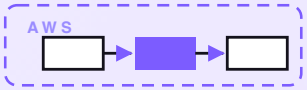
7-PART SERIES · FREE COMPANION



Document pipeline

A serverless pipeline on AWS that reads documents for you, checks each extraction, and sends the structured data to whichever tools you already use. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Document pipeline

PART 1 OF 7

APRIL 27, 2026 PART 1 OF 7 · DOCUMENT PIPELINE SERIES ~5 MIN READ

A document pipeline on AWS for a few dollars a month

Documents arrive in your business by the dozen — invoices, contracts, receipts, signed forms. Most still get typed into a sheet by hand. Here's how to build a small pipeline that reads each one for you, checks it, and sends the structured data straight to wherever it needs to go.

KEY TAKEAWAYS

- Three outside surfaces, three pieces inside AWS. Documents in, structured data out.
- A reader-validator-router trio handles every document the same way, regardless of source.
- Most documents flow straight through. A human only sees the ones the validator wasn't sure about.
- Configuration lives in a small rules file a non-engineer can edit — no deploy needed.
- Runs on AWS for a few dollars a month at typical SMB document volume.

The whole system on one page

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

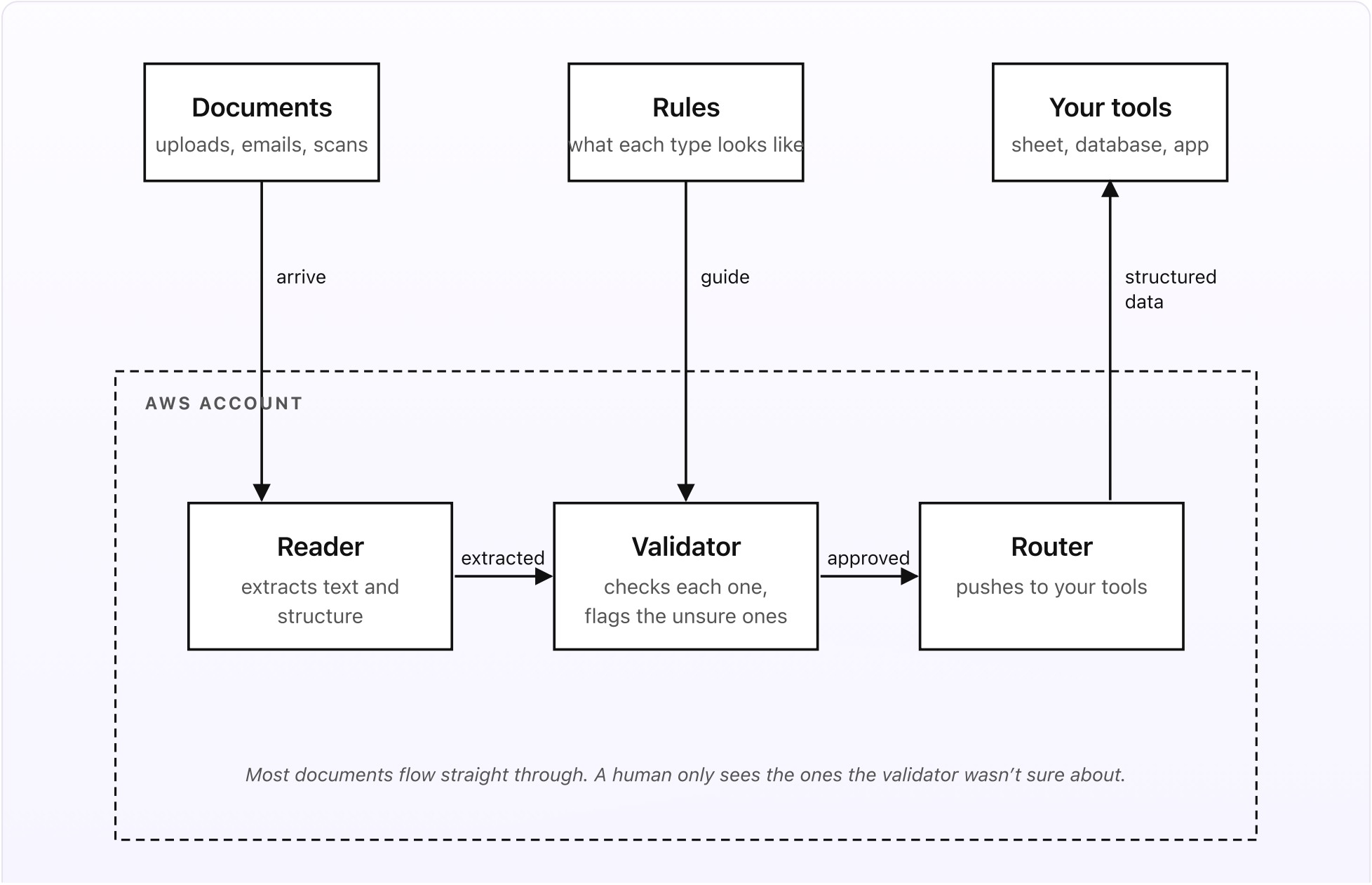

Documents
uploads, emails, scans
Rules
what each type looks like
Your tools
sheet, database, app
arrive
guide
structured data
AWS ACCOUNT
Reader
extracts text and structure
extracted
Validator
checks each one, flags the unsure ones
approved
Router
pushes to your tools
Most documents flow straight through. A human only sees the ones the validator wasn't sure about.

Fig 1. Three outside surfaces, three pieces inside AWS. Documents in, structured data out.

What you set up once (the outside)

- **A way for documents to arrive** — a small upload form, a forwarded email address, or a folder you drop scans into.
- **A short rules file** — what kinds of documents to expect (invoices, receipts, signed forms), what fields each one should have, and where the clean data should end up.
- **The tools you actually use** — a Google Sheet, a database, your accounting app. Whatever already runs your business.

