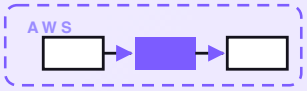
7-PART SERIES · FREE COMPANION



Deadline reminder

A serverless reminder that keeps a simple calendar of every recurring obligation your business has to meet — payroll runs, tax filings, license and permit renewals, insurance payments, compliance dates; reminds the right person early enough to act, with the right lead time for each type; escalates to the owner if one is about to be missed. The owner can mark it done, snooze, or ack-only right from the reminder. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Deadline reminder

PART 1 OF 7

MAY 19, 2026 PART 1 OF 7 · DEADLINE REMINDER SERIES ~5 MIN READ

A deadline reminder on AWS for a few dollars a month

A small business has more recurring deadlines than anyone keeps in their head. The payroll run that has to be submitted two days before payday. The quarterly sales-tax filing that is always “next week” until it is late. The business license that renews every year on a date nobody remembers. The workers’ comp policy. The annual report to the state. Most of these repeat forever, each one owned by a different person, each needing a different amount of warning. Miss one and the cost is a penalty, a lapsed permit, or a payroll that runs late. This post walks through the design of a small system that keeps a simple calendar of every recurring obligation, reminds the right person early enough to act, and escalates if one is about to be missed — then confirms when it is handled.

KEY TAKEAWAYS

- Three sources for deadlines: a Drive sheet you keep, an inbox forwarding lane, and a calendar import lane.
- Every deadline ends in one of four moves on each check: clear, first reminder, follow-up, or escalate.
- Per-type lead times: payroll gets a nudge at 5 days, a follow-up at 2, an escalation at 1.
- Reminders carry full context, respect quiet hours and weekends, and roll forward the moment a cycle is done.
- Designed on AWS for about $1.60/month at typical small-business volume.

The whole system on one page

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

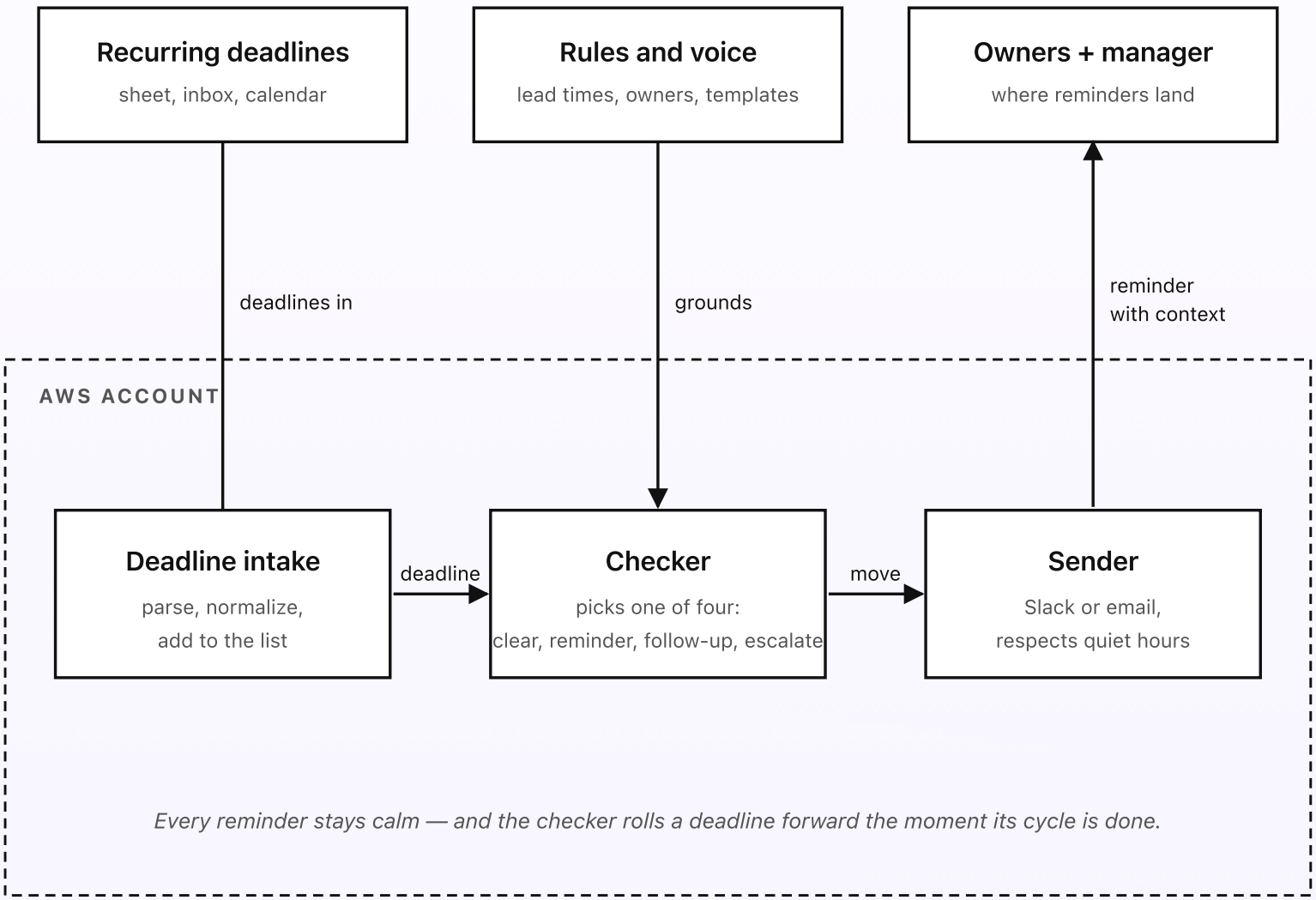

Recurring deadlines
sheet, inbox, calendar
Rules and voice
lead times, owners, templates
Owners + manager
where reminders land
deadlines in
grounds
reminder
with context
AWS ACCOUNT
Deadline intake
parse, normalize,
add to the list
deadline
Checker
picks one of four:
clear, reminder, follow-up, escalate
move
Sender
Slack or email,
respects quiet hours
Every reminder stays calm — and the checker rolls a deadline forward the moment its cycle is done.

Fig 1. Three sources outside, three pieces inside AWS. Deadlines flow in from a Drive sheet, an inbox forwarding lane, and a calendar import lane. The Checker runs daily and picks one of four moves. The Sender sends the right reminder to the right person at the right time.

What you set up once (the outside)

- **Recurring deadlines.** A Google Sheet in a Drive folder, one row per recurring obligation: name, type (payroll, tax filing, license renewal, insurance, compliance), owner email, how often it repeats (monthly, quarterly, yearly, or a custom day pattern), the next due date, the lead time, and a link to any reference document. You fill this in once with the deadlines you already know, then let new ones flow in from two other lanes covered in Part 2 — an inbox-forwarding lane (forward a renewal notice or a tax letter to a dedicated address and the system proposes a row for one-tap approval) and a calendar import lane (tag a calendar event and it becomes a proposed deadline).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the lead-time chain for each type — how many days before the due date the system should remind, and how many times. Payroll typically gets a nudge at 5 days, a follow-up at 2, and an escalation at 1; tax filings get a longer 30/14/7/2; license renewals get 60/30/14/7. The doc also lists the owner per deadline (or per type, if it overrides), the escalation target if a deadline stays unhandled, the quiet hours, and any holiday days to skip. The *voice* doc holds one reminder template per step — the gentle first nudge, the firmer follow-up, the escalation.
- **Owners and manager.** The person responsible for each deadline receives the reminders. The manager receives the escalation when a deadline is about to be

missed. Reminders land with the deadline name, the type, the days remaining, a link to any reference, and the three buttons covered in Part 5.

