7-PART SERIES · FREE COMPANION

Social scheduler

A serverless scheduler that lets a small business draft every social post in one place, holds a calendar of when each one should go out, checks each post against the rules for the platform it's headed to, posts it at the scheduled time to the chosen channels, and reports what went out and what failed. Nothing leaves until a human approves it. Seven posts on the same system — one diagram at a time — with an engineering reference at the end.

CONTENTS

Social scheduler

PART 1 OF 7

MAY 13, 2026 PART 1 OF 7 · SOCIAL SCHEDULER SERIES ~5 MIN READ

A social scheduler on AWS for a few dollars a month

A small business has more posts to push out than anyone wants to do by hand at 9am every day. The Monday tip for Facebook. The Tuesday offer for Instagram. The job-opening for LinkedIn that someone keeps meaning to write. The owner drafts them in spare minutes, then forgets to actually post them — or posts a draft that was never finished, or one that's too long for the channel it landed on. This post walks through the design of a small scheduler that lets you draft every post in one place, posts each one at the right time to the right channels, and tells you afterward what went out and what failed.

KEY TAKEAWAYS

- Three sources for posts: a Drive sheet, an optional draft-helper, and a recurring-template lane.
- Every post ends in one of four moves on each tick: resting, queue, send, or retry.
- Per-channel format rules: character limits, image sizes, and link counts are checked before anything is queued.
- Nothing posts until a human approves it. A half-finished draft never goes out by accident.
- Designed on AWS for about $2/month at typical small-business volume.

The whole system on one page

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

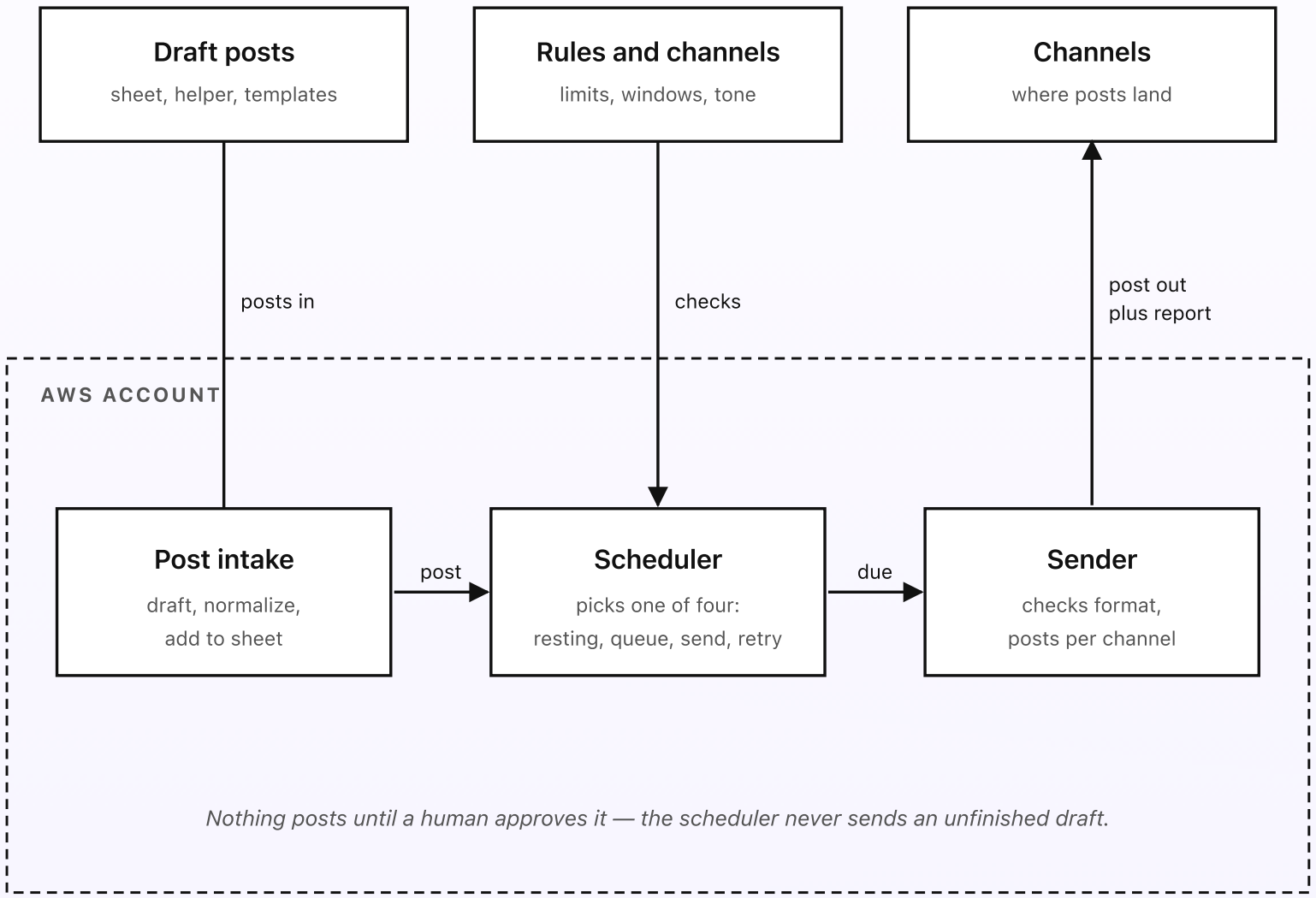

Draft posts
sheet, helper, templates
Rules and channels
limits, windows, tone
Channels
where posts land
posts in
checks
post out
plus report
AWS ACCOUNT
Post intake
draft, normalize,
add to sheet
post
Scheduler
picks one of four:
resting, queue, send, retry
due
Sender
checks format,
posts per channel
Nothing posts until a human approves it — the scheduler never sends an unfinished draft.

