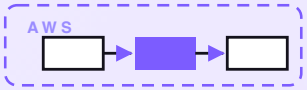
7-PART SERIES · FREE COMPANION



AWS autoposting

A scheduled Facebook poster that stays on-topic, answers questions correctly from a Google Drive knowledge base, and doesn't surprise you with a huge cloud bill. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

AWS autoposting

PART 1 OF 7

APRIL 25, 2026 PART 1 OF 7 · AWS AUTOPOSTING SERIES ~5 MIN READ

A Facebook autoposting system on AWS for $2–$5 a month

You run a Facebook page. You want it to post on a schedule, stay on-topic, answer questions correctly, and not surprise you with a huge cloud bill. Here's how to build that.

KEY TAKEAWAYS

- Three outside surfaces, two small robots, one shared brain — the smallest moving-parts count that solves the whole job.
- Posting and replying read from the same Drive-backed knowledge base, so the page never contradicts itself.
- Replies must cite a knowledge-base entry or refuse — no invented prices, promos, or promises.
- Built on always-free AWS pieces wherever possible: no NAT Gateway, no API Gateway, no always-on compute.
- Total cost lands at $2–$5 per Facebook page per month at steady volume.

The whole system on one page

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

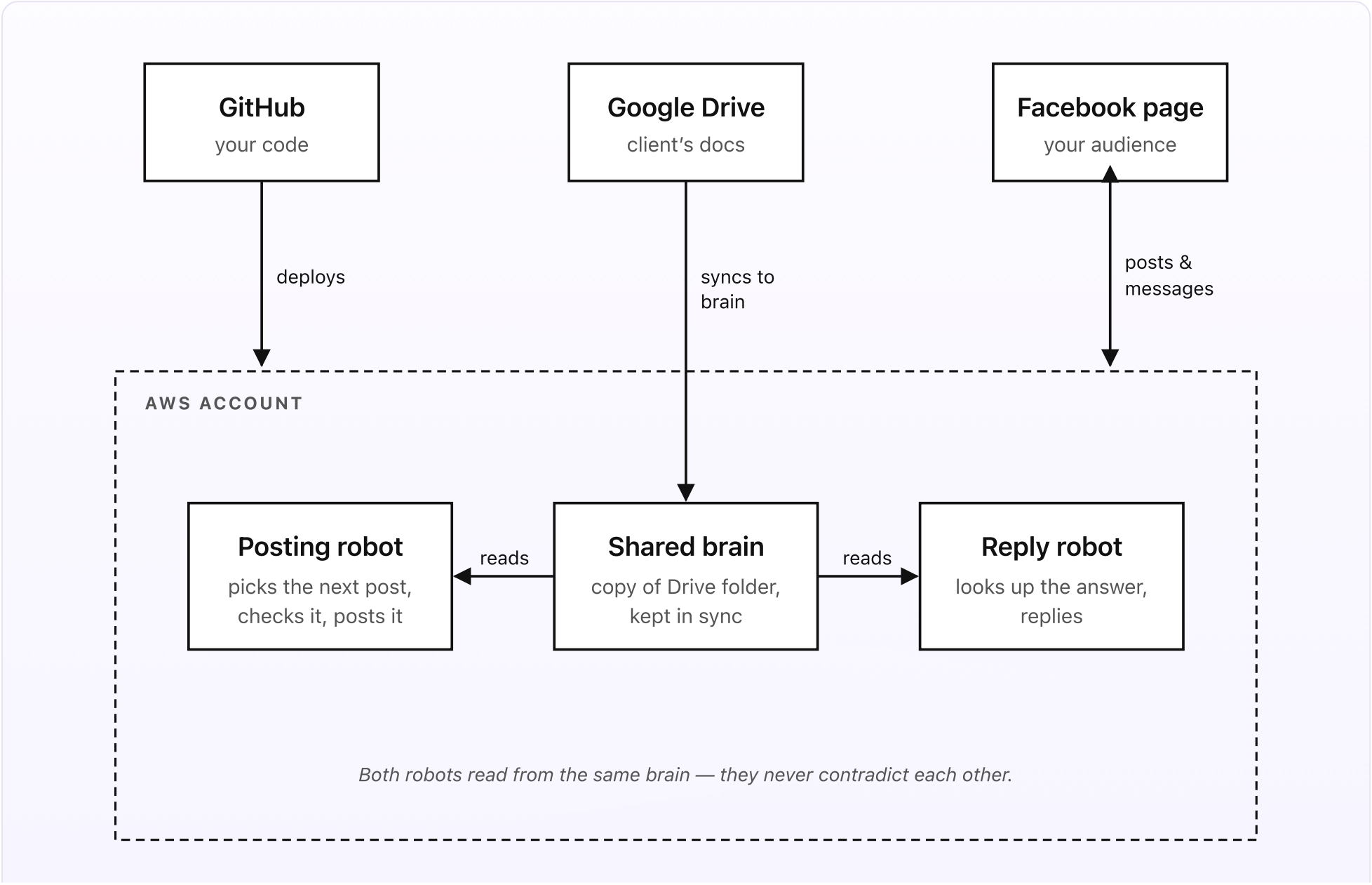

GitHub
your code
Google Drive
client's docs
Facebook page
your audience
deploys
syncs to brain
posts & messages
AWS ACCOUNT
Posting robot
picks the next post, checks it, posts it
reads
Shared brain
copy of Drive folder, kept in sync
reads
Reply robot
looks up the answer, replies
Both robots read from the same brain — they never contradict each other.

Fig 1. Three outside surfaces, three pieces inside AWS.

What you and your client touch (the outside)

- **Your code on GitHub** — where the project lives.
- **A Google Drive folder** — where the client edits content (FAQs, pricing, promos, brand rules).
- **The Facebook page** — what the world sees.

