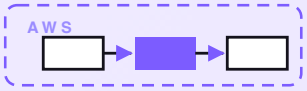
7-PART SERIES · FREE COMPANION



Meeting notetaker

A serverless notetaker that turns a meeting recording into useful notes. You drop in an audio or video file; it transcribes the meeting, writes a short summary, and pulls out the decisions and action items — who owns what, by when. Then it emails everyone a clean recap. It only summarizes what was actually said, and flags unclear owners or dates for a human to confirm. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Meeting notetaker

PART 1 OF 7

MAY 20, 2026 PART 1 OF 7 · MEETING NOTETAKER SERIES ~5 MIN READ

A meeting notetaker on AWS for a few dollars a month

Every team has the same problem after a meeting. Somebody was supposed to take notes and didn't. Or they did, and the notes are three bullet points that mean nothing a week later. Or the decision everyone agreed on never got written down, so it gets re-litigated at the next meeting. Or the action item with your name on it lives only in your memory until the day it's overdue. This post walks through the design of a small notetaker that takes a meeting recording, transcribes it, writes a short summary, pulls out the decisions and the action items — who owns what, by when — and emails everyone a clean recap. It only writes down what was actually said, and it flags anything unclear for a human to confirm.

KEY TAKEAWAYS

- Three sources for recordings: a Drive folder, a direct upload link, and an email-forwarding lane.
- Every meeting runs the same three steps: transcribe, write notes, send a recap after a human confirms.
- The notes are grounded — every decision and action item traces back to a line in the transcript.
- Unclear owners or dates are flagged for the organizer to confirm before the recap goes out.
- Designed on AWS for about $4/month at typical small-business volume.

The whole system on one page

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

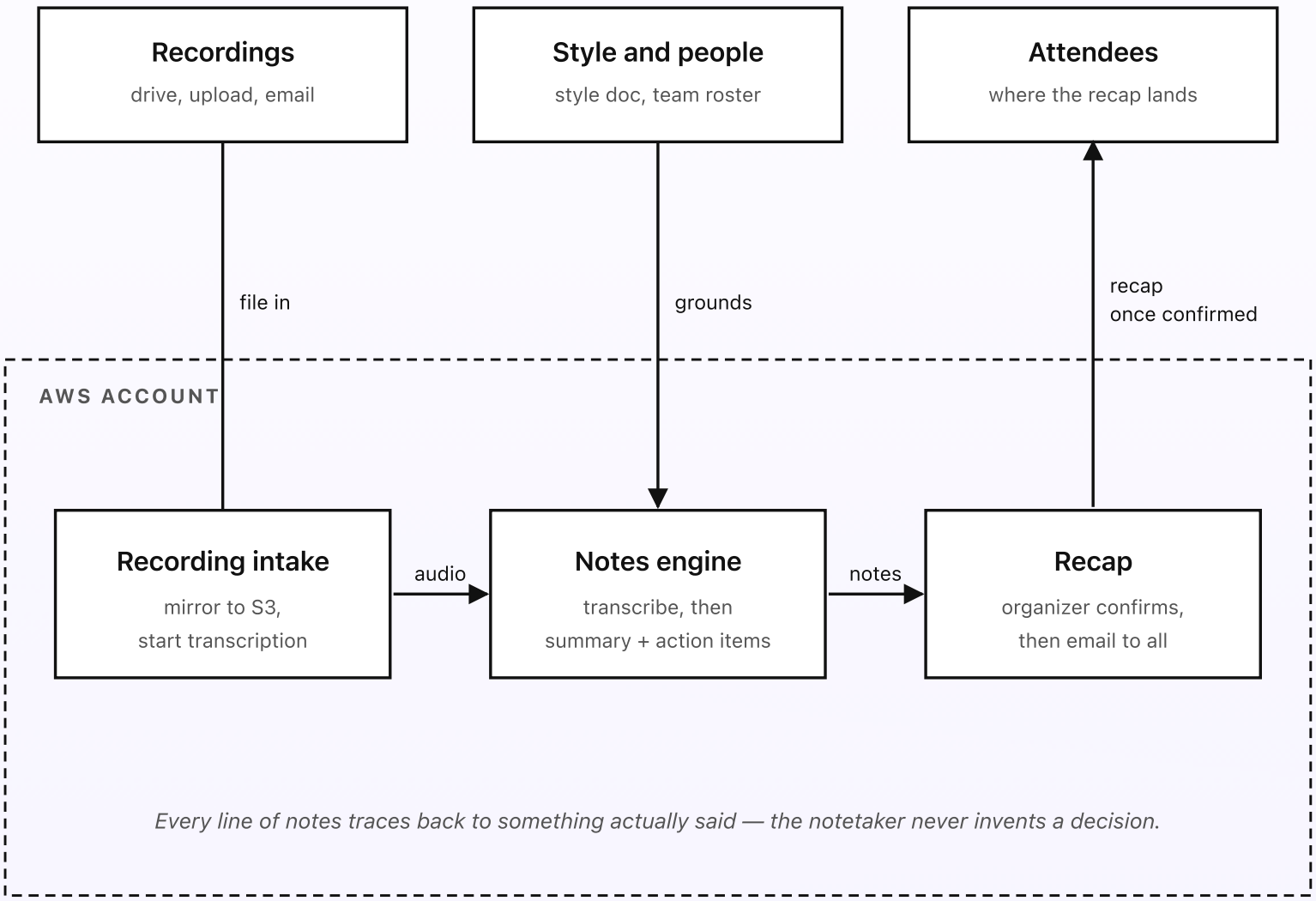

Recordings
drive, upload, email
Style and people
style doc, team roster
Attendees
where the recap lands
file in
grounds
recap
once confirmed
AWS ACCOUNT
Recording intake
mirror to S3,
start transcription
audio
Notes engine
transcribe, then
summary + action items
notes
Recap
organizer confirms,
then email to all
Every line of notes traces back to something actually said — the notetaker never invents a decision.

Fig 1. Three sources outside, three pieces inside AWS. Recordings flow in from a Drive folder, a direct upload link, and an email-forwarding lane. The Notes engine transcribes and then writes grounded notes. Recap sends the clean summary to everyone after the organizer confirms.

What you set up once (the outside)

- **Recordings.** A Google Drive folder you drop meeting files into — audio (M4A, MP3, WAV) or video (MP4, MOV). One file per meeting. Next to each file you can drop a small sidecar text file naming the meeting and listing who was in the room; if you skip it, the notetaker still runs and just asks the organizer to fill in attendees at confirm time. New recordings can also enter via two other lanes covered in Part 2 — a direct upload link (a private page that puts the file straight into S3) and an email-forwarding lane (forward the recording your conferencing tool emails you, and it gets picked up automatically).
- **A style and people folder.** Two short things in a Drive folder. The *style* doc says how the recap should read — how long the summary should be, how action items should be phrased ("Name to do X by date"), and any words to avoid. The *roster* is a short list of the team: name, email, and the way each person tends to be referred to in meetings ("Maria," "M," "the office manager") so the notetaker can match a spoken name to a real email address. Editing either one doesn't need a deploy.
- **Attendees.** The people who were in the meeting. The clean recap email lands in their inboxes — but only after the organizer has confirmed the draft. Each recap has the summary, the list of decisions, and a table of action items with an owner and a due date for each, plus a link back to the transcript moment behind every item.

What runs on every meeting (the inside)

- **The recording intake.** Three sources feed one place. A file dropped in Drive is mirrored to S3 by a small sync Lambda. A file from the upload link goes straight to S3. A forwarded recording is pulled from the email and written to S3 too. However it arrives, landing in S3 kicks off the next step: an Amazon Transcribe job. Transcribe reads the audio (it pulls audio out of video files on its own) and produces a transcript with speaker labels and a timestamp on every line.
- **The notes engine.** Once the transcript is ready, two Bedrock calls run. The first writes a short summary of the meeting — what it was about, what got decided, where things stand. The second pulls out the action items: for each one, what needs doing, who owns it, and by when. Both calls are told to use only the transcript and to cite the line behind every decision and every action item. A long meeting (over about an hour) uses the heavier Claude Sonnet 4.6 model; a normal one uses the cheaper Claude Haiku 4.5. Anything the model is unsure about — a vague owner, a fuzzy date — is marked *needs-confirm* rather than guessed.
- **The recap.** The pipeline emails the meeting organizer a draft recap with every needs-confirm item highlighted. The organizer can approve it as-is, fix an owner or a date inline, or drop an item. Only after they approve does the clean recap go to the whole attendee list via SES. Every run — the transcript, the draft, the final recap, and who confirmed it — is logged in DynamoDB so you can look back later and see exactly what was sent and why.

