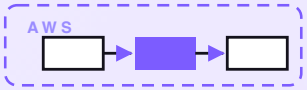
7-PART SERIES · FREE COMPANION



Daily briefing bot

An automated digest that watches the sources you care about and emails you the few items worth your time. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Daily briefing bot

PART 1 OF 7

APRIL 26, 2026 PART 1 OF 7 · DAILY BRIEFING BOT SERIES ~5 MIN READ

A daily briefing bot on AWS for a few dollars a month

You want to keep up with a dozen sources without spending an hour a day reading them. Here's how to build a small robot that reads everything for you, picks the few items worth your attention, and emails you a short digest each morning.

KEY TAKEAWAYS

- Three outside surfaces, three pieces inside AWS — one flow in, one flow out.
- Ingestor walks your source list each morning; ranker picks the keepers; postman writes the digest and sends.
- Source list and topic file live in Drive — edit a doc, save, tomorrow's digest reflects the change.
- Quiet days send nothing. The bot never trains you to ignore it.
- Runs on AWS for about $1–$3/month; most pieces sit on the always-free tier.

The whole system on one page

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

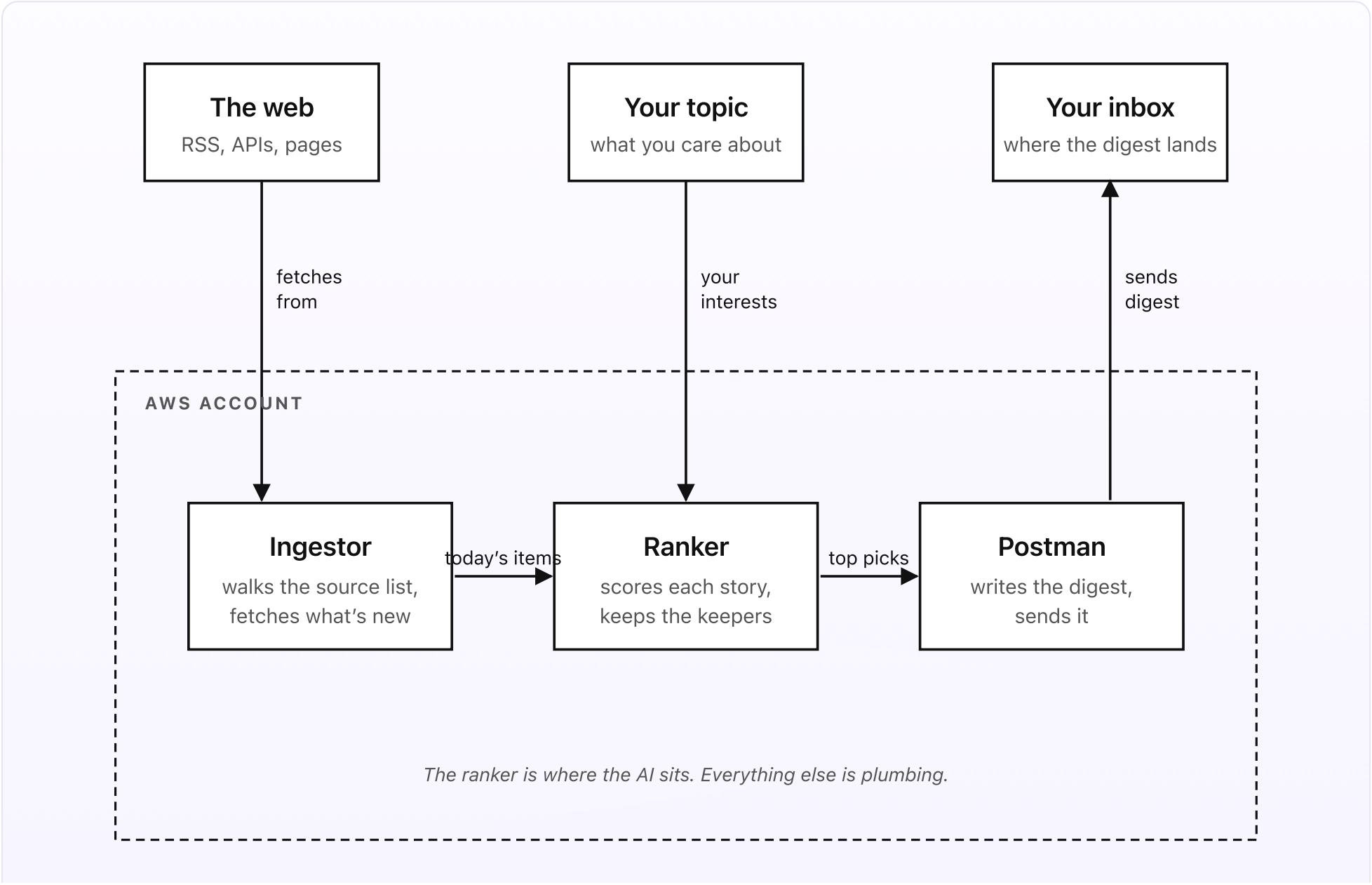

The web
RSS, APIs, pages
Your topic
what you care about
Your inbox
where the digest lands
fetches from
your interests
sends digest
AWS ACCOUNT
Ingestor
walks the source list, fetches what's new
today's items
Ranker
scores each story, keeps the keepers
top picks
Postman
writes the digest, sends it
The ranker is where the AI sits. Everything else is plumbing.

Fig 1. Three outside surfaces, three pieces inside AWS. One flow in, one flow out.