What runs quietly in the cloud (the inside)

- **The posting robot** — wakes up on schedule, picks the next post, checks it, posts it.
- **The reply robot** — wakes up when someone messages, looks up the answer, replies.
- **The shared brain** — a copy of the Drive folder, kept in sync, that both robots read from.

In plain words

You write the rules in code. Your client writes the content in Drive. Two small robots — one for posting, one for replying — do the work in the background. They both share the same brain so they never contradict each other.

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 NAT Gateway. No API Gateway. No always-on compute. No infinite log retention.
- No long-lived credentials in GitHub.
- The client's editing surface is Google Drive — they don't learn a new tool.
- Replies must cite a knowledge-base entry or refuse to answer. No hallucinated prices, no invented promises.

Why this shape

Most "autoposting" systems collapse under one of three weights: a server bill that climbs every month, a content-management workflow no one outside the dev team can use, or a chatbot that confidently invents answers.

The architecture above is the smallest set of moving parts I could find that solves all three at once. Three external surfaces, two small robots, one shared brain. Everything else is plumbing.

PART 2 OF 7

APRIL 25, 2026 PART 2 OF 7 · AWS AUTOPOSTING SERIES ~3 MIN READ

How code becomes a working system

You save your work, push to GitHub, walk away. The cloud handles the rest — and never holds a long-lived password.

KEY TAKEAWAYS

- Push to GitHub Actions, walk away — CloudFormation rolls back on failure, so a bad deploy fixes itself.
- OIDC swaps long-lived AWS keys for ~1-hour credentials issued only when the badge says “repo: yourname/yourrepo, branch: main.”
- A different repo, branch, or fork PR gets zero credentials — the trust boundary is the badge, not a stored secret.
- Nothing to leak, nothing to rotate — expired credentials are the default, not the exception.
- The whole stack is described in one `template.yaml`: Lambdas, schedules, queues, tables — one source of truth, no console clicking.

Here’s the path your code takes from your laptop to a running system in AWS.

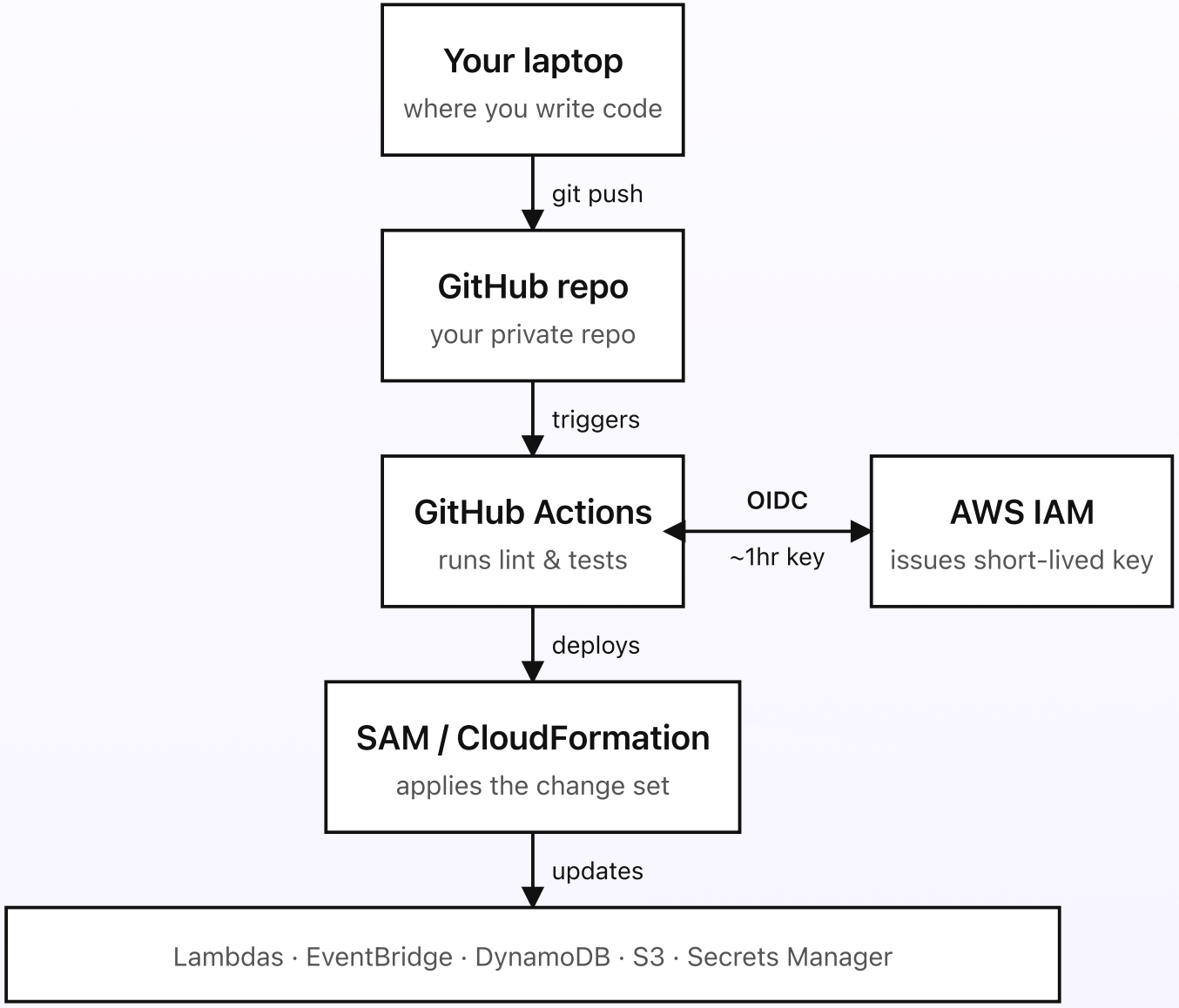

Your laptop
where you write code
git push
GitHub repo
your private repo
triggers
GitHub Actions
runs lint & tests
OIDC
~1hr key
AWS IAM
issues short-lived key
deploys
SAM / CloudFormation
applies the change set
updates
Lambdas · EventBridge · DynamoDB · S3 · Secrets Manager

Fig 2. Push to GitHub, the cloud updates itself — with a key that expires in an hour.

