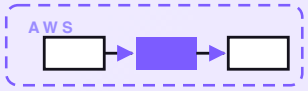
7-PART SERIES · FREE COMPANION



Tax doc collector

A serverless system that takes the pain out of tax season for an accountant or bookkeeper. For each client it knows the checklist of documents needed, sends a secure request, collects what they upload, checks off what arrives, chases what's missing, and tells the preparer the moment a file is complete. It reads each upload just enough to confirm the right document type; a human reviews before anything is marked final. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Tax doc collector

PART 1 OF 7

JUNE 14, 2026 PART 1 OF 7 · TAX DOC COLLECTOR SERIES ~5 MIN READ

A tax doc collector on AWS for a few dollars a month

Tax season is mostly chasing paper. A bookkeeper with two hundred clients spends January and February sending the same email over and over: “We still need your W-2, your mortgage interest statement, and last year’s closing statement.” The client uploads two of the three and goes quiet. Somebody has to remember who’s still missing what, send the next nudge, and notice the moment a file is finally complete so the actual work can start. This post walks through the design of a small collector that knows each client’s checklist, takes the uploads, chases the gaps, and tells the preparer when a file is ready.

KEY TAKEAWAYS

- Three ways a client file starts: the Drive checklist, a copy from last year, and a short intake form.
- Every file ends in one of four moves on each tick: complete, first request, reminder, or escalate.
- Per-client-type rules: the chase cadence and the checklist of documents owed both live in the rules doc.
- Requests respect quiet hours and your holiday calendar. A client who uploaded everything stops being chased.
- Designed on AWS for about $2 a month at typical small-practice volume.

The whole system on one page

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

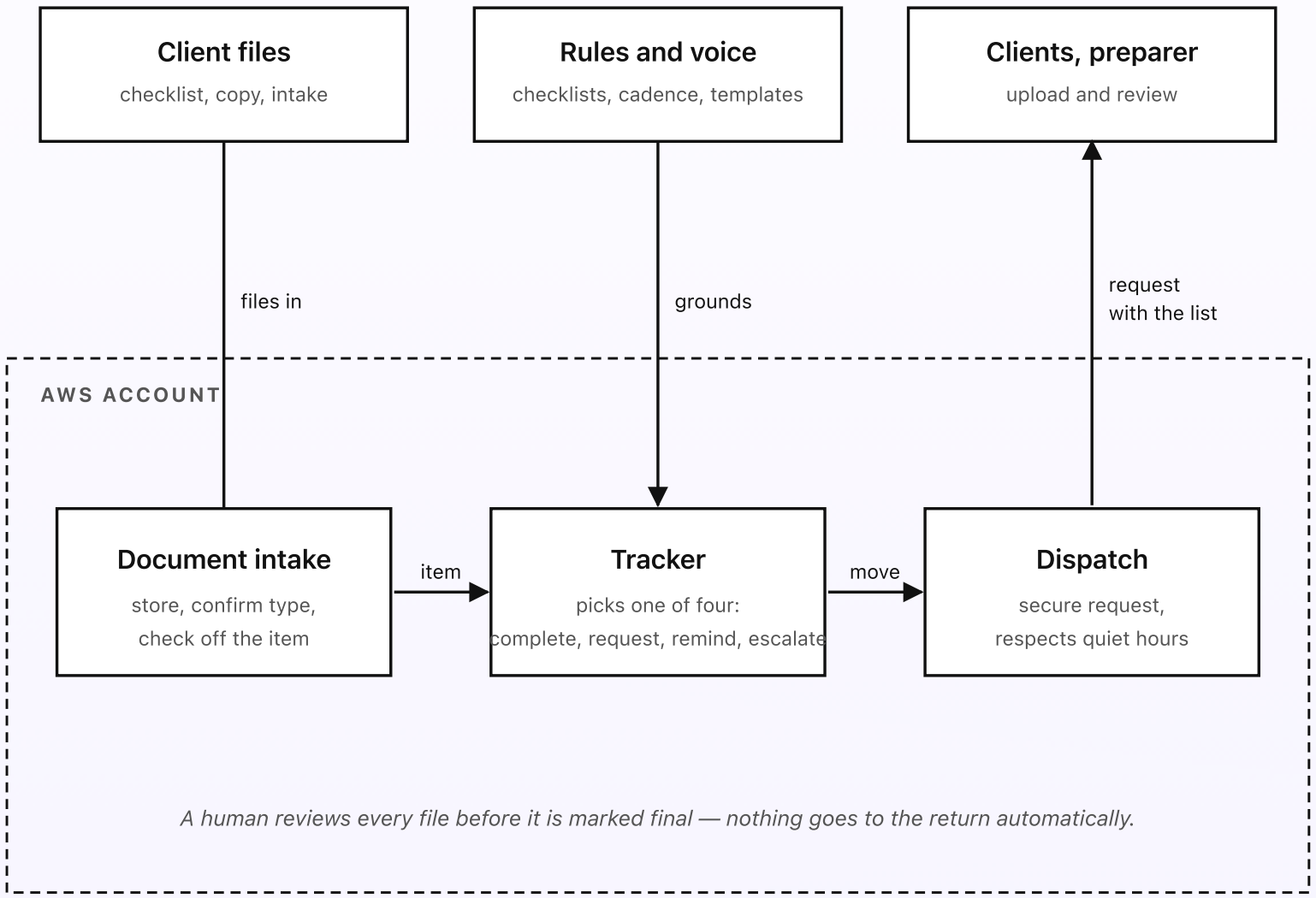

Client files
checklist, copy, intake
Rules and voice
checklists, cadence, templates
Clients, preparer
upload and review
files in
grounds
request
with the list
AWS ACCOUNT
Document intake
store, confirm type,
check off the item
item
Tracker
picks one of four:
complete, request, remind, escalate
move
Dispatch
secure request,
respects quiet hours
A human reviews every file before it is marked final — nothing goes to the return automatically.

Fig 1. Three parts outside, three pieces inside AWS. Files start from a Drive checklist, a copy from last year, or an intake form. The Tracker runs daily and picks one of four moves. Dispatch sends the right request to the right client and tells the preparer when a file is done.

What you set up once (the outside)

- **Client files.** A Google Sheet in a Drive folder, one row per client: name, client type (individual, sole trader, rental owner, small company), contact email, the checklist of documents owed for this season, a due date, and a status per item (waiting, received, accepted, rejected). You can fill it in once and forget it; new files can also start via two other lanes covered in Part 2 — a copy-from-last-year lane (one tap clones last season's file and resets every item to waiting) and a short intake-form lane (a new client fills in a few questions and the right checklist is chosen for their client type).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc lists the checklist per client type — which documents an individual owes versus a rental owner versus a small company — and the chase cadence: how many days after the first request to send a reminder, and how many times. A typical cadence is a first request on setup, then reminders at day 7, 14, and 21 for anything still missing. The doc also names the review owner (the preparer who signs off), the quiet hours, and any holiday calendars to skip. The *voice* doc holds one message template per step — what the first request and each reminder actually say.
- **Clients and preparer.** Clients upload on a secure page; each request lands with the exact list of what's still missing, a link to upload, and the option to pause the chase or hand off to a spouse or business partner. The preparer is the

accountant or bookkeeper who reviews each file once it's complete; they get a ping with a link to the per-client status board.

What runs on every tick (the inside)

- **The document intake.** A client clicks the link in their request and lands on a secure upload page (no login to remember — the link itself carries a signed, time-limited token). They drop in a PDF or a photo of a document. The file is stored privately in S3. The collector reads it just enough to confirm the type — Textract pulls the text, and Bedrock Haiku 4.5 answers one narrow question: "which checklist item does this look like?" (for example, "this looks like a W-2"). It matches the upload to a checklist item and marks that item received, pending a human review. It never reads the numbers off the document for the return; it only confirms the kind of document.
- **The tracker.** Runs once a day at 8am local. Reads each client file. For each one, works out what's still missing and how many days since the first request. Picks one of four moves. *Complete*: every item received — mark the file ready for review and tell the preparer. *First request*: the file was just set up — send the client the full list with the upload link. *Reminder*: a reminder day has passed and items are still missing — re-send, listing only what's left. *Escalate*: the due date has passed with items still missing — flag the file to the preparer so a person can call the client. The tracker calls no model on the daily tick — the move logic is plain Python.
- **Dispatch.** Reads the voice doc, fills in the request or reminder for the chosen move, and sends it. Requests go out by email through SES outbound, each carrying a signed upload link. Both honor quiet hours (no sends between 6pm and 8am local by default) and the holiday calendar. Every send writes a row to

DynamoDB so the next day's tick knows the request already went out. A weekly digest tells the preparer which files are stuck and which are close. A monthly summary writes a practice-ready paragraph: files completed, files still open, longest-waiting clients.