What runs quietly in the cloud (the inside)

- **The reader** — takes a fresh document and pulls out the text, the layout, the table cells, the signatures. Hands the result to the validator.
- **The validator** — checks the extraction against the rules. Confident reads continue. Anything fishy goes to a short review queue you can clear in seconds.
- **The router** — takes the clean structured data and sends it where it belongs.

In plain words

Documents arrive however your team already gets them. The cloud reads each one, checks it, and sends the structured data straight to the tools you already use. The system sleeps when there's nothing to read.

Total cost runs a few dollars a month, not a few hundred.

DESIGN RULES THAT SHAPED EVERY DECISION

- Stay inside the AWS always-free quotas wherever possible.
- No always-on server. No NAT Gateway. No infinite log retention.
- The AI is only one step. Cheap rules and validations do most of the gating.
- A human always has the final say on low-confidence extractions — the system never quietly invents fields.
- Configuration lives somewhere a non-engineer can edit — updating rules never needs a deploy.

Why this shape

Most “document AI” tools collapse under one of three weights: a server bill that climbs every month, an extraction that’s wrong just often enough to be unusable, or a workflow no one outside the dev team can change.

The architecture above is the smallest set of moving parts I could find that solves all three at once. One way in (your documents), one way out (your tools), a reader-validator-router pair in the middle that knows when to ask for help. Everything else is plumbing.

The next five posts walk through each piece in turn — how documents enter, how the AI reads them, how extractions stay accurate, how the data flows out, and what the whole thing actually costs. One diagram per post. A final engineering

reference at the end gives engineers the dense version with precise service names and model IDs.

PART 2 OF 7

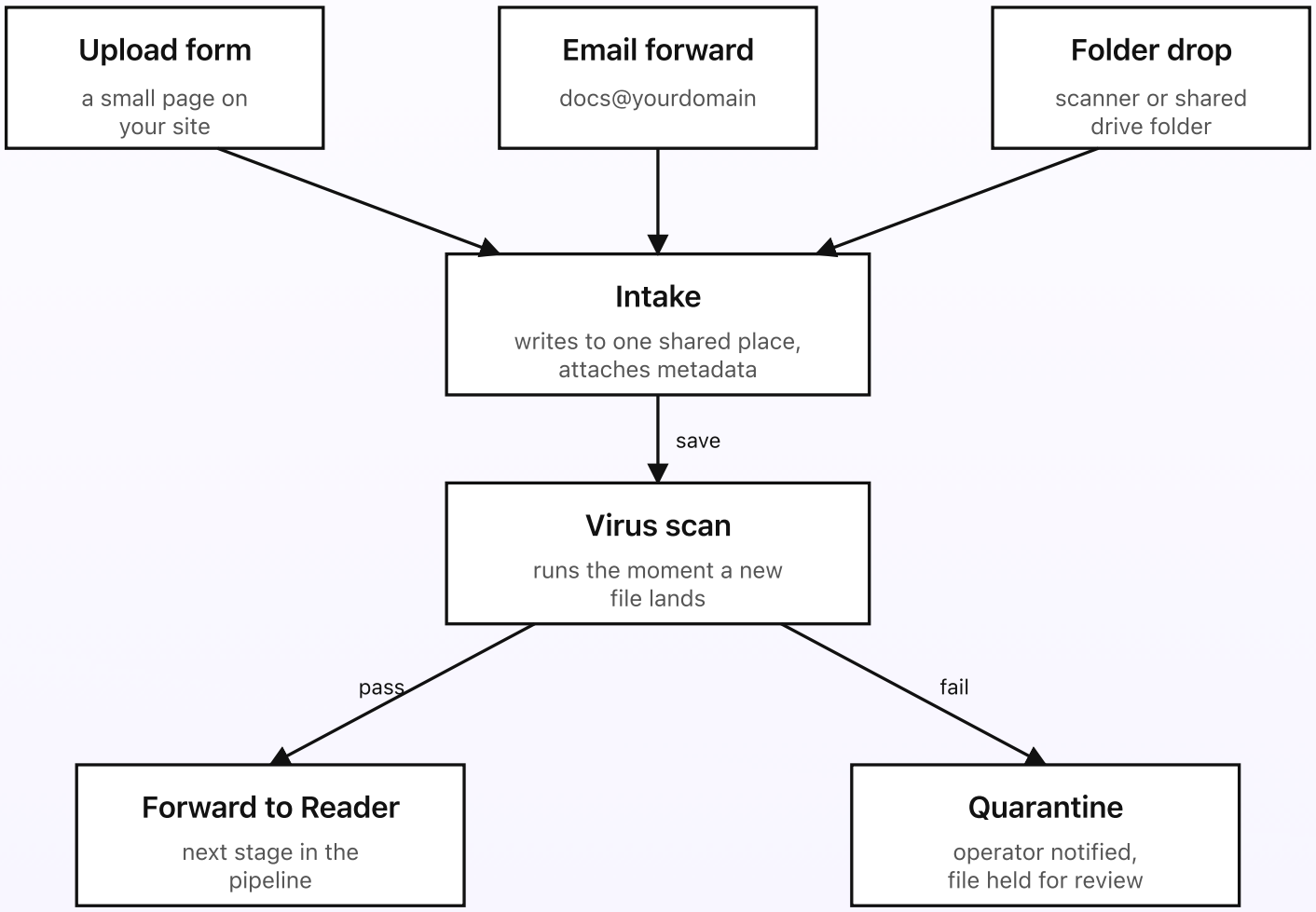
APRIL 27, 2026 PART 2 OF 7 · DOCUMENT PIPELINE SERIES ~4 MIN READ

How a document enters the pipeline

Documents arrive three ways — uploaded, emailed, or dropped into a shared folder. The intake step gets them into one shared place, screens them for trouble, and tells the rest of the pipeline what type each one is.