The old way vs the new way

Old way. Copy files to a server, SSH in, restart things, hope nothing breaks. Store the AWS password in GitHub Secrets so the deploy script can use it — and pray it never leaks.

New way. Push to GitHub, walk away. The cloud handles the rest.

If something breaks, you undo your change and push again. The cloud rolls back too — CloudFormation reverts to the previous state automatically.

Why no passwords are stored anywhere

The most common cause of compromised AWS accounts is a long-lived access key sitting in a GitHub Secrets vault — checked in by accident, shared with the wrong person, or scraped by a bot.

There's a better way. Instead of storing a key, GitHub Actions sends AWS a short-lived ID badge that says "I am the build job for this exact repo, on this exact branch." AWS checks the badge and hands back temporary credentials that expire in about an hour. (The standard for this back-and-forth is called OIDC — OpenID Connect — if you want to look it up.)

Nothing to leak. Nothing to rotate. If the temporary key ever ended up somewhere it shouldn't, it's already expired before anyone could use it.

> **THE TRUST BOUNDARY, IN ONE LINE**
>
> AWS only hands out credentials when the badge says **“repo: yourname/yourrepo, branch: main”**. A different repo, a different branch, or a pull request from someone else’s fork — no credentials. No deploy.

What gets deployed

Everything is described in one file: `template.yaml`. SAM (Serverless Application Model) is a thin layer over CloudFormation that lets you describe Lambdas, schedules, queues, and tables in a few dozen lines instead of a few hundred. One `sam deploy` applies the entire stack: a single source of truth, no clicking around in the AWS console.

PART 3 OF 7

APRIL 25, 2026 PART 3 OF 7 · AWS AUTOPOSTING SERIES ~4 MIN READ

How a post actually goes out

Five quick checks between “it’s time” and “post is live.” Cheap gates first, expensive gates only when needed.

KEY TAKEAWAYS

- Five gates between scheduler and Facebook: schema, keyword, similarity, LLM judge, verdict routing.
- Cheap checks first — most posts pass stages 1–2 for free; the LLM judge only fires on ~5% of borderline cases.
- Schema rejects a faith post tagged for a forex page before any AI runs — the wrong-page mistake is caught for zero cents.
- Three possible endings: PASS publishes, REVIEW emails you for approval, BLOCK is logged for later tuning.
- Per-page rules (allowlist, denylist, similarity thresholds) live as JSON in S3 — tune without redeploying.

The posting robot wakes up on a schedule, picks the next queued post, and runs it through a small gauntlet before letting it touch Facebook.