In plain words

Your Monday planning meeting runs 35 minutes. Six people, one recording. You drop the MP4 in the Drive folder and forget about it. Ten minutes later, Amazon Transcribe has turned it into text with speaker labels. Bedrock reads the transcript and writes: a three-sentence summary, two decisions (“Ship the pricing page Thursday,” “Pause the ad spend until the new landing page is live”), and four action items. Three of the action items are clean — clear owner, clear date. The fourth is fuzzy: somebody said “we should follow up with the vendor soon,” no name, no date. That one gets flagged *needs-confirm*. You, the organizer, get the draft. You assign the vendor follow-up to Maria, set it for Friday, and hit approve. The clean recap lands in all six inboxes two minutes later.

The cost of running this is about $4 a month at SMB volume. The cost of *not* running it is the decision everyone forgets, the action item that slips, and the half-hour each week somebody spends writing notes that still miss the one thing that mattered.

DESIGN RULES THAT SHAPED EVERY DECISION

- Only what was said. Every decision and action item cites the transcript line behind it — nothing is invented.
- Three steps, always. Transcribe, write notes, send a recap. There is no fourth.
- A human confirms before anyone else sees it. The organizer approves the draft; the recap never auto-sends by default.
- Unclear is flagged, not guessed. A vague owner or fuzzy date becomes a needs-confirm item, not a made-up answer.
- The recordings live in Drive. Adding a meeting, changing the roster, or editing the style doesn't need a deploy.
- Every run is logged. Look back next quarter and you can see exactly what recap went out and who confirmed it.

Why this shape

Most teams handle meeting notes in one of three ways: somebody volunteers and does a rushed job, nobody does and the meeting evaporates, or they pay for an always-on bot that joins every call and quietly records things they'd rather it didn't. The volunteer approach fails the moment that person is busy or away. The nobody approach fails every time. And the always-on bot is more system than a small business wants — another vendor in every meeting, another subscription, another thing recording when it shouldn't be.

The setup above is deliberately the opposite of always-on. Nothing joins your calls. You decide which meetings get notes by choosing which recordings to drop in. The notetaker wakes up only when a file appears, does its work in a few minutes, and goes back to sleep. The notes are grounded in the transcript, so they don't drift into things nobody said. And a human signs off before the recap reaches the room, so a misheard owner or a wrong date gets caught before it becomes an email everyone trusts.

The next four posts walk through each piece in turn: how a recording comes in, how the transcript becomes notes, how the notes stay grounded, and how the recap reaches everyone. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 20, 2026 PART 2 OF 7 · MEETING NOTETAKER SERIES ~4 MIN READ

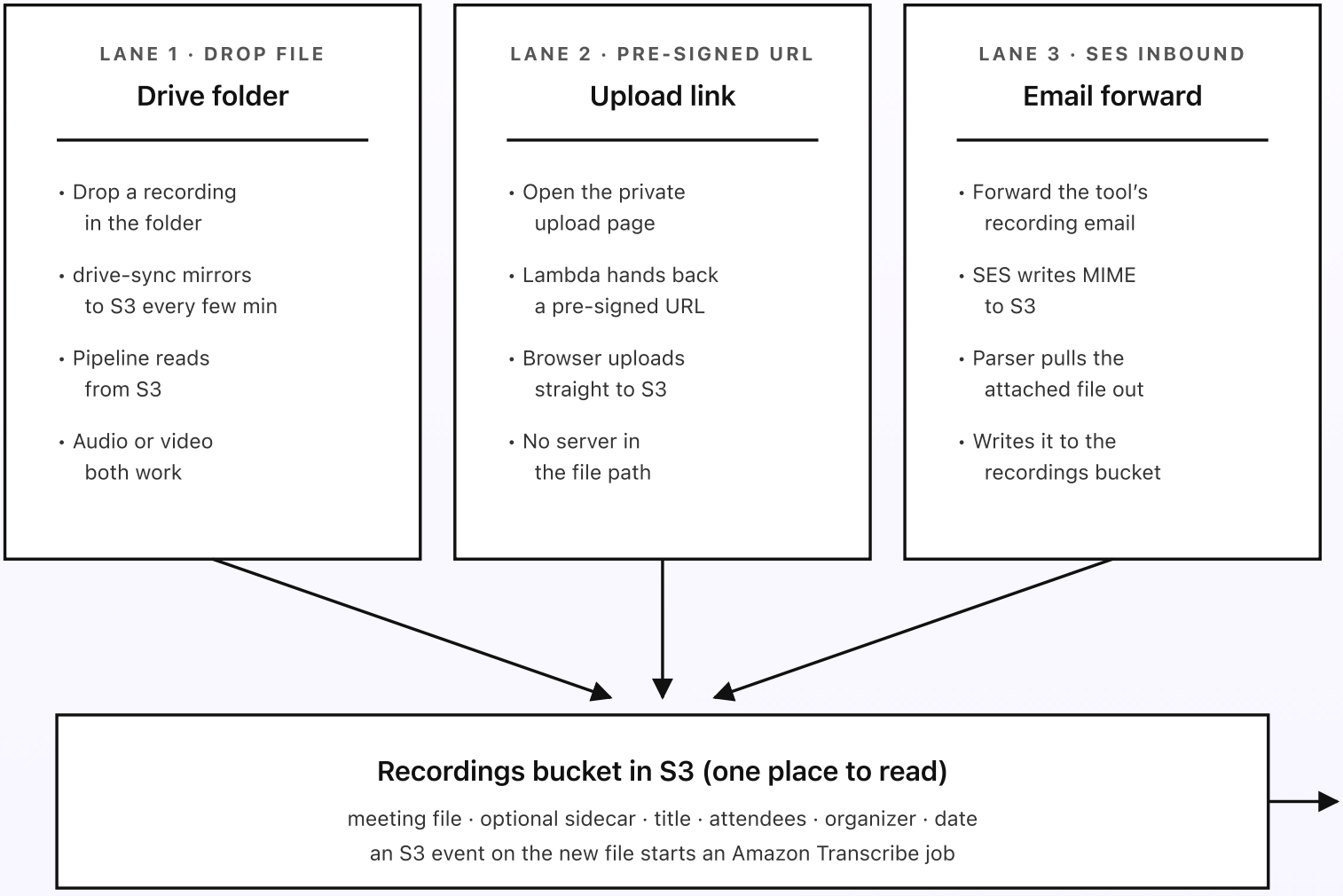
How a meeting recording comes in

The notetaker only works on recordings it actually receives. So the first job is making it easy to hand it a file, no matter where that file starts out. There are three ways a recording gets in: somebody drops it in a Drive folder, somebody uploads it through a private link, or somebody forwards the email their conferencing tool sent with the recording attached. The first one is obvious. The other two exist because in real life the recording is often already in an inbox or on a laptop, and asking people to move it into Drive first is a step they'll skip.

KEY TAKEAWAYS

- Three intake lanes feed one S3 bucket: the Drive folder, a direct upload link, and an email-forwarding lane.
- Audio and video both work — Amazon Transcribe pulls the audio out of video files on its own.
- A small sidecar file names the meeting and lists attendees; if it’s missing, the organizer fills it in later.
- The forwarding lane catches the recording your conferencing tool emails after the call.
- S3 stays the one place the pipeline reads from. The lanes are just three doors into it.

Three lanes into one bucket



S3 stays the one place the pipeline reads from — the three lanes are just doors into it.

Fig 2. Three lanes converge on one S3 bucket. The bucket is the one place the pipeline reads from; the upload link and the email lane are conveniences. A new file in the bucket starts an Amazon Transcribe job, and the pipeline takes it from there.