KEY TAKEAWAYS

- Three lanes at the door: an upload form, an email forward, and a shared folder drop.
- Every document lands in one shared place with a small label attached, regardless of source.
- The label travels through every step, so any field on any sheet traces back to its file.
- A virus scan runs before the AI reader spends a single cent reading a file.
- Failed scans are quarantined with an operator alert — the sender hears nothing automatic.



Every arriving document gets the same treatment, regardless of where it came from.

Fig 2. Three ways in, one shared mailbox, one safety check before anything else runs.

Three ways in

Different teams have different habits. The pipeline meets people where they are:

- **An upload form** — a small page on your site where a customer or staff member drags a file in. The simplest path. Good for one-off scans and customer submissions.
- **An email forward** — a real email address (something like `docs@yourdomain`). When you get an invoice in your inbox, you forward it. The pipeline picks it up automatically.
- **A folder drop** — the office scanner or a shared drive folder. Drop a file in; the pipeline takes it from there.

You don’t have to use all three. Most clients pick one and stick with it.

What gets attached at intake

The intake step doesn’t read the document yet. It just saves the file and writes a small label alongside it:

- Where the document came from (upload form, email, folder).
- Who sent it, if known (email sender, signed-in user).
- What time it arrived.

- A best-guess document type, if the source gives a hint (e.g. emailed invoices land in the “invoice” bucket by default).

That label travels with the document through every step that follows. If something goes wrong later, you can always see how it got there.

Why virus scan first

Documents arrive from outside your business. Some of them — not many, but some — arrive carrying things you don’t want. Before the AI reader spends a single cent reading a file, the system runs a quick virus scan.

Two outcomes:

- **Pass.** Almost everything passes. The document moves on to the next stage.
- **Fail.** The file is held in a quarantine spot, an alert goes to the operator, and nothing in the pipeline reads its contents. The original sender hears nothing automatic — that’s for the operator to decide.

What gets quarantined (and what happens then)

It’s rare. Most quarantines aren’t actual viruses — they’re corrupted scans, password-locked PDFs the scanner can’t open, or files in formats the system doesn’t support. The operator decides whether to release them, ask the sender to resend, or simply ignore.

The actual scanner is a small piece you can swap. The simplest version is an open-source scanner the pipeline runs itself. The cloud also offers a managed

version that does the same job without you running anything — it's the right call when it's available in your region and the pay-per-scan price fits your volume.

In plain words

Three ways in, one mailbox, one safety check. Once a document is past the virus scan, the rest of the pipeline can trust what it's reading. The label that intake attached travels with the document everywhere — so a year from now you can still trace any field on any sheet back to the exact file it came from.

PART 3 OF 7

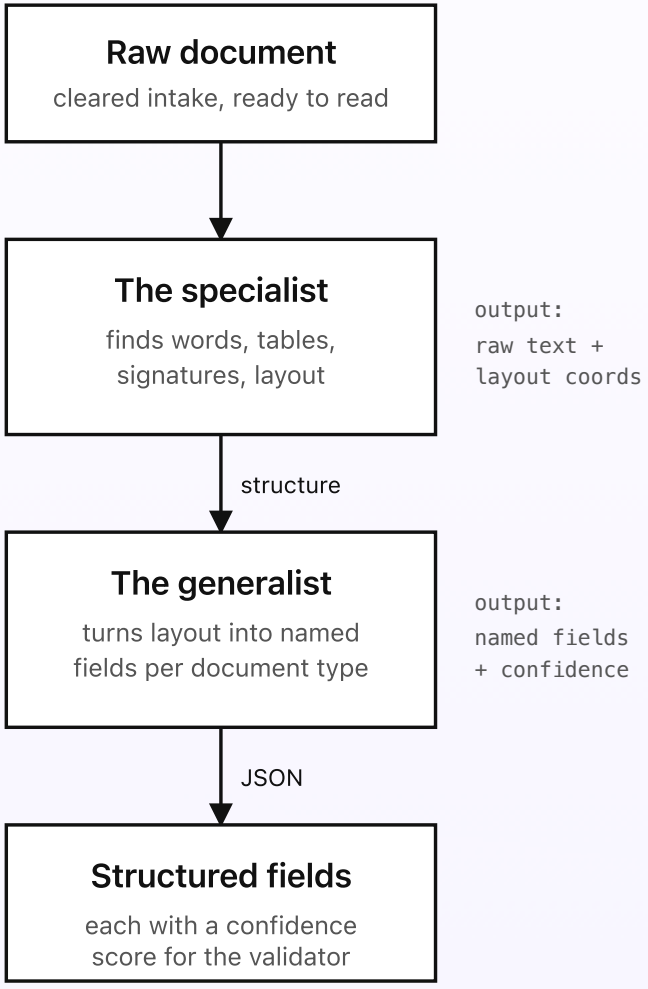
APRIL 27, 2026 PART 3 OF 7 · DOCUMENT PIPELINE SERIES ~5 MIN READ

How the AI reads a document

Reading a document is two jobs in one. First, find the words and the layout. Second, understand what they mean for this kind of document. The pipeline uses two AIs — one specialist, one generalist — to do each job.

KEY TAKEAWAYS

- Two AIs, one job each — a layout specialist for words and tables, a meaning generalist for fields.
- The specialist hands a clean map of the page to the generalist; neither does the other's job.
- Splitting the work is more accurate, cheaper, and easier to debug than one big AI doing both.
- Output is named fields with values in the right shape and a per-field confidence score.
- That confidence score is what the validator uses next to decide trust-or-review.



Two AIs, one job each. The combo is more accurate than either alone.

Fig 3. Layout specialist first, meaning generalist second. Output: structured fields the validator can score.

Two AIs, one job each

You could in theory hand the whole document to a single big AI and ask “please extract the fields.” It works — sometimes. It also costs more, makes more mistakes on tables and signatures, and gets worse the longer the document gets.

The pipeline splits the work in two. A small specialist does the boring half (where are the words?). A small generalist does the interesting half (what do they mean?). Each tool gets used for what it’s good at.