Fig 1. Three sources outside, three pieces inside AWS. Posts flow in from a Drive sheet, an optional draft-helper, and a recurring-template lane. The Scheduler runs daily and picks one of four moves. The Sender posts each one to the right channel at the right time.

What you set up once (the outside)

- **Draft posts.** A Google Sheet in a Drive folder, one row per post: the text, the channels to send to (Facebook, Instagram, LinkedIn, and so on), the scheduled date and time, a link to the image in the same folder, and an approval flag. You can fill it in whenever you have a spare minute; posts can also enter via two other lanes covered in Part 2 — a draft-helper lane (type a short note and the helper roughs out the post text for one-tap approval) and a recurring-template lane (standing posts like "Friday hours reminder" get dropped onto the calendar on a schedule you set).
- **A rules folder.** Two short Google Docs in a Drive folder. The *rules* doc covers the format limits for each platform — the character count, the image sizes and types the platform accepts, and the link count. It also lists the channels you post to, the posting windows (no posts before 8am or after 8pm by default), and any blackout days. The *voice* doc holds your brand tone — what the draft-helper should sound like when it roughs out text.
- **Channels.** The social accounts you post to. Each channel has a token stored safely in AWS that lets the scheduler post on your behalf. After each post, you get a short report — which channels it reached, a link to the live post, and any channel that failed with the reason why.

What runs on every tick (the inside)

- **The post intake.** Three sources feed the sheet. The Drive sheet itself is the canonical store. New posts can also be added via the draft-helper lane (type a short note like “remind people we’re closed Monday” and Bedrock Haiku 4.5 roughs out post text in your brand voice, then drops a one-tap approval card so you confirm before the row is added) and the recurring-template lane (standing posts get dropped onto the calendar by a small Lambda on the schedule you set).
- **The scheduler.** Runs once a day at 7am local. Reads the calendar. For each approved post, computes how far away its scheduled time is. Picks one of four moves. *Resting*: the post is more than a day away or not approved yet — do nothing. *Queue*: the post is due today — book a one-off send for the exact minute it’s scheduled. *Send*: the minute has arrived — hand it to the Sender. *Retry*: a channel failed earlier and is eligible for another attempt — queue it again with a small delay. The scheduler doesn’t call a model on the daily tick — the move logic is plain Python.
- **The Sender.** Reads the rules doc, checks the post against the format limits for each chosen channel, and posts it. A post that fails any check is flagged back to the owner with the exact reason instead of being sent. A post that passes is sent to each channel through that channel’s posting interface. Every send writes a row in DynamoDB so the report can tell what went out and what failed. A weekly digest summarizes everything that posted that week, plus what’s coming up. A monthly recap writes a short narrative: count by channel, top posts, anything that failed.

In plain words

You draft a Tuesday-morning offer for Instagram and Facebook on Sunday night. You set the time to 9am Tuesday, drop in an image, and flip the approval flag. The scheduler sees it on Monday's tick, checks the caption fits Instagram's limit and the image is the right size for both channels, and books a send for 9am Tuesday. At 9am Tuesday the Sender posts it to both channels and records two links. At 9:05 you get a short report: "Tuesday offer — posted to Facebook and Instagram. *[links]*." If Instagram had rejected the image for being too small, the report would say exactly that, and nothing would have gone out half-broken.

The cost of running this is about $2 a month at SMB volume. The cost of *not* running it is the offer that never went out because Tuesday got busy, or the caption that posted truncated mid-sentence, or the image the platform silently dropped.

DESIGN RULES THAT SHAPED EVERY DECISION

- Nothing posts until a human approves it. The approval flag is the one gate every post passes.
- Four moves, always. Resting, queue, send, retry. There is no fifth.
- Posting windows and blackout days are respected. A post scheduled for 2am waits for the morning.
- Every post is format-checked before it's queued. A post that fails is flagged, not sent.
- The sheet lives in Drive. Adding a post, changing a channel, or shifting a time doesn't need a deploy.
- Every send is logged. Look back next month and you can see every post that went out and every one that failed.

Why this shape

Most small businesses handle social posts in one of three ways: a person who posts by hand and forgets half the time, a third-party scheduling tool that costs a monthly subscription per seat, or a spreadsheet of "things we should post" that nobody opens. The by-hand way fails the first busy week. The paid tool works but charges every month whether you post once or a hundred times. And the spreadsheet, of course, is just a list of good intentions.

The setup above keeps drafting in a doc the team already edits, but adds a small system that *reads* that doc every day and posts only what's approved and due.

Posts go out on time, in the right format, to the right channels. The owner stays in control — nothing leaves without approval — and the report afterward means you never wonder whether something actually posted. The scheduler is invisible most of the day; visible only when it posts something or when something needs a fix.

The next four posts walk through each piece in turn: how a post gets drafted, how the schedule fires on time, how a post gets formatted per channel, and how a post gets approved or held. One diagram per post. A cost breakdown and a final engineering reference at the end.