What you set up once (the outside)

- **A list of sources you watch** — the feeds, public services, and pages you'd otherwise refresh by hand. Lives in a single file you can edit anytime.
- **A short description of your topic** — a paragraph or two explaining what you actually care about. The bot reads this to decide what's worth your attention.
- **An inbox** — email, a chat channel, or a fresh doc each morning. Whichever you actually check.

What runs every morning (the inside)

- **The ingestor** — wakes up on schedule, walks your source list, fetches what's new since yesterday, drops it all in one place.
- **The ranker** — reads each item, scores it against your topic description, keeps only the few worth your time.
- **The postman** — takes the keepers, writes them up as a short digest, sends it.

In plain words

You list the sources. The cloud reads them every morning. A small AI sorts the noise from the signal. You get a short digest in your inbox before your first coffee.

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

DESIGN RULES THAT SHAPED EVERY DECISION

- Stay inside the AWS always-free service quotas wherever possible.
- No always-on server. No NAT Gateway. No infinite log retention.
- The ranker must be cheap by default — most items die at a free filter before any AI sees them.
- Empty inboxes are a feature. If nothing scores high enough on a quiet day, no digest goes out.
- Your source list lives somewhere you already know how to edit — so updating it never needs a deploy.

Why this shape

Most “news aggregator” tools collapse under one of three weights: a server bill that climbs every month, a noisy digest you start ignoring within a week, or a curation pipeline that costs more in AI calls than the time it saves.

The architecture above is the smallest set of moving parts I could find that solves all three at once. One way in (your sources), one way out (your inbox), a small AI in the middle making cheap decisions. Everything else is plumbing.

The next five posts walk through each piece in turn — how the ingestor works, where the AI fits, why most items never reach it, how the digest gets written and delivered, 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

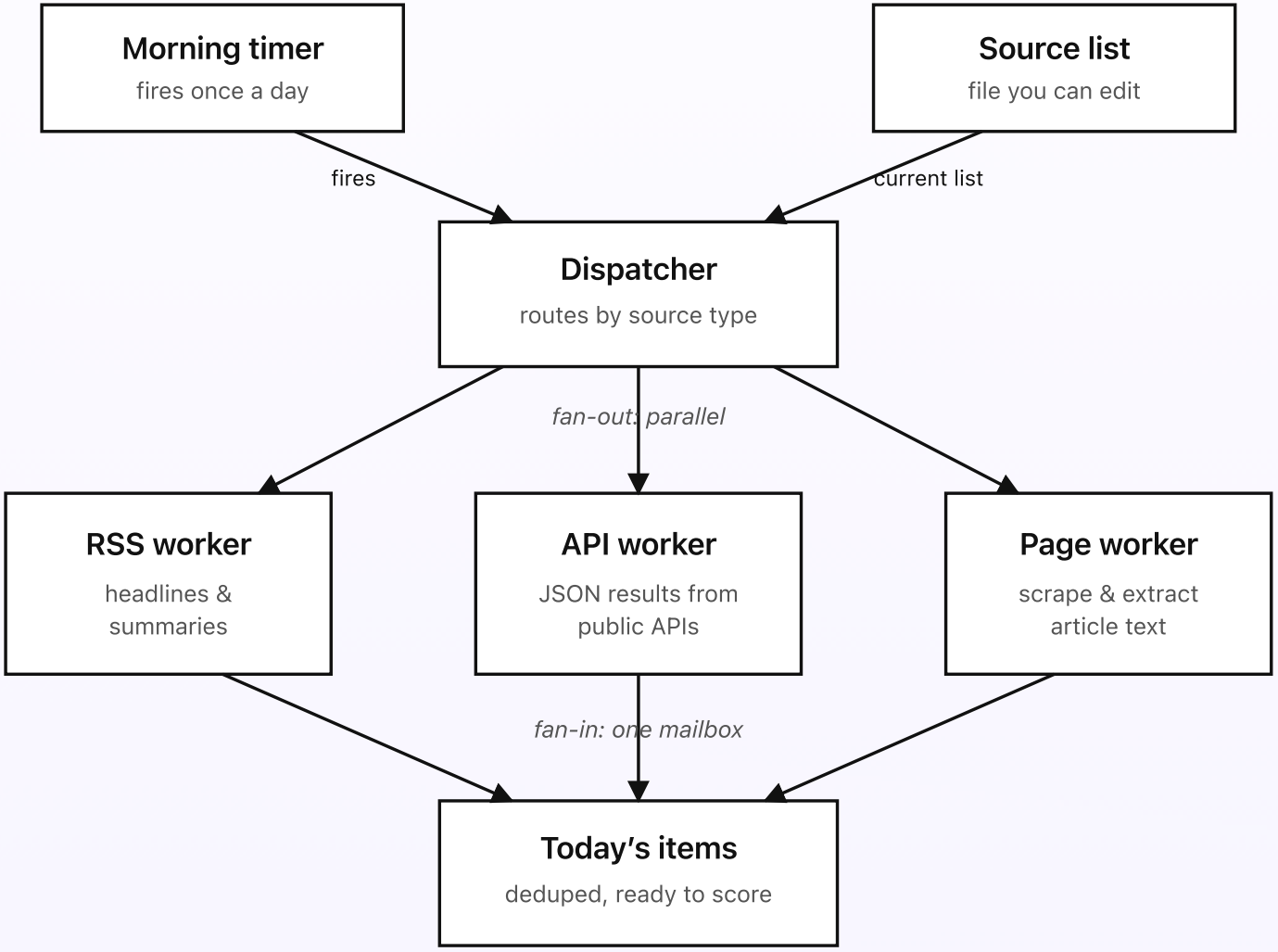
APRIL 26, 2026 PART 2 OF 7 · DAILY BRIEFING BOT SERIES ~4 MIN READ

How the ingestor walks your sources

Once a morning, a small dispatcher reads your source list, hands each source to a specialist worker, and drops everything new into one shared place. Three workers, one mailbox — the boring part of the system.

