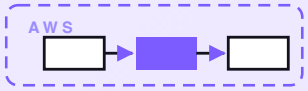
7-PART SERIES · FREE COMPANION



Content repurposer

A serverless system that turns one long piece — a blog post or a recording transcript — into a week's worth of short posts. It reads the source, pulls the strongest points, and drafts platform-sized versions in your voice: a thread, a few short posts, a quote-card caption. Every draft is grounded in the original, so it doesn't make things up. You approve, edit, or skip each one; nothing posts on its own. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Content repurposer

01 A content repurposer on AWS for a few dollars a month

02 How a long post gets loaded

03 How the strongest points get pulled

04 How a clip gets drafted

05 How a repurposed draft gets approved

06 What the content repurposer costs

07 Engineering reference: the content repurposer architecture

PART 1 OF 7

MAY 24, 2026 PART 1 OF 7 · CONTENT REPURPOSER SERIES ~5 MIN READ

A content repurposer on AWS for a few dollars a month

A small business writes one good thing a week and then lets it die. The blog post nobody links to twice. The hour-long webinar that had three lines worth quoting and zero that ever left the recording. The founder's podcast answer that would have made a great short post if anyone had pulled it out. The work to make a long piece is real; the work to turn it into a week of short posts is small but nobody has time for it. This post walks through the design of a small system that reads one long piece, pulls the strongest points, and drafts short posts in your voice — and never posts a single one without you approving it first.

KEY TAKEAWAYS

- Three sources for one long piece: a Drive folder, a transcript forwarding lane, and a paste-a-link lane.
- Every short post is grounded in the original — it ships with the exact passage it came from.
- One long piece fans out into a thread, a few short posts, and a quote-card caption, all in your voice.
- Nothing posts on its own. You approve, edit, or skip each draft. Approved ones can drip out over the week.
- Designed on AWS for about $3/month at typical small-business volume.

The whole system on one page

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

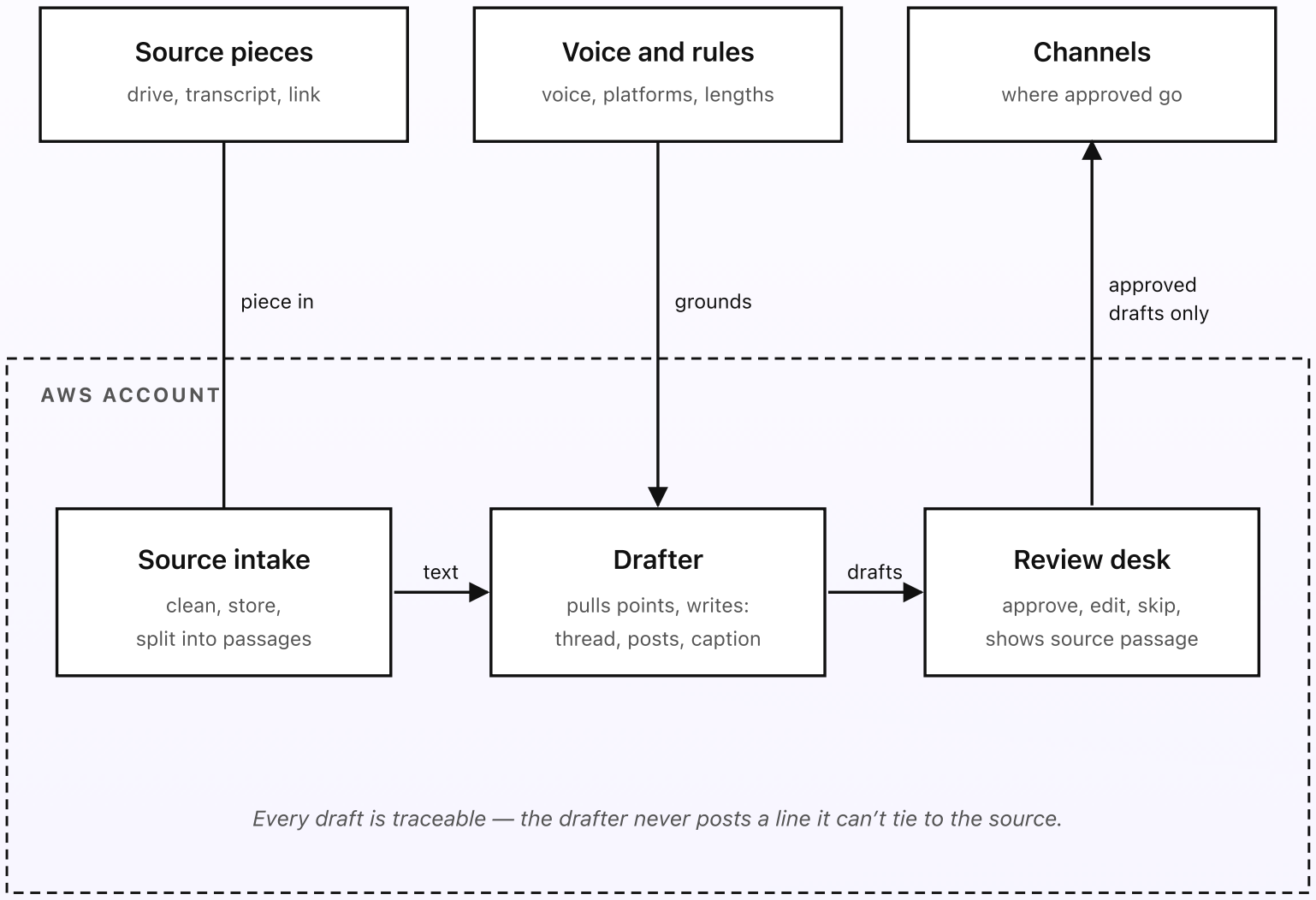

Source pieces
drive, transcript, link
Voice and rules
voice, platforms, lengths
Channels
where approved go
piece in
grounds
approved
drafts only
AWS ACCOUNT
Source intake
clean, store,
split into passages
text
Drafter
pulls points, writes:
thread, posts, caption
drafts
Review desk
approve, edit, skip,
shows source passage
Every draft is traceable — the drafter never posts a line it can't tie to the source.

Fig 1. Three sources outside, three pieces inside AWS. Long pieces flow in from a Drive folder, a transcript forwarding lane, and a paste-a-link lane. The Drafter pulls the strongest points and writes short posts. The Review desk shows each one for a human to approve, edit, or skip.

What you set up once (the outside)

- **Source pieces.** A Google Drive folder where you drop the long thing you want to reuse — a blog post as a doc, a webinar transcript, a case study. New pieces can also enter via two other lanes covered in Part 2 — a forwarding lane (forward a recording transcript to a dedicated address and the system cleans it and queues it) and a paste-a-link lane (drop a URL to a published post and the system fetches it and strips it down to plain text). You give the system one long piece; it gives you back a week of short ones.
- **A voice-and-rules folder.** Two short Google Docs in a Drive folder. The *voice* doc captures how you write — a few real examples of your own short posts, the words you use and the ones you don't, whether you use emoji, how you open and close. The *rules* doc covers which platforms to draft for and how long each one should be: a thread of five-to-eight short posts, three standalone short posts, and one quote-card caption, for example. Change a rule in the doc and the next piece drafts to the new shape — no deploy.
- **Channels.** Where an approved draft goes. The simplest setup sends approved drafts to a review channel where a person copies them out to each platform by hand. A fuller setup hands approved drafts to a post scheduler that drips them out over the week. Either way, only drafts a human approved ever reach a channel.

What runs when you give it a piece (the inside)

- **The source intake.** Three sources feed one store. A piece arrives, the intake cleans it to plain text (strips the page furniture from a fetched URL, drops the timestamps and speaker labels from a transcript), writes it to S3, and splits it into passages — small chunks of a few sentences each — so the drafter can point at the exact part a draft came from. The Drive folder stays the place you control the source.
- **The drafter.** The work happens here. First it pulls the strongest points: it reads the passages and picks the handful actually worth posting (a sharp claim, a surprising number, a clean before-and-after). Each chosen point carries the passage it came from. Then, for each point, it writes platform-sized drafts in your voice — the thread, the short posts, the caption — using the voice doc as a guide. Before any draft leaves the drafter, it's checked against the source: if a sentence can't be traced back to a passage, it's dropped. Bedrock does the reading and the writing; the grounding check keeps it honest.
- **The review desk.** Every draft lands here, never on a channel. Each one shows the draft, the platform it's for, and the source passage it came from, with three buttons: *approve*, *edit*, *skip*. Approve sends it on (to the scheduler or the review channel). Edit opens a box pre-filled with the draft. Skip drops it. Every choice is logged, so next month you can see what got used and what got cut.