What runs on every check (the inside)

- **The deadline intake.** Three sources feed the list. The Drive sheet is the canonical store. New deadlines can also be added via the inbox forwarding lane (forward a notice to `deadlines@your-company.com`, the system uses Textract to read the PDF and Bedrock Haiku 4.5 to extract name, type, due date, and repeat, then drops a one-tap approval card in the owner's Slack to confirm before the row is added) and the calendar import lane (tag a calendar event with `#deadline` and the same approval flow runs).
- **The checker.** Runs once a day at 8am local. Reads the deadlines. For each one, computes days-to-due. Compares against the per-type lead-time chain in the rules doc. Picks one of four moves. *Clear*: the due date is still far off, or this cycle is already handled — do nothing. *First reminder*: just crossed the first lead-time step — send a gentle reminder with full context. *Follow-up*: crossed a later step with no action — send a firmer reminder, mention when the previous one went out. *Escalate*: hit the final step with no action — tell the manager named in the rules doc; log it. The checker itself doesn't call a model on the daily check — the move logic is plain Python.
- **The sender.** Reads the voice doc, formats the reminder message for the chosen move and tone, and sends it. Slack is the default; email is the fallback 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 check can tell whether a reminder already went out. A weekly digest summarizes everything coming up. A monthly summary

writes a one-paragraph narrative: what was handled, what slipped, and what is due next month.

In plain words

Your quarterly sales-tax filing is due on April 30. It is owned by Dana, your bookkeeper, and the rules doc gives tax filings a 30/14/7/2 lead-time chain. On April 1, thirty days out, the system sends Dana a gentle Slack reminder: “Heads up — the Q1 sales-tax filing is due April 30. Here’s the link to last quarter’s.” Dana is busy and doesn’t act. On April 16, fourteen days out, a firmer follow-up that mentions the first one. On April 23, a week out, another. On April 28, two days out, with still nothing marked done, it escalates to the owner, Sam, so a human can make sure it happens. The moment Dana taps *Done*, the system records the filing as handled, rolls the next due date forward to July 30, and goes quiet on this one until the next cycle’s first lead time. Nobody hears about a deadline that is already handled.

The cost of running this is about $1.60 a month at SMB volume. The cost of *not* running it is the late-filing penalty, the lapsed permit that stops a job site, or the payroll that runs a day late because the person who submits it was on holiday.

DESIGN RULES THAT SHAPED EVERY DECISION

- Every reminder ships with full context — deadline name, type, days remaining, link. The owner never has to dig.
- Four moves, always. Clear, first reminder, follow-up, escalate. There is no fifth.
- The tone climbs from gentle to firm. Nothing rude ever goes out, even on the escalation.
- Quiet hours, weekends, and holidays are respected. A reminder is a finite resource; bad timing burns it.
- Marking a cycle done rolls the deadline forward on the next check. The system never reminds about a handled cycle.
- Every send and every owner action is logged. Audit a filing next year and you can see every reminder that went out.

Why this shape

Most teams track recurring deadlines in one of three ways: a person who remembers to do it when they have time, a spreadsheet of “things due” that nobody opens, or a pile of paper notices in a drawer. The person works until they are busy — and the weeks they are busiest are exactly the weeks a quarter-end stacks up. The spreadsheet is a list, not a system: it tells you what is due but does nothing about it. And the drawer is how a business ends up paying a late-filing penalty it could have avoided with one day’s warning.

The setup above keeps the deadline list in a sheet the team already maintains, but adds a small system that *looks at* that list every day and acts only when something is actually coming due. Reminders go out early enough to matter — with more warning for a tax filing than for a payroll run, because the right lead time depends on the type. They are gentle by default and only firmer as a due date nears. They carry the context the owner needs. They escalate to a manager cleanly when it is time. And they roll forward the moment a cycle is handled. The system is invisible on the days nothing is due; it only shows up when an obligation is approaching and someone needs to act.

The next four posts walk through each piece in turn: how a deadline gets on the calendar, how a deadline comes due, how a reminder reaches its owner, and how a deadline gets marked done. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 19, 2026 PART 2 OF 7 · DEADLINE REMINDER SERIES ~4 MIN READ

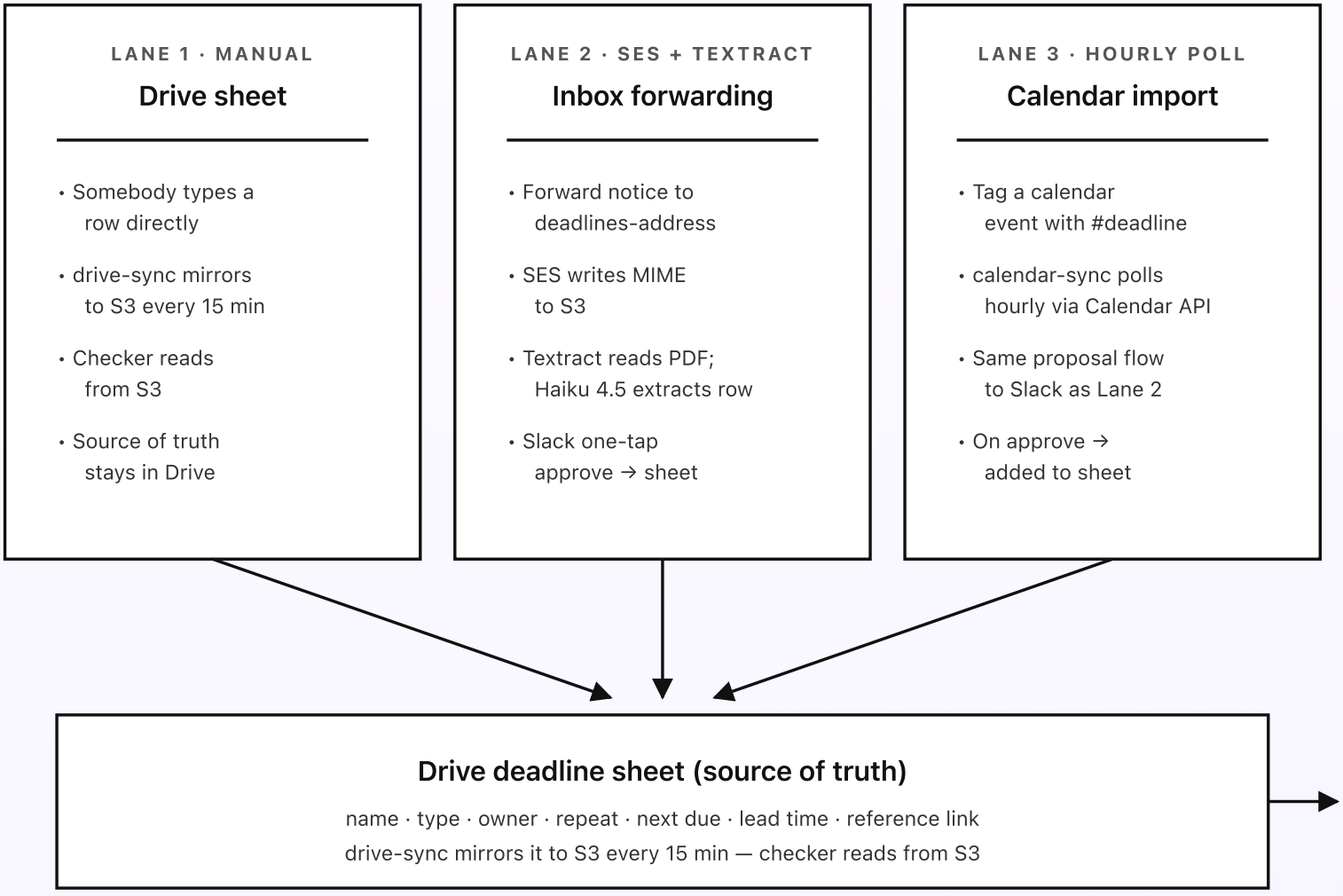
How a deadline gets on the calendar

The system only reminds you about what's on the calendar. So the first job is making sure the calendar actually reflects every recurring obligation your business has. There are three ways a deadline gets in: somebody types a row in the Drive sheet, somebody forwards a notice to a dedicated address, or somebody puts an event on their Google Calendar with a small tag. The first one is obvious. The other two exist because in real life nobody types a row in a sheet for the renewal notice that just arrived in the mail.