The specialist: layout

The first AI is built specifically for documents. It finds words on the page and tells you exactly where they are. It recognises tables and gives you each cell. It detects signatures, checkmarks, and form fields. It does this consistently — the same scan twice gives you the same result.

What it does not do: decide which words are the “invoice number” and which are the “customer reference.” That’s not its job. It hands the next stage a clean map of the page.

The generalist: meaning

The second AI is the kind that can be told what to look for in plain language. The pipeline tells it: “this is an invoice. Look for these fields: vendor name, invoice number, total amount, due date, line items.” The AI reads the layout from stage one and fills out the form.

Crucially, the generalist isn’t scanning the original document — it’s working from the specialist’s clean map. Faster, cheaper, more accurate.

Why two not one

Three reasons:

- **Accuracy.** Specialist tools beat generalist ones at the parts they specialise in. Asking a generalist to find table cells inside a scanned PDF is a recipe for hallucinated numbers.
- **Cost.** The specialist is cheap per page and predictable. The generalist is also cheap, but only when it’s working on a small clean map — not a giant blob of raw scan.
- **Trust.** When something goes wrong, you can tell which AI got it wrong. The split makes the system debuggable.

What comes out the other end

For each field the generalist extracts, you get three things:

- The **field name** (vendor, total, due date).
- The **value**, in the right shape (a number for “total”, a date for “due date”).

- A **confidence score** — how sure the AI is.

That confidence score is the next post's entire subject — it's how the validator decides whether a document goes straight through or needs a human to peek at it for two seconds.

In plain words

One AI finds the words. Another AI decides what they mean. Together they read a document better, faster, and cheaper than either could alone — and crucially, when they're wrong, the system knows.

PART 4 OF 7

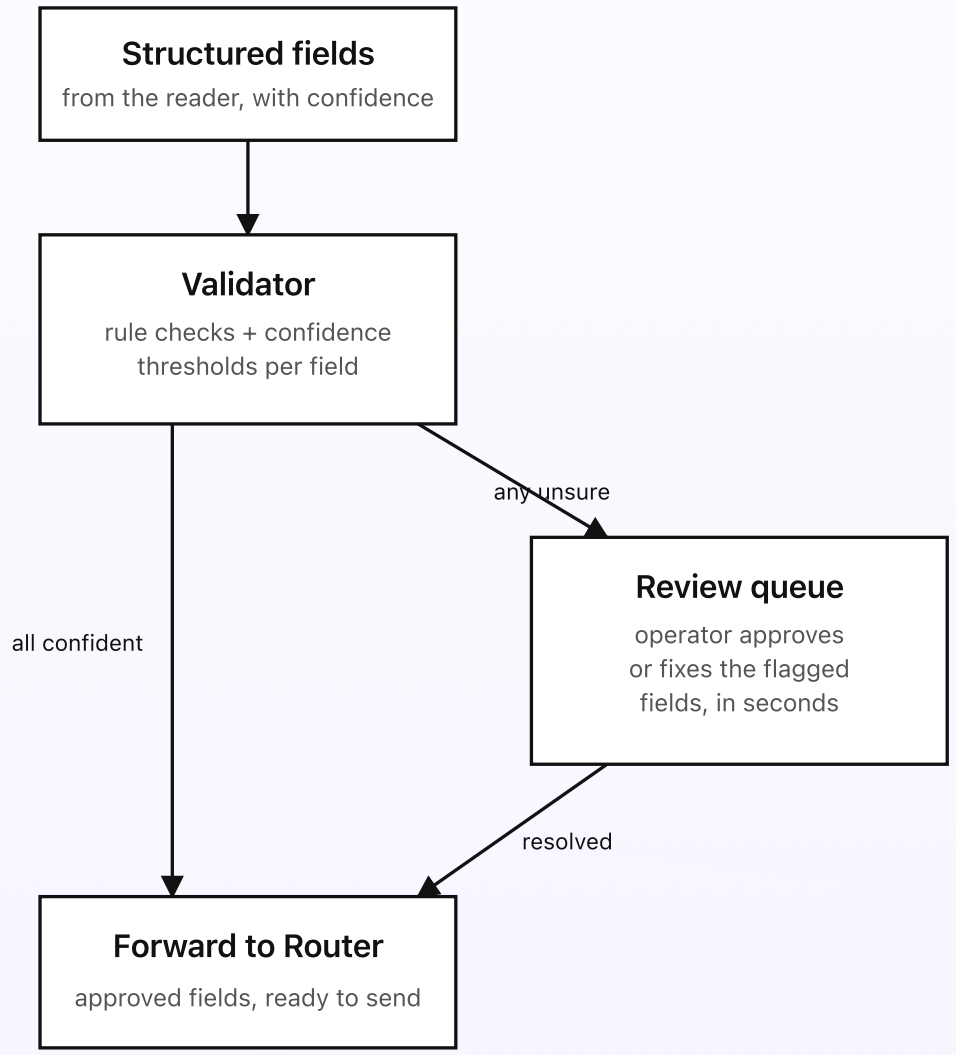
APRIL 27, 2026 PART 4 OF 7 · DOCUMENT PIPELINE SERIES ~5 MIN READ

How extraction stays accurate

AI gets things wrong. The trick is knowing when to trust it and when to ask for help. The validator scores each extraction, gates the obvious cases through, and queues the rest for a human review that takes seconds.

KEY TAKEAWAYS

- Two checks per document: do the rules hold, and is the AI sure enough on every field?
- Three verdicts — pass, review, reject. Most documents pass on their own.
- Per-field confidence thresholds are tunable: strict for amounts and dates, looser for free text.
- The review queue shows the original image next to flagged fields with one-click approve or fix.
- Wrong values never quietly slip into your tools — anything fuzzy hits a human gate first.



Most documents flow straight through. The few that need a human are quick to clear.