In plain words

You publish a 1,800-word blog post on Monday about why your shop switched suppliers. By Monday afternoon the system has read it and drafted a week of short posts: a six-part thread that walks through the switch, three standalone

posts (one with the cost number, one with the quality comparison, one with the customer reaction), and a quote-card caption pulling your sharpest line. Each draft sits on the review desk next to the exact sentence from the post it came from. You read through them over coffee — approve four, edit one to sharpen the hook, skip one that feels thin. The four approved ones drip out across the week. Nothing went out that you didn't see, and the post you spent two hours writing did a week of work instead of one day's.

The cost of running this is about $3 a month at SMB volume. The cost of *not* running it is every long piece that did one day of work and then sat in an archive nobody opens.

DESIGN RULES THAT SHAPED EVERY DECISION

- Every draft is grounded — it ships with the source passage it came from. No passage, no draft.
- Nothing posts on its own. Approve, edit, or skip. There is no fourth move.
- One long piece, one batch of drafts. The system runs when you feed it, not on a clock you can't see.
- Your voice lives in a doc. Change how it sounds without touching code.
- Platforms and lengths live in a doc too. Add a platform without a deploy.
- Every choice is logged. Look back next month and see what got approved, edited, or skipped.

Why this shape

Most teams repurpose content in one of three ways: they don't, they pay a freelancer to do it by hand, or they paste the whole post into a chatbot and hope. The first leaves money on the table — the long piece was the expensive part, and the short posts are the cheap multiplier nobody collects. The freelancer works but costs more than the system and is slow on a busy week. And the paste-into-a-chatbot approach is where things go wrong: with no grounding, the model happily invents a statistic that was never in your post, and now you've posted something you can't back up.

The setup above keeps the human where the human matters — deciding what's good and what ships — and gives the machine the parts it's reliably good at: reading a long thing, finding the quotable bits, and rewriting them to length. The grounding rule is the whole game. A short post is only allowed to say something the source said. That one constraint is the difference between a tool you trust and a tool that gets you in trouble.

The next four posts walk through each piece in turn: how a long post gets loaded, how the strongest points get pulled, how a clip gets drafted in your voice, and how a repurposed draft gets approved. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 24, 2026 PART 2 OF 7 · CONTENT REPURPOSER SERIES ~4 MIN READ

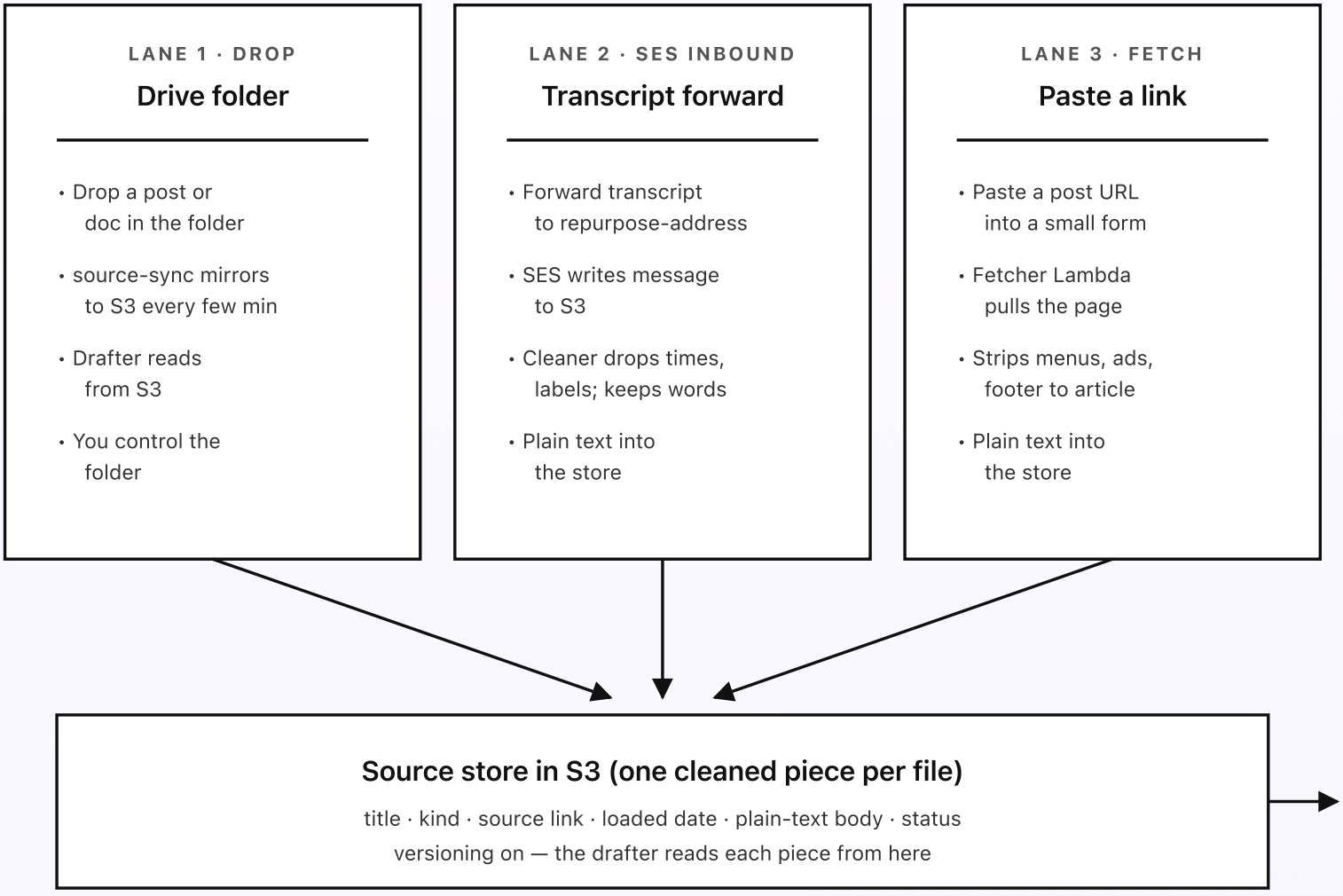
How a long post gets loaded

The repurposer only works on what you give it. So the first job is making it easy to hand it a long piece, whatever shape that piece is in. There are three ways one gets in: you drop a doc in a Drive folder, you forward a recording transcript to a dedicated address, or you paste a link to a published post. The first is for things you wrote. The other two exist because in real life the best source is often a webinar nobody transcribed cleanly or a post already live on your own site.

KEY TAKEAWAYS

- Three intake lanes feed one source store: a Drive folder, a transcript forwarding lane, and a paste-a-link lane.
- The forwarding lane cleans transcripts — drops timestamps and speaker labels, keeps the words.
- The link lane fetches a published page and strips the menus, ads, and footer down to plain text.
- Every loaded piece is mirrored to S3, so you get versioning for free and the drafter reads from there.
- The Drive folder stays the place you control. The other lanes are conveniences that write into the store.

Three lanes into one store



The Drive folder stays the place you control — the other lanes are conveniences that load pieces into the store.

Fig 2. Three lanes converge on one source store. The Drive folder is the place you control; the transcript lane and the link lane are conveniences that load pieces for you. Everything lands as cleaned plain text in S3 so the drafter can read it the same way no matter where it came from.

Lane 1: the Drive folder itself

The simplest lane. Open the Drive folder, drop in the doc you want to reuse, done. It works for anything you wrote yourself — a blog post, a case study, a long email you're proud of. A small Lambda — `source-sync` — runs every few minutes, exports any new or changed doc as plain text via the Drive API, and writes it to `s3://cr-source-store/`. The drafter reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so if you edit the source and re-run, the old version is still there.