KEY TAKEAWAYS

- Three intake lanes feed one calendar: the Drive sheet, an inbox-forwarding lane, and a calendar import.
- Inbound notices are parsed by Textract; Bedrock Haiku 4.5 reads the text and proposes a row.
- Every parsed row goes to the owner's Slack for one-tap approval before it lands in the calendar.
- Calendar events tagged `#deadline` get pulled hourly via the Google Calendar API.
- The Drive sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one calendar



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

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

Lane 1: the Drive sheet itself

The simplest lane. Open the deadline sheet in Drive, add a row, save. The columns are short: name, type, owner email, repeat interval (monthly, quarterly, yearly, or a custom pattern), next due date, lead time, and a link to any reference. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://dr-calendar-source/calendar.csv` if the sheet has changed since the last sync. The checker reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so a bad bulk-edit can be rolled back in one click.

This lane covers the cases where you already know a deadline, you know when it's due and how often it repeats, and you can spend thirty seconds typing it in. Most known obligations — payroll, the quarterly filing, the annual license — go in this way during the initial setup.

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

Set up a dedicated inbound address — something like `deadlines@your-company.com` — via Amazon SES. Anyone on the team forwards a renewal notice, a tax letter, or a compliance reminder to that address and the system takes it from there. SES writes the raw MIME to `s3://dr-raw-mime/`. The S3 PUT triggers a parser Lambda. The Lambda walks the MIME tree to the PDF attachment, runs Amazon Textract on it (Textract reads PDF, PNG, JPEG, and TIFF natively; if

somebody forwards a Word document, the parser falls back to `python-docx`), and gets back the extracted text plus any tables.

Then a Bedrock Haiku 4.5 call reads the text and emits a structured row: name, type, next due date, repeat interval (if the notice states one), and an owner-suggestion based on the "To" line of the original forward. The model prompt is short: "Extract a recurring-deadline row. Return JSON only. Mark each field with a confidence score. Do not invent a date that isn't in the text." The output goes to a small Slack interactive message that pings the person who forwarded the email: the proposed row, the confidence per field, and three buttons — *approve*, *edit*, *discard*. On *approve*, a Lambda writes the row to the Drive sheet via the Sheets API. On *edit*, the owner gets a fillable modal pre-populated with the proposal. On *discard*, the message is logged and the PDF moved to a discarded prefix in S3 for audit.

The reason every parsed row goes to a human first is simple: a due date the model misread is worse than a deadline that never made it onto the calendar at all. The misread one will quietly tell you everything is fine until the morning the filing is actually late.

Lane 3: calendar import

Some teams already track obligations on a calendar. The annual report is on the office manager's calendar. The quarterly safety inspection is on the operations calendar. The franchise-fee due date is on the owner's calendar. Forcing those teams to also type rows in a sheet is a fight you don't need to have on day one.

Lane 3 picks up calendar events tagged with `#deadline` in the description. A small `calendar-sync` Lambda runs hourly, iterates through the configured

Google Calendars (using a service-account credential stored in Secrets Manager), and pulls any events with the tag whose start time is in the future. Each pulled event becomes a proposal in the same Slack flow as Lane 2 — one-tap approve to add to the calendar, with the repeat interval read from the event's own recurrence rule if it has one. Once approved, the calendar event itself can stay where it is or be deleted; the deadline sheet now owns it.

Calendar import is the most opt-in of the three lanes. A team that doesn't use it loses nothing; a team that does avoids retyping things they already typed once.

Why the sheet stays the source of truth

Three lanes in, but only one place where the checker actually looks. That's a deliberate constraint. If two lanes both wrote directly to the checker'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 deadline, and any team member can read or edit any of it without learning a new tool. The convenience lanes are first-class for getting deadlines in, but they always pass through the sheet on the way.

Next post: how the checker actually reads the calendar, computes days-to-due, and picks one of four moves.