Lane 1: the Drive folder itself

The simplest lane. Open the recordings folder in Drive, drop in an audio or video file, done. A small Lambda — `drive-sync` — runs every few minutes, lists new files via the Drive API, and copies anything it hasn't seen to `s3://mn-recordings/`. The pipeline reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so an accidental overwrite can be rolled back in one click.

Next to the file you can drop a small sidecar text file with the same name — `monday-planning.txt` next to `monday-planning.mp4` — holding the meeting title, the date, the organizer's email, and who was in the room. If you skip the sidecar, the notetaker still runs; it just asks the organizer to fill in the attendee list when it shows them the draft. The sidecar is a convenience, not a requirement.

Lane 2: the upload link (the lane for files already on a laptop)

Sometimes the recording is just sitting on somebody's machine and Drive isn't open. For that there's a private upload page — a single web page behind a login that lets a team member pick a file and a few details (title, organizer, attendees). When they hit upload, the page asks a small Lambda for a *pre-signed URL* — a one-time, time-limited web address that lets the browser write directly to S3. The file goes straight from the laptop to the bucket; it never passes through a server

we run, which keeps things cheap and avoids any file-size limits a server would impose.

The same page writes the title and attendees into the sidecar file alongside the recording, so the notes engine has what it needs from the start. This is the lane most useful for one-off meetings and for anyone who finds dragging a file into a web page easier than syncing Drive.

Lane 3: email forward

Most conferencing tools email you a link or an attachment after a call ends. Lane 3 catches that. Set up a dedicated inbound address — something like `notes@your-company.com` — via Amazon SES. Forward the tool's recording email to that address and the notetaker takes it from there. SES writes the raw message to `s3://mn-raw-mime/`. The S3 write triggers a small parser Lambda that walks the message, finds the audio or video attachment, and copies it to the recordings bucket. If the email contains a download link instead of an attachment, the parser follows the link and pulls the file down.

The forwarding lane is the most hands-off of the three: no app to open, no folder to sync. For teams whose tool already emails them the recording, forwarding is one tap, and the notetaker handles the rest.

Why S3 stays the one place the pipeline reads

Three lanes in, but only one place the pipeline actually looks. That's a deliberate constraint. If each lane handed files to the notes engine in its own way, every "why didn't this meeting get notes?" question would mean checking three different

paths. Funneling everything through one S3 bucket means there is exactly one trigger — a new file lands, a Transcribe job starts — no matter how the file arrived. The convenience lanes are first-class for getting recordings in, but they all pass through the same door.

Next post: how the transcript gets made, and how Bedrock turns it into a short summary and a clean list of decisions and action items.

PART 3 OF 7

MAY 20, 2026 PART 3 OF 7 · MEETING NOTETAKER SERIES ~5 MIN READ

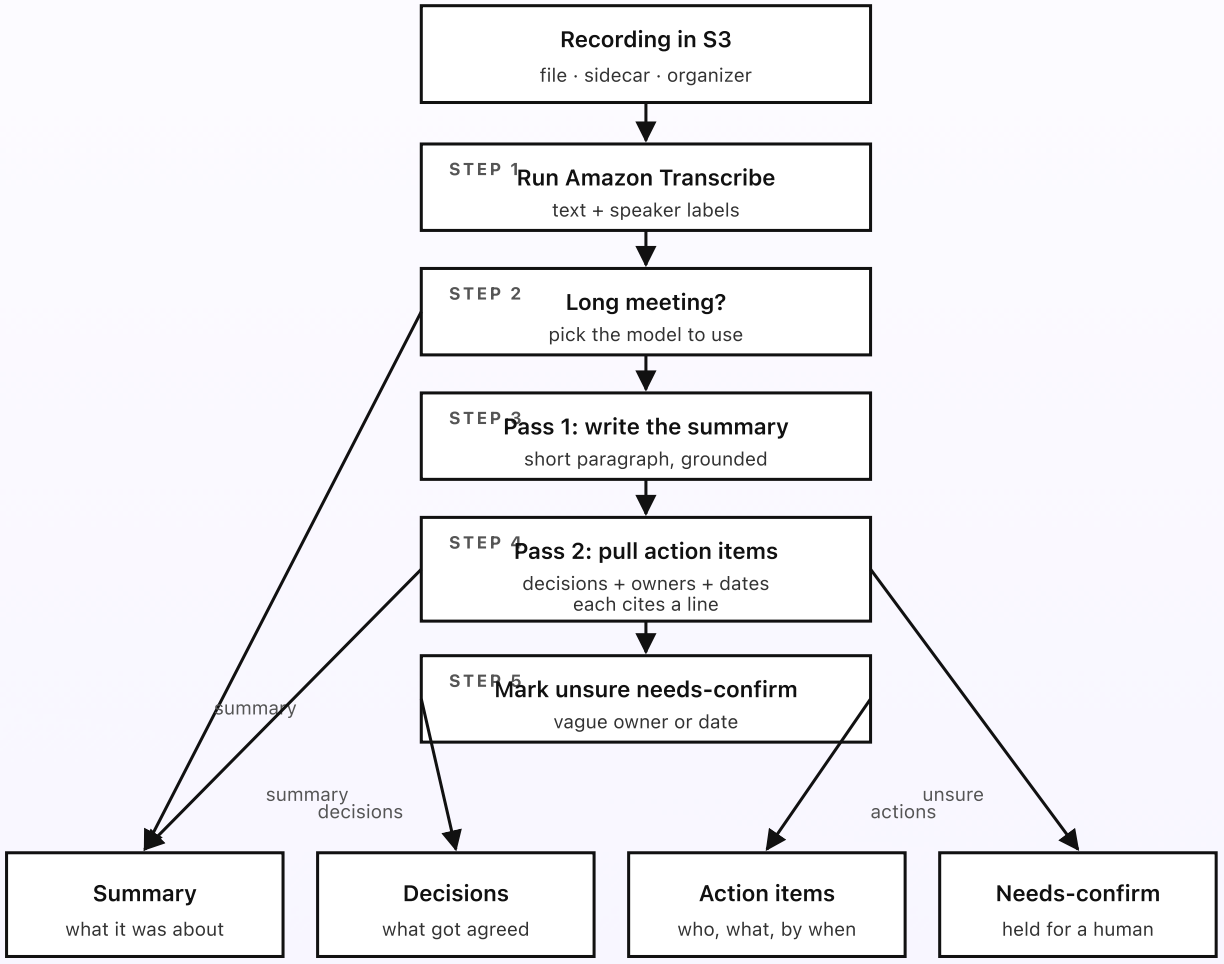
How the transcript becomes notes

A recording lands in S3. From there to a clean recap is a short chain of steps, and only two of them use a model. First Amazon Transcribe turns the audio into text with speaker labels and a timestamp on every line. Then one Bedrock call writes a short summary, and a second pulls out the decisions and action items. Both model calls are told to use only the transcript and to cite the line behind every claim. Nothing here invents; the model's whole job is to read what was said and lay it out clearly.

KEY TAKEAWAYS

- Amazon Transcribe makes the transcript — text with speaker labels and a timestamp per line.
- Two Bedrock passes: one writes the summary, one pulls out decisions and action items.
- Both passes cite the transcript line behind every claim, so nothing is invented.
- Long meetings (over ~an hour) use Claude Sonnet 4.6; normal ones use the cheaper Haiku 4.5.
- The transcript and the draft notes are saved to S3, so a run can be re-checked later.

The notes flow, per meeting



Both passes read only the transcript and cite the line behind every claim — nothing goes beyond what was said.

Fig 3. From recording to draft notes, per meeting. Transcribe makes the text; two grounded Bedrock passes write the summary and pull the action items; unsure items are flagged. The four results together form the draft saved to S3.

Transcription: text with speaker labels and timestamps

When a new file lands in `s3://mn-recordings/`, a small Lambda starts an Amazon Transcribe job. Transcribe reads the audio — and pulls audio out of video files on its own, so an MP4 works exactly like an M4A. It turns on speaker labels (Transcribe calls this speaker diarization — the feature that figures out who said what and tags each line), so the transcript reads like a script: `Speaker 1: ...`, `Speaker 2: ...`, each with a timestamp. If the sidecar named the attendees in speaking order, a small step maps `Speaker 1` to a real name; if not, the names stay generic until the organizer fixes them at confirm time.