This lane covers the case where the long piece already exists as a doc and you just want a week of short posts out of it. Most of what a small team repurposes goes in this way.

Lane 2: transcript forwarding (the messy-source lane)

Recordings are the richest source and the worst-formatted one. A webinar, a podcast, a sales call you have permission to reuse — the transcript is full of good lines buried under timestamps, speaker labels, and filler. Set up a dedicated inbound address — something like `repurpose@your-company.com` — via Amazon SES. Forward the transcript (paste it in the body, or attach the file your recorder spat out) and the system takes it from there. SES writes the raw message to `s3://cr-raw-inbound/`. The S3 PUT triggers a cleaner Lambda.

The cleaner strips the parts you don't want to post — the `00:14:32` timestamps, the `Speaker 1:` labels, the "um"s and "you know"s — and keeps the actual sentences as plain text. This part is plain code, not a model: a few rules that handle the common transcript formats. The cleaned text goes into the same source store as Lane 1, tagged as a transcript so the drafter knows it's spoken words and can lean a little more on tightening them up. Nothing is invented in the cleanup; sentences are only ever dropped or trimmed, never added.

Lane 3: paste a link

Sometimes the best source is already published — on your own blog, or a guest post you wrote for someone else. Retyping it into a doc is a waste. Lane 3 is a small form (a single text box backed by a Function URL) where you paste the URL. A `fetcher` Lambda pulls the page and strips it down to just the article: out go the navigation menus, the cookie banner, the sidebar, the ads, the footer; what's left is the title and the body text, written into the source store.

The fetcher only pulls pages you paste in — it doesn't crawl, and it doesn't go fetch anything on its own. That's on purpose: the source should always be a thing you chose to repurpose, not something the system decided to grab. Paste-a-link is the most opt-in lane; a team that never uses it loses nothing.

Why everything funnels into one store

Three lanes in, but only one place the drafter actually reads from. That's deliberate. A blog post, a transcript, and a fetched web page arrive in three completely different shapes; if the drafter had to handle each shape, it would be three times as much to get wrong. Cleaning every piece down to plain text at the

door means the drafter only ever sees one kind of thing: a title and a body. The convenience lanes are first-class for getting a piece in, but they all pass through the same cleanup and land in the same store.

Next post: how the drafter reads a loaded piece, splits it into passages, and pulls the handful of points actually worth posting — each one carrying the exact passage it came from.

PART 3 OF 7

MAY 24, 2026 PART 3 OF 7 · CONTENT REPURPOSER SERIES ~5 MIN READ

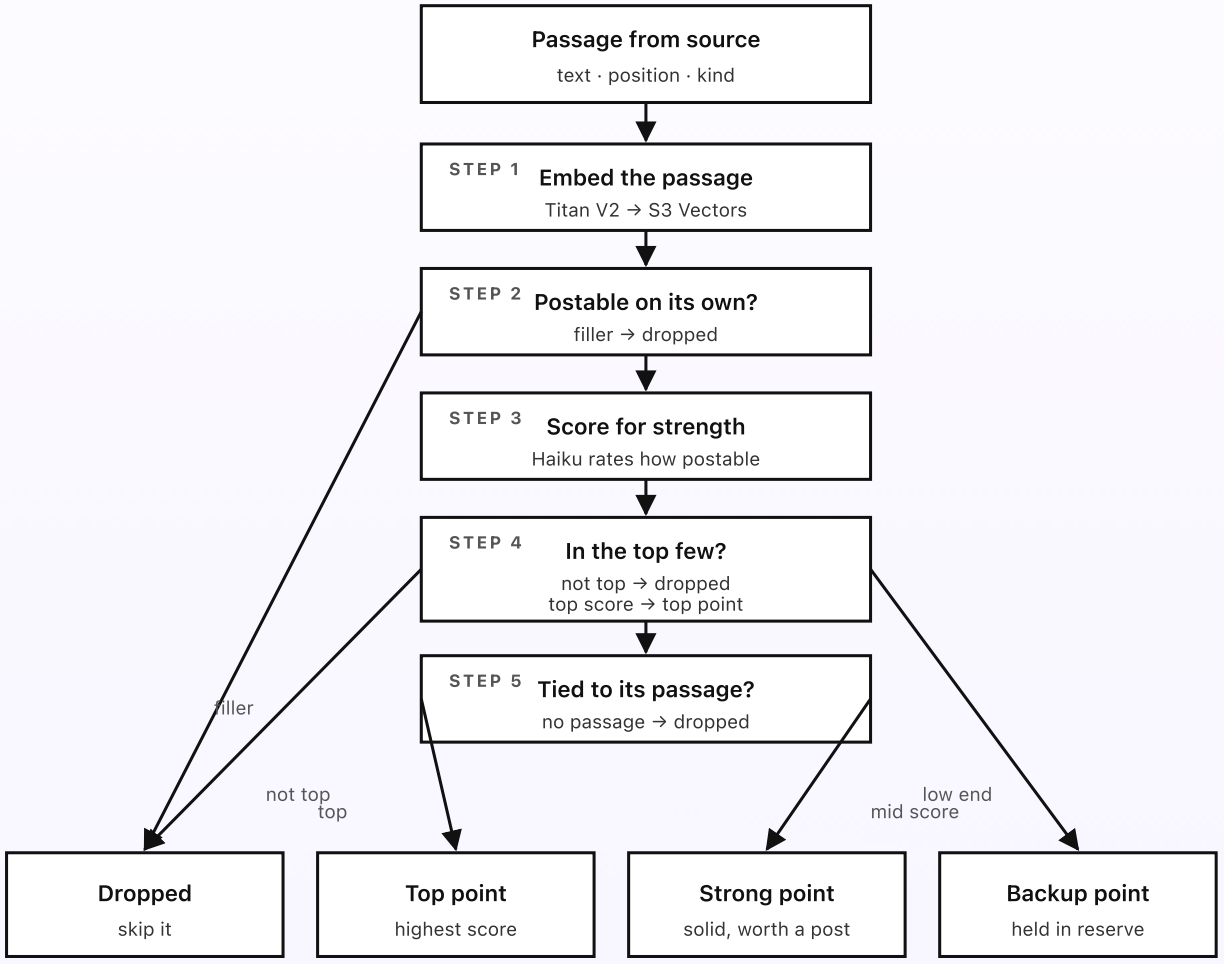
How the strongest points get pulled

A long piece has a lot of words and only a few worth posting. The job here is to find those few. The drafter splits the source into passages, indexes them so it can match ideas back to exact text, and asks Bedrock to score and pick the handful that would actually make good short posts. The important rule runs through the whole step: every point that comes out the other side carries the exact passage it came from. If a point can’t be tied to the text, it doesn’t come out.

KEY TAKEAWAYS

- The source is split into passages — small chunks of a few sentences each.
- Each passage is turned into a number-fingerprint (an embedding) and stored so it can be matched and quoted.
- Bedrock scores each candidate for how postable it is and picks the top few.
- A grounding check confirms each chosen point is backed by a real passage; ungrounded ones are dropped.
- Out comes a short list of points, each tied to the exact passage it came from.

The selection flow, per passage



Every point carries its source passage — a point with no passage never leaves this step.

Fig 3. The selection flow, per passage. Five steps decide whether a passage becomes a point and how strong it is. The grounding check is the gate; a point that can't be tied to its passage is dropped before it reaches the drafter.

Splitting into passages: small enough to quote

The first thing the drafter does with a loaded piece is cut it into passages — chunks of a few sentences each, broken at natural points like paragraph breaks or a change of subject. The reason for small chunks is the grounding rule: a short post should be able to point at the exact sentence it came from, and you can only do that if the chunks are small enough to be a real quote. A whole 1,800-word post is too big to be the "source" of a single line; a three-sentence passage is just right.

Each passage gets a position (where in the piece it is) and a note about the kind of source it came from. A passage from the opening of a post is treated a little differently from one buried in the middle — openings are often summaries, which make weaker standalone posts.