KEY TAKEAWAYS

- One morning timer fires the dispatcher, which reads the source list and routes by source type.
- Three specialist workers run in parallel: RSS feeds, public APIs, and plain web pages.
- Each item gets a URL-plus-publish-date fingerprint so duplicates never reach the ranker.
- One slow source can't hold up the others — failures are logged and the digest still ships.
- The ingestor never decides what's interesting; its only job is to fetch and dedupe.



Workers run in parallel. One slow source doesn't hold up the others.

Fig 2. The ingestor: fan-out across three worker types, fan-in to one shared mailbox.

Why three workers, not one

Not every source speaks the same language.

- **Feeds** — many sites publish a tidy stream of headlines and summaries the bot can read directly. The easy case.
- **Public services** — some sites give you the data through a public service in whatever shape they chose years ago. Needs a small adapter per service.
- **Plain web pages** — everything else. The bot pulls the article body out of the page and drops the navigation and the cookie banner.

You could write one worker that does all three. But then a single bad change breaks every source at once. Three small workers — one per kind — keep the failure contained. If the page worker chokes on a redesigned site, the other two keep flowing.

What “new” actually means

Each fetched item gets a short fingerprint — the URL plus the publish date. Before the worker does any further work, it asks: “have I seen this fingerprint before?” If yes, skip. If no, save the fingerprint and pass the item along.

This is how the bot reads the same feed every morning without flooding you with duplicates. It also means if a source republishes an old article under the same

address, the bot ignores it — same fingerprint.

Three things that go wrong (and what happens then)

- **A source is offline.** The worker waits a few seconds, tries twice more, then gives up and logs it. The other workers keep going. Tomorrow it'll try again from scratch.
- **A page changes its layout.** The page worker can't find the article body. Item is logged as "couldn't parse" and skipped. The digest still ships — you fix the parser later, not in a panic.
- **A source dumps a hundred items.** The worker takes them all and lets the next stage decide which to keep. The ingestor never makes editorial decisions. Its only job is to fetch.

In plain words

The ingestor is the boring part. It doesn't read the items, doesn't judge them, doesn't decide what's interesting. Its only job is "go to the URLs in this list, bring back what's new, put it in this one box." Three workers run at once so one slow site doesn't hold up the others. Whatever they bring home goes to the ranker — the next stop.

PART 3 OF 7

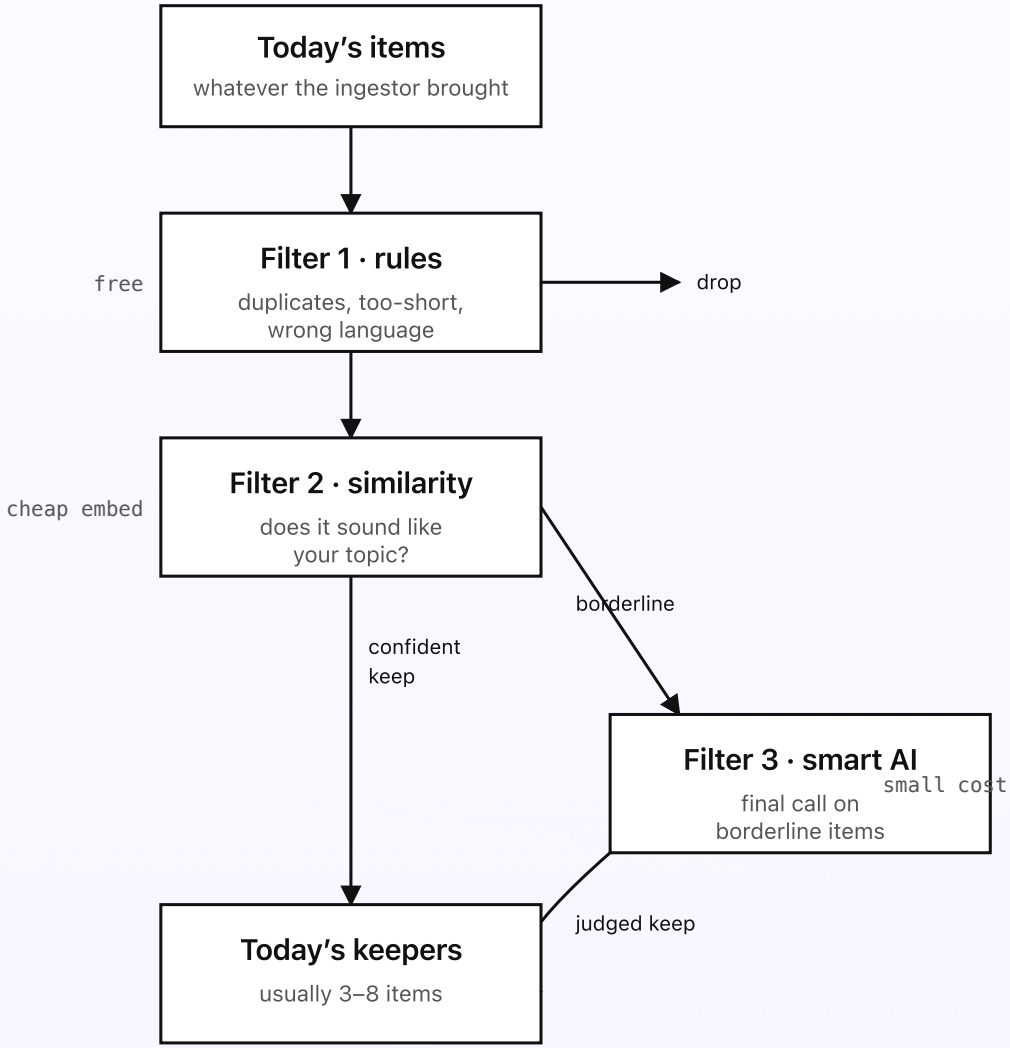
APRIL 26, 2026 PART 3 OF 7 · DAILY BRIEFING BOT SERIES ~5 MIN READ