PART 2 OF 7

MAY 13, 2026 PART 2 OF 7 · SOCIAL SCHEDULER SERIES ~4 MIN READ

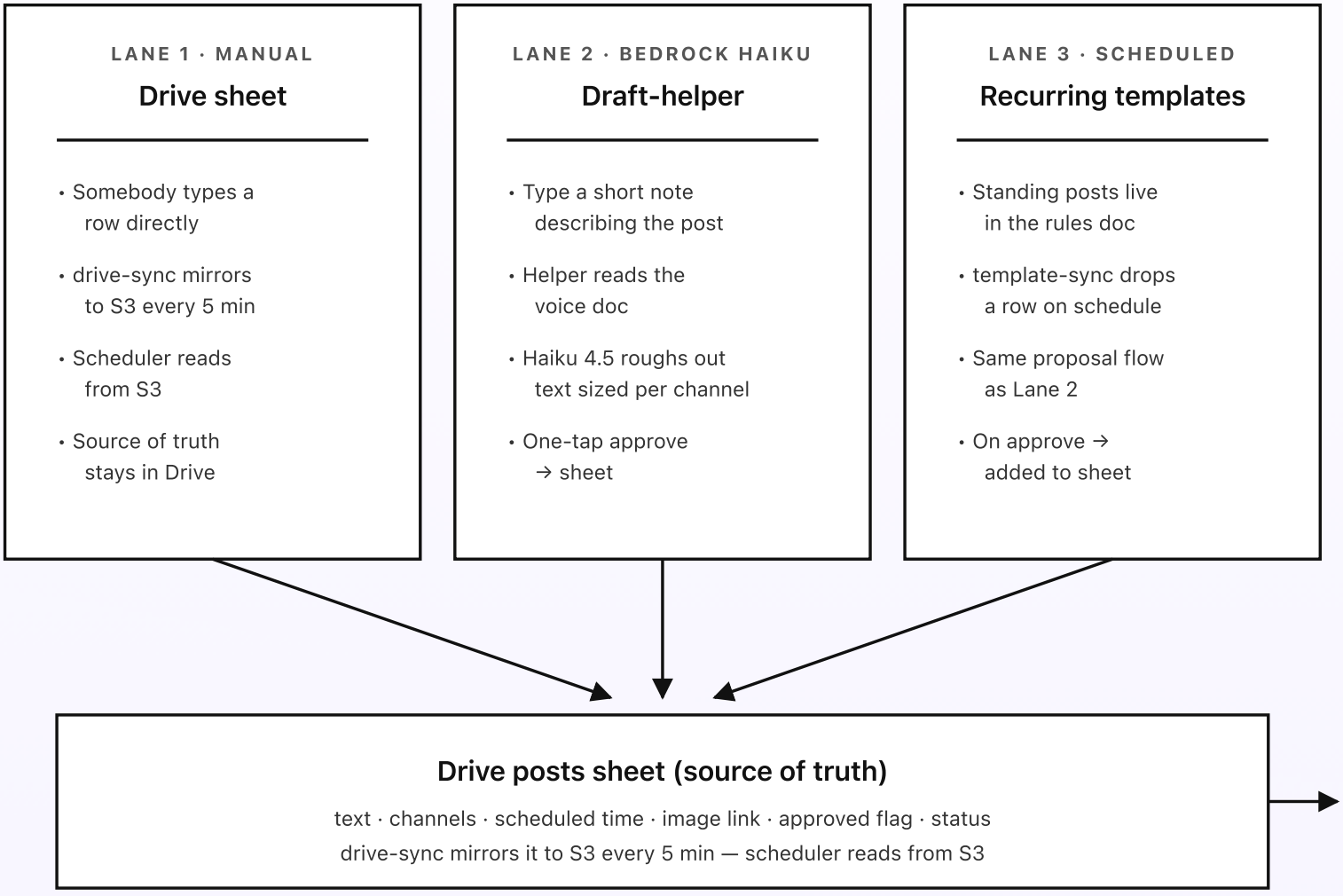
How a post gets drafted

The scheduler only sends what's in the sheet. So the first job is making it easy to get a post into the sheet in the first place. There are three ways: somebody types a row in the Drive sheet, somebody types a short note and lets the draft-helper rough out the text, or a standing post drops onto the calendar from a template. The first one is obvious. The other two exist because in real life nobody wants to write a full caption from scratch at 8am, and some posts you want to send every single week.

KEY TAKEAWAYS

- Three intake lanes feed one sheet: the Drive sheet, a draft-helper lane, and a recurring-template lane.
- The draft-helper turns a short note into post text via Bedrock Haiku 4.5 in your brand voice.
- Every helper draft goes to a one-tap approval card before it lands in the sheet.
- Recurring templates drop standing posts onto the calendar on the schedule you set.
- The sheet stays the canonical store. The other lanes are conveniences that write into it.

Three lanes into one sheet



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

Fig 2. Three lanes converge on one Drive sheet. The sheet is the source of truth; the draft-helper and the template lane are conveniences that propose rows for human approval. The drive-sync Lambda mirrors the sheet to S3 so the scheduler can read it without hitting Drive on every tick.

Lane 1: the Drive sheet itself

The simplest lane. Open the posts sheet in Drive, add a row, save. The columns are short: the text, the channels to send to, the scheduled date and time, a link to the image in the same folder, an approval flag, and a status the scheduler keeps up to date. A small Lambda — `drive-sync` — runs every five minutes, exports the sheet as plain CSV via the Drive API, and writes it to `s3://ss-posts-source/posts.csv` if the sheet has changed since the last sync. The scheduler reads from S3, not Drive directly. That keeps Drive API calls predictable and gives you S3 versioning for free, so a bad bulk-edit can be rolled back in one click.