PART 3 OF 7

MAY 19, 2026 PART 3 OF 7 · DEADLINE REMINDER SERIES ~5 MIN READ

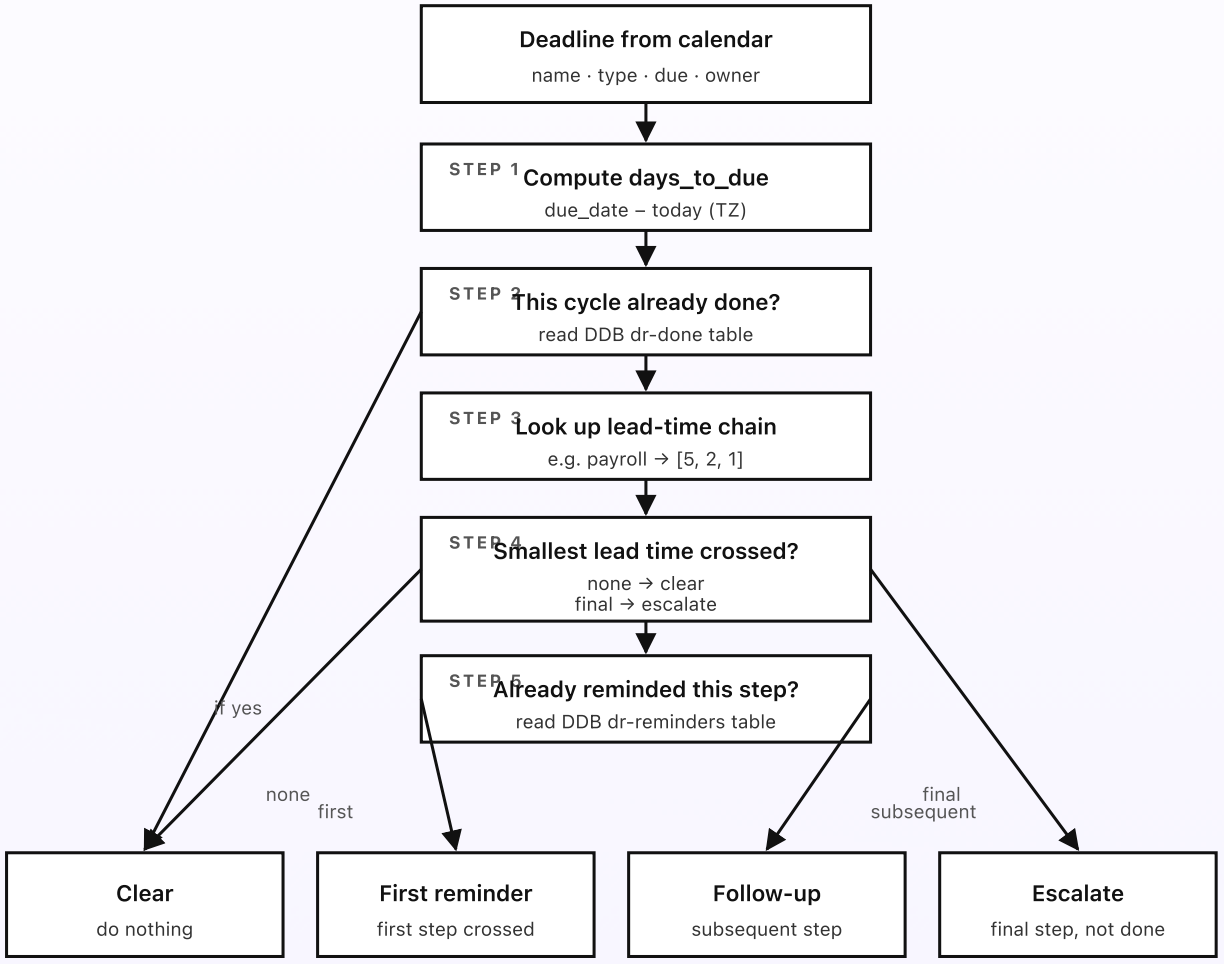
How a deadline comes due

Once a day, at 8am local time, an EventBridge Scheduler rule fires the checker Lambda. The Lambda reads the calendar, looks at one row at a time, computes the days remaining until the due date, 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 a team member can edit it without a deploy.

KEY TAKEAWAYS

- The checker runs once a day via EventBridge Scheduler at 8am local time.
- Per-type lead-time chains live in the rules doc — payroll gets 5/2/1, tax filings get 30/14/7/2, license renewals get 60/30/14/7.
- Four moves per deadline, every check: clear, first reminder, follow-up, escalate.
- DynamoDB tracks last-reminder and done state per deadline so reminders aren't duplicate spam.
- The checker itself never calls a model. The decision is entirely deterministic.

The decision flow, per deadline



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

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

Lead-time chains: 5/2/1 isn't magic, it's in the doc

The rules doc has one short section per type. Each section names the chain in plain prose: "Payroll: remind at 5, 2, and 1 days. Tax filings: 30, 14, 7, 2. License and permit renewals: 60, 30, 14, 7. Insurance payments: 21, 7, 2." The numbers are days remaining when the reminder fires. The first number is the first reminder. The last number is the escalation point — if the deadline hasn't been marked done by then, the escalation target gets reminded too.

The chains exist for a reason. A 30-day tax-filing reminder gives the bookkeeper time to pull the numbers together. A 60-day license-renewal reminder leaves room for a slow government office. A 5-day payroll reminder is the "don't forget to submit this" nudge before the cutoff, and the 1-day step is the last-chance ping before payday is at risk. Different obligations have different mechanics; the chains reflect that.

Per-deadline overrides exist too. The calendar sheet has an optional column called `chain_override`. Type a comma-separated list of days there and the checker uses your numbers instead of the type default for that one row. This is the right escape hatch for the annual report you know needs a 90-day head start because it has to go to the board first.

Four moves, always

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

- **Clear.** The due date is more than the first lead time away, or this cycle has been marked done. Do nothing. Most deadlines, most days, are clear.
- **First reminder.** The due date just crossed the first lead-time step and this cycle isn't done yet. Send a fresh reminder with full context. Write a row to the `dr-reminders` DynamoDB table marking that the first step has fired.
- **Follow-up.** A subsequent step crossed without the cycle 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 `dr-reminders`.
- **Escalate.** The final step in the chain crossed without the cycle being done. Remind the escalation target named in the rules doc — usually the owner's manager — in addition to the owner. Mark the deadline as escalated in DynamoDB; the next check will keep escalating daily until somebody marks it done. Bad timing burns reminders; an escalated deadline is one of the few cases where daily noise is the right answer.

State that makes the decision deterministic

The checker reads two DynamoDB tables every check. `dr-reminders` records every reminder that's gone out: `(deadline_id, step_index, sent_date, dispatched_via)`. `dr-done` records every completion: `(deadline_id, cycle_due_date, done_date, by_user)`. With those two tables, the move-decision logic is a few dozen lines of Python and zero magic. A given deadline with

a given due date, a given lead-time chain, and a given done/reminder history always produces the same move. Re-running the check produces no extra reminders (because the state in DDB shows what already fired).

Marking a cycle done is an explicit reset of both tables for that deadline: rows for the old cycle are kept for audit, the due date rolls forward by the repeat interval, and a fresh chain starts against the new due date. Part 5 covers the completion flow in detail.

Why the daily check uses no model

The checker could call a model on the check to write a smarter reminder message, or to decide whether to remind at all. It doesn't. Two reasons. First, the daily check should be the one part of the system that is utterly predictable — if the rules doc says remind at 14 days and the cycle isn't done, the reminder fires. A model in that loop introduces variance the team can't reason about. Second, model calls cost money, and most days most deadlines are clear, so the call would be wasted nine days out of ten.

Bedrock fires elsewhere — on the inbox parsing lane in Part 2, and on the monthly summary mentioned in Part 6. Not on the daily check. The checker itself is plain Python that reads a doc and writes events.

Next post: how a reminder finds the right owner, how quiet hours and holidays are honored, and what marking a deadline done actually does to the chain.