In plain words

The Patel family are a returning client. Their checklist this year is a W-2, a mortgage interest statement, a childcare receipt, and last year's state refund letter. The collector emails them on setup: "To start your return we need four documents — here's your secure upload link." Over the next week they upload the W-2 and the mortgage statement; the collector reads each one, confirms the type, and checks it off. On day 7 the reminder goes out, but now it only lists the two still missing — the childcare receipt and the refund letter. They upload the childcare receipt the same evening. On day 14 the last reminder names just the refund letter; Mrs Patel uploads it the next morning. The collector marks the file complete and pings the preparer: "Patel file is ready for review." The preparer opens the status board, glances at each document, accepts all four, and starts the return.

The cost of running this is about $2 a month at small-practice volume. The cost of *not* running it is the week a preparer loses every February to writing the same chase email by hand, and the file that sits half-complete until April because nobody noticed it was waiting on one receipt.

DESIGN RULES THAT SHAPED EVERY DECISION

- Every request ships with the exact list of what's still missing — never a vague "please send your documents."
- Four moves, always. Complete, first request, reminder, escalate. There is no fifth.
- Quiet hours and holidays are respected. A client who uploaded everything is never chased again.
- The collector confirms the document type only. It never reads the numbers for the return; a human always reviews.
- The checklist lives in Drive. Adding a client, changing a checklist, or shifting a due date doesn't need a deploy.
- Every upload and every action is logged. Audit a file next year and you can see exactly what came in when.

Why this shape

Most small practices chase documents in one of three places: a spreadsheet that nobody updates, an inbox full of half-finished threads, or somebody's memory. The spreadsheet works until it doesn't — one missed update and you're emailing a client for a document they sent last week. The inbox is worse: the request and the reply drift apart, attachments get buried, and you can't tell at a glance who's still missing what. And memory, of course, fails the moment the practice gets busy, which is exactly the season this matters.

The setup above keeps the source of truth in a sheet the practice already edits, but adds a small system that *looks at* that sheet every day and acts only when something needs acting on. Requests go out with the exact list, so the client never has to guess. Uploads are checked off the moment they arrive. Reminders shrink as documents come in, so a client who's sent three of four sees only the one that's left. And the preparer hears about a file exactly twice: once when it's complete, and once if it's overdue. The collector is invisible most days; visible only when a file is ready or stuck.

The next four posts walk through each piece in turn: how a client file gets set up, how a document arrives and gets checked, how a client gets chased for missing docs, and how a finished file reaches the preparer. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

JUNE 14, 2026 PART 2 OF 7 · TAX DOC COLLECTOR SERIES ~4 MIN READ

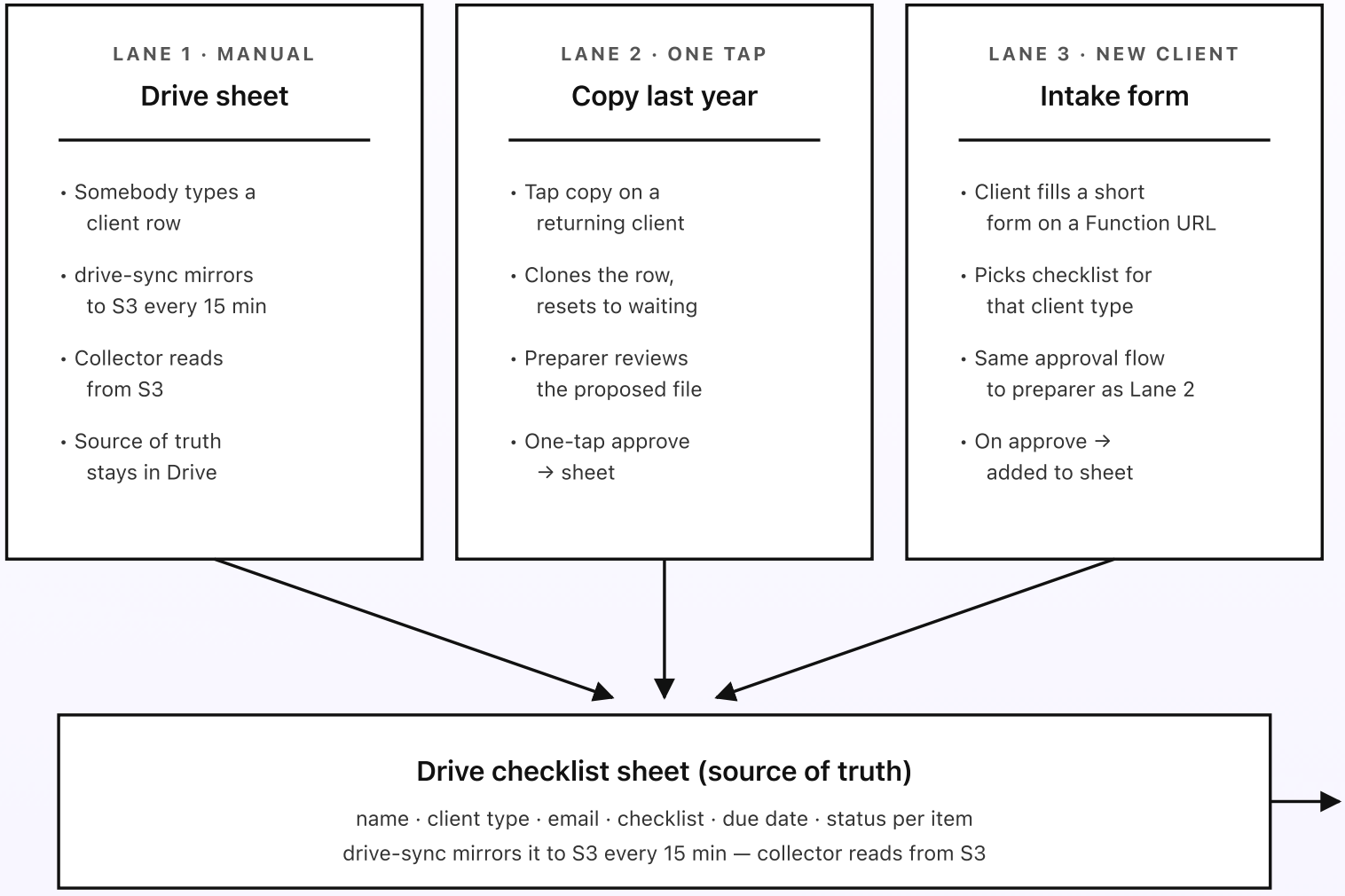
How a client file gets set up

The collector only chases what's in the checklist. So the first job is making sure each client file actually lists the right documents for that client. There are three ways a file gets started: somebody types a row in the Drive sheet, somebody taps "copy last year's file," or a new client fills in a short intake form. The first one is obvious. The other two exist because in real life nobody wants to retype last year's checklist, and a brand-new client doesn't know which documents they owe.

KEY TAKEAWAYS

- Three setup lanes feed one checklist sheet: the Drive sheet, a copy-from-last-year lane, and an intake form.
- The copy lane clones last season's row and resets every item to waiting in one tap.
- The intake form picks the right checklist for the client type, then the preparer approves it before it lands.
- Every new file goes through the preparer's one-tap approval before the first request goes out.
- The Drive sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one checklist



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

Fig 2. Three lanes converge on one Drive sheet. The sheet is the source of truth; the copy lane and the intake form are conveniences that propose files for the preparer's approval. The drive-sync Lambda mirrors the sheet to S3 so the collector can read it without hitting Drive on every tick.