This lane covers the cases where you already know exactly what you want to say and when. You type the caption, set the time, drop in the image, and flip the approval flag. Most posts go in this way.

Lane 2: the draft-helper (the lane that saves the most time)

Writing a caption from a blank box is the part everyone puts off. The draft-helper takes the friction out. Type a short note — "remind people we're closed Monday for the holiday" — into a helper row, pick the channels, and a helper Lambda takes it from there. The Lambda reads your brand tone from the *voice* doc and calls Bedrock Haiku 4.5 with a short prompt: "Write a social post from this note, in this voice, sized for these channels. Return one version per channel. Do not invent facts the note doesn't contain."

The output goes to a small interactive review card: the proposed text per channel, a note on the character count for each, and three buttons — *approve*, *edit*, *discard*. On *approve*, a Lambda writes the row (or rows) to the Drive sheet via the Sheets API with the approval flag unset, so it still passes through a final approval before it can post. On *edit*, you get a fillable box pre-populated with the draft. On *discard*, the proposal is logged and dropped.

The reason every helper draft goes to a human first is simple: a caption the model got slightly wrong is worse than no caption at all. The wrong one will happily post your business's name next to a sentence you'd never have written. The helper writes; a person decides.

Lane 3: recurring templates

Some posts you want to send on a rhythm. The Friday "weekend hours" reminder. The first-of-the-month "here's what's new" note. The Monday motivational line. Typing those out every week is exactly the kind of chore that quietly stops happening.

Lane 3 keeps standing posts in a *templates* section of the rules doc, each with its own schedule ("every Friday at 9am," "first Monday of the month"). A small `template-sync` Lambda runs on those schedules, drops a fresh row onto the calendar a few days ahead, and runs it through the same review card as Lane 2. You get a chance to tweak the wording or skip it for the week before it's queued. A template you never touch just keeps proposing the same reliable post; one you edit each time still saves you the blank box.

Recurring templates are the most opt-in of the three lanes. A team that doesn't use them loses nothing; a team that does never forgets the standing posts again.

Why the sheet stays the source of truth

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

Next post: how the scheduler reads the calendar, computes when each post is due, and picks one of four moves.