How the bot picks what matters

Most items in the morning's haul aren't worth your time. The ranker's job is to throw out the noise as cheaply as possible and only pay for AI on the items that survive. Three filters in order, cheapest first.

KEY TAKEAWAYS

- Three filters in order: free rules, cheap embeddings, smart AI — an item must survive each one.
- Free rules clear roughly half the morning's haul (duplicates, too short, wrong language, muted source).
- Embeddings score every survivor against your topic for a fraction of a cent each, even on heavy news days.
- Only borderline items reach the smart AI — usually a handful per morning, not a hundred.
- What survives is typically three to eight keepers per day, occasionally zero.



Most items die at the free filter. The smart AI only sees the few borderline cases.

Fig 3. The ranker: three filters in order. Cheap rules first; smart AI last; most items never reach it.

Three filters, cheapest first

If you handed every fetched item to a smart AI, your bill would explode — and most of the AI's time would be spent looking at things that obviously don't fit. So the ranker runs three filters in order. An item only moves to the next filter if it survived the previous one.

Filter 1 — is this even worth looking at?

Free, runs in milliseconds. Throws out:

- Items already in your digest archive (duplicate)
- Items too short to carry meaning (a tweet-length blurb with no body)
- Items in the wrong language for your topic
- Items from sources you've quietly muted

This usually clears out roughly half the morning's haul without any AI involved.

Filter 2 — does it sound like your topic?

A small AI converts both your topic description and each item into a list of numbers. The closer the lists, the more related the item is to your topic.

Items well above the threshold pass through as confident keepers. Items well below it are dropped. Items in the borderline middle get one more look.

The cost is a fraction of a cent per item, even on a heavy news morning.

Filter 3 — is it actually relevant?

For the borderline items only, a smarter AI reads the title and summary and makes the final call. It returns one of:

- **Keep** — relevant, include in digest.
- **Skip** — not actually a fit on closer reading.
- **Skip but explain** — borderline, log the reason so you can tune your topic later.

Only borderline items hit this filter. On a typical morning that's a handful, not a hundred — which is the entire point.

What survives

Whatever made it through becomes the digest — usually three to eight items, occasionally zero. Zero is fine. The next post explains why empty digests are a feature, not a bug.

In plain words

Most items die at the free filter. A small AI sorts most of the rest. The smart AI only sees the few borderline cases. The whole pipeline costs pennies a day, even when

the news cycle is loud. Cheap rules first, smart AI last — the same shape used by serious content guardrails everywhere, applied to a different job.

PART 4 OF 7

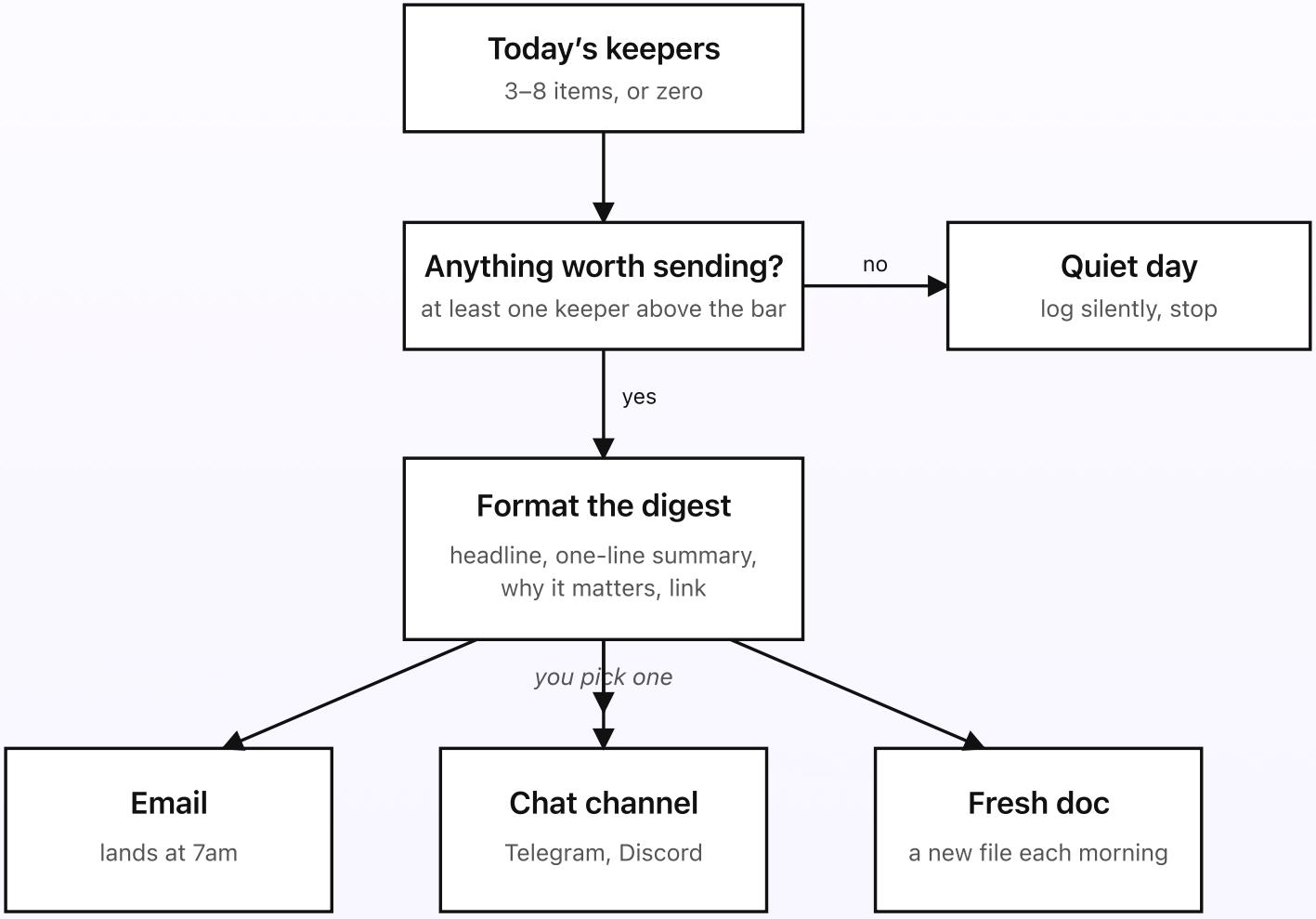
APRIL 26, 2026 PART 4 OF 7 · DAILY BRIEFING BOT SERIES ~3 MIN READ