Transcribe writes its output JSON back to S3. A short Lambda flattens it into a plain transcript — one line per turn, each line carrying its timestamp — and saves that too. The flat transcript is what the model reads, and the timestamps are what let every action item link back to the exact moment it was said.

Two grounded passes, not one

It would be possible to ask the model for everything in a single call. The notetaker doesn't, for a simple reason: a summary and an action-item list are different jobs, and splitting them keeps each prompt short and each output easy to check.

Pass 1 writes the summary. The prompt is short: “Here is the transcript. Write a summary of at most N sentences in the style described. Use only what is in the transcript. Do not add background the speakers didn’t mention.” The style and length come from the style doc, so the team controls the tone without touching code.

Pass 2 pulls the action items. The prompt asks for a list of decisions and a list of action items in a fixed shape: for each item, the task, the owner, the due date, and the transcript line behind it. It’s told explicitly: “Return JSON only. For each action item include the verbatim transcript line it came from and its timestamp. If the owner or the date wasn’t said out loud, set it to needs-confirm — do not guess.” That last instruction is the whole game, and Part 4 is about how it’s enforced.

Choosing the model by meeting length

A 20-minute standup and a two-hour strategy offsite are different problems. The short one is easy: Claude Haiku 4.5 reads it, writes a tidy summary, and pulls a handful of action items, for a fraction of a cent. The long one has more threads to follow, more half-decisions, and more chances to confuse who owns what — so for transcripts over about an hour, the pipeline switches to Claude Sonnet 4.6, which reasons more carefully across a long conversation. The cutoff is a number in the style doc; most meetings fall under it and use the cheap model.

Both models run on Bedrock through Global cross-Region inference, which spreads the call across regions for capacity without you managing any of that. The notetaker never holds a model open or runs one on a schedule — it calls Bedrock twice per meeting and that’s it.

What gets saved, and why

Three things land in S3 for every meeting: the Transcribe output, the flat transcript, and the draft notes (summary, decisions, action items, needs-confirm). All three are versioned. If a recap later looks wrong — an owner that shouldn't have been on an item, a date that reads oddly — you can open the transcript and the draft and see exactly what the model had in front of it. Because every action item carries the transcript line and timestamp behind it, checking a claim is a matter of reading one line, not re-listening to the whole meeting.

Next post: the four checks that keep the notes honest — how cite-or-drop, owner resolution, date sanity, and the needs-confirm queue make sure nothing invented ever reaches the recap.

PART 4 OF 7

MAY 20, 2026 PART 4 OF 7 · MEETING NOTETAKER SERIES ~5 MIN READ

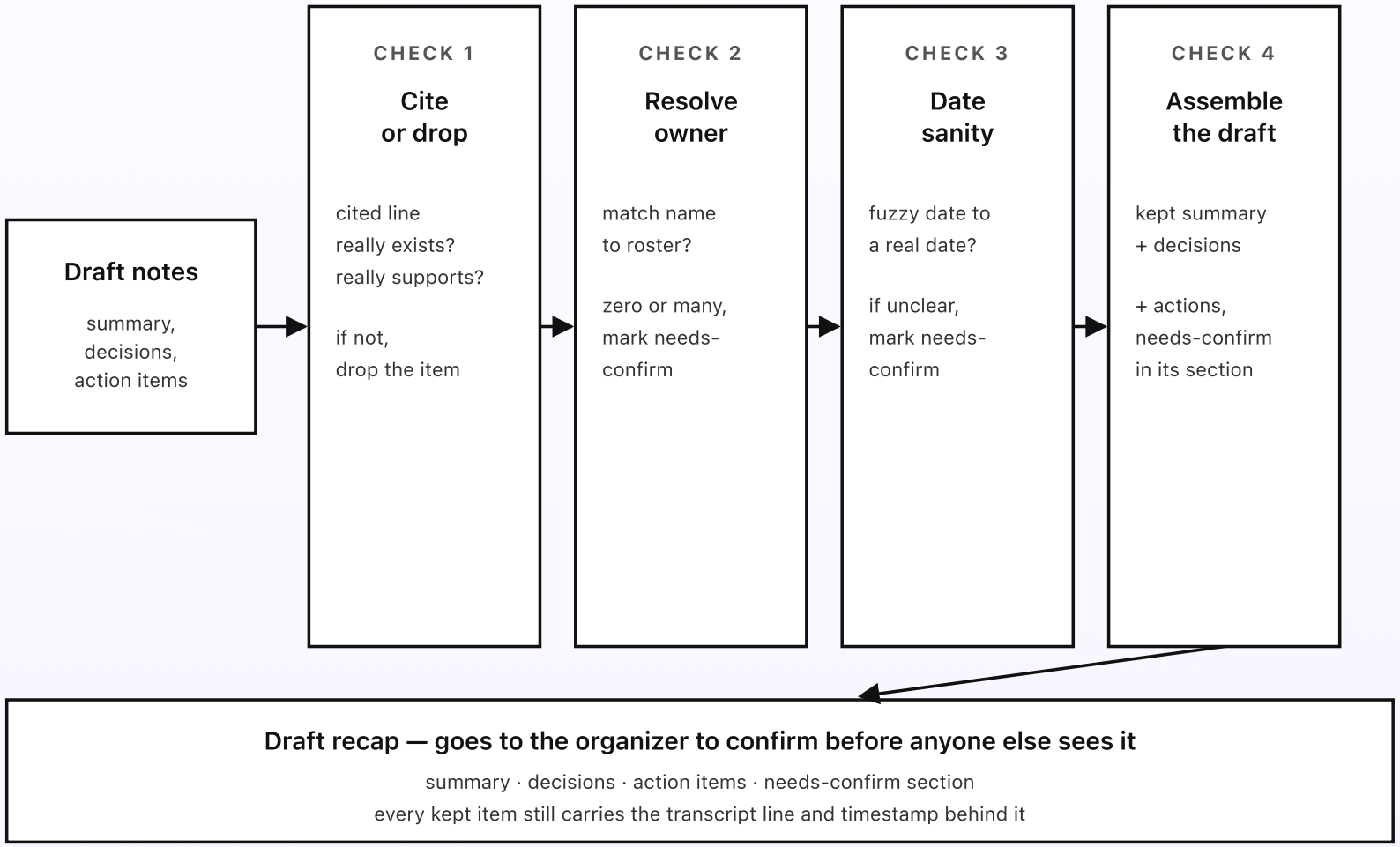
How the notes stay grounded

The model wrote a draft — a summary, some decisions, a list of action items. Now the question that matters: can you trust it? A wrong owner on an action item, a date that was never said, a decision that sounds firmer than it was — any of those, in an email everyone reads, is worse than no notes at all. So the draft never goes straight to the recap. Four small checks sit between the model's output and the recap, and each one's job is to catch a different way notes can go wrong.

KEY TAKEAWAYS

- Cite-or-drop: every decision and action item must point to a real transcript line, or it’s removed.
- Owner resolution: a spoken name is matched to a real person from the roster, or flagged.
- Date sanity: a fuzzy date (“next Friday”) becomes a real date, or it’s flagged.
- Needs-confirm queue: anything the checks can’t settle is held for the organizer.
- Every check is plain code, not another model — predictable, and free.

Four checks on every draft



Every check is plain code, not another model — nothing the transcript didn't support survives.

Fig 4. Four checks between the model's draft and the recap. Confirm every citation. Resolve every owner. Pin down every date. Assemble what survives, with the unsure items collected for the organizer. Then on to the recap step.

Check 1: cite-or-drop

In Part 3 the model was told to attach the transcript line behind every decision and action item. This check holds it to that. For each item, the code looks up the cited line in the saved transcript and confirms two things: that the line actually exists (the model didn't invent a quote), and that the line plausibly supports the claim (the words line up). If an item cites a line that isn't there, or cites a line that doesn't say what the item claims, the item is dropped — not kept, not guessed at. Better to miss one real action item the organizer can add by hand than to ship one the meeting never produced.

This is the single most important check, because it's the one that catches a model that's drifted into making things up. Everything that survives this gate is anchored to a real moment in the recording.