PART 3 OF 7

MAY 13, 2026 PART 3 OF 7 · SOCIAL SCHEDULER SERIES ~5 MIN READ

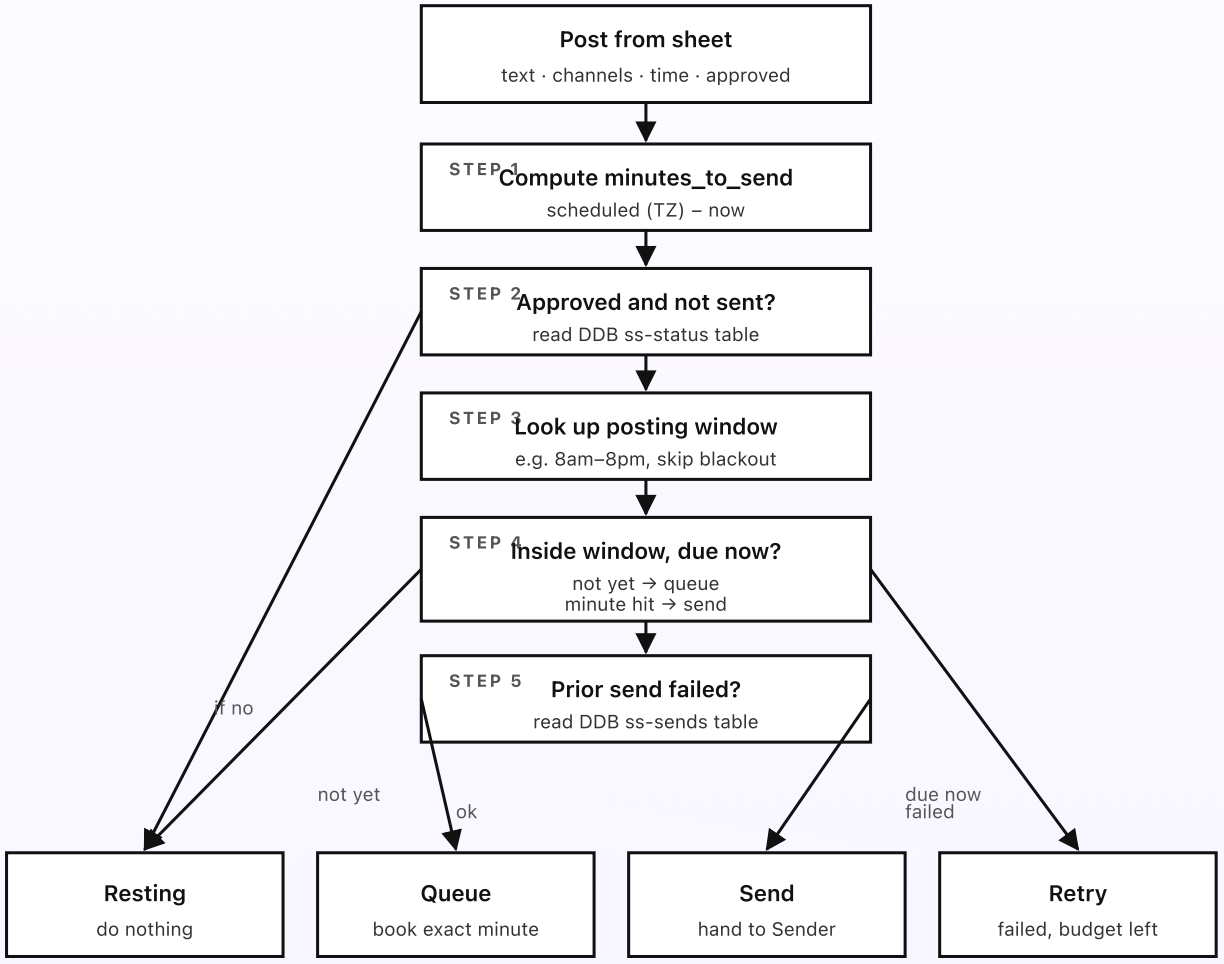
How the schedule fires on time

Once a day, at 7am local time, an EventBridge Scheduler rule fires the scheduler Lambda. The Lambda reads the calendar, looks at one approved post at a time, works out when it's due, and decides whether to leave it alone or to book it for sending — and if it's already due, hand it off. The whole decision is plain Python. No model. No guesswork. Every posting window lives in the rules doc, where the owner can edit it without a deploy.

KEY TAKEAWAYS

- The scheduler runs once a day via EventBridge Scheduler at 7am local time.
- Posting windows live in the rules doc — default is no posts before 8am or after 8pm, plus blackout days.
- Four moves per post, every tick: resting, queue, send, retry.
- DynamoDB tracks each post's status so a post is never double-sent.
- The scheduler itself never calls a model. The decision is entirely deterministic.

The decision flow, per post



The rules doc holds the windows — change one and tomorrow's tick uses the new value.

Fig 3. The scheduler's decision tree, per post, per daily tick. Five steps decide which of four moves applies. The rules doc holds the posting windows; the scheduler only enforces them.

Posting windows: 8am to 8pm isn't magic, it's in the doc

The rules doc has one short section for the posting windows. It names them in plain prose: "Post only between 8am and 8pm local. No posts on Sundays. Skip these blackout dates: Dec 25, Jan 1." The window is when posts are allowed to leave. A post scheduled for 2am isn't sent at 2am; the scheduler nudges it to the first allowed minute — 8am the same morning, or the next non-blackout day if the whole day is blocked.

The windows exist for a reason. A post that lands at 3am gets buried by morning and barely seen. A post on a blackout day — a holiday you're closed for, a day you'd rather stay quiet — is one you scheduled by mistake or want to skip. Different businesses have different rhythms; the windows reflect that.

Per-post overrides exist too. The sheet has an optional column called `send_exact`. Set it to yes and the scheduler honors the exact minute you typed, even outside the normal window — the right escape hatch for the time-sensitive announcement you want out at 6am sharp.

Four moves, always

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

- **Resting.** The post is more than a day away, or it hasn't been approved, or it's already gone out. Do nothing. Most posts, most ticks, are resting.
- **Queue.** The post is approved and due today but the minute hasn't arrived. Book a one-off EventBridge Scheduler rule for the exact send minute. Write a row to the `ss-status` DynamoDB table marking the post as queued so the next tick doesn't book it twice.
- **Send.** The send minute has arrived. Hand the post to the Sender, which does the format check and the actual posting (Part 4). Mark the post as sending in DynamoDB.
- **Retry.** A channel failed on a previous attempt — the platform was briefly down, the token needed a refresh — and the retry budget for that post isn't spent. Queue it again after a short delay (a few minutes, growing on each attempt, up to a small cap). When the budget runs out, the post is marked failed and the owner is told. Most sends never need a retry; the budget is there for the rare flake.

State that makes the decision deterministic

The scheduler reads two DynamoDB tables every tick. `ss-status` records each post's current state: `(post_id, state, queued_for)` where state is one of draft, queued, sending, sent, or failed. `ss-sends` records every actual send attempt: `(post_id, channel, attempt, result, posted_url_or_error)`. With those two tables, the move-decision logic is a few dozen lines of Python and zero magic.

A given post with a given scheduled time, a given approval flag, and a given send history always produces the same move. Re-running the tick produces no extra posts (because the state in DynamoDB shows what already fired).

Holding a post is an explicit reset: its state drops back to draft and any queued one-off send is cancelled. Sending it now is the opposite: the state jumps straight to sending. Part 5 covers those actions in detail.

Why the daily tick uses no model

The scheduler could call a model on the tick to pick a “better” time to post, or to decide whether a post is worth sending at all. It doesn’t. Two reasons. First, the daily tick should be the one part of the system that is utterly predictable — if the sheet says post at 9am and it’s approved, it posts at 9am. A model in that loop introduces variance the owner can’t reason about. Second, model calls cost money, and most posts most ticks are resting, so the call would be wasted nearly every time.

Bedrock fires elsewhere — on the draft-helper lane in Part 2, and on the monthly recap mentioned in Part 6. Not on the daily tick. The scheduler itself is plain Python that reads a calendar and books sends.

Next post: how a post gets formatted per channel, how the format checks run, and what the Sender does when a check fails.