Lane 1: the Drive sheet itself

The simplest lane. Open the checklist sheet in Drive, add a row, save. The columns are short: name, client type, contact email, the checklist of documents owed, a due date, and a status per item. A small Lambda — `drive-sync` — runs every fifteen minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://td-clients-source/clients.csv` if the sheet has changed since the last sync. The collector 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 the client and their checklist and you can spend thirty seconds typing it in. It's also how you hand-tune any file the other two lanes propose — the sheet is always editable by hand.

Lane 2: copy last year (the lane most practices use)

Most clients in February are returning clients. Last year you collected a W-2, a mortgage statement, and a childcare receipt; this year you'll need the same kinds of document, just the new versions. So the most-used lane is a one-tap copy. In the per-client status board, each returning client has a *Start this year's file* button. Tapping it calls a Function URL Lambda that clones last season's row: it keeps the name, client type, email, and the checklist, but resets every item's status to

waiting and clears last year's uploaded files (they stay archived for reference). It sets a fresh due date from the rules doc.

The cloned file is shown to the preparer for a quick look before anything goes out — a small approval card with the proposed checklist and three buttons: *start*, *edit*, *cancel*. On *start*, the row is written to the Drive sheet via the Sheets API and the file enters the chase cadence (Part 4). On *edit*, the preparer gets a fillable view to add or drop an item — useful for the client who sold a rental this year and now owes a closing statement they didn't owe last year. On *cancel*, nothing is written.

The reason every cloned file goes to a human first is simple: the wrong checklist is worse than no checklist. A file that asks a client for a document they don't have wastes their time and yours; a file that forgets to ask for one means a scramble in April. A ten-second preparer glance catches both.

Lane 3: the intake form for new clients

A brand-new client has no last-year file to copy, and they don't know which documents they owe. Lane 3 is a short intake form, served from a Function URL, that you link from your website or a welcome email. The client answers a few plain questions — are you filing as an individual, a sole trader, a rental owner, a small company; do you have children; do you own your home — and submits.

A Function URL Lambda reads the answers and builds the right checklist from the rules doc: the rules doc has one checklist template per client type, plus a few conditional items ("if they own a home, add the mortgage interest statement"). The proposed file goes into the same preparer approval card as Lane 2 — one tap to start, with an edit option. Once approved, the new row lands in the Drive sheet and the first request goes out.

The intake form is the most opt-in of the three lanes. A practice that doesn't use it loses nothing; a practice that does saves the back-and-forth of figuring out what a new client needs to send.

Why the sheet stays the source of truth

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

Next post: how a document actually arrives, how it's stored privately, and how the collector reads it just enough to check off the right checklist item.

PART 3 OF 7

JUNE 14, 2026 PART 3 OF 7 · TAX DOC COLLECTOR SERIES ~5 MIN READ

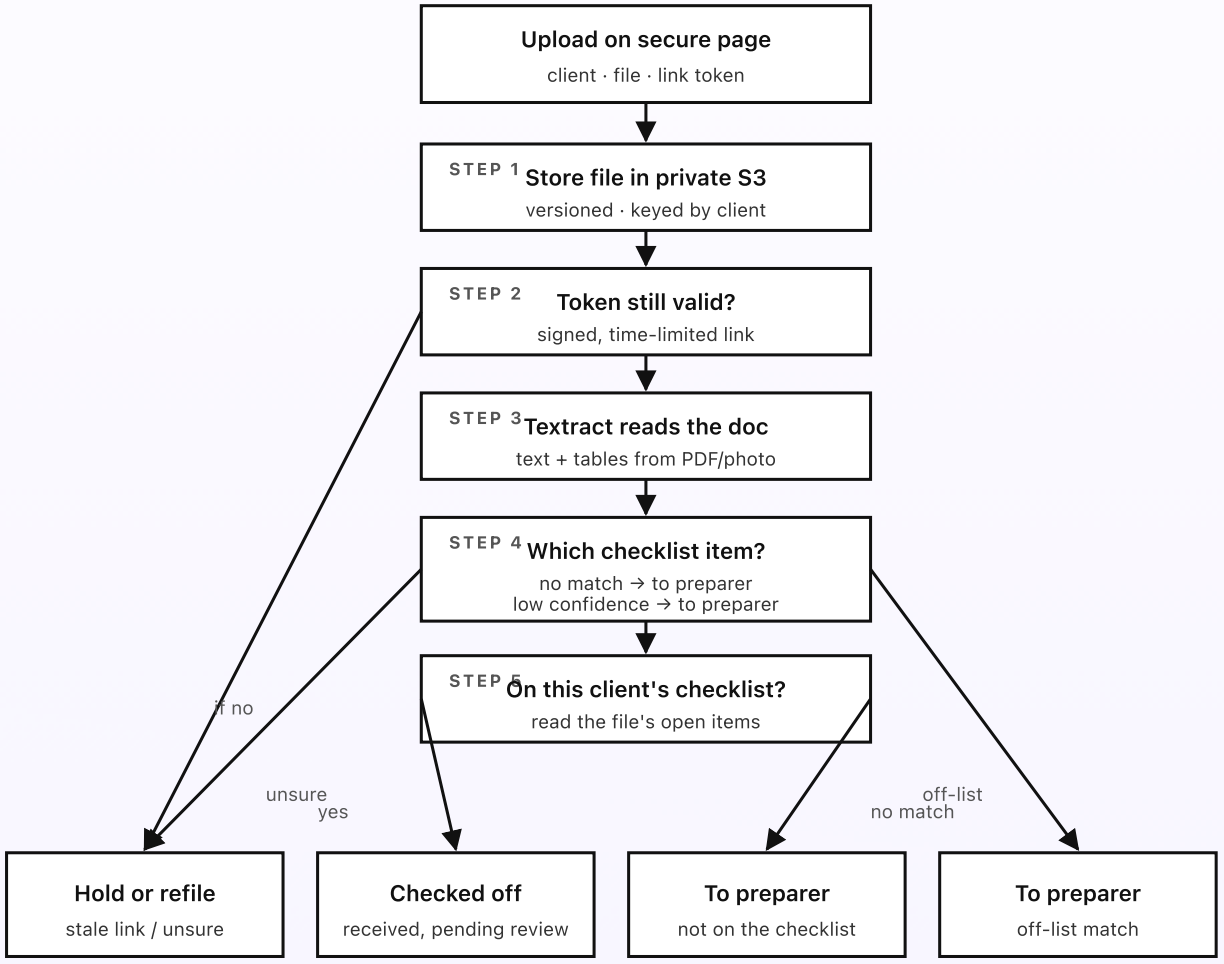
How a document arrives and gets checked

A client clicks the upload link in their request and lands on a secure page. They drop in a PDF of their W-2. From there the collector has to store it safely, read it just enough to recognize what kind of document it is, match it to the right item on that client's checklist, and check it off — without ever pretending it's done the preparer's job. The whole flow is a few small steps, and the one model call answers exactly one narrow question.

KEY TAKEAWAYS

- The upload page is a Function URL with a signed, time-limited link — no password for the client to remember.
- Files land in a private, versioned S3 bucket; nothing is public.
- Textract reads the text; Bedrock Haiku 4.5 answers one question — which checklist item is this?
- A confident match checks the item off; a low-confidence match goes to the preparer to file by hand.
- The collector confirms the document type only. It never reads numbers for the return.

The check flow, per upload



The collector confirms the kind of document only — a human reviews the contents before final.

Fig 3. The check flow, per upload. Five steps decide whether a document checks off an item, gets held, or goes to the preparer. The model answers one question — which checklist item is this — and never reads figures for the return.

The secure upload page

The request email carries a link like `https://<function-url>/u/<token>`. The token is signed and time-limited — it carries the client id and an expiry, and it's signed with a key in Secrets Manager so it can't be forged or guessed. The client doesn't need a password; the link is the credential. When they open it, a Function URL Lambda checks the signature and expiry, then serves a plain upload page that lists exactly which items are still missing, with a drop zone for each. They can upload a PDF or a photo from their phone.

If a client tries an old link — say, one from last week that's now expired — the page shows a friendly "this link has expired, here's a fresh one" message and emails them a new link. Links expire on purpose: a tax document link that lives forever in an old email is a small risk that's easy to remove.