Check 2: resolve the owner

An action item with no owner is a wish. So each action item carries a spoken name — "Maria," "Sam," "the design team" — and this check tries to turn that into a real person. It compares the spoken name against the roster (name, email, and the nicknames each person goes by). One clear match: attach that person's email and move on. Zero matches, or several plausible ones (two people named Sam): the owner is marked *needs-confirm* and left for the organizer.

The roster is deliberately the only source for owners. The check never invents an email address or assigns an item to whoever was loudest. If the meeting genuinely didn't name an owner, the right answer is "a human decides," not a guess.

Check 3: date sanity

People say dates loosely. "By next Friday," "end of the month," "before the launch," "in a couple of weeks." This check turns the ones it can into real calendar dates, measured from the meeting date — "next Friday" in a meeting on the 22nd becomes a specific date. The conversions are plain rules, not a model: a small table maps common phrases to offsets. Anything the rules can't pin down — "before the launch" when no launch date was set, "soon," "sometime" — is marked *needs-confirm* with the original phrase shown, so the organizer can set a real date.

The reason for being strict here is the same as everywhere else: a confidently wrong date is more dangerous than an obviously missing one. "Due: needs-confirm" prompts a fix; "Due: May 29" when nobody said May 29 quietly becomes a deadline somebody trusts.

Check 4: assemble, with the unsure items collected

The last step builds the draft recap from what survived: the summary, the kept decisions, and the action items that have a real owner and a real date. Everything that got marked *needs-confirm* along the way — a dropped citation worth a second look, an unresolved owner, an unpinned date — is gathered into its own section at the top of the draft, so the organizer sees exactly what needs a human

before anything else. Every kept item still carries its transcript line and timestamp, so the organizer can check any of them in one read.

None of these four checks call a model. They're plain Python: look up a line, match a name against a list, convert a date phrase, sort items into buckets. That's on purpose. The model's job is to read the meeting; the checks' job is to be utterly predictable about what's allowed through. A predictable gate you can reason about beats a smarter gate you can't.

Why grounding is the whole point

The reason people don't trust automatic meeting notes is that they've all seen the bad version: a confident recap with an action item nobody remembers agreeing to, assigned to somebody who wasn't even there. One bad recap and the team stops reading them. These four checks exist so that every line in the recap can be traced back to the recording, and so that the things the system isn't sure about are shown as questions, not stated as facts. The notetaker would rather hand the organizer three clean items and two questions than five items where one is quietly wrong.

Next post: how the recap reaches everyone — how the organizer confirms the draft, fixes the needs-confirm items, and sends the clean version to the whole room.

PART 5 OF 7

MAY 20, 2026 PART 5 OF 7 · MEETING NOTETAKER SERIES ~5 MIN READ

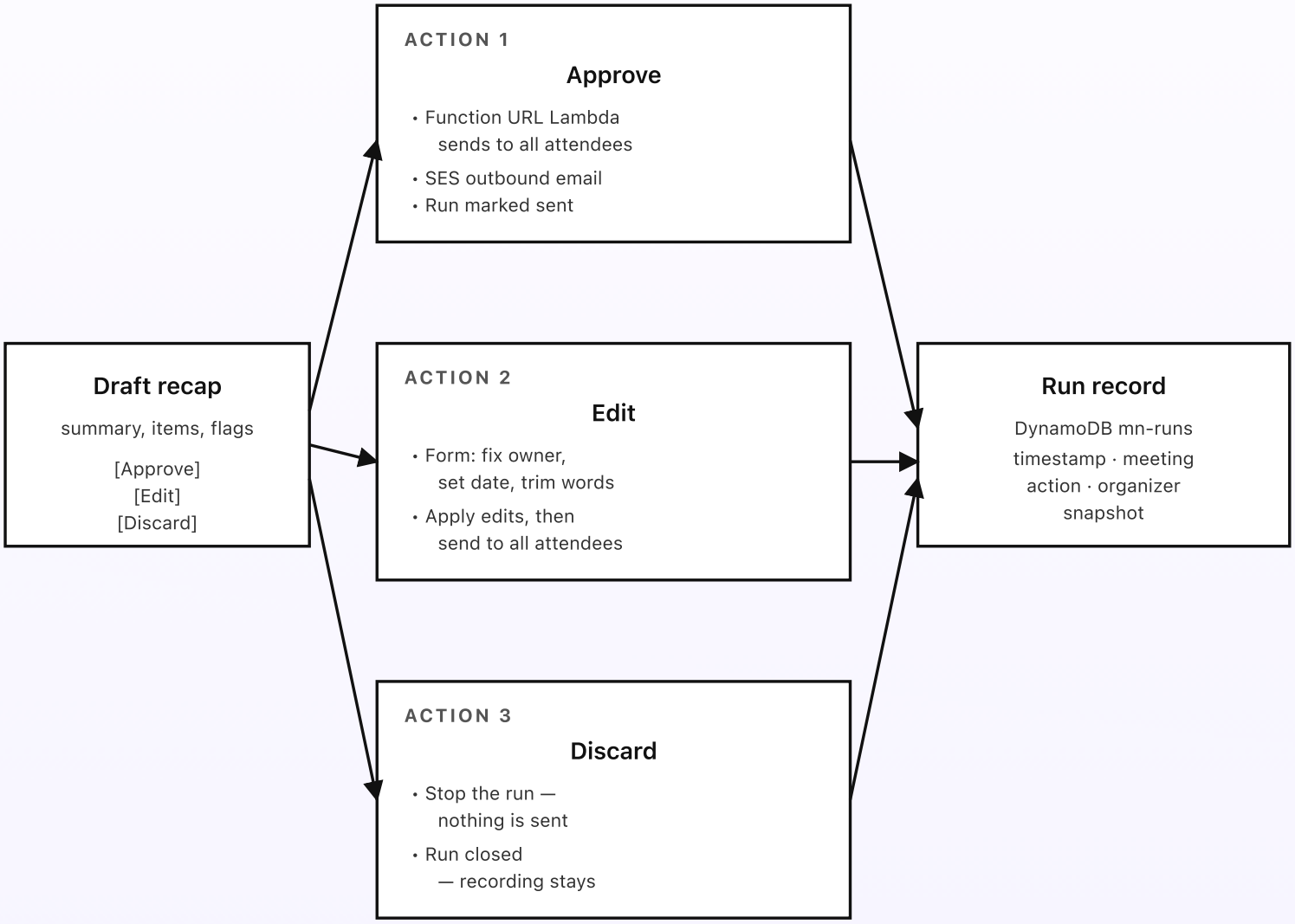
How the recap reaches everyone

The draft is clean and the unsure items are flagged. A recap email lands in the organizer's inbox at 9:12am, a few minutes after the meeting ended. There's a summary, a list of decisions, a table of action items, and a short needs-confirm section at the top. What happens next is the whole reason this system can be trusted: a human looks at it before anyone else does. This post walks through the three things the organizer can do — approve, edit, discard — and how the run record and the audit trail stay in sync.

KEY TAKEAWAYS

- Three actions on the draft: *approve* (send to all), *edit* (fix inline, then send), *discard* (stop).
- The clean recap goes to the whole attendee list only after the organizer approves.
- Editing lets the organizer resolve a needs-confirm owner or date before anyone sees it.
- Auto-send is opt-in, per meeting type, and only when the draft has no needs-confirm flags.
- The approve and send buttons are backed by a Function URL; every action is logged.

Three actions on the draft



The recap never auto-sends by default — a human confirms before the room sees it.

Fig 5. Three actions on the draft, three different effects. Approve sends the clean recap to everyone. Edit fixes the flagged items first. Discard stops the run. Every action writes to the run record.

Action 1: approve (the most common)

If the draft is clean — no needs-confirm flags, owners and dates all resolved — the organizer reads it, agrees, and taps *Approve*. The button submits to a Function URL Lambda (a small public web address attached straight to a Lambda, no API Gateway in front of it). The Lambda formats the clean recap as an HTML email, drops the needs-confirm section since there's nothing left in it, and sends it to the whole attendee list via SES outbound. Then it marks the run as `sent` in the `mn-runs` table with the timestamp and the organizer's name.

The attendees get a tidy email: a short summary, the decisions, and a table of action items each with an owner and a due date. Every action item links back to the transcript moment behind it, so anyone who wants to check a line can, in one click.