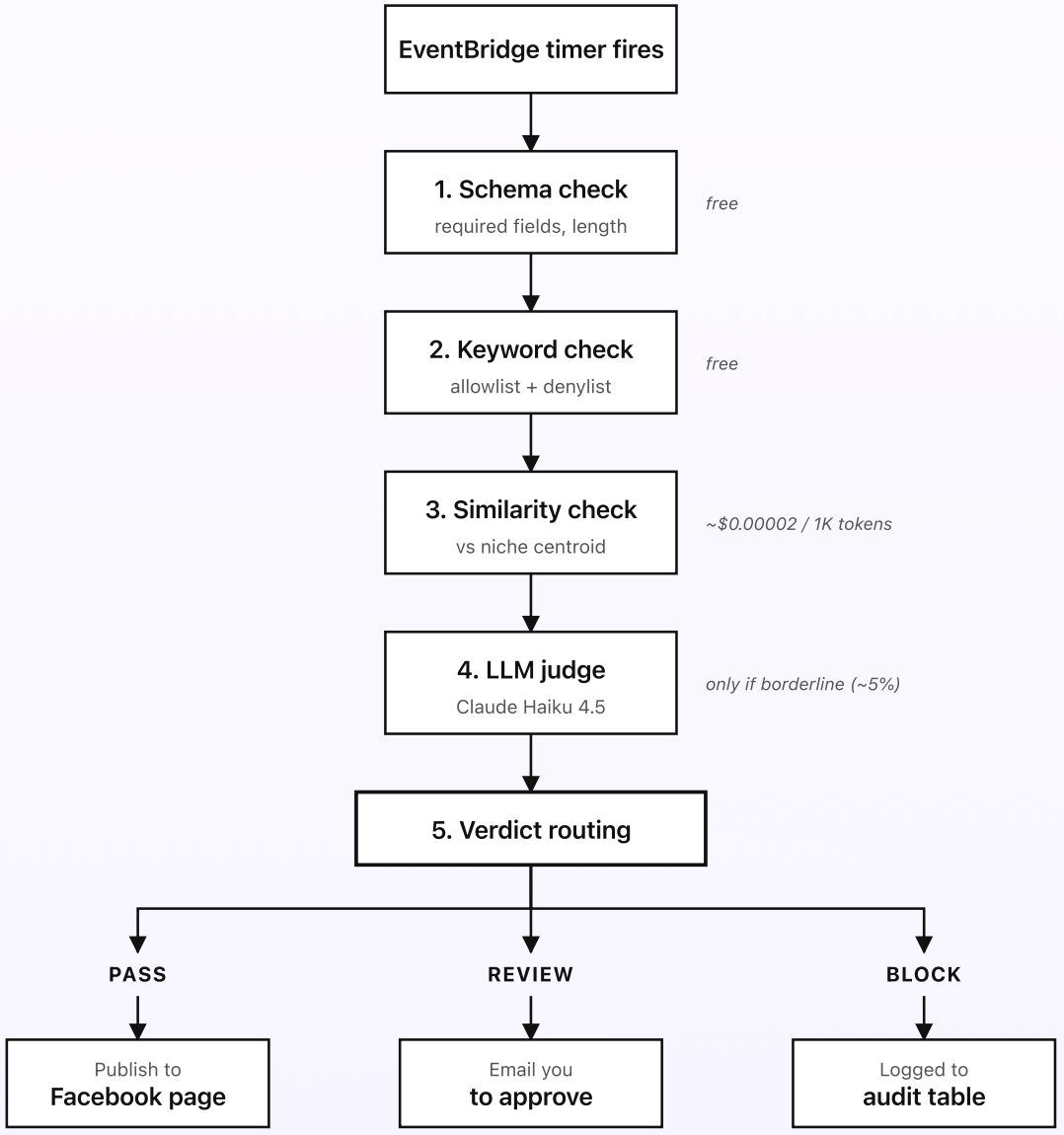

EventBridge timer fires
1. Schema check
required fields, length
free
2. Keyword check
allowlist + denylist
free
3. Similarity check
vs niche centroid
~$0.00002 / 1K tokens
4. LLM judge
Claude Haiku 4.5
only if borderline (~5%)
5. Verdict routing
PASS
REVIEW
BLOCK
Publish to
Facebook page
Email you
to approve
Logged to
audit table

Fig 3. Five gates between “it’s time” and “post is live.”

Cheap checks first, expensive checks only when needed

Most posts pass all five stages in well under a second. A typo or a missing required hashtag is caught for free in stage 1 or 2 — no AI runs, no money spent. The cheap embedding model only weighs in at stage 3, and that costs a fraction of a cent per post. Only when the score lands in the borderline band does the smarter, more expensive judge get called — about five percent of posts in steady state.

Why bother?

The point isn’t paranoia — it’s catching the moment when a faith post accidentally gets queued against the DailyScalper page (because someone got distracted while drafting), before the world sees it. The schema gate alone catches that: a post tagged with the wrong page is rejected at stage 1, no AI required.

The keyword denylist catches the next layer: a post containing “XAUUSD” or “FTMO” cannot reach a faith page, no matter how it slipped through.

The semantic check is the safety net for everything else: subtle drift, off-brand voice, content that’s technically allowed but doesn’t fit the page’s identity.

Three possible endings

- **PASS** — the post goes to the Graph API, gets a published timestamp, and the audit table records it.
- **REVIEW** — you get an email with the post text and an approve link. Your call.
- **BLOCK** — the post is marked blocked in the queue with a reason. Never posts. Logged for review so you can tune the rules later.

> **PER-PAGE RULES LIVE IN S3, NOT IN CODE**
>
> Each page has a small JSON config in S3: which hashtags it requires, which terms it blocks, what its niche centroid is, where its similarity thresholds sit. Tuning the rules — raising a threshold, adding a denylist term — doesn't require a redeploy. Edit the JSON, save, the next post uses the new rules.

PART 4 OF 7

APRIL 25, 2026 PART 4 OF 7 · AWS AUTOPOSTING SERIES ~4 MIN READ

How the Drive folder powers everything

The client edits a Google Doc. The system updates itself — with a safety layer that keeps the old version live if anything looks broken.

KEY TAKEAWAYS

- Four documents, one fixed shape: FAQs, pricing, promos, don't-say list — the system always knows where to look.
- A validator runs before any save reaches S3 — broken markdown or missing pricing rows are rejected with a Drive comment.
- Replies read from S3, never from Drive — a Drive outage or rate limit can't take the customer-facing path down.
- S3 versioning gives one-click rollback on the raw KB bucket — no backup ritual, no engineer at 2am.
- The client's entire interface is the Google Doc itself — live-as-of timestamps and rejection comments land in the doc they're editing.

The client's editing surface is Google Drive — a tool they already know. They never see a dashboard, never click a deploy button, never file a ticket. They open a doc, edit, save. A few seconds later the live system reflects the change.