The fingerprint that lets the system match and quote

Each passage is turned into an *embedding* — a list of numbers that captures what the passage is about, using Amazon Titan Text Embeddings V2. Think of it as a fingerprint: two passages about the same idea have similar fingerprints, even if they use different words. The fingerprints are stored in Amazon S3 Vectors, a cheap place to keep them and search them.

This matters later, in the drafting step, for the grounding check. When a draft says something, the system can take that draft's fingerprint and find the passage that best matches it — that's the source it gets attached to. The fingerprints are what make "show me the exact passage this came from" a one-step lookup instead of a guess.

Scoring and picking the few worth posting

Now the model earns its place. Bedrock Haiku reads each candidate passage and scores it for one thing: how good a short post would it make on its own? A sharp claim scores high. A surprising number scores high. A clean before-and-after, a strong opinion, a concrete example — all high. A transition sentence, a throat-clearing intro, a "as we discussed earlier" — all low. The prompt is short and specific: "Rate this passage from 1 to 5 for how strong a standalone short post it would make. Return the score and one line of why. Do not rewrite it yet."

Then the system keeps only the top few across the whole piece. A long post might have thirty passages; you don't want thirty short posts, you want the best six or eight. The rules doc sets how many to keep. The highest scorers become *top points*, the next tier *strong points*, and a few more are held as *backup points* in case you skip several at review time and want more. Everything below the line is dropped — not deleted from the source, just not turned into a draft.

The grounding gate

Before any point moves on to drafting, one last check: can this point be tied back to its exact passage? For points pulled straight from the text this is automatic —

the passage *is* the source. The check matters more in the next step, when the model rewrites a point into a post and might drift away from what the source actually said. Setting up the tie here, at selection time, is what makes that later check possible. A point with no passage behind it never becomes a draft.

This is the same instinct as a careful writer who won't put a number in a post unless they can point to where it came from. The system makes that instinct a hard rule instead of a habit you're trusting someone to remember.

Next post: how each pulled point becomes platform-sized drafts in your voice — the thread, the short posts, the quote-card caption — and how the grounding tie is checked one more time before a draft reaches you.

PART 4 OF 7

MAY 24, 2026 PART 4 OF 7 · CONTENT REPURPOSER SERIES ~5 MIN READ

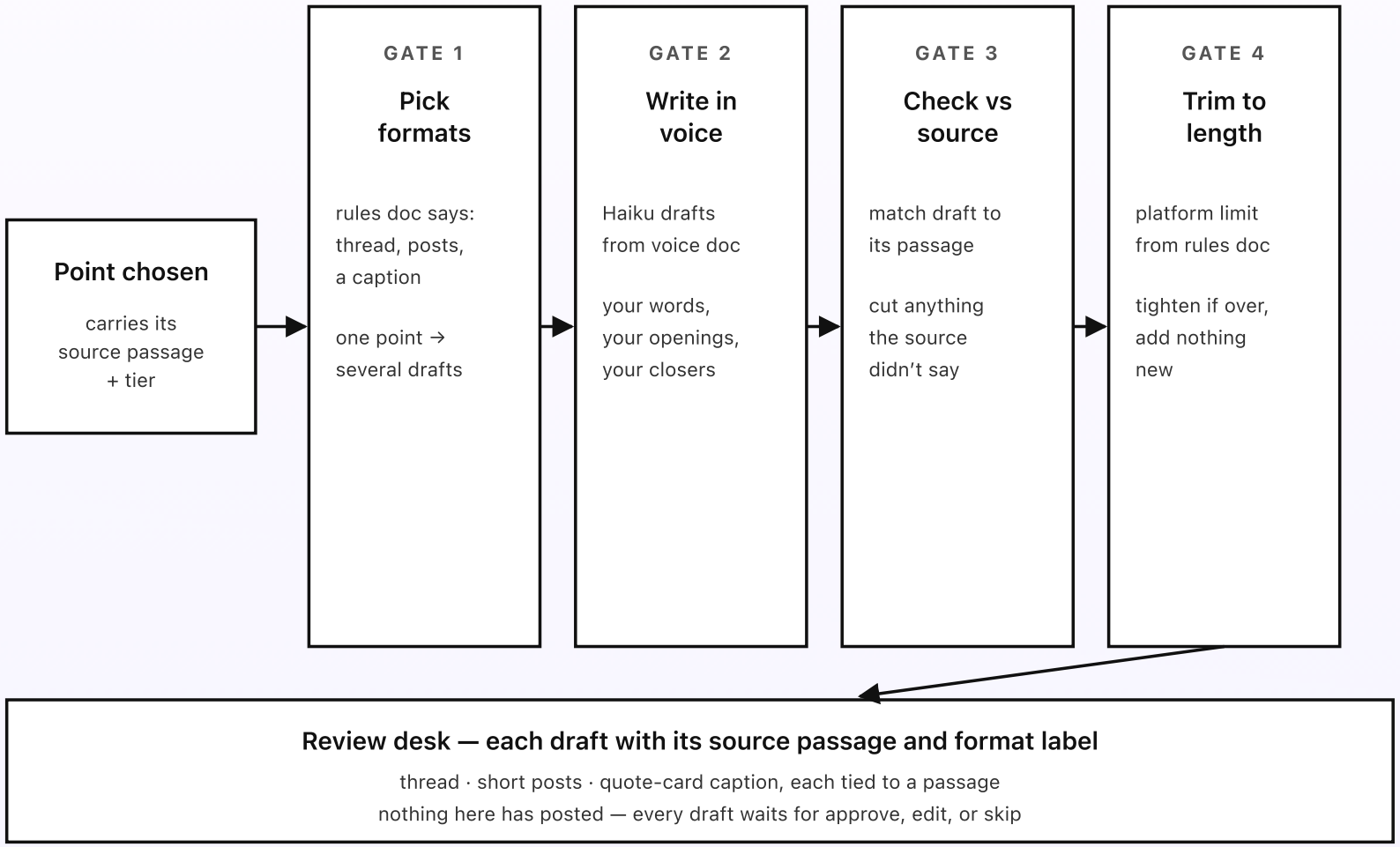
How a clip gets drafted

A point has been picked and it carries the passage it came from. Now the drafter has to turn it into something you'd actually post — the right format for each platform, in your voice, the right length, and still true to what the source said. Get any of those wrong and the draft is more work than writing from scratch: a post that sounds like a robot, a thread that's too long, a caption that says something the source never did. Four small gates sit between a chosen point and a draft on your desk.

KEY TAKEAWAYS

- Each point is drafted into the formats the rules doc asks for: a thread, short posts, a quote-card caption.
- The voice doc steers the wording so drafts sound like you, not like a generic assistant.
- Every draft is checked back against the source passage; anything the source didn't say is cut.
- Each format is trimmed to its platform's length before it's allowed through.
- A draft only reaches the review desk if it passed all four gates, with its source passage attached.

Four gates on every draft



Every gate is a plain check — a draft that can't be tied to the source never reaches you.

Fig 4. Four gates between a chosen point and a draft on the review desk. Pick the formats. Write in your voice. Check against the source. Trim to length. Then the draft lands on the desk with its source passage attached, waiting for you.

Gate 1: pick the formats

One point can become several drafts, and the rules doc decides which. A typical setup asks for three formats per point: a *thread* (a handful of short posts that build on each other), one or two *standalone short posts* (a single punchy post that stands alone), and a *quote-card caption* (a short caption to pair with a graphic that shows the quote). Not every point gets every format — a top point might get all three, a strong point just a short post and a caption. The rules doc holds the recipe, so you can add a platform or change the mix without anyone touching code.

The point keeps its source passage through this whole step. Every draft that comes out of it inherits that same passage, which is what the source-check in Gate 3 needs.

Gate 2: write it in your voice

This is where Bedrock writes. The drafter takes the point, the source passage, and the voice doc, and asks Haiku 4.5 to draft each format. The voice doc is doing the heavy lifting on tone: it holds three or four real examples of your own short posts, a short list of words you use and words you avoid, whether you use emoji, and how you tend to open and close. The prompt points at all of it: “Write a [format]

from this point. Match the voice in these examples. Use only what the source passage supports."