Storing the file safely

The uploaded file goes straight into a private S3 bucket, keyed by client and a fresh id: `s3://td-uploads/<client_id>/<upload_id>`. The bucket blocks all public access, has versioning on (so a re-upload never silently overwrites), and is encrypted at rest. Nothing about a client's documents is ever served from a public URL; the preparer views them through the status board, which generates short-lived signed links on demand. The S3 PUT triggers the next step.

Reading just enough to recognize the document

Amazon Textract reads the file — it handles PDF, PNG, JPEG, and TIFF natively, which covers a scan, a phone photo, or a download from a payroll portal. Textract returns the text and any tables. Then a single Bedrock Haiku 4.5 call answers one narrow question: of the items still open on this client's checklist, which one does this document look like? The prompt is short and bounded: "Here is the text of an uploaded document and a list of the checklist items still open for this client. Return the single best-matching item and a confidence score. If none match, say so. Do not extract any amounts or figures."

That last sentence matters. The collector is not reading the wages off a W-2 or the interest off a mortgage statement — that's the preparer's job, and getting it wrong silently would be far worse than not doing it at all. The model's only job is to recognize *what kind of document* this is so the right box gets checked.

A confident match checks off; everything else goes to a human

If the match is confident and the named item is on this client's checklist, the collector marks that item *received, pending review* in the file and writes a row to the `td-uploads` DynamoDB table linking the upload to the item. The client's next reminder (if any) will no longer list that document. The item is not yet *accepted* — that happens only when the preparer reviews it in Part 5.

Two cases route to the preparer instead. If the model is unsure — a blurry photo, an unusual layout, a document it can't confidently name — the upload lands in a small "needs filing" queue on the status board for a human to match by hand. And

if the document clearly matches something that isn't on this client's checklist — the client uploaded a brokerage statement when their checklist has no investment item — it also goes to the preparer, who can add the item to the checklist or set the document aside. Both cases are normal, and both keep a person in the loop exactly where a machine would be guessing.

Why this shape

The expensive, risky work in document collection isn't storing files — it's the silent mistake: a document filed against the wrong client, or a number misread into a return. This design removes the second risk entirely by never reading figures, and it keeps the first risk small by checking the signed token, keying every file to a client, and sending anything uncertain to a human. The one model call is cheap, bounded, and easy to reason about: it answers one question and is wrong in only the safe direction (it sends things to a person rather than guessing).

Next post: how the daily tick reads each file, sees what's still missing, and decides whether to send a first request, a reminder, an escalation, or nothing at all.

PART 4 OF 7

JUNE 14, 2026 PART 4 OF 7 · TAX DOC COLLECTOR SERIES ~5 MIN READ

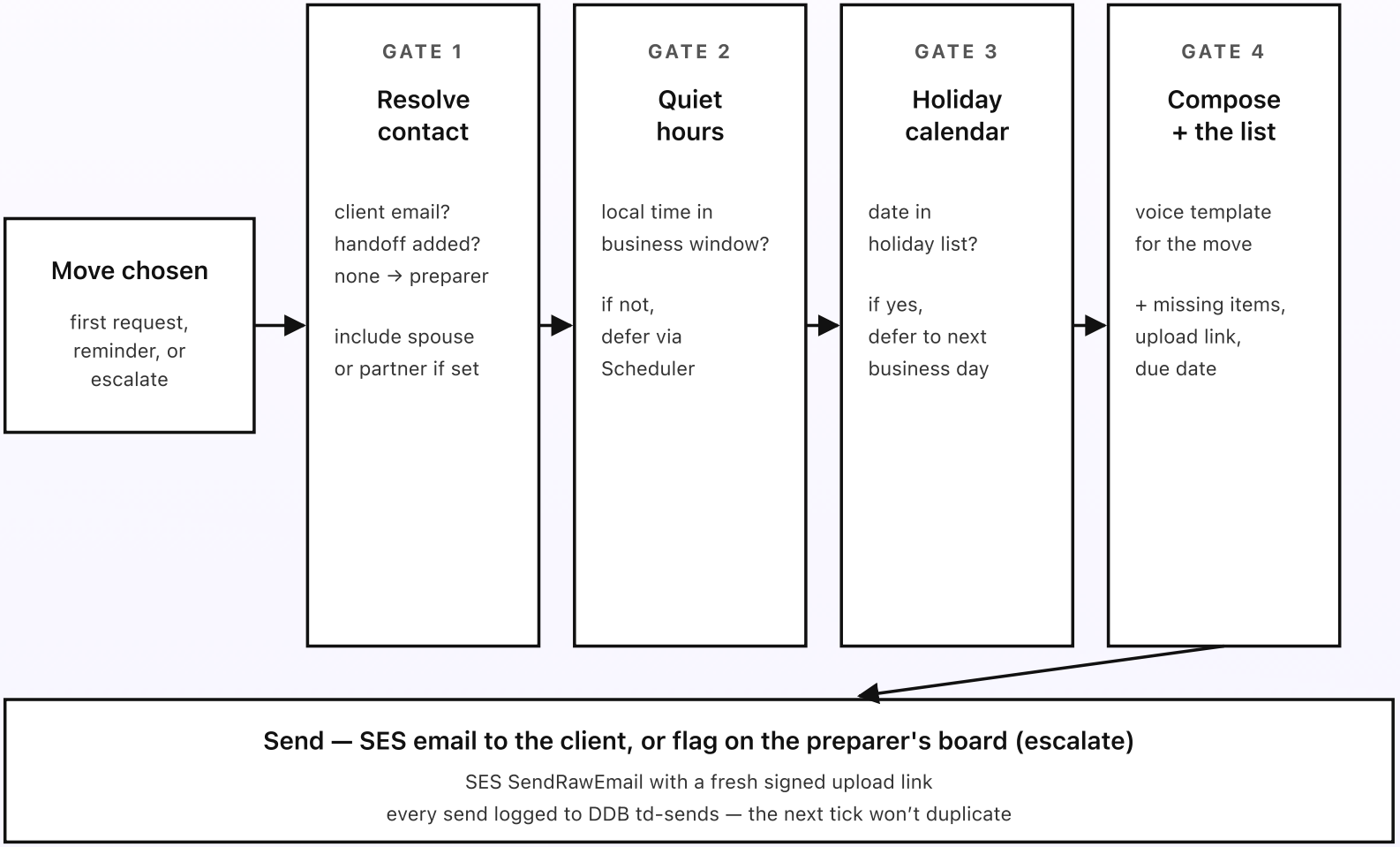
How a client gets chased for missing docs

Once a day, at 8am local time, an EventBridge Scheduler rule fires the tracker Lambda. It reads each client file, sees what's still missing and how many days since the first request, and picks one of four moves: do nothing, send the first request, send a reminder, or escalate to the preparer. When a move means sending something, four small guardrails decide who gets it, on what day, and with what wording — so a client never gets a 2am email or a vague "please send your documents." The whole decision is plain Python. No model on the tick.

KEY TAKEAWAYS

- The tracker runs once a day via EventBridge Scheduler at 8am local time.
- The chase cadence lives in the rules doc — first request on setup, then reminders at day 7, 14, and 21.
- Four moves per file, every tick: complete, first request, reminder, escalate.
- Reminders list only what's still missing — a client who sent three of four sees only the one that's left.
- Four guardrails on every send: resolve the contact, honor quiet hours, skip holidays, compose with the exact list.

Four guardrails on every send



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

Fig 4. Four guardrails between the move and the sent request. Resolve the contact. Honor quiet hours. Skip holidays. Compose with only the missing items and a fresh upload link. Then ship via email and log the send so the next tick doesn't duplicate.

The four moves, per file, per tick

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

- **Complete.** Every checklist item is received. The tracker marks the file *ready for review* and tells the preparer (Part 5). It stops chasing the client. Most files, once finished, sit here.
- **First request.** The file was just set up and no request has gone out. Send the full list with the upload link. Write a row to the `td-sends` DynamoDB table marking that the first request fired.
- **Reminder.** A reminder day in the cadence has passed and items are still missing. Send a follow-up that lists *only* what's left, not the whole checklist again. The client who sent three of four documents sees a short note about the one that's missing. Write the new send to `td-sends`.
- **Escalate.** The due date has passed and items are still missing. Don't keep emailing the client into the void — instead, flag the file on the preparer's status board so a person can pick up the phone. The client may still get a final reminder, but the human now owns it. Mark the file escalated in DynamoDB.