Action 2: edit (resolve the flags first)

More often, the draft has a flag or two — the vendor follow-up with no owner, the "before the launch" date that never got pinned down. The organizer taps *Edit*. A simple form opens showing each needs-confirm item next to the transcript line behind it, so they have the context without re-listening. They pick an owner from the roster, set a real date, and can also trim wording on any item or shorten the

summary. On save, the Function URL Lambda applies the edits, clears the flags, and sends the clean recap to everyone — the same send path as approve.

This is the step that makes the whole system honest. The model surfaced what it wasn't sure about instead of guessing, and the one person who actually knows — the organizer, who was in the room — resolves it in a few seconds. The attendees never see the uncertainty; they see a clean recap that happens to have had a human pass over the tricky parts.

Action 3: discard (the stop)

Sometimes the recap shouldn't go out at all. The recording was a one-on-one that doesn't need a group email. The wrong file got uploaded. The meeting was sensitive and the notes shouldn't be circulated. The organizer taps *Discard* and the run stops: nothing is sent, no attendee is emailed. The recording and transcript stay in S3 (so a discard isn't a delete — it's a "don't send"), and the run is closed in `mn-runs` with action `discarded` and the organizer's name, so there's a record of the choice.

Discard exists because the default has to be safe. A system that sends unless told not to will eventually send something it shouldn't. This one sends only when a human says go.

Auto-send, for the meetings that earn it

The default is always a human in the loop. But some meetings are routine — the daily standup, the weekly check-in — and the same person approves a clean draft

every single time. For those, the style doc can turn on *auto-send* for a named meeting type, with one hard condition: the draft has zero needs-confirm flags. If everything resolved cleanly, the recap goes out on its own and the organizer just gets a copy. The moment a draft has a single flag — an unclear owner, a fuzzy date — auto-send is skipped and it falls back to the normal approve-or-edit flow. So auto-send only ever ships drafts that would have been one-tap approvals anyway.

Every run is logged, every recap is recoverable

The `mn-runs` table records every meeting: the recording, the transcript location, the draft, the action taken (sent, edited-then-sent, discarded), the organizer, the timestamp, and a snapshot of exactly what email went out. Three months later, when somebody asks “wait, who was supposed to handle the vendor thing?”, the answer is one lookup away, with the transcript line that started it. The recap email everyone half-remembers is the working memory of the meeting; the run record is the permanent one.

Next post: the cost breakdown. The whole pipeline runs in coffee-money territory at SMB volume; Part 6 explains exactly where the dollars go — mostly transcription — and how it scales.

PART 6 OF 7

MAY 20, 2026 PART 6 OF 7 · MEETING NOTETAKER SERIES ~3 MIN READ

What the meeting notetaker costs

The notetaker is a cheap system, and almost all of its cost is one line: transcription. Amazon Transcribe is billed per minute of audio, so the bill tracks how many minutes of meetings you run, not how many features the system has. Bedrock fires just twice per meeting — once for the summary, once for the action items — so it's a small sliver. Everything else (Lambda, DynamoDB, S3, SES) rounds to nothing. At typical SMB volume, the bill is a few dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $4/month at typical SMB volume (about 40 meetings a month, ~30 minutes each).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- Transcription is the dominant cost — billed per minute of audio.
- Bedrock fires twice per meeting (summary + action items); it stays a small sliver.
- At 100 meetings the bill is around $9. At 200 meetings it's around $18.

Cost at three volumes

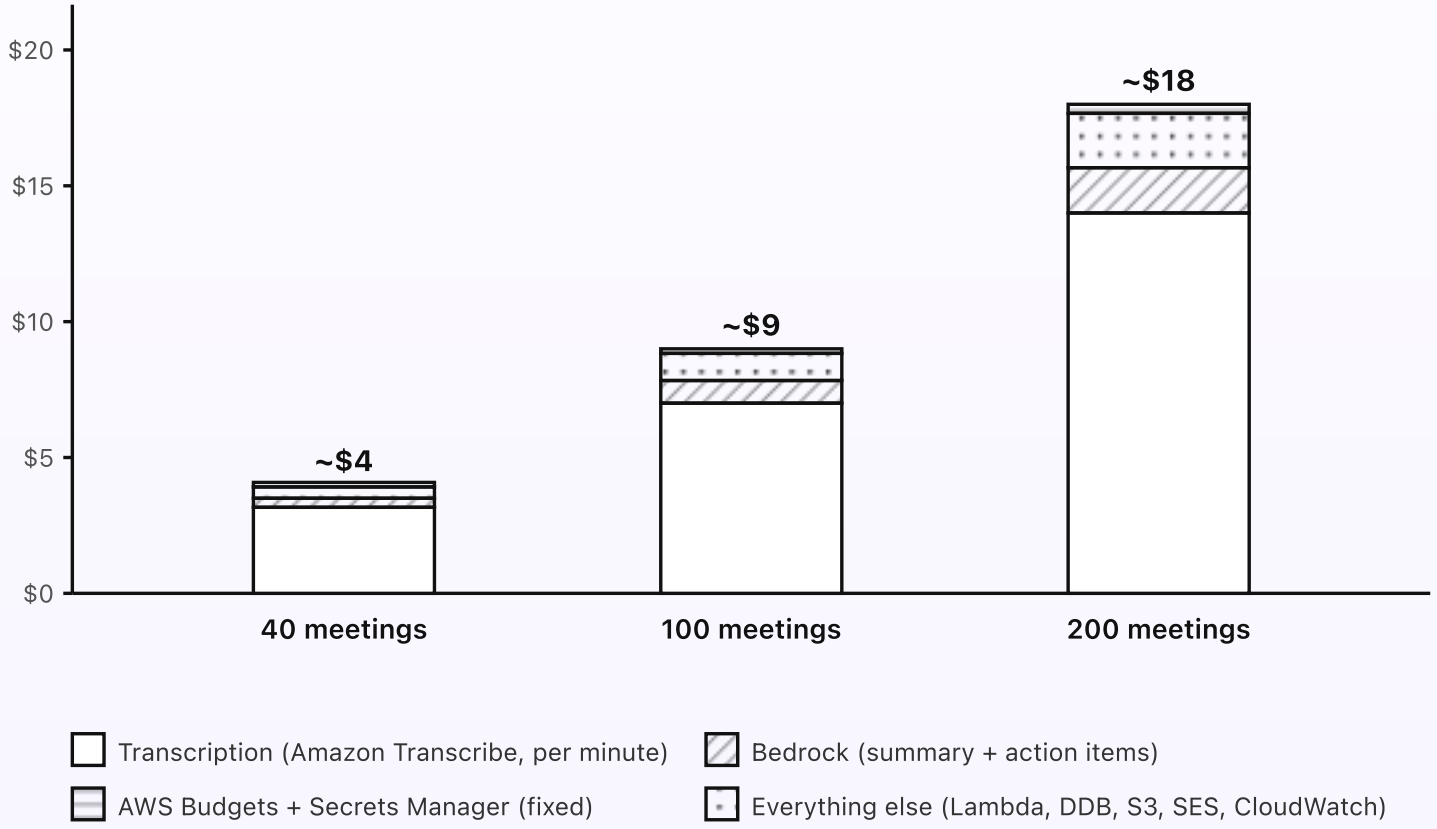


Fig 6. Monthly cost at three meeting volumes (~30 minutes each). Transcription is the dominant slice and grows with minutes of audio. Bedrock stays small because it's only two short calls per meeting. Everything else rounds to nothing.

Where the dollars actually go

Transcription (the bulk). Amazon Transcribe is billed per minute of audio. A 30-minute meeting is 30 minutes of transcription. At 40 meetings a month that's about 20 hours of audio; at 200 meetings it's about 100 hours. This is the line that moves the bill, and it moves with minutes, so a month of long offsites costs more than a month of short standups even at the same meeting count. There's nothing to optimize here except recording shorter meetings — which is good advice anyway.

Bedrock (two calls per meeting). The summary pass and the action-item pass. Each reads the transcript (a few thousand tokens for a normal meeting) and writes a short result. On Haiku 4.5 that's a fraction of a cent per meeting. Long meetings use Sonnet 4.6, which costs more per token but still lands at cents per meeting. Across a month, Bedrock is a small sliver next to transcription.