Most pieces draft fine on Haiku 4.5, which is the cheap, fast model. The harder long pieces — a dense technical post, a rambling transcript where the good lines are tangled up — can be sent to Claude Sonnet 4.6 instead, which reasons better at the cost of being a little slower and dearer. The rules doc picks which model by source kind, so you only pay for the heavier model when a piece actually needs it.

Gate 3: check it against the source

The most important gate. A model rewriting a point for punch can quietly add something the source never said — a rounder number, a stronger claim, a detail that sounds right. Gate 3 catches that. It takes the finished draft, finds the passage that best matches it (using the fingerprints from Part 3), and checks that every claim in the draft is actually backed by the source. Anything that isn't gets cut. If the draft has drifted too far to fix by trimming, it's sent back to Gate 2 to be rewritten once, with a note to stay closer to the passage.

This is the rule that makes the whole system safe to use. Without it, repurposing with a model is a slow-motion way to publish things you can't back up. With it, the worst case is a draft that's a little flat — never a draft that's a little false.

Gate 4: trim to the platform's length

Each platform has its own limits, and the rules doc holds them: a short post capped at a certain number of characters, each post in a thread kept tight, a

caption short enough to read at a glance. Gate 4 checks each draft against its limit and, if it's over, tightens it — cutting words, not adding them. The constraint is one-directional on purpose: trimming can only remove text the source already supported, so it can't reintroduce a claim Gate 3 just cut.

A draft that passes all four gates lands on the review desk with everything you need to judge it fast: the draft itself, the format it's for, the point tier, and the exact source passage it came from. Nothing has posted. Every draft is waiting for you.

Next post: the review desk itself — the three buttons on every draft, what approve, edit, and skip each do, and how every choice gets logged so you can see what worked next month.

PART 5 OF 7

MAY 24, 2026 PART 5 OF 7 · CONTENT REPURPOSER SERIES ~5 MIN READ

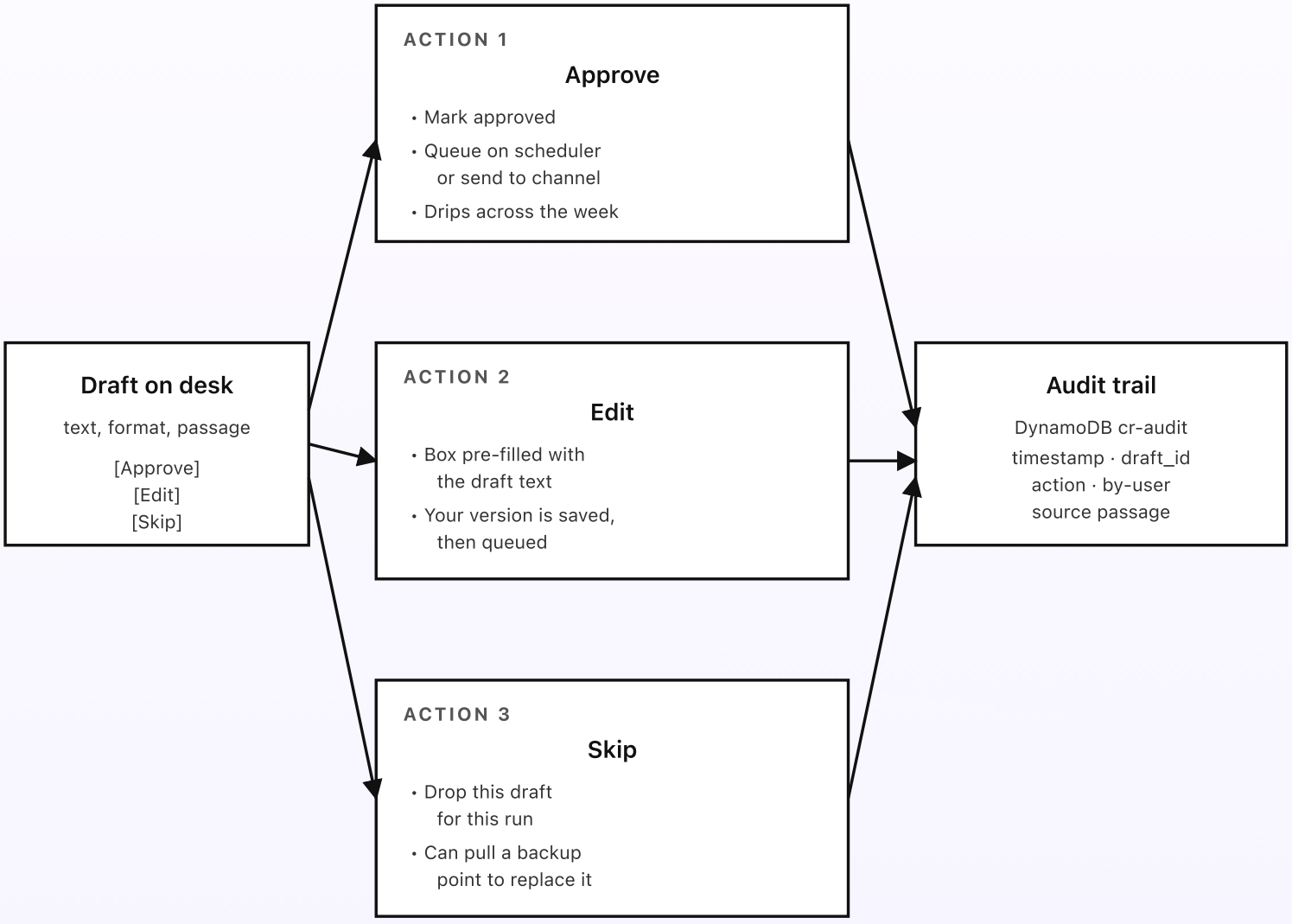
How a repurposed draft gets approved

A batch of drafts lands on the review desk on Monday afternoon. A six-part thread, three short posts, a quote-card caption — each with the source passage it came from sitting right next to it. There's an Approve button, an Edit button, and a Skip button on every one. What happens when you tap each? This post walks through the three actions, what each does to the draft and the schedule, and how every choice gets logged so nothing posts that you didn't wave through.

KEY TAKEAWAYS

- Three actions per draft: *approve* (send it on or queue it), *edit* (tweak it), *skip* (drop it).
- Approve hands the draft to your scheduler or your review channel — never straight to the public.
- Edit opens a box pre-filled with the draft; your edit is what gets saved and sent.
- Skip drops the draft and can pull a backup point off the bench to replace it.
- Every action is logged, so next month you can see what got approved, edited, or skipped — and why.

Three actions on a draft



Nothing posts on its own — a human taps every draft before it leaves the desk.

Fig 5. Three actions per draft, three different effects. Approve queues it or sends it to a channel. Edit saves your version and then queues it. Skip drops it and can pull a backup point to replace it. Every action writes to the audit trail.

Action 1: approve (the common one)

You read the draft, it's good, you tap *Approve*. A Function URL Lambda marks the draft as approved and decides where it goes based on your setup. In the simple setup, it lands in a review channel where a person copies it out to the right platform by hand — a small extra step, but it means the system never touches a public account. In the fuller setup, it's queued on a post scheduler that drips approved drafts out across the week at the times the rules doc sets — say, one a day at 9am, never two in an hour. Either way, an `approved` row is written to the audit trail with the draft, the source passage, and your name.

The drip matters. A week's worth of posts approved in one sitting shouldn't all go out in one sitting. The scheduler spaces them so your feed looks like a person posting through the week, not a batch job that fired at 2pm Monday.

Action 2: edit (the tweak)

Most drafts are close but not perfect. The hook could be sharper, a word feels off, you want to swap the opening line. *Edit* opens a box pre-filled with the draft text. You change what you want and save. The Function URL Lambda stores *your* version as the approved one — the model's draft is kept too, but yours is what gets sent — and writes an `edited` row with the before and after, then queues it the same way an approve does.

Keeping both versions is worth it. Over a few months, the gap between what the model drafted and what you actually posted is the clearest signal of where the voice doc needs work. If you keep rewriting the openings, the voice doc's examples probably don't show enough of how you open. The edit history is how you tune the system without guessing.