Fig 4. Confident extractions go straight through. Unsure ones get a 5-second human check, then merge back into the same flow.

Two checks, in order

For every extracted document, the validator runs two checks before letting it move on.

Check 1 — do the rules hold?

The rules file describes what each document type should look like:

- The **required fields** for this type (an invoice without a total isn't an invoice).
- The **shape** of each value (a date should look like a date; an amount should be a number).
- **Cross-checks** between fields where they apply (line items should add up to the total, give or take rounding).

These rules are written once, in plain language, and live in the same Drive folder as the rest of your config. They're free to run.

Check 2 — is the AI sure?

Every field came out of the reader with a confidence score — how sure the AI was about that specific value. The validator compares each score to a threshold for that field type (you can set strict thresholds for amounts and dates, looser ones for free-text notes).

Any field below its threshold raises a flag.

Three possible verdicts

- **Pass.** All rules hold, all fields are confident enough. The document moves on to the router untouched. Most documents land here.
- **Review.** A rule is questionable, or a field came in fuzzy. The document goes to a small review queue. An operator opens it, sees the flagged field highlighted next to the original document image, approves or fixes, moves on. Five seconds, maybe ten.
- **Reject.** The document is so badly extracted that the rules can't hold at all (e.g. an invoice with no recognisable line items). The document is sent back to the operator with a clear note saying what looked wrong, and the original sender hears nothing automatic.

Why the loop matters

This is the part that separates a useful document pipeline from a frustrating one.

Without the validator, the system would either stop trusting the AI entirely (everything goes to a queue, the human is the bottleneck again) or trust it too much (wrong totals quietly land in your books). The validator's job is to know when to trust the AI for free and when to spend five seconds of a human's time. Confidence scores plus rule checks make that decision precise.

What the operator actually sees

When a document lands in the review queue, the operator sees:

- The **original document image** on one side.

- The **extracted fields** on the other side, with the flagged ones highlighted.
- One button per flagged field: approve as-is, fix and approve, or reject.

No code, no terminal. The whole queue can be cleared between coffee sips.

In plain words

The AI is allowed to be confident, and the validator decides how to respond when it isn't. Most documents pass on their own. The few that don't are obvious to spot and quick to fix. Wrong values never quietly slip into your tools — they always hit a human gate first.

PART 5 OF 7

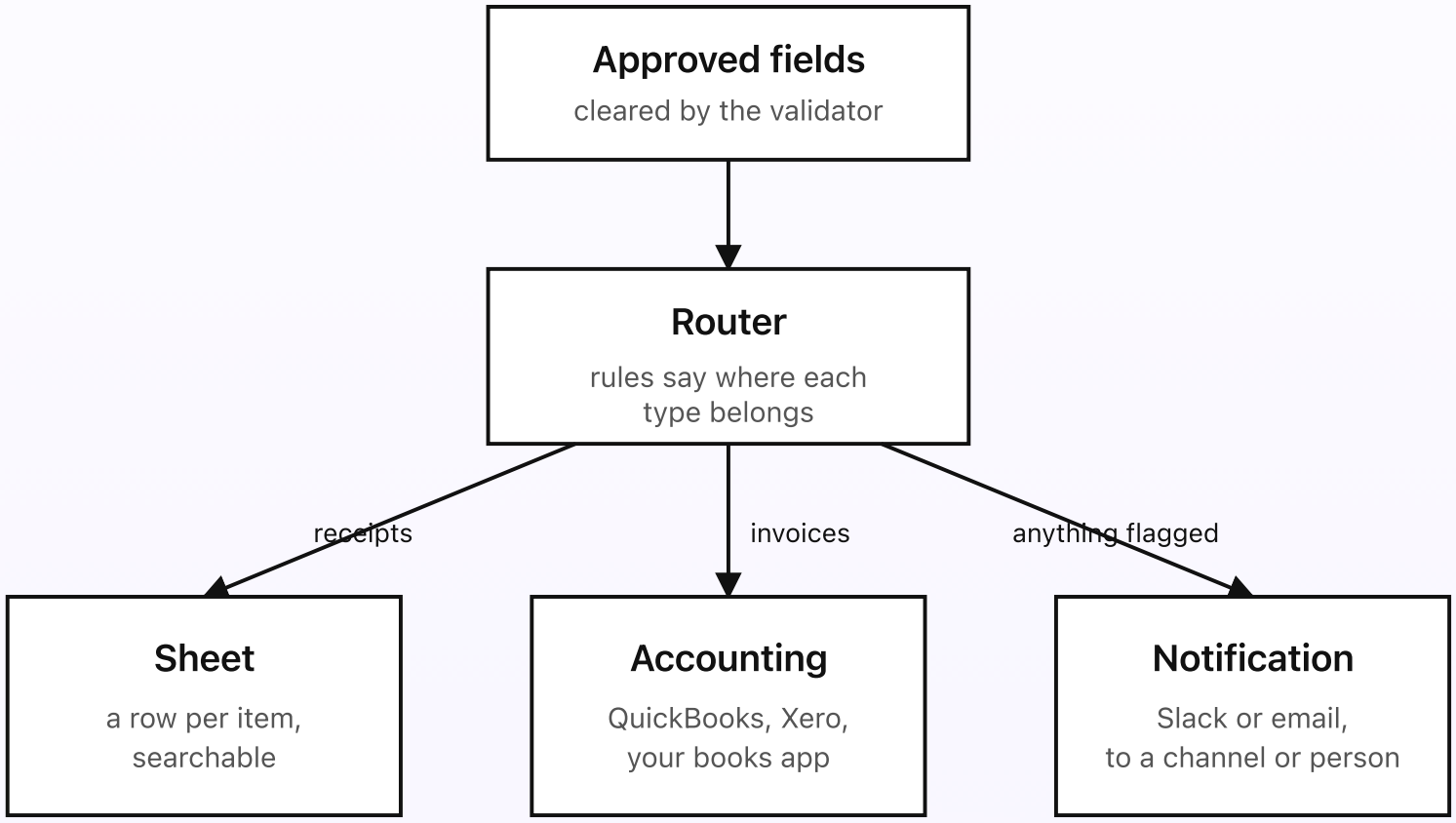
APRIL 27, 2026 PART 5 OF 7 · DOCUMENT PIPELINE SERIES ~4 MIN READ

How the data flows out

Once a document is read and validated, the structured data goes wherever it's actually useful — your sheet, your accounting software, a Slack channel. Each destination is a small pluggable piece.