The cadence: day 7, 14, 21 isn't magic, it's in the doc

The rules doc has one short section per client type. Each names the cadence in plain prose: “Individuals: first request on setup, then remind at day 7, 14, and 21. Small companies: first request on setup, then remind at day 5, 10, 15, 20.” The numbers are days since the first request. The last number is the escalation point — if items are still missing by then, the file is handed to the preparer. A file with an earlier due date gets a tighter cadence; a file set up in early January with an April due date gets a looser one. The cadence reflects how much runway the client actually has.

Per-file overrides exist too. The checklist sheet has an optional `cadence_override` column. Type a comma-separated list of days and the tracker uses your numbers for that one client — the right escape hatch for the client who’s travelling for a month and asked you not to nudge until they’re back.

Gate 1: resolve the contact

The first place the dispatch looks is the file’s `contact_email`. If the client has added a handoff — a spouse on a joint return, or a business partner who actually handles the paperwork — that person is included too. If no contact is set at all (a half-finished file), the request isn’t sent into the void; the file is flagged for the preparer to fix. The fallback should never fire in steady state; if it does, the weekly digest names every file that hit it so the sheet can be corrected.

Gates 2 and 3: quiet hours and holidays

The tracker runs at 8am local, so the first send of a tick is already in business hours. But deferred sends and one-offs can land later. Gate 2 reads the quiet-

hours setting (default 6pm to 8am) and, if the current local time is outside it, creates a one-off EventBridge Scheduler rule that fires at the next business minute and exits without sending. Gate 3 checks the date against the holiday list and, if it's a configured holiday, defers to the next non-holiday business day. Both gates exist for the same reason: a request that lands at a sensible hour gets opened; one that lands at 11pm on a holiday gets buried.

Gate 4: compose with only what's missing

The voice doc has one template per move — a first-request message and a reminder message. Gate 4 fills the template, but the crucial part is the list: it includes *only* the checklist items still open for this file, never the whole checklist. It attaches a fresh signed upload link (the old one may have expired) and the due date. For an escalate, the wording is different and the recipient is the preparer: it names how long the file has been waiting and which items are still missing, so the human has the full picture before calling the client.

Every send — request, reminder, or escalate flag — writes a row to `td-sends` in DynamoDB. The next day's tick reads that row and knows not to send the same step again. That state is what makes the whole decision deterministic: a given file with a given cadence and a given send history always produces the same move, and re-running the tick sends nothing extra.

Why the daily tick uses no model

The tick could call a model to write a warmer reminder, or to decide whether to nudge at all. It doesn't. The daily tick should be the one part of the system that's

utterly predictable — if the cadence says remind at day 14 and items are missing, the reminder fires. A model in that loop adds variance the practice can't reason about, and it would burn a model call on the nine files out of ten that are healthy. Bedrock fires only on the upload lane in Part 3 and on the monthly summary in Part 6. The tracker itself is plain Python that reads a sheet and writes sends.

Next post: how a finished file reaches the preparer — the review, the three actions on a complete file, and how a rejected item turns into a fresh request for just that one document.

PART 5 OF 7

JUNE 14, 2026 PART 5 OF 7 · TAX DOC COLLECTOR SERIES ~5 MIN READ

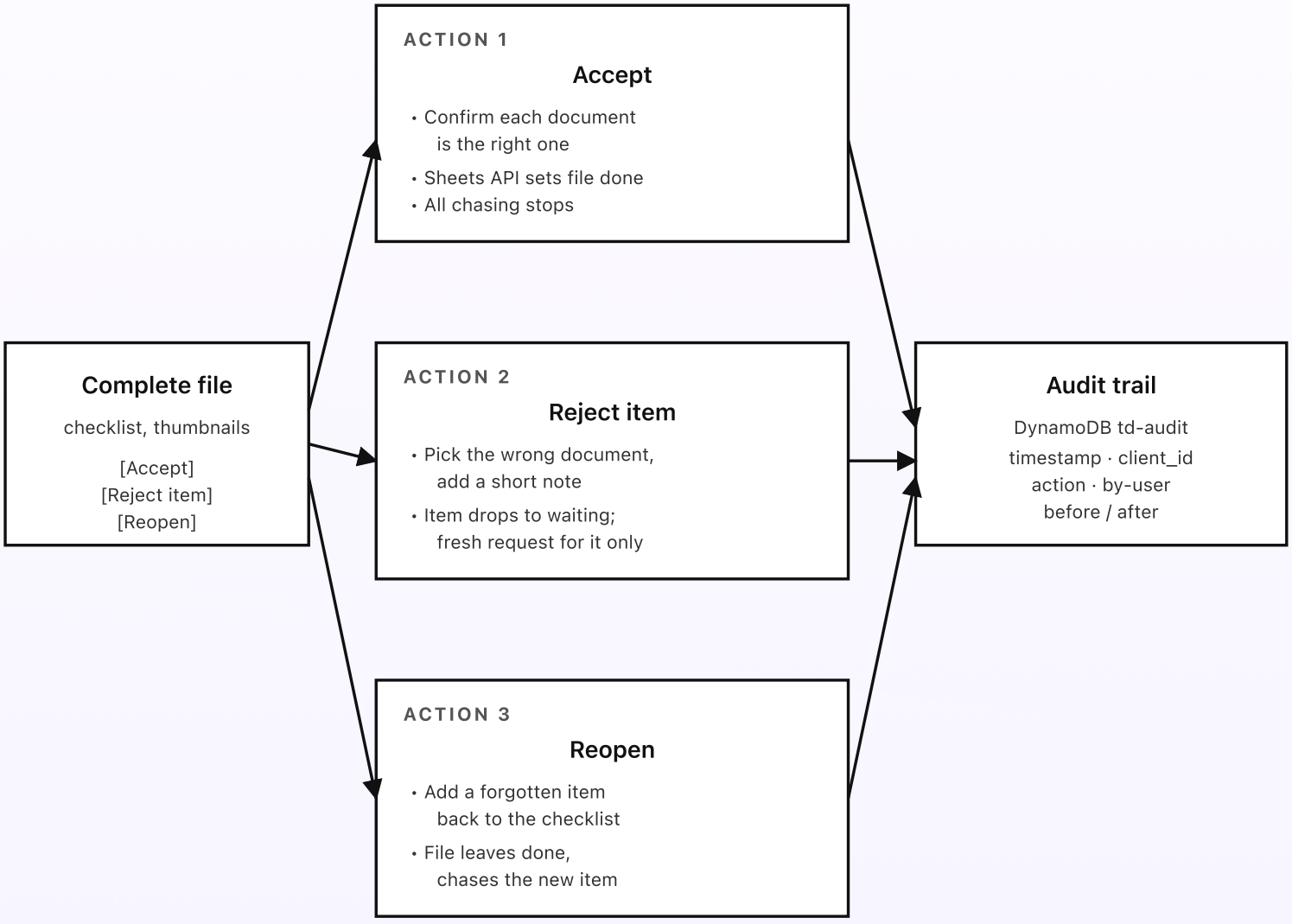
How a finished file reaches the preparer

The Patel file just went complete — all four documents received. The preparer gets a ping: “Patel file is ready for review.” There’s a link to the status board. What happens when they open it? The honest answer is “it depends on what the documents actually are.” This post walks through the three things the preparer can do on a complete file — accept, reject one item, reopen — and how the file, the chase state, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three actions per review: *accept* (file to done), *reject one item* (fresh request for just that item), *reopen* (add a forgotten item back).
- Each action updates the file via the Sheets API and writes an audit row.
- A rejected item drops back to waiting and the client gets a request for that one document only.
- Nothing is marked final until a human has looked at every document.
- The status board and its action buttons are backed by Function URLs.

Three actions on review



Nothing is marked final until a human has looked at every document — the collector only proposed.

Fig 5. Three actions per review, three different effects. Accept moves the file to done. Reject one item sends the client a fresh request for just that document. Reopen adds a forgotten item back. Every action writes to the audit trail.