Lambda runtime. A handful of short functions per meeting: start the transcribe job, flatten the output, run the two model calls, run the four checks, send the draft, and (on approve) send the recap. Milliseconds each. Pennies a month at all three volumes.

DynamoDB on-demand. One small table, `mn-runs`, with a row per meeting plus the audit snapshots. Writes are a few per meeting; reads are occasional lookups. Pennies.

S3 + storage. The recording, the transcript, and the draft notes per meeting. A 30-minute recording is tens of megabytes; transcripts and notes are kilobytes. A few gigabytes a month at most, with lifecycle rules moving old recordings to cheaper storage. A dollar or two at the high end, less at SMB volume.

SES. Outbound for the draft to the organizer and the recap to attendees: $0.10 per thousand emails. A couple of cents a month even at 200 meetings.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the approve and send buttons.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate, no bot sitting in your calls. The pipeline wakes only when a recording lands.
- **A Knowledge Base.** The notes come from the meeting's own transcript, not from a searchable corpus. No embeddings, no Knowledge Base, no S3 Vectors needed.
- **Models on a schedule.** Bedrock fires twice per meeting and never on a timer. Nothing runs when no meetings do.

How the cost scales

Transcription grows linearly with minutes of audio, and Bedrock grows linearly with meeting count, so the total tracks how much your team actually meets. At 400 meetings a month the bill is around $35; at 1,000 it's around $85, almost all of it transcription. Past those volumes the math doesn't change shape — there's no cliff — you're simply paying for more minutes of audio. The only real lever is meeting length, and the system can't shorten your meetings for you.

Set an AWS Budgets alarm at $30/month so anything unusual pages you before the bill matters. The notetaker'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, the Transcribe job config, and the Bedrock model IDs.

PART 7 OF 7

MAY 20, 2026 PART 7 OF 7 · MEETING NOTETAKER SERIES ~8 MIN READ

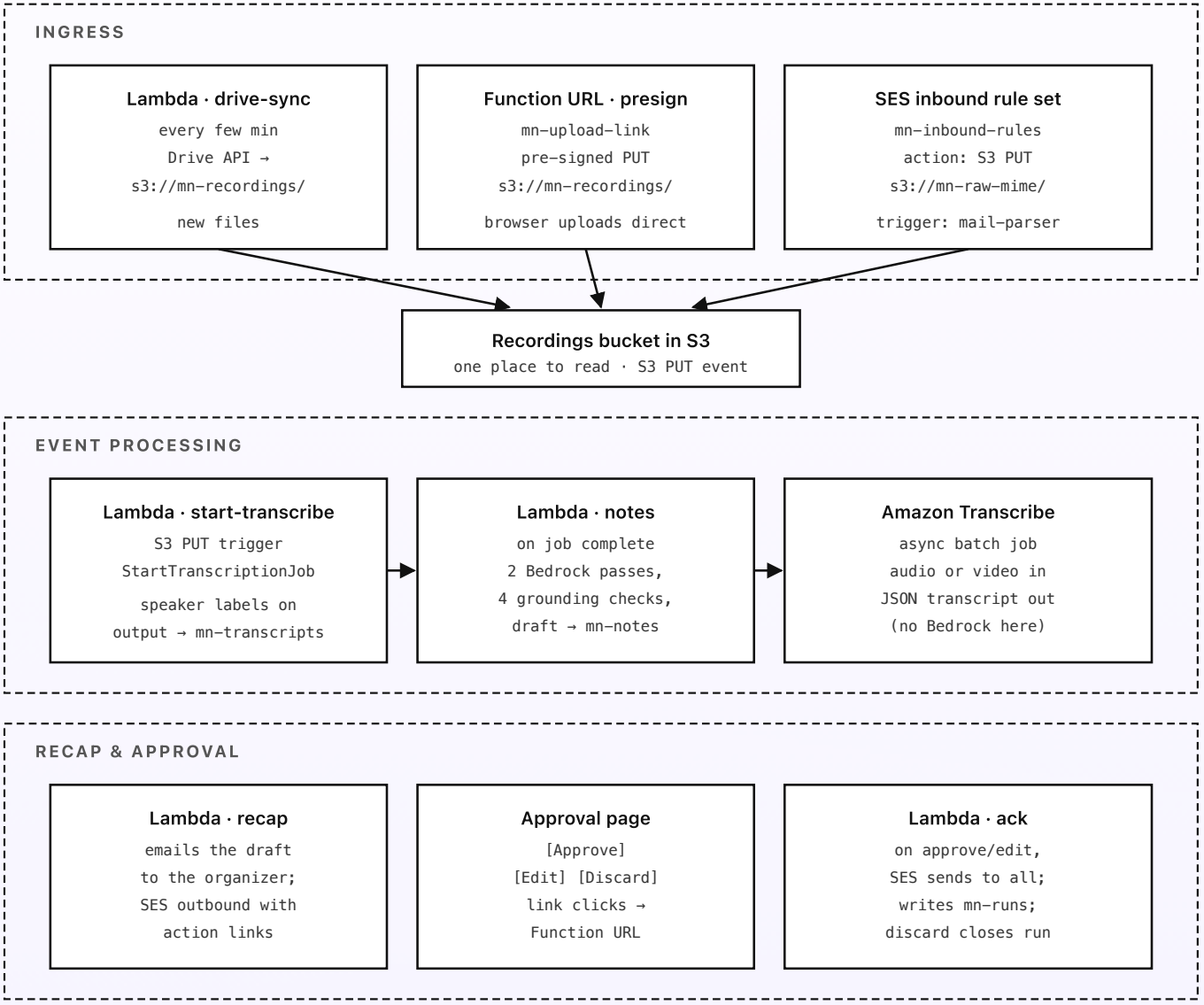
Engineering reference: the meeting notetaker architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, Lambda inventory, IAM scopes, the Amazon Transcribe job config, EventBridge wiring, the DynamoDB schemas, and the approve-and-send flow. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). Amazon Transcribe, Bedrock cross-Region inference, SES, and EventBridge are all available there. A second region for resilience isn't worth the setup work at SMB volume — the failure mode for an SMB is one meeting's recap landing a few minutes late, not a regional outage. One AWS account dedicated to the notetaker (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Every recap is confirmed by a human — and every run is logged to mn-runs.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the recordings bucket), event processing (transcribe, then the grounded notes draft), recap and approval (the organizer confirms and the recap ships). Every Lambda is event-driven; nothing is synchronous-chained.

Lambda functions

All Lambdas use the `arm64` architecture, the smallest memory size that meets latency targets, 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 few minutes. Uses the Google Drive API (service-account credentials in Secrets Manager under `mn/drive/sa`) to list new files in the recordings folder and copy anything unseen to `s3://mn-recordings/`, along with the matching sidecar if present. Same pattern syncs the style doc and roster to `s3://mn-config-source/`. Memory: 256 MB. Timeout: 60 s.
- **`presign`** — Lambda Function URL behind the private upload page. Issues a pre-signed S3 `PutObject` URL scoped to a single key under `s3://mn-recordings/` with a short expiry, and writes the supplied title/organizer/attendees into the sidecar object. The browser uploads directly to S3; no file passes through Lambda. Memory: 256 MB. Timeout: 15 s.
- **`mail-parser`** — S3 PUT trigger on `s3://mn-raw-mime/`. Parses MIME, extracts the audio/video attachment (or follows a download link in the body), and writes the media to `s3://mn-recordings/` with a sidecar derived from the email headers. Memory: 512 MB. Timeout: 120 s.