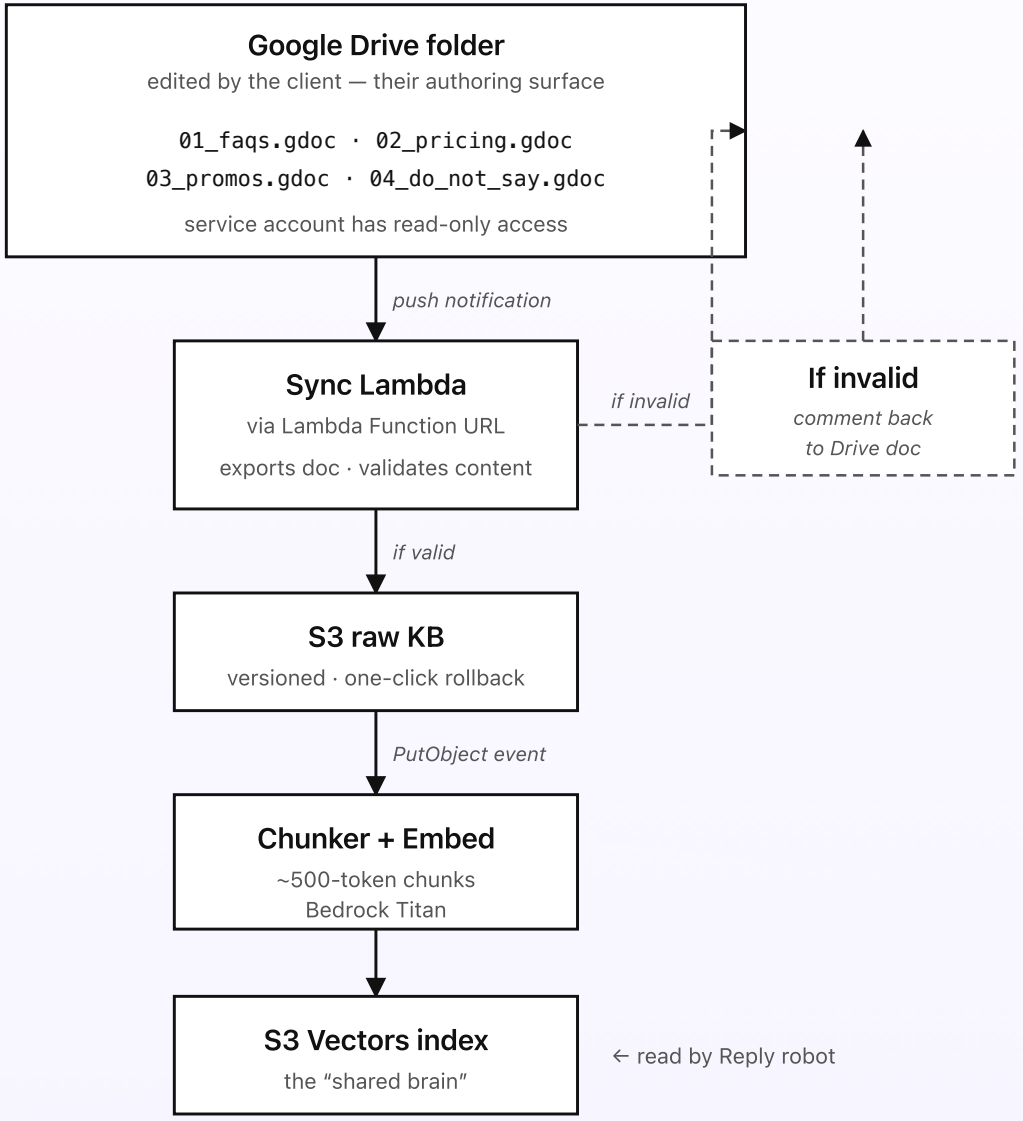

Google Drive folder
edited by the client — their authoring surface
01_faqs.gdoc · 02_pricing.gdoc
03_promos.gdoc · 04_do_not_say.gdoc
service account has read-only access
push notification
Sync Lambda
via Lambda Function URL
exports doc · validates content
if invalid
If invalid
comment back
to Drive doc
if valid
S3 raw KB
versioned · one-click rollback
PutObject event
Chunker + Embed
~500-token chunks
Bedrock Titan
S3 Vectors index
the "shared brain"
← read by Reply robot

Fig 4. Drive is the source of truth; S3 is what the live system reads from.

Four docs, one job

The folder always has the same four documents:

- **FAQs** — the questions customers ask all the time, with the actual answers.
- **Pricing** — service tiers, what each costs, what's included.
- **Promos** — current campaigns with start and end dates.
- **Don't-say list** — topics to avoid, things to escalate to a human, refusals.

The shape never changes. Only the contents do. The system knows where to look for what.

Why a safety check first

If the client breaks something — deletes the pricing table, leaves a section blank, pastes formatting that breaks the parse — the live system shouldn't go down with them. So before anything is saved to S3, a small validator runs:

- Markdown parses cleanly
- Required sections are present
- Pricing table is structurally intact

If the validator complains, nothing is saved. The Sync Lambda writes a comment back into the Drive doc itself: "Pricing table looks empty — check rows 4–6." The

old version of the knowledge base stays live until the doc is fixed.

> **WHY DRIVE ISN'T ON THE HOT PATH**
>
> Replies read from S3, never from Drive directly. A Drive outage doesn't take the live page down. Drive's API rate limits can never throttle the customer-facing reply path. Drive is the editing surface; S3 is the runtime surface. They're decoupled by design.

Versioning is free safety

S3 versioning is enabled on the raw KB bucket. Every save is a new version. Rolling back a bad change is one click in the AWS console — no "restore from backup" ritual, no recovery point objective math, no engineer needed at 2am.

The client's feedback loop is the doc itself

The client never logs into AWS. They never see a dashboard. The only place they get feedback is the doc they're editing:

- *"Live as of 14:23"* — their change is now in production.
- *"Pricing table is missing values in row 5"* — their change was rejected; here's why; here's where to look.

That's the entire interface. They already know how to use Google Docs. There's nothing else to learn.

PART 5 OF 7

APRIL 25, 2026 PART 5 OF 7 · AWS AUTOPOSTING SERIES ~5 MIN READ

How replies work without making things up

The bot answers from the client's docs only — or escalates to a human. Citation required, no exceptions.

KEY TAKEAWAYS

- The Meta webhook is acked in milliseconds; the slow RAG work happens off the critical path on SQS.
- A confidence gate refuses to answer when the top vector match scores below threshold — no answer beats a guessed answer.
- Claude Haiku 4.5 is instructed to answer using ONLY retrieved chunks and to return the IDs of the chunks it used.
- The citation guardrail blocks any reply that didn't cite a chunk — the structural guarantee against hallucinated prices.
- Every new client starts in draft mode — replies route to your inbox until you trust each category, then you flip them to auto-send.

AI reply bots have a bad reputation, and they've earned it. They quote prices that don't exist, promise features that aren't real, give medical or financial advice they shouldn't. The fix isn't to make the model bigger — it's to take away its freedom to make things up.