Action 1: accept (the most common)

The preparer opens the Patel file on the status board. They see the four checklist items, each marked *received, pending review*, with a thumbnail and a one-line note from the collector ("looks like a W-2"). They glance at each — the W-2 is the W-2, the mortgage statement is the right year, the childcare receipt has the provider's name on it, the refund letter is the state one. They tap *Accept all*.

The tap submits to a Function URL Lambda. Two things happen, in order. First, the Sheets API updates the row: every item moves from *received* to *accepted*, and the file status becomes *done*, with today's date and the reviewing user in the `reviewed_by` column. Second, an `action: accepted` row is written to `td-audit` with the user, timestamp, and the file snapshot. From this point the tracker treats the file as done — no more chasing, no more reminders. The preparer can now start the return with every document in one place.

Action 2: reject one item (the targeted re-ask)

Sometimes a document is the right kind but the wrong one. The W-2 is from last year. The mortgage statement is for the wrong property. The receipt is a blurry photo with the amount cut off. The collector correctly recognized the *kind* of document, but only a human can tell it's not the one needed for this return.

The preparer taps *Reject* on that one item, picks a reason from a short list or types a note ("this is your 2024 W-2, we need 2025"), and saves. The Function URL Lambda drops just that item back to *waiting*, leaves every other accepted item alone, and sends the client a fresh request — but only for the one document, with the preparer's note included so the client knows exactly what was wrong. The file leaves *done* and re-enters the chase cadence for that single item; when the client re-uploads, it goes through the Part 3 check flow again and comes back for review. The rest of the file never moves.

This is the action that makes the whole system honest. Without it, "complete" would mean "the client sent four files," not "the preparer has four usable documents." Reject-one-item keeps the gap between those two small and closes it with a request the client can actually act on.

Action 3: reopen (the forgotten item)

Sometimes the checklist itself was wrong. The client mentioned on a call that they bought a rental property in November — which means they now owe a closing statement nobody added to the checklist. The file went *complete* against a checklist that was missing an item.

Reopen lets the preparer add an item back. They pick from the rules-doc list of document types (or type a custom one), and save. The Function URL Lambda adds the item in *waiting*, sets the file status back from *done* to *collecting*, and the file re-enters the chase cadence — but only for the new item. The client gets a request for that one document. Everything already accepted stays accepted.

Reopen is the escape hatch for “we asked for the wrong list” that doesn’t force the preparer to start the file over.

Every action is logged, every action is reversible

The `td-audit` table records every accept, reject, and reopen with the user who took the action, the timestamp, and a snapshot of the file before and after. If a file gets accepted by mistake (wrong client, fat-fingered button), a preparer 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 stays clean. This matters most a year later, when a question about a client’s return comes up and the only memory anyone has is the trail: who collected what, who reviewed it, and when.

And it’s worth saying once more, plainly: the collector never decided a file was finished. It recognized document types and checked boxes; a person accepted every one. The machine did the chasing and the filing; the human kept the judgment.

Next post: the cost breakdown. The whole pipeline above runs in coffee-money territory at small-practice volume; Part 6 explains exactly where the dollars go and why the chase tick is nearly free.

PART 6 OF 7

JUNE 14, 2026 PART 6 OF 7 · TAX DOC COLLECTOR SERIES ~3 MIN READ

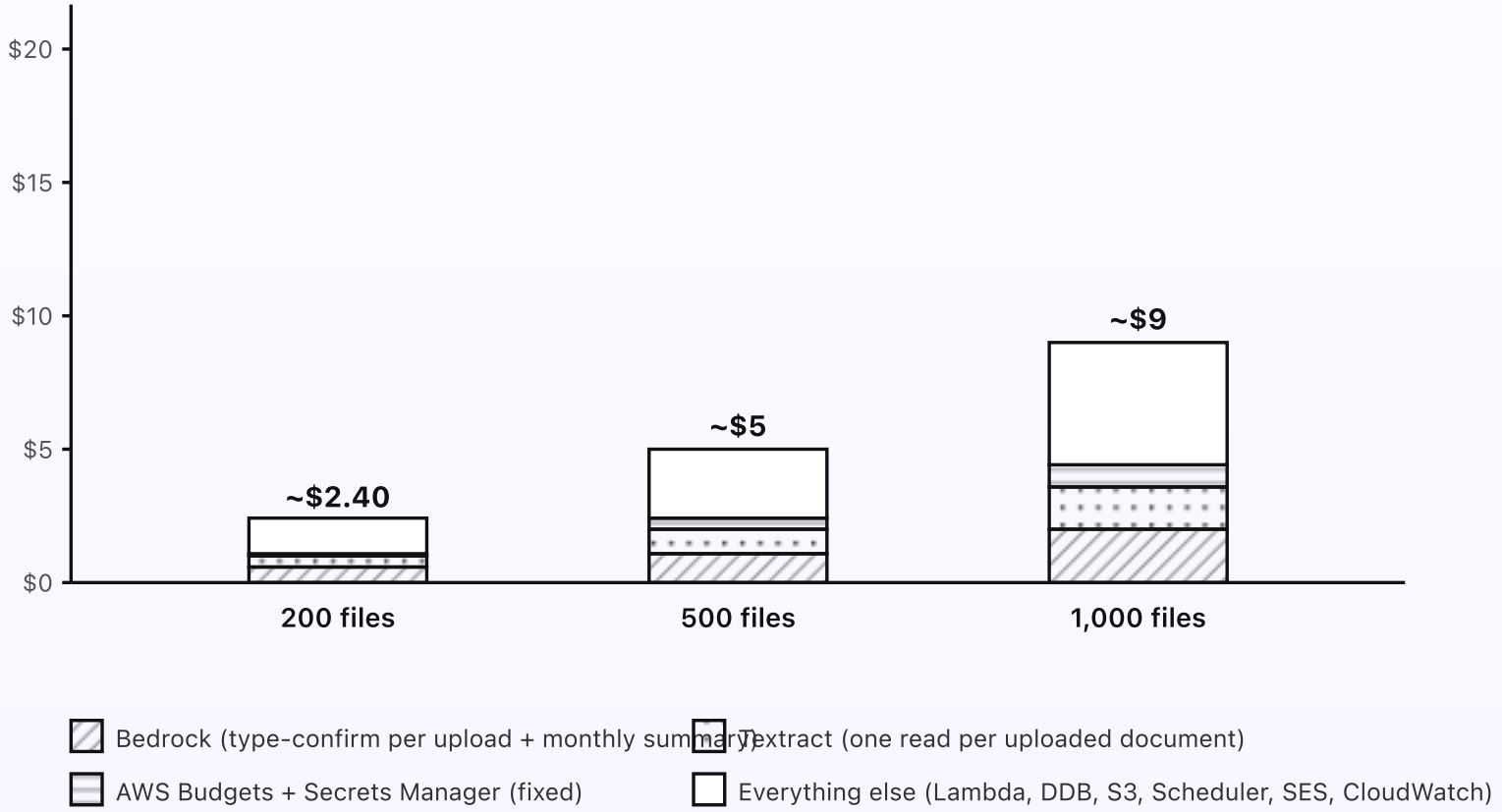
What the tax doc collector costs

The collector is a cheap system to run. The daily chase tick reads a CSV from S3, does some date arithmetic, writes a few rows to DynamoDB, and sends a handful of emails. It calls no models on the tick. The cost that does add up is the document-reading lane: every upload gets read by Textract and named by Bedrock. Even so, at typical small-practice volume the bill is a few dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2.40/month at typical small-practice volume (around 200 active client files).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily chase tick costs pennies — no model calls.
- Textract and Bedrock fire only when a client uploads a document, plus the monthly summary.
- At 500 active files the bill is around $5. At 1,000 files it's around $9.

Cost at three volumes



The document-reading lane scales with volume — the daily chase tick stays nearly free.

Fig 6. Monthly cost at three active-file volumes. Textract and Bedrock are real slices because every upload is read, but they only fire when a document arrives. The dominant cost is the everything-else bucket: the daily tick and the dispatch emails.

Where the dollars actually go

Lambda runtime (the bulk). The chase tick runs once a day. Each tick reads the checklist CSV from S3, iterates the rows, works out what's missing for each, and decides on a move. At 200 files that's a few hundred milliseconds; at 1,000 it's a couple of seconds. Add the dispatch Lambda for each send, the upload-page and action Function URLs, and the drive-sync Lambda every fifteen minutes — the Lambda total still lands under a dollar at all three volumes.