How the digest gets delivered

The keepers from the ranker still need to become a thing you'll actually open. The postman writes the digest, ships it to whatever inbox you check first thing — and knows when not to send.

KEY TAKEAWAYS

- Each keeper becomes four lines: headline, one-line summary, why it matters, link.
- You pick one delivery channel — email, a chat channel, or a fresh doc each morning.
- If the ranker returned zero keepers, no email goes out. Quiet days stay quiet.
- The subject line is the headlines of the top two keepers, so you know whether to open before you do.
- Five short blocks read in under a minute — the digest you'll actually open.



If nothing scored high enough, no digest goes out. The bot never trains you to ignore it.

Fig 4. Format, then ship — or stay silent. One channel only; the bot doesn't spray.

What goes in the digest

Each item becomes a small block with four lines:

- **Headline** — the original title.
- **One-line summary** — the bot's compression of the article in plain English.
- **Why it matters** — one sentence on how it relates to your topic.
- **Link** — straight to the source.

That's it. Five items, twenty lines, you read the digest in under a minute.

Where it gets sent

You pick one delivery channel when you set up the bot:

- **Email.** Most common. Lands in your inbox at 7am. Pennies a month at this volume.
- **A chat channel.** Telegram, Discord, or similar. Useful if your inbox is already chaos.
- **A fresh doc.** A new Google Doc every morning. Searchable history without inbox clutter.

The bot only ships to one channel. Adding a second means another delivery worker; usually not worth it.

The quiet-day rule

If the ranker returned zero keepers, the bot doesn't send a "your daily digest (0 items)" email. It logs the empty result, goes back to sleep, tries again tomorrow.

This sounds obvious but most aggregators get it wrong — they ship the same empty email every quiet day until you unsubscribe. The bot never trains you to ignore it.

Why the subject line matters

The subject line summarizes the digest — the headlines of the top two keepers, separated by a slash. So at a glance you know whether to open it now, later, or not at all. The body is the full version.

If you open the email three weeks in a row, the bot is doing its job. If you stop opening, the threshold gets adjusted — or the topic description in your Drive folder needs a tune-up. The next post explains how that happens without touching code.

In plain words

Five short blocks in your inbox at 7am. Nothing on quiet days. Subject line tells you what's inside before you open it. Whichever inbox you actually check is the one to send to — the bot doesn't care, but you do.