KEY TAKEAWAYS

- The router fans out by document type to a sheet, accounting software, or a Slack notification.
- Routing is configured in the rules file — change destinations without a code deploy.
- Each destination is independent: one going down doesn't hold up the others.
- Failed sends retry with backoff, then land in a holding pen with a single short operator alert.
- A new destination is roughly twenty lines of code — the rest of the pipeline doesn't change.



Each destination is a small pluggable piece. Add new ones without touching the rest of the pipeline.

Fig 5. The router fans out by document type. Each destination is its own small piece — pluggable, removable, replaceable.

Three common destinations

You don't need to use all of these. Most clients pick one or two.

- **A sheet.** One row per processed document. Easiest to start with, easiest to share with a non-technical colleague. Good for receipts, expense lists, simple form data.
- **Your accounting software.** QuickBooks, Xero, or whatever already runs your books. The router talks to the software directly and writes the invoice or bill straight in — no manual re-entry.
- **A notification.** A Slack message or an email when something needs eyes — a flagged extraction, a contract that just got signed, an invoice over a certain amount.

Per-type routing

The rules file from earlier doesn't just describe what each document type looks like — it also says where each one should go.

For instance: receipts always go to the expense sheet. Invoices go to accounting *and* a Slack notification if the amount is above a threshold. Signed contracts go to a Drive folder and notify the legal contact. The router doesn't make these decisions itself — it reads the rules and follows them.

This means you can change where things land by editing the rules file. No code change, no deploy.

Failures and retries

Destinations live outside your system. Sometimes they're slow. Sometimes they're down. Sometimes they reject your request because they decided to require a new field this morning.

The router treats each destination call independently:

- If a call fails, it retries a few times with a small delay.
- If it still fails, the message goes to a holding pen for failed sends — not lost.
- The operator gets a single short notification: "3 documents couldn't reach your accounting software."

One destination going down doesn't hold up the others. The sheet still updates even when accounting is offline.

Adding a new destination

A new destination is a small bit of code that takes a structured field set and pushes it somewhere — one function, maybe twenty lines. The rest of the pipeline doesn't know or care that it exists.

So when a client says "can you also push these to our internal CRM?" the answer is yes, in an afternoon, without touching the reader, the validator, or any other destination.

In plain words

Structured data flows out the same way it came in — one piece at a time, where it's actually needed. The router is the smallest, dumbest part of the pipeline; the smarts already happened. From here on, it's plumbing.

PART 6 OF 7

APRIL 27, 2026 PART 6 OF 7 · DOCUMENT PIPELINE SERIES ~3 MIN READ

What the document pipeline costs

A pipeline that reads documents for you doesn't have to cost real money. Most of the system runs on free tiers; the only meaningful spend is the AI extraction itself, and even that's pennies per document.

KEY TAKEAWAYS

- About $1–$5/month at typical SMB volume of around 100 documents per month.
- Lambda runs, orchestration, webhook URLs, queues, alerts, and small tables all sit free.
- Only meaningful spend: page reading (pennies/page) and AI structuring (about a cent per doc).
- No always-on server, no per-document SaaS minimum, no infinite logs — three traps avoided.
- A $15 budget alarm catches anything weird before the bill becomes a surprise.

Where the dollars go in a typical month

ALWAYS FREE

$0

Lambda runs	`free`
Orchestration	`free`
Webhook URLs	`free`
Queues, alerts	`free`
Small tables	`free`

all under the perpetual free tier

TINY FIXED COST

~ $0.50/mo

S3 storage	`cents`
Password vault	`~$0.40 each`

flat, regardless of volume

GROWS WITH USE

~ $0.50–$4/mo

Reading pages	`pennies/page`
AI structuring	`~¢/doc`
Notifications	`<1¢/mo`

scales with how busy the inbox is

TOTAL, TYPICAL MONTH

about $1–$5 / month at SMB volume

~100 documents/month; budget alarm at $15 catches anything weird

More documents means more spend — but still cents per document, not dollars.

Fig 6. Three tiers of cost: free, tiny fixed, variable. The variable column is where most of the bill lives, and even there the unit cost is cents per document.

Free at this scale

- **The robots** — the cloud gives you a million runs a month for free, more than the pipeline will ever need.
- **Pipeline orchestration** — the workflow engine that strings the steps together charges per state transition. At SMB volume that's fractions of a cent per month — below the noise floor of the bill.
- **Webhook URLs** — the upload endpoint and the destination callbacks. Free at this volume.
- **Queues and alerts** — the small bookkeeping pieces. All free.
- **Small tables** — the audit log, the doc metadata. Comfortably under the always-free quota.

Costs pennies to a dollar each month

- **Storage** — the original documents and the structured outputs. Pennies per month at SMB volume; cheaper if you move old files to cold storage automatically.
- **Password vault** — about 40 cents a month for each secret you store (the access keys for your destinations).

Grows with how busy the inbox is

- **Reading pages.** The specialist AI charges per page. The default cheap mode (text + layout) rounds to pennies per page — great for receipts and most invoices. For forms-heavy documents that need precise table extraction, a richer mode is available at about a nickel per page; the rules file picks per type.
- **AI structuring.** The generalist AI works on the small clean output of the reader, not the original document — so even on long documents the cost stays around a cent or two per document.
- **Notifications.** Slack messages are free. Emails are a fraction of a cent.