PART 4 OF 7

MAY 19, 2026 PART 4 OF 7 · DEADLINE REMINDER SERIES ~5 MIN READ

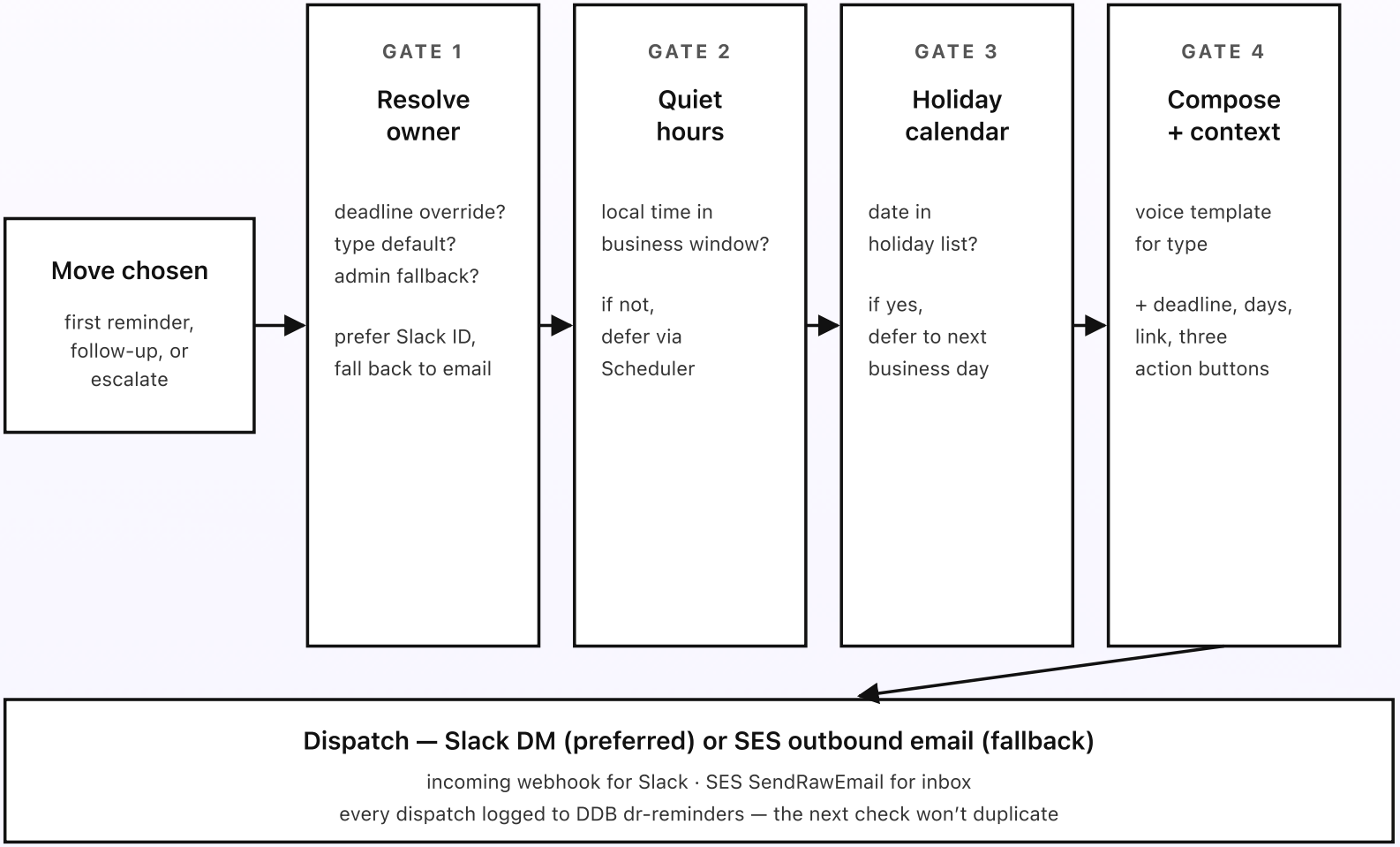
How a deadline reminder reaches its owner

The checker picked a move — first reminder, follow-up, or escalate. Now the dispatch 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 changed roles three months ago. Four small guardrails sit between the move and the actual reminder.

KEY TAKEAWAYS

- Owner resolution: per-deadline override beats per-type 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 deadline, type, days remaining, link, and the Done / Snooze / Ack-only buttons.
- 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 check doesn't duplicate.

Gate 1: resolve the owner

Three places the dispatch Lambda looks for the owner of a deadline, in order. First, the calendar sheet's per-deadline `owner_email` column — if a row has a specific person assigned, that person owns it regardless of the type default. Second, the per-type default in the rules doc ("all tax filings default to the bookkeeper"). Third, the configured admin fallback — the person who set up the system and gets every unowned reminder. The fallback should never fire in steady state; if it does, the weekly digest names every deadline 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 action buttons is more useful than an email link. Email is the fallback so nobody falls through the cracks.

Gate 2: quiet hours

The checker itself runs at 8am local time, so the first time a move fires it's already in business hours. But follow-ups and escalations that result from a check can fire later in the day. And one-off computed dispatches (a second-step reminder that takes effect at the same time as the first) 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 system 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. One exception: when a deadline is one or two days out, the holiday gate yields, because deferring a last-chance payroll reminder past a holiday could push it past the due date itself.

Gate 4: compose with full context, then ship

The voice doc has one Slack message template per type: a short message with placeholders for the deadline name, type, days remaining, and link to any reference. The dispatch Lambda fills the placeholders, attaches the three action

buttons (Done, Snooze, Ack-only), and ships the message via the Slack incoming webhook. The webhook URL itself lives in Secrets Manager.

For email fallback, the same template is wrapped in a small HTML email with the same fields and links that, when clicked, hit a Function URL that records the action — the email equivalent of the Slack buttons.

An escalate move adds a second recipient: the escalation target named in the rules doc for that type. 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 close the deadline now is, so the manager has the full picture at hand.

Every dispatch — Slack or email, owner or escalate — writes a row to `dr-reminders` in DynamoDB. The next day's check reads that row and knows not to remind the same step again.

Why the guardrails exist

None of these gates are exotic. They're the kind of small care a thoughtful person 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 a deadline gets marked done once an owner has acted — how the system records the completion, rolls the due date forward by the repeat interval, and starts a fresh chain for the next cycle.