DynamoDB on-demand. Small tables: `td-sends`, `td-uploads`, `td-state`, `td-audit`. Reads dominate during the daily tick (one read per file, plus state). Writes are sends, uploads, and audit rows. Pennies a month at any of these volumes.

S3 + storage. The mirrored checklist CSV plus every uploaded document. A typical client file is a handful of PDFs and photos — a few megabytes. Even at 1,000 files that's low single-digit gigabytes. A dollar or two of storage, and that's being generous.

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

SES. Inbound (if you let clients reply or forward): $0.10 per thousand received. Outbound for the requests and reminders: $0.10 per thousand sent. A 200-file practice sends maybe a few hundred emails across a season — cents.

Textract (one read per upload). Per-page pricing; a typical tax document is one to three pages. A few cents per document. This is the band that scales with volume: more files means more uploads to read. At 200 files with their documents,

it's under a dollar; at 1,000 files it lands around a couple of dollars across the busy months.

Bedrock (only when something fires it). The daily tick uses no Bedrock. The type-confirm fires Haiku 4.5 once per uploaded document: the Textract text in, a short JSON answer out — a small fraction of a cent per call. The monthly summary is one larger call that writes a practice-ready paragraph. Bedrock stays a modest slice even at 1,000 files.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the upload page, the intake form, and the action endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The collector sleeps until a tick fires or a client uploads.
- **A Knowledge Base.** The checklist is structured rows, not free text — deterministic lookup beats vector search here. No embeddings, no Knowledge Base, no S3 Vectors.
- **Models on the tick.** The daily decision is plain Python. Bedrock fires only on uploads and the monthly summary.

How the cost scales

Lambda runtime and DynamoDB grow roughly linearly with file count, because every file is evaluated on every tick. Textract and Bedrock grow with upload count,

which roughly tracks file count too (each file is a few documents). So the bill at 2,500 active files is around $22; at 5,000 it's around $42. Past those volumes you'd batch the type-confirm calls and read documents only on the first upload of a session, but those are optimizations for a large practice — not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. A normal-volume small practice stays well under that ceiling, even in the thick of February.

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

JUNE 14, 2026 PART 7 OF 7 · TAX DOC COLLECTOR SERIES ~8 MIN READ

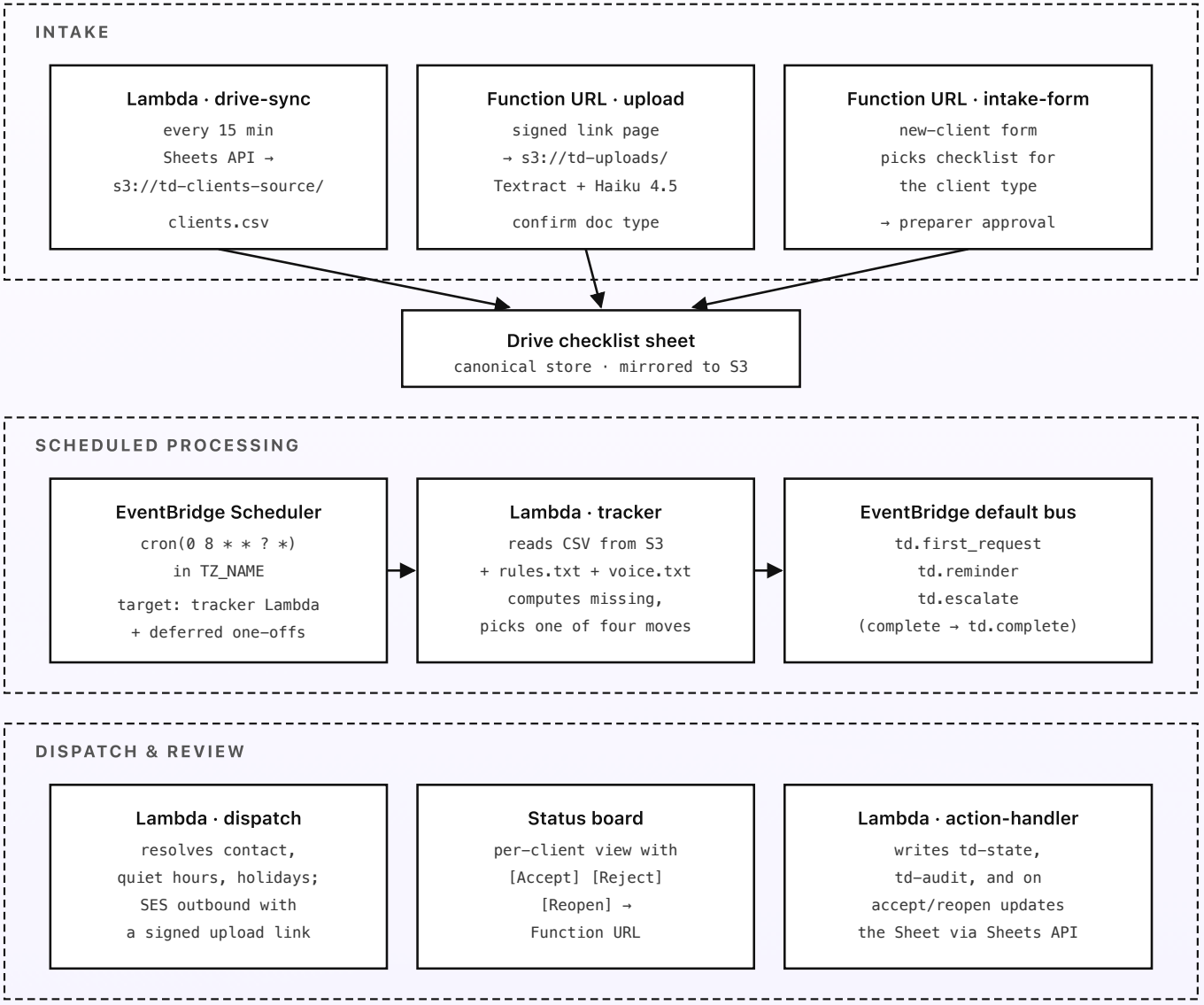
Engineering reference: the tax doc collector architecture

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 secure-upload flow. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). SES inbound, Bedrock Global cross-Region inference, Textract, and EventBridge Scheduler are all in good shape there. A second region for multi-region resilience isn't worth the extra setup work at small-practice volume — the failure mode for a practice is a reminder that goes out a day late, not a regional outage. One AWS account dedicated to the collector (separate from your other workloads) keeps the IAM blast radius small, isolates client documents, and lets a single AWS Budgets alarm cover the whole system.

Topology



A human reviews every file before final — and every interaction is logged to td-audit.

Fig 7. AWS topology, in three regions of the diagram: intake (setup lanes and the secure upload path into the checklist), scheduled processing (the daily chase tick emitting events), dispatch and review (the request ships and the preparer's decision 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 `td/drive/sa`) to export the checklist sheet as CSV and write to `s3://td-clients-source/clients.csv` only if the sheet has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://td-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`upload-handler`** — Lambda Function URL, public with `AuthType: NONE`; every request carries a signed, time-limited token (HMAC over `client_id` + `exp`, key in Secrets Manager under `td/upload/signing-key`). On GET, serves the upload page listing the file's open items. On POST, validates the token, writes the file to `s3://td-uploads/<client_id>/<upload_id>`, and enqueues the read job. The S3 PUT triggers `intake-classify`. Memory: 512 MB. Timeout: 30 s.