Three traps you're avoiding

- **No always-on server** — would be $30+ a month before reading anything.
- **No paid document-reading service** — those charge a dollar or more per document and usually require a monthly minimum. The pipeline has no minimum and costs less per document.
- **No infinite logs** — 7-day retention; logs can't pile into a slow-growing surprise bill.

In plain words

For an SMB processing a hundred or so documents a month, the bill is a coffee. Push that to a thousand and the bill goes up — but per document it's still cents, not dollars. Set a budget alarm that fits the volume you actually expect, and the bill can't surprise you.

PART 7 OF 7

APRIL 27, 2026 PART 7 OF 7 · DOCUMENT PIPELINE SERIES ~3 MIN READ

Engineering reference: the document pipeline architecture

Same system as the rest of the series, drawn purely for engineers. Service names, resource identifiers, region, Bedrock model IDs, and the actual flow operations — everything you'd need to recreate this in your own AWS account.

KEY TAKEAWAYS

- Single AWS account, region `ap-southeast-1`, Bedrock invoked via Global cross-Region inference.
- Step Functions Express orchestrates virus scan, Textract, Bedrock Haiku 4.5, and the validator.
- Bedrock Structured Outputs (JSON schema) eliminate the "almost-valid JSON" class of bug.
- Three intake lanes (Function URL, SES inbound, S3 PutObject) all converge on one S3 prefix.
- CloudWatch logs at 7-day retention, $15 monthly budget alarm, weekly archive to Glacier IR.

Posts 1–6 walk through the system in plain language. This page is the dense version — no softening, just the architecture as you'd sketch it on a whiteboard during a design review.