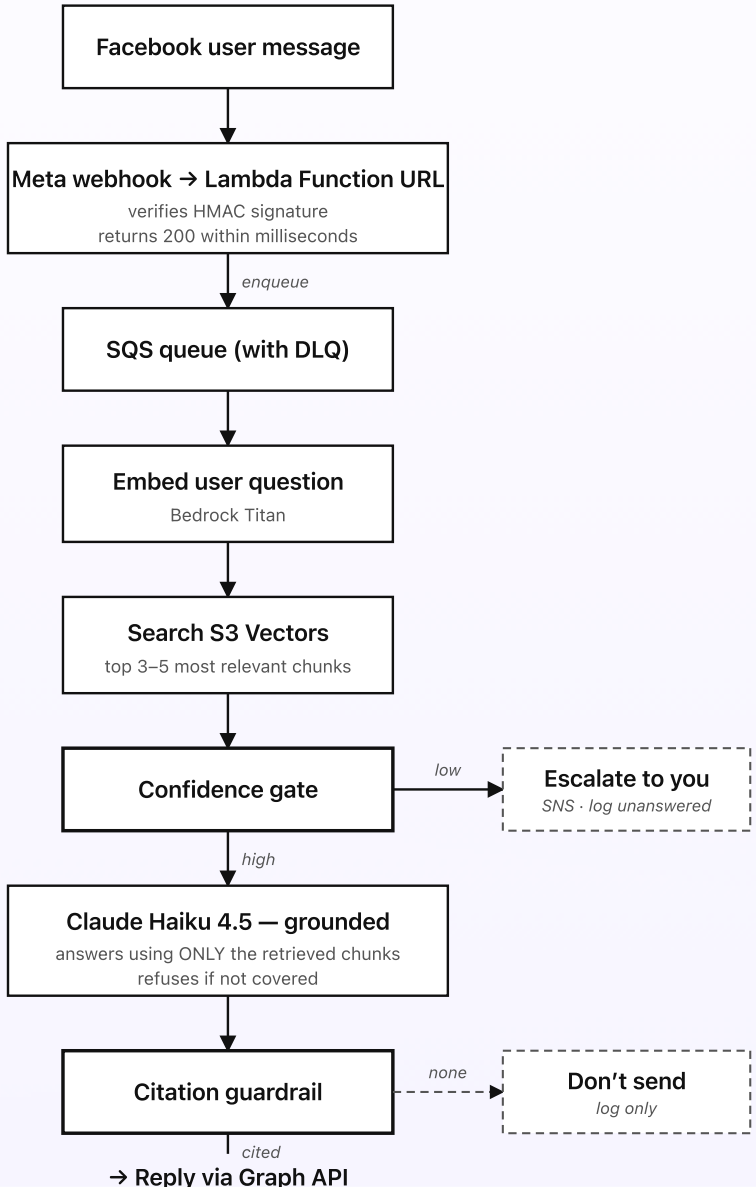

Facebook user message
Meta webhook → Lambda Function URL
verifies HMAC signature
returns 200 within milliseconds
enqueue
SQS queue (with DLQ)
Embed user question
Bedrock Titan
Search S3 Vectors
top 3–5 most relevant chunks
Confidence gate
low
Escalate to you
SNS · log unanswered
high
Claude Haiku 4.5 — grounded
answers using ONLY the retrieved chunks
refuses if not covered
Citation guardrail
none
Don't send
log only
cited
→ Reply via Graph API

Fig 5. Two guardrails: a confidence gate before the AI runs, a citation check before the reply ships.

Why webhook then queue

Facebook's webhook expects a quick "got it" within a couple of seconds, or it'll keep retrying the same message. So the receiver does the bare minimum — checks the message is real, drops it into a queue, and answers Facebook right away. The slow work (looking things up, calling the AI) happens after, off the critical path. If the AI happens to be slow that day, Facebook still gets its "got it" on time. And if a message fails partway through, it lands in a separate "something went wrong" queue you can look at later, instead of vanishing.

Why a confidence gate

The vector search returns the top relevant chunks ranked by similarity. If the top match is weak — below 0.6, say — that's the system telling you "I don't actually know this." Forcing the model to answer anyway is exactly how hallucinations happen.

So the confidence gate is brutal: low score, the message lands in your inbox as an alert and gets logged in an "unanswered" list. That list becomes the next batch of FAQ entries the client should add to their Drive doc. The bot improves over time without anyone training a model.

Why a citation guardrail

Even when given the right context, models sometimes free-style anyway. So the instructions to the model are blunt:

- Answer using ONLY the provided context.
- Return both the reply text *and* the IDs of the chunks you actually used.

The Reply Lambda checks the response. **No cited chunk = no send.** The reply is logged but never reaches Facebook. This is the structural guarantee that the bot can't invent things — if the model didn't ground its answer in actual content from the client's docs, the reply doesn't go out.

> **SAME DENYLIST AS POSTS**
>
> The reply pipeline reuses the page's denylist from the posting pipeline. A reply on the SFC page can't mention "XAUUSD," even by accident. A reply on DailyScalper can't talk about theology. The same per-page rules govern both directions of the conversation.

Start in draft mode

For every new client, replies start in **draft mode** by default: instead of going to Facebook, they go to your inbox. You review them in batches for a week or two. Once you trust the system on a category — say, pricing questions — you flip that category to auto-send. Broader categories like general FAQ come later.

The cost is a slower rollout. The benefit is that no theological or financial mishap reaches a real user before you've seen what the bot would have said.

PART 6 OF 7

APRIL 25, 2026 PART 6 OF 7 · AWS AUTOPOSTING SERIES ~3 MIN READ

What this all costs

A coffee a month, not a Netflix subscription. Here's where the dollars actually go.

KEY TAKEAWAYS

- $2–$5/month per Facebook page at steady volume — three small line items: storage, password vault, AI calls.
- Always-free at this scale: Lambda, EventBridge Scheduler, DynamoDB, SQS, SNS, Lambda Function URLs.
- Small fixed cost is dominated by Secrets Manager (~$1.20) plus pennies for S3 storage and the S3 Vectors index.
- S3 Vectors (GA December 2025) replaces an OpenSearch line item that used to cost real money even when idle.
- Adding a second page roughly doubles only the variable bucket — marginal cost ~$1–$3, not $30+.

The bill stays small because the system sleeps when there's nothing to do. There's no always-on server idling between events. Every cost line is either zero or proportional to actual use.

A QUICK NOTE ABOUT AWS'S FREE TIER

AWS changed how the free tier works in July 2025. New accounts now get six months of access plus up to $200 in credits, instead of the old twelve-month version. **The always-free pieces this article relies on weren't touched** — the small list below is still free for everyone, old accounts and new. The math here works either way.