PART 5 OF 7

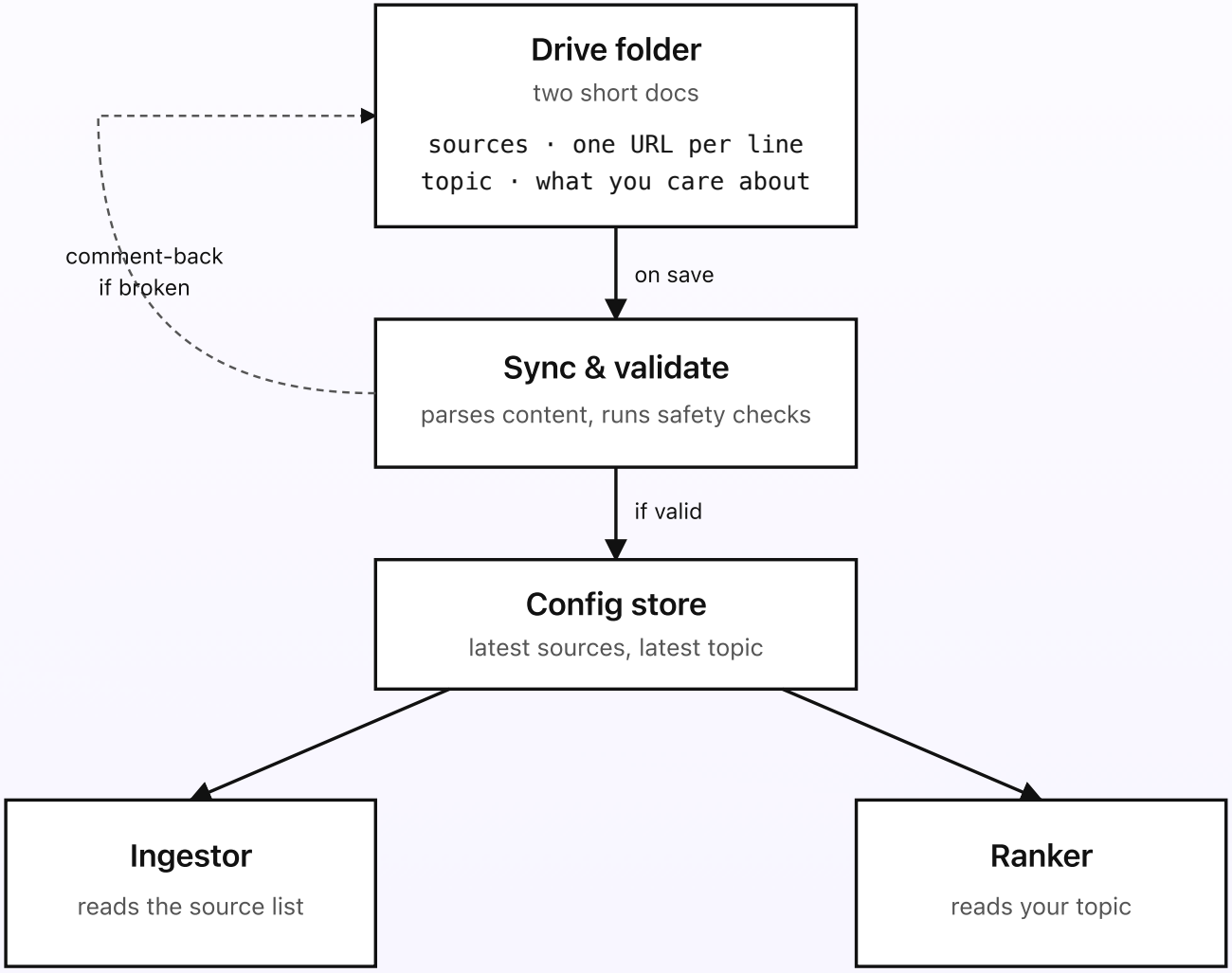
APRIL 26, 2026 PART 5 OF 7 · DAILY BRIEFING BOT SERIES ~3 MIN READ

How you change what the bot watches

Your interests change. So does the list of sources worth reading. The bot is built so you can update either one in 30 seconds without touching code, opening a terminal, or deploying anything.

KEY TAKEAWAYS

- Two Drive docs are the whole interface: a sources file (one URL per line) and a topic file (plain English).
- Saving a doc pings a sync worker that validates the new content before promoting it.
- If a change is broken, the old version stays live and you get a comment in the doc explaining why.
- Both robots read the latest config on every run — tomorrow's digest reflects today's edit.
- No deploy, no terminal, no admin panel; configuration is as easy to change as a shopping list.



Edit a doc, save, tomorrow's digest reflects the change. No deploy, no waiting.

Fig 5. The bot reads its config from Drive. The dashed loop is the safety net: bad changes never go live.

Two files you edit

In a small Drive folder are two short documents:

- **sources** — a list of links, one per line. Optionally a tag for source type (feed, service, or page).
- **topic** — a paragraph or two describing what you actually care about. Plain English. The ranker reads this every morning when scoring items.

That's the entire interface. No dashboard, no admin panel. You already know how to edit a Doc.

What happens when you save

The moment you save either file, Drive pings the cloud. A small worker pulls the new version, makes sure it parses (URLs are URLs, the topic isn't empty), and saves it to a place both robots read from. The next morning's run picks it up automatically.

You don't deploy anything. You don't need to know it worked. Tomorrow's digest will just behave differently.

If you break something

If you paste a malformed URL or accidentally clear the topic, the safety check stops the bad version from going live. The old version stays in use. You get a small note back in the doc explaining what looked wrong — “line 8 isn’t a valid URL”, “topic file is empty”.

Fix the typo, save again, same flow. No downtime, no panic.

In plain words

Edit a Drive doc. Save. Tomorrow’s digest reflects your change. No git push, no deploy, no waiting on anyone. The bot’s configuration is exactly as easy to change as a shopping list — because that’s what it is.

PART 6 OF 7

APRIL 26, 2026 PART 6 OF 7 · DAILY BRIEFING BOT SERIES ~3 MIN READ

What the briefing bot costs

A small daily digest doesn't have to cost real money. Most of the system runs on free tiers; the only meaningful spend is the AI for ranking, and even that's pennies. Here's where the dollars actually go.

KEY TAKEAWAYS

- Lambda runs, the morning timer, queues, alerts, and small tables all sit inside the AWS always-free tier.
- Tiny fixed cost: pennies of S3 storage and ~$0.40/month per Secrets Manager secret.
- Variable cost: cheap embeddings, the smart-AI judge on borderline items, and SES email — cents to about a dollar a month.
- No always-on server, no managed scraper, no infinite logs — the three traps a daily digest usually falls into.
- Total: about $1–$3/month at typical volume; a $5 AWS Budget alarm catches anything weird before it grows.

Where the dollars go in a typical month

ALWAYS FREE

$0

Lambda runs	free
Morning timer	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 activity

GROWS WITH USE

~ $0.50–$2.00/mo

Cheap embeddings	pennies
Smart AI (judge)	cents to ~$1
Email send (SES)	<1¢/mo

scales with how loud the news is

TOTAL, TYPICAL MONTH

about $1–$3 / month

budget alarm at $5 catches anything weird before it grows

A coffee a month, possibly less. The bot only spends when it's actively reading and ranking.

