one row per write action of any kind. PK `(task_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.
- **DynamoDB** · **`ct-reminders-archive`** — archived cycles after a completion. Same shape as `ct-reminders`; PK `(task_id, cycle_id, chain_index)`. On-demand.
- **S3** · **`ct-tasklist-source`** — mirrored CSV from the Drive task sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · **`ct-rules-source`** — mirrored rules and voice docs as plain text, plus the starter-pack templates. Versioning enabled.
- **S3** · **`ct-raw-mime`** — raw inbound MIME from forwarded policies. Lifecycle to Glacier at 30 days; expiry at 7 years.
- **S3** · **`ct-evidence`** — uploaded proof files (photos, signed forms, PDFs), keyed by `<task_id>/<cycle>/`. Versioning enabled; this is the durable evidence store, kept for 7 years.

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`. Two callsites: `intake-ses-parser` for the forwarded-policy parsing and `done-handler` / `summary` for the evidence summary and monthly board narrative. Claude Sonnet 4.6 (`anthropic.claude-sonnet-4-6-`

`20250930-v1:0`) is wired but unused by default — reserved for the rare case where a forwarded policy is long and ambiguous enough that Haiku's proposal needs a heavier second pass.

- **Embeddings.** Not used. The task 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 scheduler itself doesn't call Bedrock; the parsing and evidence lanes fire a few times a month at most.

EventBridge Scheduler config

- **`ct-daily-tick`** — `cron(0 8 * * ? *)` in the SMB's timezone. Target: `scheduler` Lambda.
- **`ct-drive-sync`** — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- **`ct-weekly-digest`** — `cron(0 18 ? * SUN *)` in TZ. Target: `digest` Lambda.
- **`ct-monthly-summary`** — `cron(0 9 ? * 2#1 *)` (first Monday at 9am) in TZ. Target: `summary` Lambda.
- **One-off rules** — created on the fly by `dispatch` when a quiet-hours or holiday defer is needed. Use `at(YYYY-MM-DDTHH:MM:SS)` expressions with `--action-after-completion DELETE` so the rule self-cleans.

SES inbound and outbound

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

- SES inbound rule set `ct-inbound-rules`: one rule with recipient `controls@your-company.com` → spam scan → S3 PUT to `s3://ct-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-ses-parser`.
- SES outbound for the email-fallback reminders: verify a sender identity at `tracker@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request.

IAM (least privilege per Lambda)

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

- **scheduler role:** `s3:GetObject` on the task list, rules, and voice keys; `dynamodb:Query` + `GetItem` on `ct-reminders`, `ct-done`; `events:PutEvents` on the default bus. *No* `bedrock:*`.
- **dispatch role:** `scheduler:CreateSchedule` for the deferred-reminder one-offs; `secretsmanager:GetSecretValue` on the Slack bot-token secret; `ses:SendRawEmail` from the verified sender identity; `dynamodb:PutItem` on `ct-reminders`; outbound network access to `slack.com`.
- **done-handler role:** `dynamodb:PutItem` on `ct-done` and `ct-audit`; `s3:PutObject` on `ct-evidence`; `textract:*` + `bedrock:InvokeModel` for the evidence summary; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret; outbound network access to `sheets.googleapis.com`; on done, `dynamodb:BatchWriteItem` for archiving the old cycle to `ct-reminders-archive`.
- **intake-ses-parser role:** `s3:GetObject` on `ct-raw-mime`; `textract:StartDocumentTextDetection` + `StartDocumentAnalysis`;

`bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the Slack bot token.

- **drive-sync and starter-pack-loader roles:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` and `s3:GetObject` on the task list and rules buckets; outbound network to `www.googleapis.com`.

Slack interactive flow

The reminder messages are posted via the `chat.postMessage` Web API with Block Kit blocks containing the action buttons. Button clicks are sent by Slack to the configured Interactivity request URL, which is the `done-handler` Function URL. `done-handler` verifies the Slack signing secret on the inbound request, parses the action_id (`done`, `attach`, `snooze`), opens a modal if needed (Attach and Snooze open modals; Done is one-tap), and processes the response when the modal is submitted. The Attach modal uses Slack's `files.upload` flow so the proof file lands in S3 rather than only in Slack.

The Slack app needs `chat:write`, `im:write`, `files:read`, and the Interactivity URL configured. The bot token lives in Secrets Manager under `ct/slack/bot-token`. The signing secret is `ct/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:** scheduler Lambda failures > 0 in a day (the daily tick is the one piece that has to run); dispatch failure rate > 1% in 24h; done-handler signature-verification failures > 5/hour (might mean the Slack secret rotated).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $15/month threshold, alarm at 80% and 100%, posts to SNS topic `ct-cost-alarm` subscribed to the on-call admin's email and Slack.

Config and secrets

Service-account credentials for Drive, Sheets, and Calendar APIs all live in Secrets Manager under `ct/drive/sa` (one service account with scopes for all three APIs). Slack bot token and signing secret under `ct/slack/*`. SES sender identity lives in IAM and the verified-domain config. The configured timezone, holiday list reference, quiet-hours window, `max_snoozes_per_cycle`, and admin fallback owner all live in Parameter Store under `/ct/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), building with 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 `ct-tasklist-source`, `ct-rules-source`, and `ct-evidence` so a bad Drive edit can be rolled back in one click and proof files are never silently overwritten, and version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily tick in UTC after a CI rotation. Total deployable surface: around 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.