ALWAYS FREE AT THIS SCALE — $0.00 / month

Lambda	*1M requests + 400K GB-s/month forever*
EventBridge Scheduler	*14M invocations/month*
DynamoDB	*25 GB storage always free; on-demand req < $0.01/mo here*
SQS · SNS · Lambda Function URLs	*covered by the free tier at this volume*

SMALL FIXED — ~$1.30–$1.70 / month

Secrets Manager (3 secrets: FB token, Drive key, app secret)	**~$1.20**
S3 storage + S3 Vectors index	**~$0.10–$0.50**

Per Facebook page. Adds linearly with each new client page.

VARIABLE WITH ENGAGEMENT — ~$0.60–$2.30 / month

Titan Text Embeddings v2	**~$0.00002 / 1K tokens**
Claude Haiku 4.5 replies	**~$0.00125 each**
Claude Haiku 4.5 guardrail judge	**~5% of posts**

Worked example: 1,000 replies in a month ≈ about $1.25 of AI cost.

REALISTIC TOTAL

$2 – $5 / month

per Facebook page, with steady posting and replies

AWS Budget alarm set at $10/month catches anything weird before it grows.

Fig 6. Three tiers: free, small fixed, variable with use. Plus a hard ceiling alarm at $10.

The brain got cheaper to store last December

Until late 2025, the standard place to keep the brain (the searchable copy of the client's docs) was a service called OpenSearch — which costs real money even when nobody's using it. In December 2025, AWS launched a much cheaper option called S3 Vectors. For one Facebook page, the brain is small (well under a gigabyte) and gets a few hundred lookups a day. On the new service, that's pennies a month. If you'd planned this a year ago, you can take a whole line off the bill.

The traps you're deliberately avoiding

Most cloud bills go sideways because of a few specific decisions made early. The architecture above sidesteps each of them on purpose:

- **No always-on networking gateway.** The networking gateway most cloud setups need (called a NAT Gateway) is about $33 a month before you send a single byte through it. The robots here don't need it — they talk to other AWS services on the internal network and to Facebook over the regular public internet. So the line item is zero.
- **No API gateway.** The robots receive webhooks directly. The fancy version of the gateway is $3.50 per million requests; the cheaper version is $1 per million. We use neither. (If you do end up needing one later, pick the cheaper version.)

- **No bottomless log bucket.** Every log stream is set to delete itself after seven days. Logs can't quietly pile up and get billed for years.
- **No keep-the-engine-warm fee.** It's fine for the robots to take a moment to wake up cold — we're not running a low-latency website. Paying to keep them warm would be money for nothing.
- **No CDN, no private network endpoints, no container registry.** Nice things to have for big systems; not needed for this one.

The math that makes this work

The trick isn't cleverness — it's discipline about what doesn't exist. Every always-on piece was swapped for one that wakes only when there's work to do. Every paid service was checked against a free alternative first. The result is a steady bill made of three small line items: a few cents for storage, a dollar or two for the password vault, and the AI calls you actually made that month.

Adding a second Facebook page roughly doubles only the variable bucket and the storage line. The free-tier components stay free. So the marginal cost of a new client page is closer to $1–$3/month than the $30+ a traditional setup would cost.

> **A COFFEE A MONTH**
>
> For one page with steady posting and replies, all-in cost is about $2–$5/month. For five pages, more like $10–$20/month. The architecture scales with the variable line, not with a fixed-cost staircase.

PART 7 OF 7

APRIL 25, 2026 PART 7 OF 7 · AWS AUTOPOSTING SERIES ~3 MIN READ

Engineering reference: the full architecture

Same system as the rest of the series, drawn purely for engineers. Service names, resource identifiers, region, 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 (`global.` model IDs).
- Five subsystems: Build & Deploy, Posting (scheduled), KB Sync (Drive→S3→Vectors), Reply (RAG), Cross-cutting.
- Models: `global.anthropic.claude-haiku-4-5-20251001-v1:0` + `amazon.titan-embed-text-v2:0` ; vector store is S3 Vectors (GA Dec 2025).
- Naming conventions: `fn-*` Lambdas, `tbl-*` DynamoDB, `q-*` SQS with paired DLQ, `t-*` SNS, `vec-*-{page}` per-page vectors.
- Day-one paperwork: Meta Page access token rotation (60-day expiry), Drive service account scope, AWS Budget alarm at $10/month.

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.