PART 4 OF 7

MAY 13, 2026 PART 4 OF 7 · SOCIAL SCHEDULER SERIES ~5 MIN READ

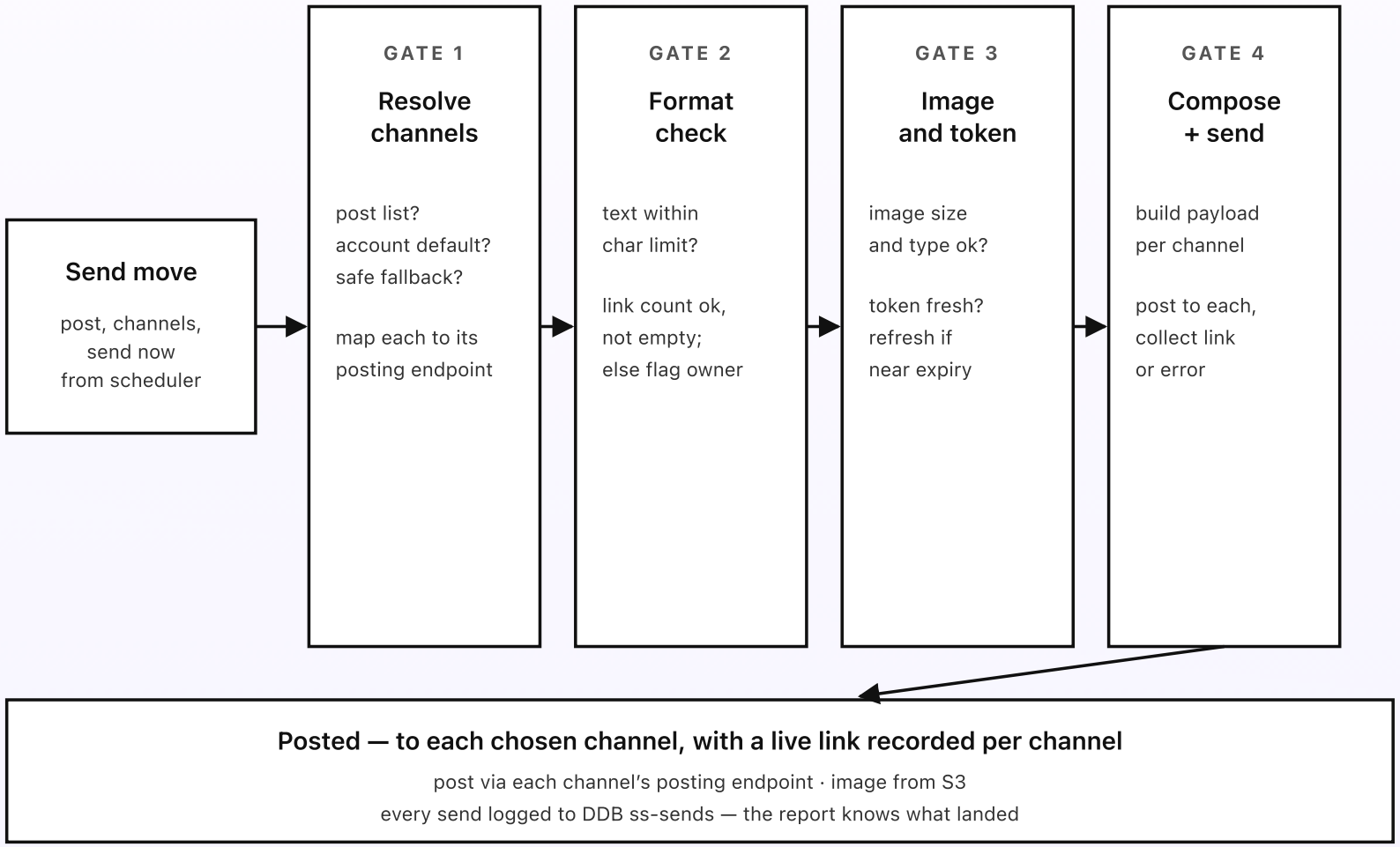
How a post gets formatted per channel

The scheduler said send. Now the Sender Lambda has to figure out which channels this post is headed to, whether the text and image fit each one's rules, whether the channel token still works, and only then post it. Get any of those wrong and the post is worse than no post: a caption chopped off mid-word, an image the platform silently dropped, a link that broke the layout. Four small guardrails sit between the send move and the post actually landing.

KEY TAKEAWAYS

- Channel resolution: per-post channel list beats the account default beats a safe fallback.
- Text is checked against each channel's character limit and link count before anything goes out.
- Images are checked for size and type each channel accepts; a bad image fails the post, not silently.
- The channel token is checked for freshness; a stale token is refreshed before the send.
- A post that fails any check is flagged back to the owner with the exact reason instead of being sent.

Four guardrails on every send



A post that fails any gate is flagged to the owner with the reason — it is never sent half-broken.

Fig 4. Four guardrails between the send move and the posted message. Resolve the channels. Check the text. Check the image and token. Compose and send. Then post to each channel and log the result so the report knows what landed.

Gate 1: resolve the channels

Three places the Sender looks for the channels of a post, in order. First, the sheet's per-post `channels` column — if a row names specific channels, those are the ones, regardless of the account default. Second, the account-default channel list in the rules doc ("every post goes to Facebook and Instagram unless told otherwise"). Third, a single safe fallback channel — usually the main page — so a post with a blank channel cell still has somewhere to go rather than failing silently.