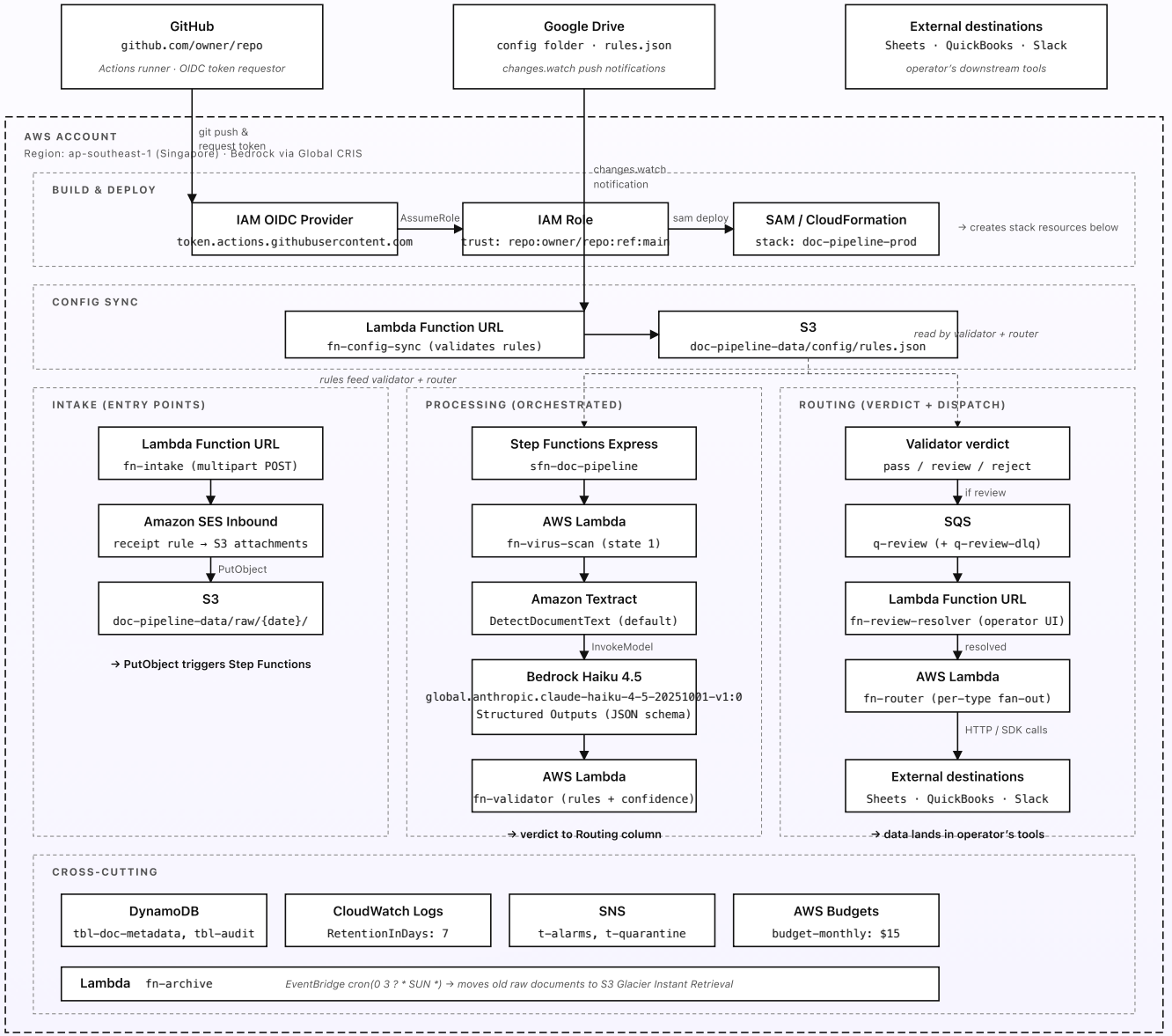

GitHub
github.com/owner/repo
Actions runner · OIDC token requestor
Google Drive
config folder · rules.json
changes.watch push notifications
External destinations
Sheets · QuickBooks · Slack
operator's downstream tools
AWS ACCOUNT
Region: ap-southeast-1 (Singapore) · Bedrock via Global CRIS
git push & request token
changes.watch notification
BUILD & DEPLOY
IAM OIDC Provider
token.actions.githubusercontent.com
AssumeRole
IAM Role
trust: repo:owner/repo:ref:main
sam deploy
SAM / CloudFormation
stack: doc-pipeline-prod
→ creates stack resources below
CONFIG SYNC
Lambda Function URL
fn-config-sync (validates rules)
S3
doc-pipeline-data/config/rules.json
read by validator + router
rules feed validator + router
INTAKE (ENTRY POINTS)
Lambda Function URL
fn-intake (multipart POST)
Amazon SES Inbound
receipt rule → S3 attachments
PutObject
S3
doc-pipeline-data/raw/{date}/
→ PutObject triggers Step Functions
PROCESSING (ORCHESTRATED)
Step Functions Express
sfn-doc-pipeline
AWS Lambda
fn-virus-scan (state 1)
Amazon Textract
DetectDocumentText (default)
InvokeModel
Bedrock Haiku 4.5
global.anthropic.claude-haiku-4-5-20251001-v1:0
Structured Outputs (JSON schema)
AWS Lambda
fn-validator (rules + confidence)
→ verdict to Routing column
ROUTING (VERDICT + DISPATCH)
Validator verdict
pass / review / reject
if review
SQS
q-review (+ q-review-dlq)
Lambda Function URL
fn-review-resolver (operator UI)
resolved
AWS Lambda
fn-router (per-type fan-out)
HTTP / SDK calls
External destinations
Sheets · QuickBooks · Slack
→ data lands in operator's tools
CROSS-CUTTING
DynamoDB
tbl-doc-metadata, tbl-audit
CloudWatch Logs
RetentionInDays: 7
SNS
t-alarms, t-quarantine
AWS Budgets
budget-monthly: $15
Lambda fn-archive
EventBridge cron(0 3 ? * SUN *) → moves old raw documents to S3 Glacier Instant Retrieval

Fig 7. Full architecture, ap-southeast-1. White boxes = AWS resources; dashed AWS container; dashed grey boxes = subsystem groupings; dashed grey arrows = config feed to runtime stages.

Read this top-down, then column-by-column

Top row is the three external surfaces. Below it, the AWS account contains five subsystems: Build & Deploy across the top, then Config Sync, then three runtime columns (Intake, Processing, Routing), with a Cross-cutting strip at the bottom. The dashed grey arrows from the Config Sync output to the validator and router show the only cross-subsystem data dependency — both read the latest rules from S3 on every invocation.

Naming conventions used in the diagram

- **Lambda functions:** `fn-<purpose>` — e.g. `fn-intake`, `fn-virus-scan`, `fn-validator`, `fn-router`.
- **Step Functions workflow:** `sfn-doc-pipeline` (Express type, since per-document runtimes stay well under 5 minutes).
- **DynamoDB tables:** `tbl-<name>` — `tbl-doc-metadata`, `tbl-review-queue`, `tbl-audit`.
- **SQS queues:** `q-<name>` with paired `q-<name>-dlq`.
- **SNS topics:** `t-alarms` for general failures, `t-quarantine` for virus-scan rejections.

- **S3 layout:** single bucket `doc-pipeline-data` with prefixes `config/`, `raw/{date}/`, `parsed/{date}/`, `structured/{date}/`, `quarantine/`.

Region and Bedrock model access

Everything runs in `ap-southeast-1` (Singapore) for low latency from the Philippines. Bedrock model invocations use the **Global cross-Region inference** profile (model IDs prefixed with `global.`) — data at rest stays in Singapore; inference may route to other regions for capacity. Pricing is the same as on-demand Singapore pricing.

Bedrock Anthropic models support **Structured Outputs** on the Converse and InvokeModel APIs (GA Feb 2026) — the pipeline uses a JSON schema to enforce the field shape per document type, eliminating the "the model returned almost-valid JSON" class of bug.

Textract calls are synchronous for single-page documents (`DetectDocumentText`) and asynchronous for multi-page or richer extraction (`StartDocumentAnalysis` → `GetDocumentAnalysis`), with the choice driven by document type in the rules file. Default is `DetectDocumentText` — cheap and works for most receipts and invoices, with Bedrock doing the heavy lifting on layout interpretation. `AnalyzeDocument` with `FORMS` and `TABLES` features is reserved for forms-heavy documents where the cost (about a nickel per page) is justified by the layout-extraction quality.

What's deliberately not on the diagram

- IAM policy details — per-Lambda execution role inline policies are minimal (one bucket prefix, one table, one queue as appropriate).
- Per-document-type schemas — the `rules.json` file contains schemas for each type (invoice, receipt, contract, etc.), the field thresholds, and the destination mappings.
- X-Ray tracing — on for the Step Functions workflow, sampling 100% during ramp-up, 10% in steady state.
- The CloudFormation parameter for the Bedrock model ID and the Textract feature is templated, so swapping models or features doesn't require code changes.
- **GuardDuty Malware Protection for S3** — an AWS-managed alternative to a custom ClamAV-on-Lambda virus scan. Cheaper to operate at low volume and zero infrastructure to maintain. Worth swapping in when available in your region.
- **Bedrock Nova as a single-stage alternative** — `amazon.nova-lite-v1:0` and `amazon.nova-pro-v1:0` can read PDFs directly via vision and return structured output in one call, skipping Textract entirely. Cheaper at low volume on simpler documents; the Textract+Haiku two-stage shape stays the default for accuracy on forms and tables.
- **Bedrock Batch Inference** — for nightly reprocessing or backfilling old documents under a new schema, Bedrock Batch is roughly half the cost of on-demand with a 24-hour SLA.

IF YOU'RE RECREATING THIS

Start with Build & Deploy alone (a single Lambda, no triggers). Once `git push` reliably updates an empty stack, add Intake next so you have a place to put your documents. Then the Processing pipeline against a single hard-coded type. Then the Validator with one rule. Then the Router pointing at one destination. Cross-cutting (audit, logs, alarms, budget, archive) goes in from day one.