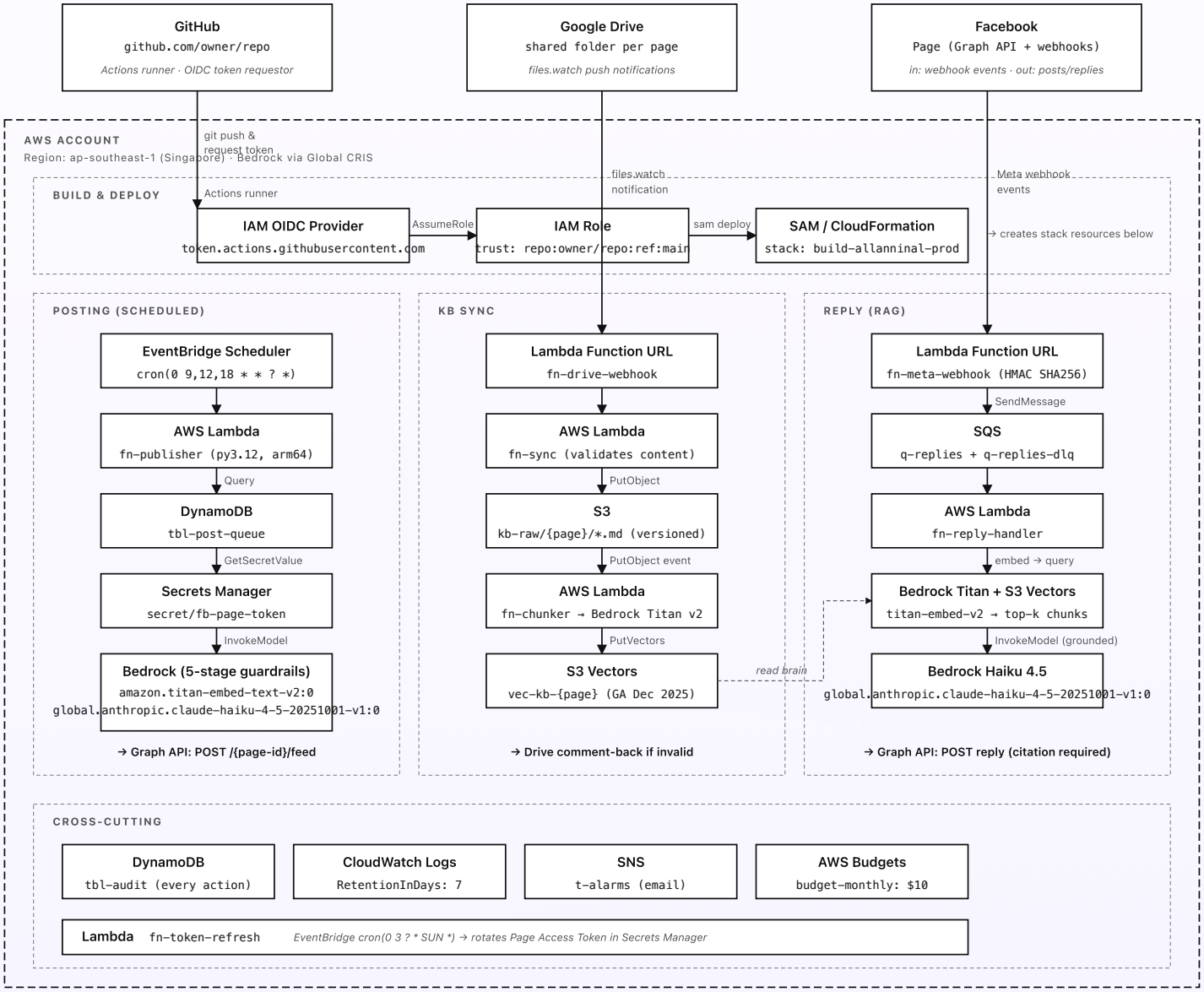

GitHub
github.com/owner/repo
Actions runner · OIDC token requestor
Google Drive
shared folder per page
files.watch push notifications
Facebook
Page (Graph API + webhooks)
in: webhook events · out: posts/replies
AWS ACCOUNT
Region: ap-southeast-1 (Singapore) · Bedrock via Global CRIS
git push & request token
files.watch notification
Meta webhook events
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: build-allanninal-prod
→ creates stack resources below
POSTING (SCHEDULED)
EventBridge Scheduler
cron(0 9,12,18 * * ? *)
AWS Lambda
fn-publisher (py3.12, arm64)
Query
DynamoDB
tbl-post-queue
GetSecretValue
Secrets Manager
secret/fb-page-token
InvokeModel
Bedrock (5-stage guardrails)
amazon.titan-embed-text-v2:0
global.anthropic.claude-haiku-4-5-20251001-v1:0
→ Graph API: POST /{page-id}/feed
KB SYNC
Lambda Function URL
fn-drive-webhook
AWS Lambda
fn-sync (validates content)
PutObject
S3
kb-raw/{page}/*.md (versioned)
PutObject event
AWS Lambda
fn-chunker → Bedrock Titan v2
PutVectors
S3 Vectors
vec-kb-{page} (GA Dec 2025)
read brain
→ Drive comment-back if invalid
REPLY (RAG)
Lambda Function URL
fn-meta-webhook (HMAC SHA256)
SendMessage
SQS
q-replies + q-replies-dlq
AWS Lambda
fn-reply-handler
embed → query
Bedrock Titan + S3 Vectors
titan-embed-v2 → top-k chunks
InvokeModel (grounded)
Bedrock Haiku 4.5
global.anthropic.claude-haiku-4-5-20251001-v1:0
→ Graph API: POST reply (citation required)
CROSS-CUTTING
DynamoDB
tbl-audit (every action)
CloudWatch Logs
RetentionInDays: 7
SNS
t-alarms (email)
AWS Budgets
budget-monthly: $10
Lambda fn-token-refresh
EventBridge cron(0 3 ? * SUN *) → rotates Page Access Token in Secrets Manager

Fig 7. Full architecture, ap-southeast-1. White boxes = AWS resources; dashed AWS container; dashed grey boxes = subsystem groupings; dashed grey arrow = cross-subsystem data dependency.

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, three runtime columns (Posting, KB Sync, Reply) in the middle, and a Cross-cutting strip at the bottom. The dashed grey arrow from the KB Sync output to the Reply column shows the only cross-subsystem data dependency — the Reply pipeline reads the same vectors index that KB Sync writes.

Naming conventions used in the diagram

- **Lambda functions:** `fn-<purpose>` — e.g. `fn-publisher`, `fn-sync`, `fn-reply-handler`.
- **DynamoDB tables:** `tbl-<name>`.
- **SQS queues:** `q-<name>` with paired `q-<name>-dlq`.
- **SNS topics:** `t-<name>`.
- **S3 Vectors indexes:** `vec-<purpose>-{page}` — one index per Facebook page.
- **S3 buckets:** `kb-raw` partitioned by `{page}/` prefix.

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.

What's deliberately not on the diagram

- IAM policy details — per-Lambda execution role inline policies are minimal (one secret, one table, one bucket as appropriate).
- Per-page configuration JSON in S3 (read by Publisher and Reply Lambdas; lets you tune thresholds without redeploys).
- X-Ray tracing — on for the Reply Lambda only, sampling 10%.
- 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 Posting next. Don't add Reply until KB Sync is producing a usable vectors index — the Reply pipeline depends on that brain existing. Cross-cutting (audit, logs, alarms, budget) goes in from day one.