Once the Sender knows which channels to post to, it maps each one to its posting endpoint and the token that authorizes it. The token for each channel lives in Secrets Manager. Different platforms have different posting interfaces, but from the Sender's point of view each channel is just "an endpoint plus a token plus a set of format rules."

Gate 2: format check

The rules doc lists the format limits for each platform: the maximum character count for the text, the maximum number of links the platform renders cleanly, and any required fields. Gate 2 checks the post's text against the limits for every chosen channel. A caption that fits Facebook but overruns Instagram fails for Instagram with the exact count — "caption is 2,310 characters; Instagram's limit is 2,200" — rather than being chopped off when it posts.

A post that fails the format check isn't sent. Instead the Sender writes the failure to the post's status with the reason, and the owner gets a flag: "This post is too long for Instagram. Shorten it or drop Instagram from the channel list." The post goes back to draft and waits for a fix. Catching the problem before the send beats catching it after, when the truncated post is already public.

Gate 3: image and token

If the post has an image, Gate 3 fetches it from S3 (the drive-sync Lambda mirrored it there from the Drive folder) and checks its size and type against what each chosen channel accepts. Platforms reject images that are too small, too large, or in a format they don't support — and they often do it silently, posting the text with no picture. Gate 3 turns that silent failure into an explicit one: a bad image fails the post with the reason, so the owner can swap it before anything goes out.

Gate 3 also checks the channel token. Social tokens expire. If a token is close to expiry, the Sender refreshes it (using the refresh credential in Secrets Manager) before the send, so a post never fails just because a token lapsed overnight. If a token can't be refreshed — the account was disconnected, the password changed — the post fails with "the Facebook connection needs to be re-linked," which is a fixable, clear message rather than a mystery.

Gate 4: compose and send

With the channels resolved, the text checked, and the image and token verified, Gate 4 builds the final payload for each channel — the text sized for that platform,

the image attached, any required fields filled — and posts it through that channel's posting endpoint. Each channel is posted independently, so if Facebook succeeds and Instagram briefly times out, the success on Facebook still counts and only Instagram is queued for a retry.

For each channel, the Sender collects either the live post link (on success) or the error message (on failure) and writes a row to `ss-sends` in DynamoDB. That row is what the report reads: "Tuesday offer — posted to Facebook *[link]*, posted to Instagram *[link]*." A failure shows up the same way with the reason, so the owner always knows exactly what happened.

Why the guardrails exist

None of these gates are exotic. They're the kind of small care a thoughtful person would take if they were posting by hand — check it's going to the right places, make sure it fits, make sure the picture works, then post it and grab the link. Putting them in code as four small sequential gates makes them part of the design, not a feature you're trusting the writer of any one post to remember at 9am.

Next post: how a post gets approved or held once it's in the queue — the three actions on the Review button and how the sheet, the status, and the audit trail stay in sync.