Action 3: skip (the no)

Some drafts just don't land. The point was thinner than it looked, the angle doesn't fit this week, you already posted something similar. *Skip* drops the draft for this run. Nothing is sent. A `skipped` row is written so the reason isn't a mystery later.

Skip has one extra trick: the backup points from Part 3. If you skip a draft and there's a backup point sitting on the bench, the system can pull one off and draft a replacement, so a week of skips doesn't leave you short. This is optional — the rules doc decides whether skips auto-refill or just drop — but it's the difference between "I skipped three and now I have nothing" and "I skipped three and three fresh ones showed up."

Every action is logged, every draft is traceable

The `cr-audit` table records every approve, edit, and skip with the user who did it, the timestamp, the draft, and the source passage it came from. That last part is the quiet payoff of the grounding rule: months later, you can look at any post that went out and see not just that a human approved it, but the exact sentence in the

original piece it traces back to. If someone ever asks “where did this claim come from?”, the answer is one lookup away.

This is also how you measure whether the system is earning its keep. The log shows your approve rate, your edit rate, and what you skip most. A high skip rate on one format means the rules doc is asking for a format that doesn’t work for you; a high edit rate means the voice doc needs more examples. The review desk isn’t just a gate — it’s the feedback loop that makes next month’s drafts better.

Next post: the cost breakdown. The whole pipeline above runs in coffee-money territory at SMB volume; Part 6 explains exactly where the dollars go and why the Bedrock calls are the only line that really moves.

PART 6 OF 7

MAY 24, 2026 PART 6 OF 7 · CONTENT REPURPOSER SERIES ~3 MIN READ

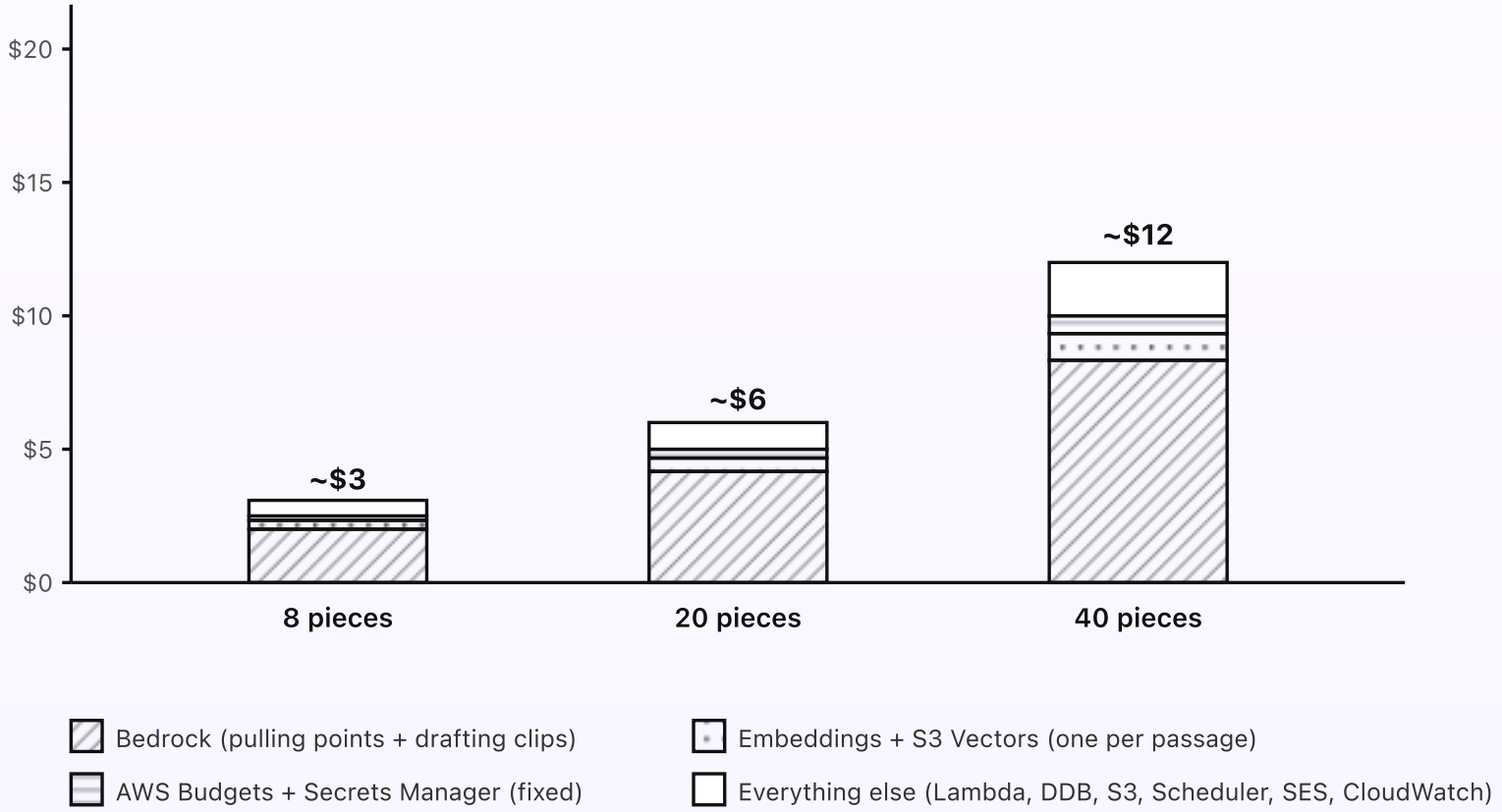
What the content repurposer costs

This system only does work when you hand it a long piece. There's no clock ticking in the background, no always-on anything. When a piece arrives, it reads it, pulls the points, drafts the posts, and goes quiet again. The cost is almost all model calls — the reading and the writing — and even those are cheap because the model used is the small, fast one. At typical SMB volume, the bill is a few dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $3/month at typical SMB volume (about eight long pieces a month).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The cost is dominated by Bedrock — pulling points and drafting each clip.
- Embeddings and S3 Vectors are a small slice; everything else is pennies.
- At 20 pieces a month the bill is around $6. At 40 it's around $12.

Cost at three volumes



The model calls are the dominant cost — and even those are a fraction of a cent per draft.

Fig 6. Monthly cost at three piece volumes. Bedrock is the dominant slice because reading and drafting are the real work; embeddings and the everything-else bucket are small. The cost scales with how many pieces you repurpose, not with a clock running in the background.

Where the dollars actually go

Bedrock (the bulk). Two jobs cost real money, and both are model calls. First, pulling points: scoring each passage in a piece for how postable it is. A 1,800-word post is maybe thirty short passages, so thirty cheap reads. Second, drafting: writing each chosen point into a thread, short posts, and a caption, then the source-check rewrite if a draft drifted. That's the larger share. Both run on Haiku 4.5, the small fast model, so even a piece that fans out into a dozen drafts costs a few cents. The harder pieces that go to Sonnet 4.6 cost more per call, but they're the minority. At eight pieces a month, Bedrock is a couple of dollars.

Embeddings + S3 Vectors. Each passage gets one fingerprint, made once with Titan Text Embeddings V2 and stored in S3 Vectors. A piece of thirty passages is thirty small embedding calls and thirty tiny vectors. Storing and searching them is cheap. A small slice of the bill, growing slowly with the number of pieces.

Lambda runtime. The intake, the points lane, the drafter, the Function URL handler for the buttons, the scheduler jobs. None run long. The drafter is the heaviest because it waits on model calls, but waiting on Bedrock is billed as Bedrock time, not Lambda time. Lambda lands under a dollar at all three volumes.

DynamoDB on-demand. A couple of small tables: the draft state and the `cr-audit` log. Writes are approvals, edits, and skips; reads are when you open the desk. Pennies a month at any of these volumes.

S3 + storage. The cleaned source pieces, the raw inbound from forwarded transcripts, the drafts. A few MB total at SMB volume. Effectively free.