- **`intake-classify`** — S3 PUT trigger on `s3://td-uploads/`. Runs Textract via `StartDocumentTextDetection` + `StartDocumentAnalysis` (asynchronously to handle multi-page documents). 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 name the best-matching open checklist item with a confidence score. On a confident match, marks the item *received* in `td-state` and links the upload in `td-uploads`; otherwise routes to the preparer's needs-filing queue. The prompt is bounded to type-confirmation only and never extracts amounts. For DOCX uploads (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 path that runs a few times per client, that's acceptable; the community fork `python-docx-oss` is a drop-in alternative if extraction precision becomes a concern. Memory: 512 MB. Timeout: 60 s.
- **`intake-form`** — Lambda Function URL for the new-client intake form. On submit, reads the answers, builds the right checklist for the client type from the rules doc (including conditional items), and posts a preparer approval card. On approve, writes the new row to the Drive sheet via the Sheets API. Memory: 256 MB. Timeout: 30 s.
- **`tracker`** — EventBridge Scheduler target, daily at 8am local time (the schedule expression runs in `TZ_NAME` set to the practice's timezone, e.g. `Asia/Singapore`). Reads `s3://td-clients-source/clients.csv` and the rules and voice docs. For each row, computes the still-missing items and days-since-first-request, reads send state from `td-sends` and item state from `td-state`, decides on a move. Emits one event per row that needs action:

`td.first_request`, `td.reminder`, `td.escalate`, or `td.complete`, with the file context as the event payload. Healthy in-progress files emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*

- **`dispatch`** — EventBridge rule on the request/reminder/escalate events. Resolves contact (per-file email plus any handoff), checks quiet hours and holiday calendar, formats the request from the voice template with only the missing items and a fresh signed upload link, and sends via SES `SendRawEmail`. On `td.complete`, notifies the preparer with a status-board link instead. 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 `td-sends` after a successful send. Memory: 256 MB. Timeout: 30 s.
- **`action-handler`** — Lambda Function URL for the status-board actions; authenticated by the preparer's session cookie (the board is a small authenticated app). Handles Accept, Reject-item, and Reopen. Writes to `td-state` and `td-audit`; on accept sets the file done; on reject-item drops one item to waiting and triggers a single-item request; on reopen adds an item and re-enters the cadence. Updates the Drive sheet via the Sheets API. Memory: 256 MB. Timeout: 15 s.
- **`digest`** — EventBridge Scheduler target, weekly Monday 7am. Reads `td-state` and the checklist; sends the preparer a digest summarizing files complete this week, files stuck, and longest-waiting clients. No Bedrock; a plain summary table. Memory: 256 MB.
- **`summary`** — EventBridge Scheduler target, monthly on the first Monday at 9am. Reads the past month's `td-sends`, `td-state`, and `td-audit`; calls Bedrock

Haiku 4.5 to write a one-paragraph practice narrative; emails it via SES to the configured partner list. Memory: 512 MB.

Storage

- **DynamoDB** · `td-state` — one row per checklist item per file. PK `(client_id, item_id)`; attributes: `status` (waiting/received/accepted/rejected), `upload_id`, `confirmed_type`, `confidence`, `reviewed_by`. On-demand.
- **DynamoDB** · `td-sends` — one row per dispatch. PK `(client_id, step)`; attributes: `sent_date`, `move` (first_request/reminder/escalate), `recipient`, `missing_count`. On-demand. No TTL.
- **DynamoDB** · `td-uploads` — one row per uploaded file. PK `(client_id, upload_id)`; attributes: `s3_key`, `matched_item`, `confidence`, `uploaded_at`, `review_state`. On-demand.
- **DynamoDB** · `td-audit` — one row per write action of any kind. PK `(client_id, ts)`; attributes: `action`, `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.
- **S3** · `td-clients-source` — mirrored CSV from the Drive checklist sheet. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 7 years.
- **S3** · `td-rules-source` — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3** · `td-uploads` — client-uploaded documents. Block all public access; SSE encryption; versioning enabled; lifecycle to Glacier at 180 days; expiry at 7 years. Access only via short-lived presigned URLs from the status board.

- **S3 · `td-archive`** — prior-season files and documents, kept for reference when a returning client's file is copied forward.

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-classify` for the per-upload type-confirmation, and `summary` for the monthly practice narrative. Claude Sonnet 4.6 (`global.anthropic.claude-sonnet-4-6-20250930-v1:0`) is available as a fallback for uploads the Haiku pass flags as low-confidence, but in practice tax documents are recognizable enough that Haiku handles them, and a low-confidence upload routes to a human anyway.
- **Embeddings.** Not used. The checklist is structured rows; deterministic lookup beats vector retrieval here. No Knowledge Base, no S3 Vectors.
- **Quotas.** Default account quotas are more than enough at small-practice volume. The tracker itself doesn't call Bedrock; the classify lane fires once per uploaded document.

EventBridge Scheduler config

- **`td-daily-tick`** — `cron(0 8 * * ? *)` in the practice's timezone. Target: `tracker` Lambda.
- **`td-drive-sync`** — `rate(15 minutes)`. Target: `drive-sync` Lambda.
- **`td-weekly-digest`** — `cron(0 7 ? * MON *)` in TZ. Target: `digest` Lambda.

- `td-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. `docs.your-practice.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com` if you want clients to be able to reply or forward documents by email.
- SES inbound rule set `td-inbound-rules`: one rule with recipient `docs@your-practice.com` → spam scan → S3 PUT to `s3://td-uploads/inbound/<message-id>` → stop. The S3 PUT triggers `intake-classify` via the same path as an upload.
- SES outbound for the requests and reminders: verify a sender identity at `docs@your-practice.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:

- **tracker role:** `s3:GetObject` on the clients, rules, and voice keys; `dynamodb:Query` + `GetItem` on `td-sends`, `td-state`; `events:PutEvents` on the default bus. *No* `bedrock:*`.

- **dispatch role:** `scheduler:CreateSchedule` for the deferred-send one-offs; `secretsmanager:GetSecretValue` on the upload signing key; `ses:SendRawEmail` from the verified sender identity; `dynamodb:PutItem` on `td-sends`.
- **upload-handler role:** `s3:PutObject` on `td-uploads`; `secretsmanager:GetSecretValue` on the upload signing key; `dynamodb:PutItem` on `td-uploads`.
- **intake-classify role:** `s3:GetObject` on `td-uploads`; `textract:StartDocumentTextDetection` + `StartDocumentAnalysis`; `bedrock:InvokeModel` on the Haiku ARN; `dynamodb:PutItem` on `td-state` and `td-uploads`.
- **action-handler role:** `dynamodb:PutItem` on `td-state` and `td-audit`; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret; outbound network to `sheets.googleapis.com`; `s3:GetObject` on `td-uploads` for presigned review links.
- **drive-sync role:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the clients and rules buckets; outbound network to `www.googleapis.com`.

Secure upload and review flow

Upload links are signed tokens, not session cookies: an HMAC over `client_id`, `file_set`, and an expiry, signed with the key in `td/upload/signing-key`. `upload-handler` verifies the signature and expiry on every request; an expired link renders a “request a fresh link” page that triggers a new `td.reminder`. The bucket is fully private; the preparer’s status board generates short-lived presigned

GET URLs on demand to render thumbnails and previews, so document bytes are never served from a durable public URL.

The status board itself is a small authenticated app (the practice's staff log in); its action buttons post to `action-handler` with the preparer's session. Client-facing surfaces (upload page, intake form) are unauthenticated but token-gated; staff-facing surfaces (status board, actions) require login. That split keeps clients out of each other's files without making them manage a password.

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:** tracker Lambda failures > 0 in a day (the daily tick has to run); intake-classify failure rate > 1% in 24h; upload-handler token-verification failures > 20/hour (might mean a leaked or stale link being retried).
- **X-Ray:** off by default. Not worth the cost at small-practice volume.
- **AWS Budgets:** $15/month threshold, alarm at 80% and 100%, posts to SNS topic `td-cost-alarm` subscribed to the on-call partner's email.

Config and secrets

Service-account credentials for Drive and Sheets APIs live in Secrets Manager under `td/drive/sa` (one service account with scopes for both APIs). The upload signing key is `td/upload/signing-key`. SES sender identity lives in IAM and the

verified-domain config. The configured timezone, holiday list reference, quiet-hours window, default due date, and the per-client-type checklists all live in Parameter Store under `/td/config/` (with the larger checklist templates in the Drive rules doc). Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

Whichever IaC you prefer. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning and block-public-access for `td-uploads` so a client document is never exposed and a re-upload never silently overwrites, and version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily tick in UTC after a CI rotation. CDK with a Python stack file works well; SAM also fits, and matches the GitHub Actions + OIDC deploy with no long-lived keys. Total deployable surface: around nine Lambdas, four DynamoDB 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 practice, see Work with me.