PART 5 OF 7

MAY 19, 2026 PART 5 OF 7 · DEADLINE REMINDER SERIES ~5 MIN READ

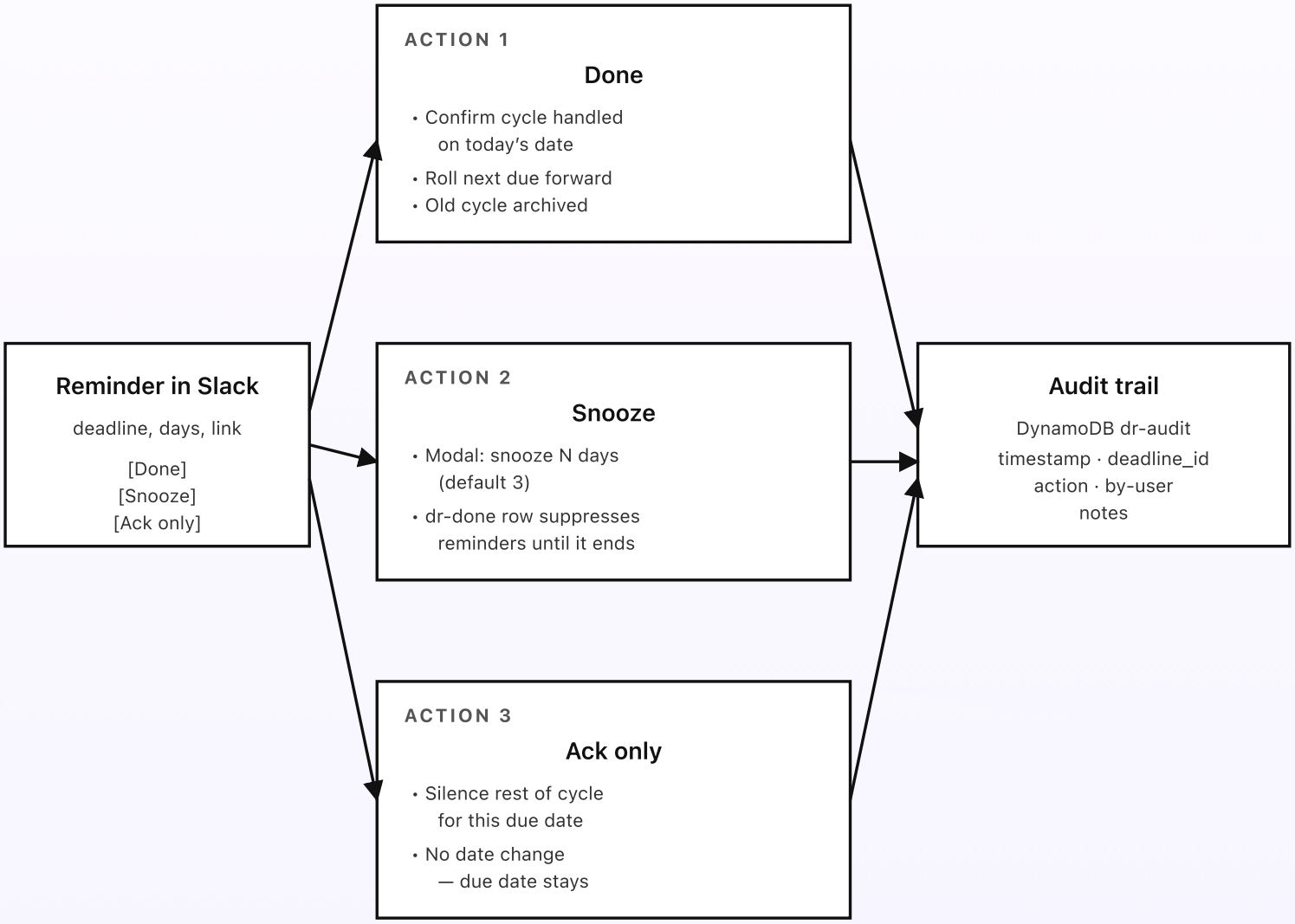
How a deadline gets marked done

A reminder lands in Dana’s Slack DM at 8:03am. The Q1 sales-tax filing is due in 14 days. 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 system can do on a reminder — done, snooze, ack-only — and how the calendar, the chain state, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three actions per reminder: *done* (roll the date forward, fresh chain), *snooze* (delay), *ack-only* (silence this cycle).
- Each action updates the calendar sheet via the Sheets API and writes an audit row.
- Marking done archives the old cycle's chain to a separate sheet for history.
- Snooze is bounded — you can only snooze a few times before the system escalates anyway.
- The Done button is a Slack interactive message backed by a Function URL.

Three actions on Done



Ack only doesn't roll the date — it stops the noise. The next cycle starts on schedule.

Fig 5. Three actions per reminder, three different effects. Done rolls the due date forward and starts a fresh chain. Snooze delays without dismissing. Ack-only silences the cycle without rolling the date. Every action writes to the audit trail.

Action 1: done (the most common)

Dana pulled the numbers, submitted the filing through the state portal, and now needs to tell the system it's handled. She taps *Done*. For most deadlines that's a one-tap confirmation; for a few the system opens a small Slack modal with one field — *Date handled* (defaulting to today, which is right most of the time) — in case she's clearing a cycle she actually finished yesterday.

The tap submits to a Function URL Lambda. Three things happen, in order. First, the system computes the next due date by adding the repeat interval to the current cycle's due date (a monthly payroll rolls forward one month; a quarterly filing rolls forward three; a yearly license rolls forward a year), then the Sheets API updates the row in the calendar sheet with the new `next_due` and a note in the `last_done` column with today's date and the user who acted. Second, the existing cycle's rows in `dr-reminders` are copied to `dr-reminders-archive` with a cycle id, and the live chain is cleared. Third, a `action: done` row is written to `dr-audit` with the user, timestamp, old due date, and new due date.

Tomorrow's check reads the calendar, sees the new due date is more than the first lead time out, and lands at *clear*. The owner won't hear from the system about this deadline again until the new cycle's first step crosses.

Action 2: snooze (the deferral)

Some cycles take time the chain doesn't plan for. The accountant is waiting on one last figure. The signer is travelling. The portal is down for maintenance. Dana isn't ready to mark it done, but she's also handling it — she just needs the system to be quiet for a few 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 `dr_done` with `(deadline_id, snooze_until)`. The next day's check reads that row in the "this cycle already done?" step from Part 3 and treats the deadline as clear until the snooze ends. When the snooze ends, the system re-evaluates the chain from where it was — if the deadline is now in escalation territory, the next reminder is 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 deadline; please mark it done or escalate" reply, and the next check reminds normally regardless. This is a soft constraint that exists because the most dangerous failure mode is repeatedly snoozing a filing past its actual due date.

Action 3: ack-only (the "I've got it")

Sometimes the owner can't mark it done yet and doesn't want to be reminded again either. Maybe the obligation is being handled entirely outside the system this cycle (the accountant is filing on their behalf). Maybe the deadline is being retired. Maybe the owner just wants to confirm they've seen it and will handle it manually.

Ack only writes a row to `dr-done` with `(deadline_id, ack_until: due_date)` — effectively, "don't remind me about this cycle again until its due date passes." The due date itself isn't rolled forward. The chain is silenced for the rest of this cycle. If the cycle ends up actually getting handled via Lane 1 (somebody just edits the next-due date in the calendar sheet directly), the new date resets the chain naturally on the next check.

If the due date passes without the cycle being marked done, the system does one more thing: it writes the deadline to a separate *missed* sheet in the same Drive folder, with the date it lapsed and the last-known owner. The missed sheet is what the monthly summary in Part 6 reports on. A missed deadline that was ack-only'd gets a small note — "ack-only by Dana on 2026-04-15" — so the audit trail isn't a mystery later.

Every action is logged, every action is reversible

The `dr-audit` table records every done, snooze, and ack-only with the user who took the action, the timestamp, and a snapshot of the row before and after. If a wrong roll-forward gets entered (a quarterly deadline marked done a cycle early, say), a team member 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 obligations you'll only think about a few times a year. The next time the annual report comes up, it'll be Dana again or it'll be the person who took her job after she moved on. Either way, the audit trail is 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.