SES + Scheduler. SES inbound for the transcript lane and outbound for any email summaries: a few cents a year at this scale. EventBridge Scheduler for the weekly drip and the one-off jobs: pennies.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the paste-a-link form and the approve/edit/skip buttons.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The system does nothing between pieces.
- **A big always-on index.** S3 Vectors is pay-for-what-you-store, not a running cluster. No vector database humming when no one's repurposing anything.
- **Heavy-model-by-default.** Haiku 4.5 does the routine reading and drafting. Sonnet 4.6 fires only on the harder pieces, so you don't pay for the big model on every clip.

How the cost scales

The bill tracks how many long pieces you feed it, because every piece means passages to score and drafts to write. Bedrock and embeddings grow roughly linearly with piece count; Lambda and DynamoDB grow with them but stay small. There's no background cost that grows on its own — a month where you repurpose nothing costs essentially nothing. So the bill at 80 pieces a month is around $24, and at 160 it's around $48. Past those volumes you're running a

content operation, not a side system, and you'd batch the model calls and cache embeddings harder — but those are tunings, not redesigns.

Set an AWS Budgets alarm at $15/month so an unusual spike (a giant transcript, a runaway retry loop) pages you before the bill matters. The normal-volume bill stays well under that ceiling.

Last post in the series: the engineering reference. Same system, drawn for engineers — service names, Lambda inventory, IAM scopes, the S3 Vectors index, DynamoDB schemas, and EventBridge Scheduler config.

PART 7 OF 7

MAY 24, 2026 PART 7 OF 7 · CONTENT REPURPOSER SERIES ~8 MIN READ

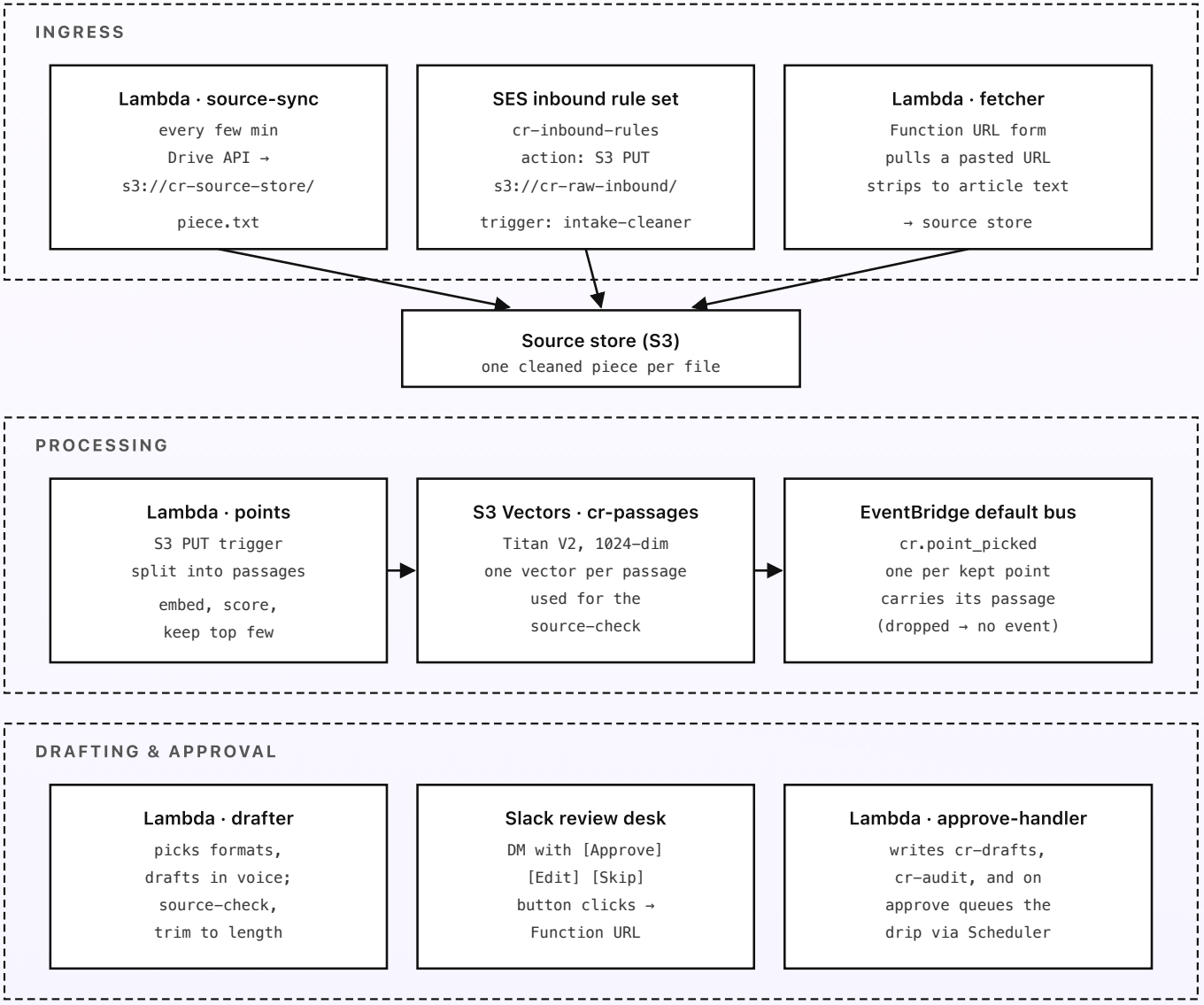
Engineering reference: the content repurposer architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, the S3 Vectors index, Lambda inventory, IAM scopes, the SES inbound rule set, EventBridge Scheduler config, the DynamoDB schemas, and the Slack interactive flow for the review desk. 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 cross-Region inference, S3 Vectors, and EventBridge Scheduler are all in good shape there. A second region for resilience isn't worth the setup at SMB volume — the failure mode here is a draft that shows up an hour late, not a regional outage, and nothing in this system is on a hard real-time path. One AWS account dedicated to the repurposer (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Every draft is grounded — and every approval is logged to cr-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the source store), processing (split, embed, score, emit a point event), drafting and approval (the drafter writes, the review desk shows, the approval 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.

- **`source-sync`** — EventBridge Scheduler target, fires every few minutes. Uses the Google Drive API (service-account credentials in Secrets Manager under `cr/drive/sa`) to export any new or changed doc in the source folder as plain text and write it to `s3://cr-source-store/<piece-id>.txt` only if it has changed since the last sync. Same pattern syncs the voice and rules docs to `s3://cr-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`intake-cleaner`** — S3 PUT trigger on `s3://cr-raw-inbound/`. Parses the forwarded MIME, pulls the transcript body (or attachment), and strips timestamps, speaker labels, and filler with a small set of format rules (Otter, Zoom, Fireflies, and plain-paste formats handled; unknown formats fall back to a generic line-and-timestamp regex). Writes the cleaned plain text into `s3://cr-source-store/` tagged `kind=transcript`. No Bedrock. Memory: 256 MB. Timeout: 30 s.

- **`fetcher`** — Lambda Function URL (the paste-a-link form). Fetches the single pasted URL, runs a readability extraction (`trafilatura`, with a `readability-lxml` fallback) to strip navigation, ads, and footer down to the article body, and writes plain text into `s3://cr-source-store/` tagged `kind=web`. Only fetches the exact URL submitted; no crawling, with an allowlist of schemes and a request timeout. Memory: 512 MB. Timeout: 30 s.
- **`points`** — S3 PUT trigger on `s3://cr-source-store/`. Splits the piece into passages (paragraph- and topic-boundary chunking, ~3–5 sentences each), calls Titan Text Embeddings V2 to embed each passage into the S3 Vectors index `cr-passages`, calls Bedrock Haiku 4.5 to score each passage for postability, keeps the top N per the rules doc, and emits one `cr.point_picked` event per kept point with the point and its source passage as payload. Dropped passages emit nothing. Memory: 512 MB. Timeout: 120 s.
- **`drafter`** — EventBridge rule on `cr.point_picked`. Reads the voice and rules docs, drafts each requested format with Bedrock Haiku 4.5 (`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`); routes the hard pieces (`kind=transcript` or a length/complexity flag) to Claude Sonnet 4.6 (`global.anthropic.claude-sonnet-4-6-20250930-v1:0`). Runs the source-check by embedding the draft and querying `cr-passages` for the nearest passage, dropping any claim not supported and re-prompting once if the draft drifted. Trims to the platform length. Writes each draft to `cr-drafts` and posts it to the Slack review desk. Memory: 512 MB. Timeout: 120 s.
- **`approve-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a Slack signature on the request body. Triggered by Slack interactive button clicks (Approve/Edit/Skip) and the Edit modal submission. Writes to `cr-`

`drafts` and `cr-audit`; on approve, queues the draft on the post scheduler (a one-off EventBridge Scheduler rule per drip slot) or routes it to the review channel; on skip, optionally requests a backup point from `points`. Memory: 256 MB. Timeout: 15 s.