- **`start-transcribe`** — S3 PUT trigger on `s3://mn-recordings/` (media keys only; sidecar writes are ignored). Calls `StartTranscriptionJob` with `ShowSpeakerLabels: true`, `MaxSpeakerLabels` from the sidecar attendee count, output to `s3://mn-transcripts/`, and a job name keyed to the meeting id. Memory: 256 MB. Timeout: 30 s. *No Bedrock calls.*
- **`notes`** — triggered by an EventBridge rule on the Transcribe job-state-change event (`COMPLETED`). Reads the transcript JSON, flattens it to a timestamped line list, maps speaker labels to roster names, then calls Bedrock: pass 1 for the summary, pass 2 for decisions + action items as JSON with a cited line per item. Selects Haiku 4.5 by default; Sonnet 4.6 when the audio duration exceeds the configured cutoff. Runs the four grounding checks (cite-or-drop, owner resolution, date sanity, assemble) in plain Python. Writes the draft to `s3://mn-notes/`. Memory: 1024 MB. Timeout: 120 s.
- **`recap`** — triggered after `notes` writes a draft (S3 PUT on `s3://mn-notes/`). Renders the draft as an HTML email and sends it to the organizer via SES `SendRawEmail`, with Approve/Edit/Discard links pointing at the `ack` Function URL carrying a signed token. If the meeting type has auto-send enabled and the draft has zero needs-confirm flags, skips the organizer step and sends straight to attendees. Memory: 256 MB. Timeout: 30 s.
- **`ack`** — Lambda Function URL, public with `AuthType: NONE`; verifies a signed token (HMAC with a secret in Secrets Manager) on every request. Handles Approve (send the clean recap to attendees via SES, mark run `sent`), Edit (render the edit form, apply submitted changes, then send), and Discard (close the run, send nothing). Writes `mn-runs` on every action. Memory: 256 MB. Timeout: 15 s.

- `digest` — EventBridge Scheduler target, weekly Monday 8am. Reads `mn-runs` for the past week and emails the organizer set a roll-up of meetings processed and any still-open action items. No Bedrock; a plain summary table. Memory: 256 MB.

Storage

- **DynamoDB** · `mn-runs` — one row per meeting run. PK `meeting_id`; sort key `ts`; attributes: `action` (sent/edited/discarded), `organizer`, `attendees`, `transcript_key`, `notes_key`, `snapshot` (the recap that was sent). On-demand. No TTL — this is the long-term audit trail.
- **DynamoDB** · `mn-action-items` — one row per action item across all meetings. PK `(meeting_id, item_index)`; attributes: `task`, `owner_email`, `due_date`, `cite_line`, `cite_ts`, `status` (open/done). On-demand. Backs the weekly digest's open-items view.
- **S3** · `mn-recordings` — uploaded media plus sidecars. Versioning enabled. Lifecycle to Glacier at 30 days; expiry at 1 year (recordings are the largest objects).
- **S3** · `mn-transcripts` — raw Transcribe JSON and the flattened transcript. Versioning enabled.
- **S3** · `mn-notes` — the draft notes JSON and the rendered recap HTML. Versioning enabled.
- **S3** · `mn-raw-mime` — raw inbound MIME from the email lane. Lifecycle to Glacier at 30 days; expiry at 1 year.

- **S3 · `mn-config-source`** — mirrored style doc and roster as plain text/JSON. Versioning enabled.

Amazon Transcribe

- **Job type.** Asynchronous batch (`StartTranscriptionJob`), not streaming — the recording is already a complete file, and batch is cheaper and simpler than streaming for after-the-fact notes.
- **Speaker labels.** `Settings.ShowSpeakerLabels: true`, `MaxSpeakerLabels` set from the sidecar attendee count (default 10). Output written to `s3://mn-transcripts/<meeting-id>.json`.
- **Language.** `IdentifyLanguage: true` by default so mixed-language teams work without per-job config; pin `LanguageCode` if the team is always one language for slightly better accuracy.
- **Completion.** Transcribe emits a `Transcribe Job State Change` event to the default EventBridge bus; an EventBridge rule on `COMPLETED` triggers `notes`. No polling.

Bedrock

- **Foundation models.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0` for normal meetings; `anthropic.claude-sonnet-4-6-20250930-v1:0` via `global.anthropic.claude-sonnet-4-6-20250930-v1:0` for meetings over the duration cutoff. Two callsites in `notes`: the summary pass and the action-item pass.

- **Embeddings.** Not used. Notes come from the meeting's own transcript; there's no corpus to search. If a future feature needs retrieval across past meetings, the path would be Amazon Titan Text Embeddings V2 (1024-dim) into Amazon S3 Vectors — not added here.
- **Prompting.** Both passes are given the flat transcript and instructed to use only it, to cite the verbatim line and timestamp behind every claim, and to emit `needs_confirm` rather than guess an owner or date. Pass 2 returns strict JSON validated against a schema before the grounding checks run.
- **Quotas.** Default account quotas are more than enough at SMB volume — two short calls per meeting.

EventBridge wiring

- **`mn-drive-sync`** — Scheduler, `rate(5 minutes)`. Target: `drive-sync` Lambda.
- **`mn-transcribe-complete`** — rule on the default bus matching `source: aws.transcribe`, `detail-type: Transcribe Job State Change`, `detail.TranscriptionJobStatus: COMPLETED` with a name prefix of `mn-`. Target: `notes` Lambda.
- **`mn-weekly-digest`** — Scheduler, `cron(0 8 ? * MON *)` in `TZ_NAME`. Target: `digest` Lambda.
- **S3 notifications** — `mn-recordings` PUT → `start-transcribe`; `mn-raw-mime` PUT → `mail-parser`; `mn-notes` PUT → `recap`. Suffix filters keep sidecar and JSON writes from triggering the wrong function.

SES outbound and inbound

- Set the MX record on a dedicated subdomain (e.g. `notes.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com` for the forwarding lane.
- SES inbound rule set `mn-inbound-rules`: one rule with recipient `notes@your-company.com` → spam scan → S3 PUT to `s3://mn-raw-mime/<message-id>` → stop. The S3 PUT triggers `mail-parser`.
- SES outbound for the draft and the recap: verify a sender identity at `notes@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:

- **start-transcribe role:** `s3:GetObject` on `mn-recordings`; `transcribe:StartTranscriptionJob`; `s3:PutObject` on `mn-transcripts`. *No* `bedrock:*`.
- **notes role:** `s3:GetObject` on `mn-transcripts` and `mn-config-source`; `bedrock:InvokeModel` on the Haiku and Sonnet ARNs; `s3:PutObject` on `mn-notes`; `dynamodb:PutItem` on `mn-action-items`.
- **recap role:** `s3:GetObject` on `mn-notes`; `ses:SendRawEmail` from the verified sender; `secretsmanager:GetSecretValue` on the token-signing secret.
- **ack role:** `s3:GetObject` on `mn-notes`; `ses:SendRawEmail`; `dynamodb:PutItem` on `mn-runs` and `UpdateItem` on `mn-action-items`; `secretsmanager:GetSecretValue` on the token-signing secret.

- **drive-sync and presign roles:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on `mn-recordings` and `mn-config-source`; outbound network to `www.googleapis.com`. `presign` also needs `s3:PutObject` grantable via pre-signed URL on the recordings prefix.

Approval flow and tokens

The draft email's Approve/Edit/Discard links each carry a token: `base64(meeting_id . action . expiry)` with an HMAC signature using the secret in `mn/token/signing`. `ack` recomputes the HMAC and rejects any link whose signature doesn't match or whose expiry has passed, so a forwarded email can't be used to send a recap days later. Edit returns a minimal HTML form (no SPA, no build step) pre-filled with the needs-confirm items and the transcript line behind each. On submit, the form posts back to the same `ack` Function URL with the resolved owners and dates. Auto-send, when enabled for a meeting type, bypasses the token flow but only fires when the draft has zero needs-confirm flags.

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:** `notes` Lambda failures > 0 in a day; Transcribe job `FAILED` events > 0; `ack` token-verification failures > 5/hour (might mean the signing secret rotated).

- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $30/month threshold, alarm at 80% and 100%, posts to SNS topic `mn-cost-alarm` subscribed to the admin's email.

Config and secrets

The Google service-account credential for Drive lives in Secrets Manager under `mn/drive/sa`. The approval-token signing secret is `mn/token/signing`. The configured timezone, the long-meeting duration cutoff, the auto-send meeting-type list, the summary length, and the organizer-set address all live in Parameter Store under `/mn/config/`. The style doc and roster live in `mn-config-source` in S3, mirrored from Drive so a non-engineer can edit tone and people without a deploy. 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 every bucket so a bad sync can be rolled back in one click, and keep the EventBridge rule that matches Transcribe completion tight with a `mn-` job-name prefix so it never fires on unrelated jobs in the same account. Total deployable surface: around eight Lambdas, two DDB tables, six S3 buckets, two EventBridge rules 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.