Fig 6. Three tiers of cost: free, tiny fixed, variable. Most of the bill is the variable column — and that column is still pennies on a quiet news week.

Free at this scale

- **The robots** — the cloud gives you a million function runs a month for free. The bot uses a tiny fraction of that.
- **The morning timer** — 14 million wake-ups a month free. The bot needs 30.
- **Queues and alerts** — the small bookkeeping pieces are all on free tiers at this volume.
- **Small tables** — the small lists the bot keeps (which items it's already seen, which digests it's sent) fit comfortably in the always-free tier.

Costs pennies to a dollar each month

- **Storage** — the raw items, today's digest, the config files. Cents per month.
- **Password vault** — about 40 cents a month for each secret you store (any access keys for the sources you watch).

Grows with how busy your sources are

- **Quick scoring** — the small AI looks at each item for about a fraction of a cent. A heavy news day with 200 items still rounds to pennies.
- **Smart AI for borderline ranking** — about a tenth of a cent per call. Only borderline items get this treatment, so it's usually a handful per day. A few

cents to a dollar a month.

- **Email delivery** — about a tenth of a cent per send (priced per thousand messages). A daily digest is fractions of a cent per month.

Three traps you're avoiding

- **No always-on server** — that alone would be $30+ a month before doing anything.
- **No managed scraper service** — the page worker is a small bit of code that runs only when there's something to fetch. Managed scraping services charge per page and add up fast.
- **No infinite logs** — 7-day retention; logs can't pile into a slow-growing surprise bill.

In plain words

A coffee a month, possibly less. The bot only spends money when it's actively reading and ranking; the rest of the day it costs nothing. Set a budget alarm at $5 a month and you'll know within a day if anything weird ever starts to happen.

PART 7 OF 7

APRIL 26, 2026 PART 7 OF 7 · DAILY BRIEFING BOT SERIES ~3 MIN READ

Engineering reference: the briefing bot 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 in `ap-southeast-1` (Singapore); Bedrock via Global cross-Region inference.
- Five subsystems: Build & Deploy, Config Sync, Ingestor (scheduled fan-out), Ranker (3-stage filter), Postman (event-driven on S3 PutObject).
- Models: `global.anthropic.claude-haiku-4-5-20251001-v1:0` in JSON mode for borderline items, plus `amazon.titan-embed-text-v2:0` for similarity.
- Two crons: ingestor at 06:00 SGT (`cron(0 22 * * ? *)`), ranker at 06:15 SGT (`cron(15 22 * * ? *)`); postman is S3-event-driven.
- Drive `changes.watch` push notifications drive config sync; `fn-watch-renewer` runs weekly to renew before the 7-day expiry.

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.