- **`drip`** — EventBridge Scheduler one-off target. Sends one approved draft to its channel at its scheduled time (Slack channel post, or a webhook to the configured post scheduler). Reads `cr-drafts`, marks the draft `sent`, writes a `sent` row to `cr-audit`. No Bedrock. Memory: 256 MB. Timeout: 15 s.
- **`recap`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads the past week's `cr-audit`; sends a recap to a configured Slack channel: pieces repurposed, drafts approved, edited, skipped, and the approve rate per format. The message is a plain summary table; no Bedrock. Memory: 256 MB.

Storage

- **DynamoDB · `cr-drafts`** — one row per draft. PK `(piece_id, draft_id)`; attributes: `format` (thread/post/caption), `point_tier`, `passage_ref` (S3 Vectors id of the source passage), `model_text`, `final_text`, `status` (pending/approved/edited/skipped/sent). On-demand.
- **DynamoDB · `cr-audit`** — one row per write action of any kind. PK `(draft_id, ts)`; attributes: `action` (approve/edit/skip/sent), `by_user`, `before`, `after`, `passage_ref`. On-demand. No TTL — this is the long-term audit trail.
- **DynamoDB · `cr-pieces`** — one row per loaded piece. PK `piece_id`; attributes: `title`, `kind` (drive/transcript/web), `source_link`, `loaded_at`, `passage_count`, `status`. On-demand.

- **S3 Vectors** · `cr-passages` — one vector per passage, Titan Text Embeddings V2 at 1024 dimensions, with metadata `(piece_id, passage_id, position, text)`. Queried by the `points` scorer context and the `drafter` source-check.
- **S3** · `cr-source-store` — cleaned plain-text pieces, one file each. Versioning enabled. Lifecycle to Glacier at 90 days; expiry at 2 years.
- **S3** · `cr-rules-source` — mirrored voice and rules docs as plain text. Versioning enabled.
- **S3** · `cr-raw-inbound` — raw inbound MIME from forwarded transcripts. Lifecycle to Glacier at 30 days; expiry at 1 year.

Bedrock

- **Foundation models.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile for passage scoring and the routine drafting; `anthropic.claude-sonnet-4-6-20250930-v1:0` via its Global profile for the hard pieces only, selected by the `drafter` from source kind and a complexity flag.
- **Embeddings.** `amazon.titan-embed-text-v2:0` at 1024 dimensions, one call per passage, written to the S3 Vectors index `cr-passages`. This is what makes the grounding source-check a single nearest-neighbour lookup.
- **Quotas.** Default account quotas are more than enough at SMB volume. The system only calls Bedrock when a piece is loaded; there is no background traffic.

EventBridge Scheduler config

- `cr-source-sync` — `rate(5 minutes)`. Target: `source-sync` Lambda.
- `cr-weekly-recap` — `cron(0 18 ? * SUN *)` in `TZ_NAME`. Target: `recap` Lambda.
- **Drip one-offs** — created by `approve-handler` per approved draft at the slot the rules doc assigns (e.g. one a day at 9am local). Use `at(YYYY-MM-DDTHH:MM:SS)` expressions with `--action-after-completion DELETE` so each rule self-cleans after it fires `drip`.
- **Backup-refill one-offs** — created by `approve-handler` on a skip when auto-refill is enabled, targeting `points` to draft a replacement from a backup point.

SES inbound and outbound

- Set the MX record on a dedicated subdomain (e.g. `repurpose.your-company.com`) to `inbound-smtp.ap-southeast-1.amazonaws.com`.
- SES inbound rule set `cr-inbound-rules`: one rule with recipient `repurpose@your-company.com` → spam scan → S3 PUT to `s3://cr-raw-inbound/<message-id>` → stop. The S3 PUT triggers `intake-cleaner`.
- SES outbound for the weekly recap email (optional, if you prefer email to Slack): verify a sender identity at `repurposer@your-company.com` with DKIM and SPF on the parent domain. Out of sandbox by request.

IAM (least privilege per Lambda)

Each Lambda has its own role with policies scoped to exact ARNs. Sketch:

- **points role:** `s3:GetObject` on the source store; `bedrock:InvokeModel` on the Titan and Haiku ARNs; `s3vectors:PutVectors` + `QueryVectors` on `cr-passages`; `events:PutEvents` on the default bus; `dynamodb:PutItem` on `cr-pieces`.
- **drafter role:** `s3:GetObject` on the rules source; `bedrock:InvokeModel` on the Haiku and Sonnet ARNs; `s3vectors:QueryVectors` on `cr-passages`; `dynamodb:PutItem` on `cr-drafts`; `secretsmanager:GetSecretValue` on the Slack bot token; outbound network to `slack.com`.
- **approve-handler role:** `dynamodb:PutItem` + `UpdateItem` on `cr-drafts` and `cr-audit`; `scheduler:CreateSchedule` for the drip one-offs; `secretsmanager:GetSecretValue` on the Slack signing secret; `events:PutEvents` for the optional backup-refill.
- **intake-cleaner role:** `s3:GetObject` on `cr-raw-inbound`; `s3:PutObject` on `cr-source-store`. No Bedrock, no network egress.
- **source-sync and fetcher roles:** `secretsmanager:GetSecretValue` on the Google service-account secret (source-sync only); `s3:PutObject` on the source and rules buckets; outbound network to `www.googleapis.com` (source-sync) or the open web with a scheme allowlist (fetcher).

Slack interactive flow

Drafts are posted to the review desk via the `chat.postMessage` Web API with Block Kit blocks: the draft text, the format and point tier, the source passage in a context block, and three action buttons. Button clicks are sent by Slack to the configured Interactivity request URL, which is the `approve-handler` Function URL. `approve-handler` verifies the Slack signing secret on the inbound request,

parses the `action_id` (`approve`, `edit`, `skip`), opens a modal for Edit (pre-filled with the draft), and processes the response. Approve and Skip are one-tap; Edit submits through the modal.

The Slack app needs `chat:write` and `im:write`, and the Interactivity URL configured. The bot token lives in Secrets Manager under `cr/slack/bot-token`; the signing secret is `cr/slack/signing-secret`.

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:** `drafter` failure rate > 1% in 24h; source-check drop rate > some threshold (a spike means the model is drifting and the prompt or model choice needs a look); approve-handler signature-verification failures > 5/hour (might mean the Slack secret rotated).
- **X-Ray:** off by default. Not worth the cost at SMB volume.
- **AWS Budgets:** $15/month threshold, alarm at 80% and 100%, posts to SNS topic `cr-cost-alarm` subscribed to the admin's email and Slack.

Config and secrets

Service-account credentials for the Drive API live in Secrets Manager under `cr/drive/sa`. Slack bot token and signing secret under `cr/slack/*`. The post-scheduler webhook (if you drip to an external scheduler instead of a Slack

channel) under `cr/scheduler/webhook`. The configured timezone, drip slots, format mix, top-N point count, and model-routing thresholds all live in Parameter Store under `/cr/config/`, with the voice and rules docs themselves in Drive (mirrored to S3). Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC into a deploy role (no long-lived keys), building and shipping with AWS SAM. The opinionated bits: deploy the SES rule set as a separate stack (rule-set changes affect mail flow), turn on S3 versioning for `cr-source-store` and `cr-rules-source` so a bad edit can be rolled back in one click, and keep the S3 Vectors index in its own stack so re-indexing never forces a full app redeploy. Total deployable surface: around eight Lambdas, three DDB tables, one S3 Vectors index, three 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 business, see Work with me.