PART 6 OF 7

MAY 19, 2026 PART 6 OF 7 · DEADLINE REMINDER SERIES ~3 MIN READ

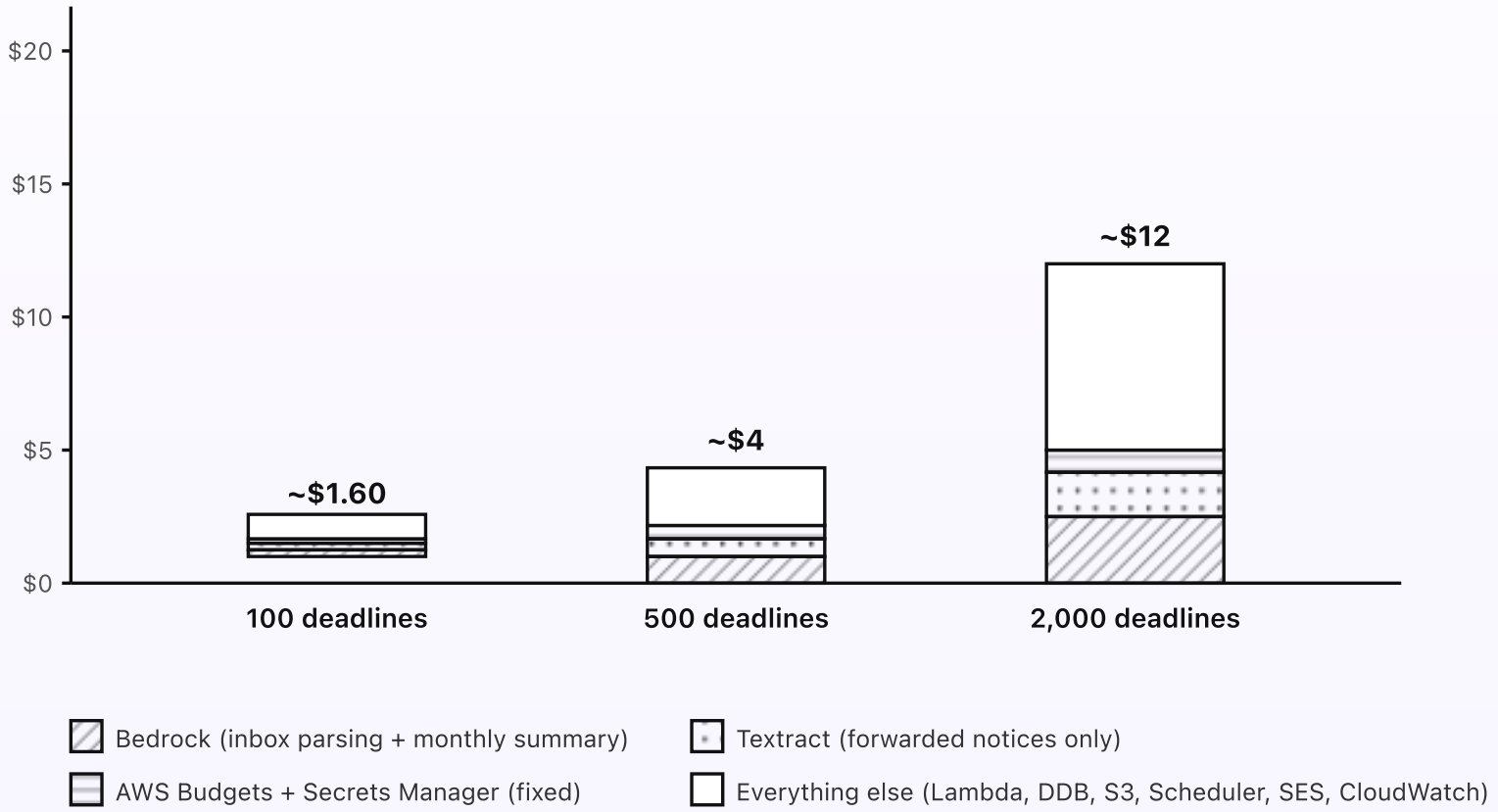
What the deadline reminder costs

The deadline reminder is one of the cheapest systems in this whole series. The daily check reads a CSV from S3, does some date arithmetic, writes a few rows to DynamoDB, and posts a handful of messages to Slack. It calls no models on the check. Bedrock fires only when somebody forwards a notice 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 $1.60/month at typical SMB volume (around 100 recurring deadlines).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily check costs pennies — no model calls.
- Bedrock fires only on inbox notice parsing (a few times a month) and the monthly summary.
- At 500 tracked deadlines the bill is around $4. At 2,000 it's around $12.

Cost at three volumes



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

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

Where the dollars actually go

Lambda runtime (the bulk). The checker runs once a day. Each check reads the calendar CSV from S3, iterates the rows, computes `days_to_due` for each, and decides on a move. At 100 deadlines, that's a few hundred milliseconds. At 2,000 it's a couple of seconds. Either way it's pennies a month. Add the dispatch Lambda firing for each reminder (around five to fifteen reminders a month at 100 deadlines, thirty to sixty at 2,000), the Function URL Lambda for done/snooze/ack-only, the calendar-sync Lambda running hourly, 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: `dr-reminders`, `dr-done`, `dr-audit`. Reads are dominant during the daily check (one read per deadline per check, plus chain history). Writes are dispatch events and audit rows. Pennies a month at any of these volumes.

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

EventBridge Scheduler. The daily check rule plus deferred dispatch rules from the 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 check uses no Bedrock. The inbox parsing lane fires Haiku 4.5 once per forwarded notice: a few thousand input

tokens (the Textract output) and a few hundred output tokens (the proposed row JSON), so a fraction of a cent per parse. At a few forwarded notices a month, Bedrock costs cents. The monthly summary is one larger call: write a paragraph that summarizes the month's reminders, completions, and misses; a couple of cents.

Textract (only on forwarded notices). Per-page pricing; a typical notice is one to four pages. A few cents per parse. At a few notices a month, Textract is a few cents. At 2,000 tracked deadlines with twenty notices forwarded per month, it lands around a dollar.

What doesn't cost money

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

How the cost scales

Lambda runtime grows roughly linearly with deadline count, because every deadline is evaluated on every check. DynamoDB grows linearly too. Bedrock and Textract are uncorrelated with deadline count — they only fire when somebody forwards a notice or it's the first of the month. So the bill at 5,000 tracked deadlines is around $30; at 10,000 it's around $60. Past those volumes the daily-check model probably stops being right (you'd switch to a partial-check that only evaluates deadlines inside the union of all type lead-time windows), but those are optimizations for very large calendars — not redesigns.

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

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

PART 7 OF 7

MAY 19, 2026 PART 7 OF 7 · DEADLINE REMINDER SERIES ~8 MIN READ

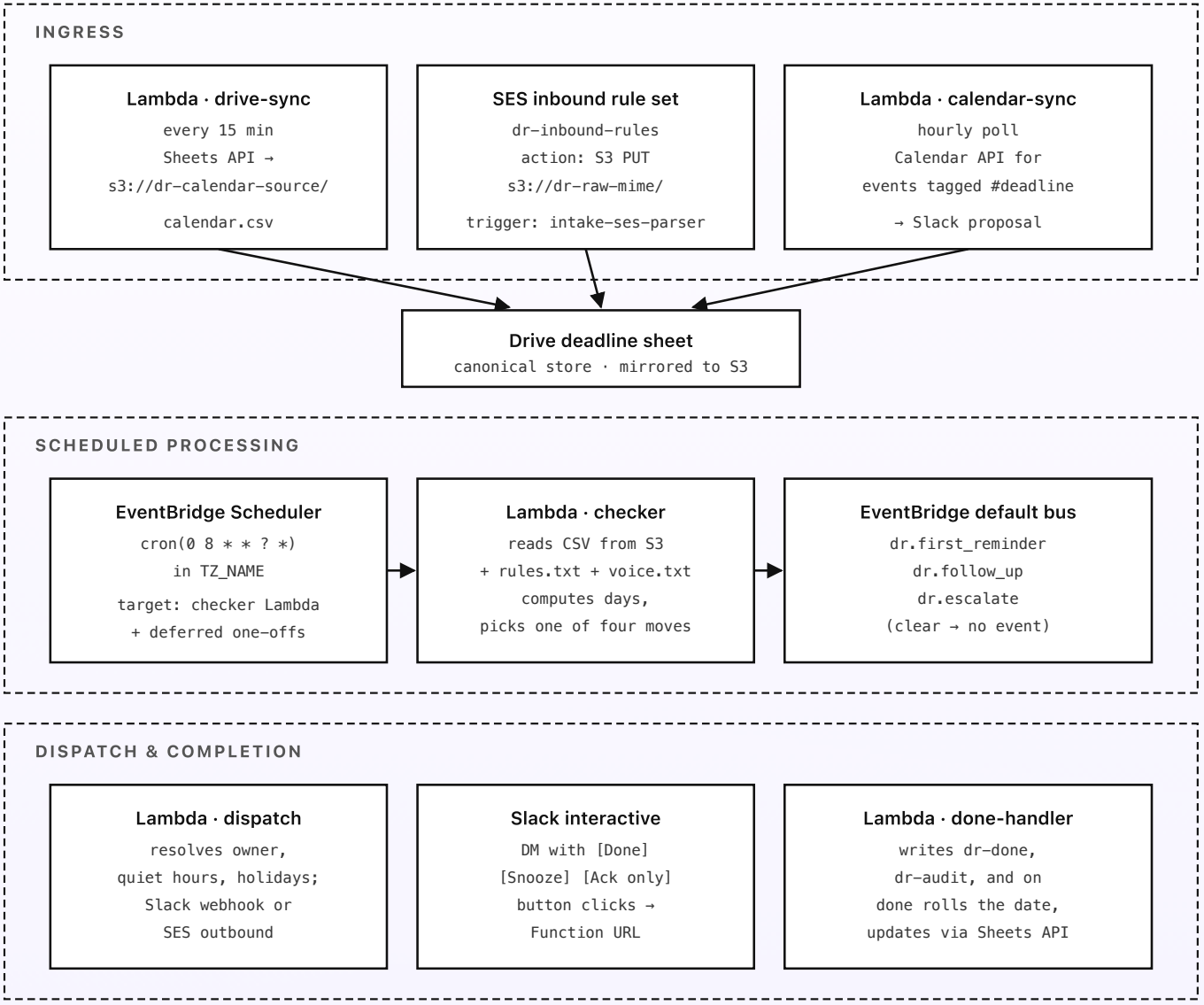
Engineering reference: the deadline reminder architecture