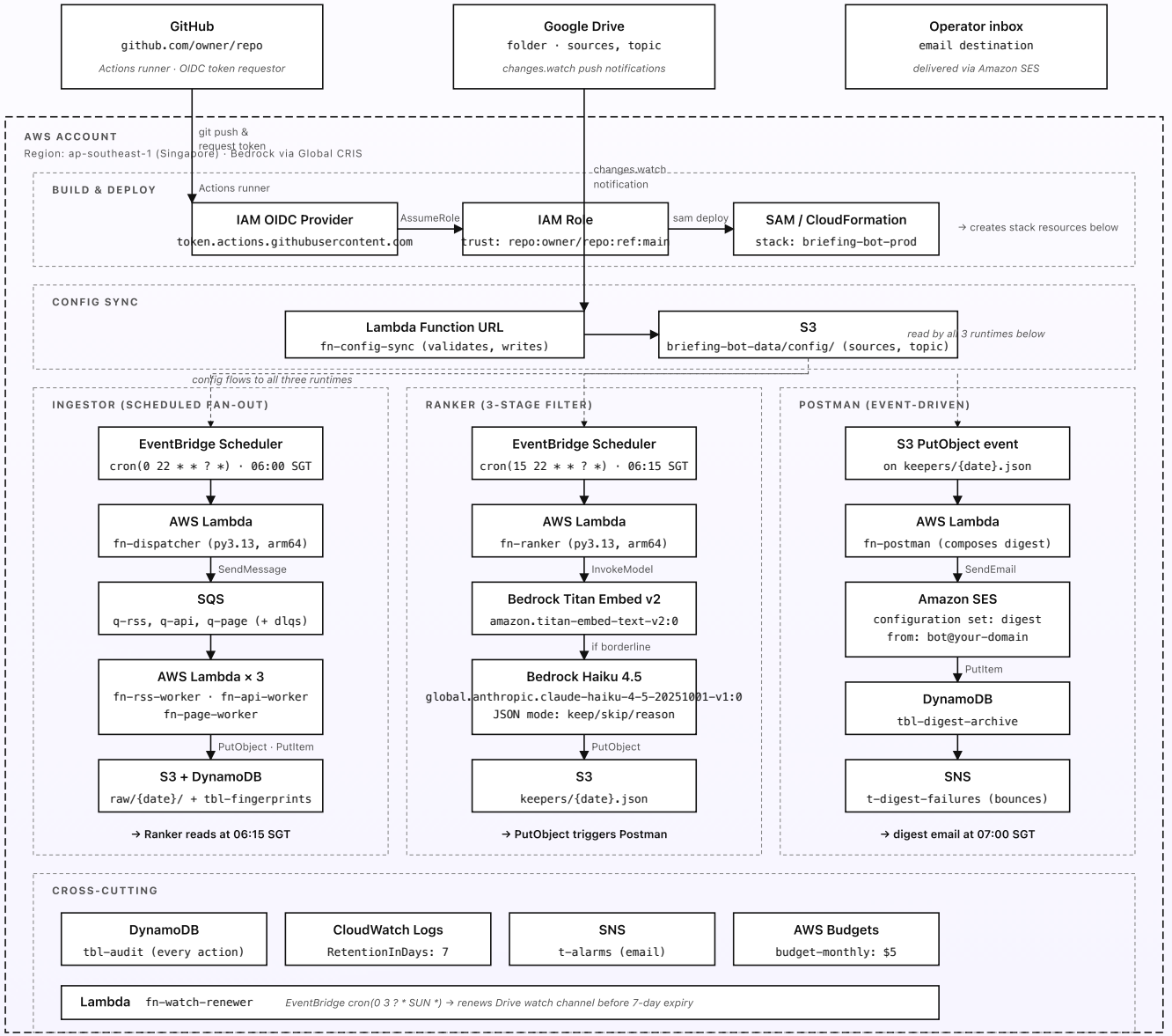

GitHub
github.com/owner/repo
Actions runner · OIDC token requestor
Google Drive
folder · sources, topic
changes.watch push notifications
Operator inbox
email destination
delivered via Amazon SES
AWS ACCOUNT
Region: ap-southeast-1 (Singapore) · Bedrock via Global CRIS
git push & request token
changes.watch notification
BUILD & DEPLOY
Actions runner
IAM OIDC Provider
token.actions.githubusercontent.com
AssumeRole
IAM Role
trust: repo:owner/repo:ref:main
sam deploy
SAM / CloudFormation
stack: briefing-bot-prod
→ creates stack resources below
CONFIG SYNC
Lambda Function URL
fn-config-sync (validates, writes)
S3
briefing-bot-data/config/ (sources, topic)
read by all 3 runtimes below
config flows to all three runtimes
INGESTOR (SCHEDULED FAN-OUT)
EventBridge Scheduler
cron(0 22 * * ? *) · 06:00 SGT
AWS Lambda
fn-dispatcher (py3.13, arm64)
SendMessage
SQS
q-rss, q-api, q-page (+ dlqs)
AWS Lambda × 3
fn-rss-worker · fn-api-worker
fn-page-worker
PutObject · PutItem
S3 + DynamoDB
raw/{date}/ + tbl-fingerprints
→ Ranker reads at 06:15 SGT
RANKER (3-STAGE FILTER)
EventBridge Scheduler
cron(15 22 * * ? *) · 06:15 SGT
AWS Lambda
fn-ranker (py3.13, arm64)
InvokeModel
Bedrock Titan Embed v2
amazon.titan-embed-text-v2:0
if borderline
Bedrock Haiku 4.5
global.anthropic.claude-haiku-4-5-20251001-v1:0
JSON mode: keep/skip/reason
PutObject
S3
keepers/{date}.json
→ PutObject triggers Postman
POSTMAN (EVENT-DRIVEN)
S3 PutObject event
on keepers/{date}.json
AWS Lambda
fn-postman (composes digest)
SendEmail
Amazon SES
configuration set: digest
from: bot@your-domain
PutItem
DynamoDB
tbl-digest-archive
SNS
t-digest-failures (bounces)
→ digest email at 07:00 SGT
CROSS-CUTTING
DynamoDB
tbl-audit (every action)
CloudWatch Logs
RetentionInDays: 7
SNS
t-alarms (email)
AWS Budgets
budget-monthly: $5
Lambda fn-watch-renewer
EventBridge cron(0 3 ? * SUN *) → renews Drive watch channel before 7-day expiry

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 runtimes.

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 (Ingestor, Ranker, Postman), with a Cross-cutting strip at the bottom. The dashed grey arrows from the Config Sync output to each runtime column show the only cross-subsystem data dependency — all three runtime Lambdas read the latest config from S3 on every invocation.

Naming conventions used in the diagram

- **Lambda functions:** `fn-<purpose>` — e.g. `fn-dispatcher`, `fn-ranker`, `fn-postman`.
- **DynamoDB tables:** `tbl-<name>` — e.g. `tbl-fingerprints`, `tbl-digest-archive`, `tbl-audit`.
- **SQS queues:** `q-<name>` with paired `q-<name>-dlq`. One queue per source type.
- **SNS topics:** `t-<name>` — `t-alarms` for general failures, `t-digest-failures` for SES bounces.
- **S3 layout:** single bucket `briefing-bot-data` with prefixes `config/`, `raw/{date}/`, `keepers/{date}.json`, `archive/`.

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.

Crons are written in UTC (`cron(0 22 * * ? *)` = 06:00 SGT next day) so the digest lands before the operator's morning coffee.

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-channel delivery alternates — if you use Telegram or Slack instead of SES, the Postman column substitutes a Lambda Function URL call to the platform's API and a corresponding secret in Secrets Manager.
- X-Ray tracing — on for the Ranker Lambda only, sampling 10% — useful when tuning the similarity threshold.
- The CloudFormation parameter for Bedrock model ID is templated, so swapping models doesn't require code changes.

IF YOU'RE RECREATING THIS

Start with Build & Deploy alone (a single Lambda, no triggers). Once `git push` reliably updates an empty stack, add Config Sync next so you have a place to put your sources and topic. Then the Ingestor, then the Ranker, then the Postman — each on a real source, end to end, before adding the next. Cross-cutting (audit, logs, alarms, budget, watch renewer) goes in from day one.