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

Topology



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

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the calendar), scheduled processing (the daily checker 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 `dr/drive/sa`) to export the deadline sheet as CSV and write to `s3://dr-calendar-source/calendar.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://dr-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`calendar-sync`** — EventBridge Scheduler target, hourly. Uses the Google Calendar API `events.list` to scan configured calendars for events with `#deadline` in the description; for any new events, creates a Slack interactive proposal message, reading the recurrence rule for the repeat interval if the event has one. For lower-latency setups you can switch to `events.watch` and have Calendar push notifications to a Function URL instead of polling, at the cost of renewing the channel before it expires (Calendar push channels have a finite TTL and need a small refresh job). Memory: 256 MB. Timeout: 30 s.

- **`intake-ses-parser`** — S3 PUT trigger on `s3://dr-raw-mime/`. Parses MIME, extracts the PDF attachment, runs Textract via `StartDocumentTextDetection` + `StartDocumentAnalysis` (asynchronously to handle multi-page notices). 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 deadline row, including a repeat interval if the notice states one. Posts the proposal to Slack via the incoming webhook 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 notice-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.
- **`checker`** — 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://dr-calendar-source/calendar.csv` and the rules and voice docs. For each row, computes `days_to_due`, reads chain state from `dr-reminders` and `dr-done`, decides on a move. Emits one event per row that needs action: `dr.first_reminder`, `dr.follow_up`, or `dr.escalate`, with the deadline context as the event payload. Clear deadlines 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 incoming webhook (`dr/slack/webhook` 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 `dr-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/Snooze/Ack-only) and by email-link clicks. Writes to `dr-done` and `dr-audit`; on done, rolls the next due date forward by the repeat interval, updates the Drive sheet via the Sheets API, and archives the old cycle in `dr-reminders-archive`. Memory: 256 MB. Timeout: 15 s.
- **`digest`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads `dr-reminders` for the past week and the calendar; sends a digest message to a configured Slack channel summarizing reminders sent and deadlines 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 `dr-reminders`, `dr-done`, and `dr-audit`; calls Bedrock Haiku 4.5 to write a one-paragraph owner narrative (what was handled, what slipped, what is due next month); emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `dr-reminders`** — one row per dispatch. PK `(deadline_id, step_index)`; attributes: `sent_date`, `dispatched_via` (slack/email), `recipient`, `move` (first_reminder/follow_up/escalate). On-demand. No TTL.
- **DynamoDB · `dr-done`** — one row per completion or deferral. PK `deadline_id`; sort key `cycle_due_date`; attributes: `action`

(done/snooze/ack-only), `by_user`, `snooze_until` (if action = snooze), `old_due`, `new_due` (if action = done). On-demand.

- **DynamoDB** · `dr-audit` — one row per write action of any kind. PK `(deadline_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.
- **DynamoDB** · `dr-reminders-archive` — archived chains after a cycle is done. Same shape as `dr-reminders`; PK `(deadline_id, cycle_id, step_index)`. On-demand.
- **S3** · `dr-calendar-source` — mirrored CSV from the Drive deadline sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `dr-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3** · `dr-raw-mime` — raw inbound MIME from forwarded notices. Lifecycle to Glacier at 30 days; expiry at 7 years.
- **S3** · `dr-source-notices` — the parsed source notices after the inbox parser handles them, kept for reference if the deadline row links to one.

Bedrock

- **Foundation model.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0`. Two callsites: `intake-ses-parser` for the inbox notice parsing, and `summary` for the monthly owner narrative. Claude Sonnet 4.6 (`anthropic.claude-sonnet-4-6-20250930-v1:0`) is wired but unused by

default — it's only worth switching the parser to Sonnet if you start forwarding dense multi-page legal notices where Haiku's extraction precision falls short.

- **Embeddings.** Not used. The calendar 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 checker itself doesn't call Bedrock; the parsing lane fires a few times a month at most.

EventBridge Scheduler config

- `dr-daily-check` — `cron(0 8 * * ? *)` in the SMB's timezone. Target: `checker` Lambda.
- `dr-drive-sync` — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- `dr-calendar-sync` — `rate(1 hour)`. Target: `calendar-sync` Lambda.
- `dr-weekly-digest` — `cron(0 18 ? * SUN *)` in TZ. Target: `digest` Lambda.
- `dr-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. `deadlines.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.

- SES inbound rule set `dr-inbound-rules`: one rule with recipient `deadlines@your-company.com` → spam scan → S3 PUT to `s3://dr-raw-mime/<message-id>` → stop. The S3 PUT triggers `intake-ses-parser`.
- SES outbound for the email-fallback reminders: verify a sender identity at `deadlines@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:

- **checker role:** `s3:GetObject` on the calendar, rules, and voice keys; `dynamodb:Query` + `GetItem` on `dr-reminders`, `dr-done`; `events:PutEvents` on the default bus. *No* `bedrock:*`.
- **dispatch role:** `events:ListSchedules` + `CreateSchedule` for the deferred-dispatch one-offs; `secretsmanager:GetSecretValue` on the Slack webhook secret; `ses:SendRawEmail` from the verified sender identity; `dynamodb:PutItem` on `dr-reminders`; outbound network access to `hooks.slack.com`.
- **done-handler role:** `dynamodb:PutItem` on `dr-done` and `dr-audit`; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret; outbound network access to `sheets.googleapis.com`; `dynamodb:Query` for chain state lookup; on done, `dynamodb:BatchWriteItem` for archiving the old cycle to `dr-reminders-archive`.
- **intake-ses-parser role:** `s3:GetObject` on `dr-raw-mime`; `textract:StartDocumentTextDetection` + `StartDocumentAnalysis`;

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

- **drive-sync and calendar-sync roles:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the calendar 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 reminder 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 (`done`, `snooze`, `ack_only`), opens a modal if needed (Snooze opens a modal; Done is one-tap or a small date confirm; Ack-only 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 `dr/slack/bot-token`. The signing secret is `dr/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:** checker Lambda failures > 0 in a day (the daily check 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 `dr-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 `dr/drive/sa` (one service account with scopes for all three APIs). Slack bot token, signing secret, and webhook URL all under `dr/slack/*`. SES sender identity lives in IAM and the verified-domain config. The configured timezone, holiday list reference, quiet-hours window, repeat-interval defaults, and admin fallback owner all live in Parameter Store under `/dr/config/`. Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC into a deploy role (no long-lived keys) and AWS SAM for the stack. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning for both `dr-calendar-source` and `dr-rules-source` so a bad Drive edit can be rolled back in one click, and version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily check in UTC after a CI rotation. A SAM template covers it cleanly; CDK with a Python stack file also fits. 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.