PART 5 OF 7

MAY 13, 2026 PART 5 OF 7 · SOCIAL SCHEDULER SERIES ~5 MIN READ

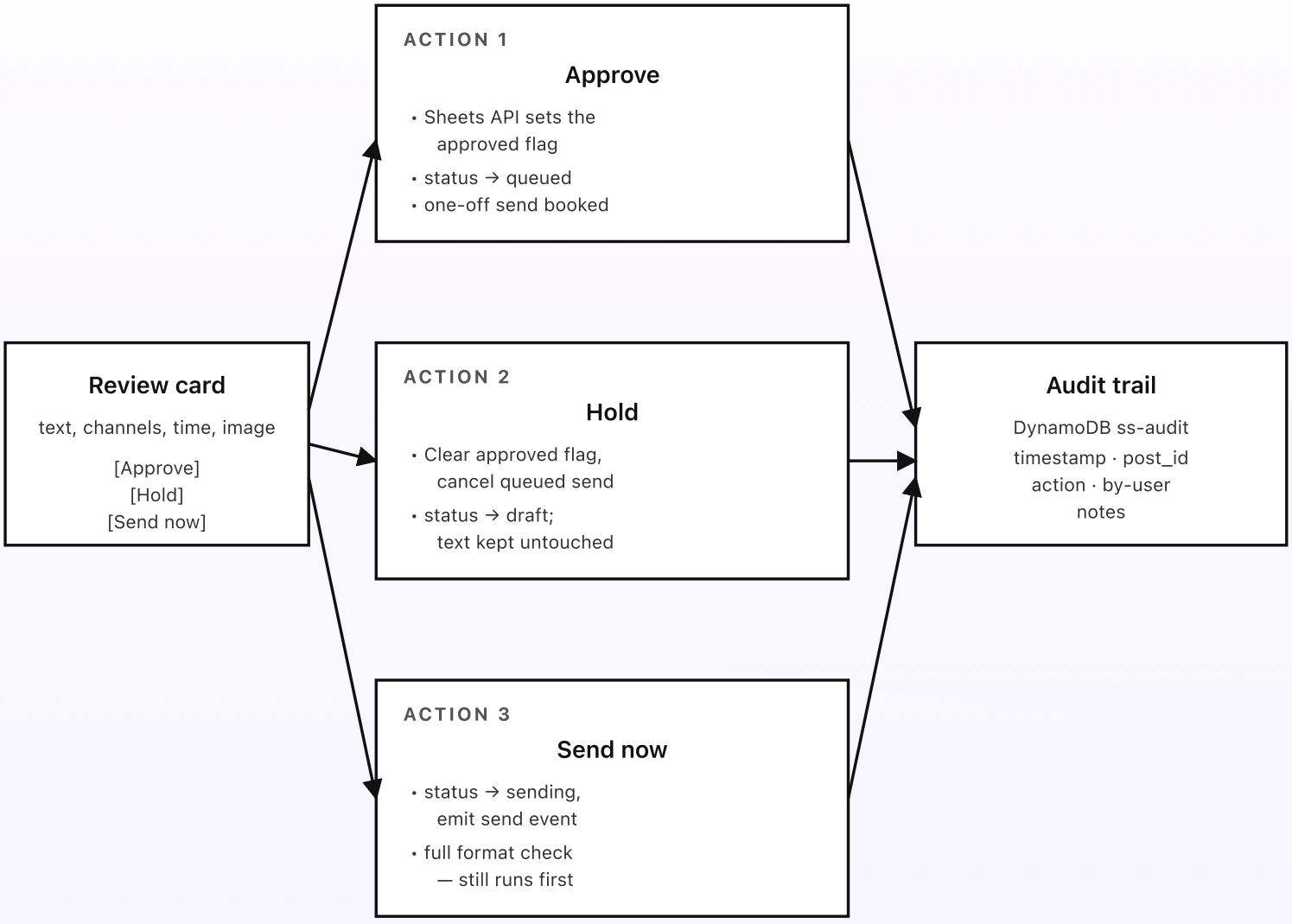
How a post gets approved or held

A review card lands in the owner's inbox the morning before a post is due. The Tuesday offer is drafted and scheduled for 9am tomorrow. There's a Review button. What happens when she taps it? The honest answer is "it depends on what she actually did." This post walks through the three things she can do — approve, hold, send now — and how the sheet, the post's status, and the audit trail all stay in sync.

KEY TAKEAWAYS

- Three actions per review: *approve* (lock it in for its time), *hold* (pull it back to draft), *send now* (post immediately).
- Each action updates the posts sheet via the Sheets API and writes an audit row.
- Approve books a one-off EventBridge Scheduler send for the exact minute the post is due.
- Hold cancels any queued send and drops the post back to draft without losing the text.
- The Review button is an interactive card backed by a Function URL.

Three actions on Review



Hold never deletes a post — it pulls it back to draft. Nothing posts without approval.

Fig 5. Three actions per review, three different effects. Approve locks the post in and books its send. Hold drops it back to draft without losing it. Send now posts it immediately, still through the full format check. Every action writes to the audit trail.

Action 1: approve (the most common)

The owner read the Tuesday offer, likes it, and wants it to go out at 9am as planned. She taps *Approve*. The Review card's buttons submit to a Function URL Lambda, and three things happen, in order. First, the Sheets API sets the `approved` flag on the row in the posts sheet, with a small note in the `approved_by` column naming her and the time. Second, the post's status in `ss-status` moves to *queued*, and a one-off EventBridge Scheduler rule is booked for the exact send minute — 9am Tuesday. Third, an `action: approved` row is written to `ss-audit` with the user, the timestamp, and the scheduled time.

At 9am Tuesday the one-off rule fires, the Sender runs the format check, and the post goes out. The owner won't need to touch it again unless she wants to change something — in which case Hold brings it back.

Action 2: hold (the "not yet")

Sometimes a post isn't ready. The offer changed. The image is wrong. The timing no longer makes sense because something else is going on that day. The owner isn't deleting the post — she just doesn't want it to go out as-is.

Hold clears the `approved` flag, cancels any queued one-off Scheduler rule for the post, and drops the status back to *draft*. The text, the image, the channels, and

the scheduled time all stay exactly as they were — nothing is lost. The post simply sits in draft until the owner edits it and approves it again, or changes the time, or decides to drop it. Because Hold cancels the queued send, a held post can't surprise anyone by going out anyway.

Hold is the safety valve that makes the whole approval model work. Knowing that any queued post can be pulled back at any time, right up until the send minute, is what lets the owner approve posts a day or two ahead without worrying about being locked in.

Action 3: send now (the "right now")

Sometimes a post can't wait for its scheduled minute. Something just happened — a sold-out item is back, a weather closure, a last-minute event — and the owner wants it out immediately.

Send now sets the status to *sending* and emits a send event right away, skipping the wait for the scheduled minute. It does *not* skip the format check: the post still runs through all four guardrails from Part 4 before anything posts, so a "send now" can still fail cleanly if the caption is too long or the image is wrong. The only thing Send now changes is the timing — it never lowers the bar on what's allowed to go out.

If a post that was scheduled for later is sent now, its original one-off Scheduler rule is cancelled so it doesn't fire a second time. The audit row records that it went out early and who pushed it, so the history is clear.

Every action is logged, every action is reversible

The `ss-audit` table records every approve, hold, and send-now with the user who took the action, the timestamp, and a snapshot of the row before and after. If a post went out that shouldn't have, or a wrong time slipped in, a quick look at the audit trail shows exactly what changed and who changed it. Holding a post that's already gone out can't un-post it — nothing can — but the audit trail at least makes it instantly clear how it happened, so the fix is to delete it on the platform and adjust the template, not to go hunting.

This kind of clear record matters most when more than one person can draft posts. The next time a post goes out at an odd time, the audit trail is the only memory anyone needs to see who approved it and when.

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 it's cheaper than a per-seat scheduling tool.

PART 6 OF 7

MAY 13, 2026 PART 6 OF 7 · SOCIAL SCHEDULER SERIES ~3 MIN READ

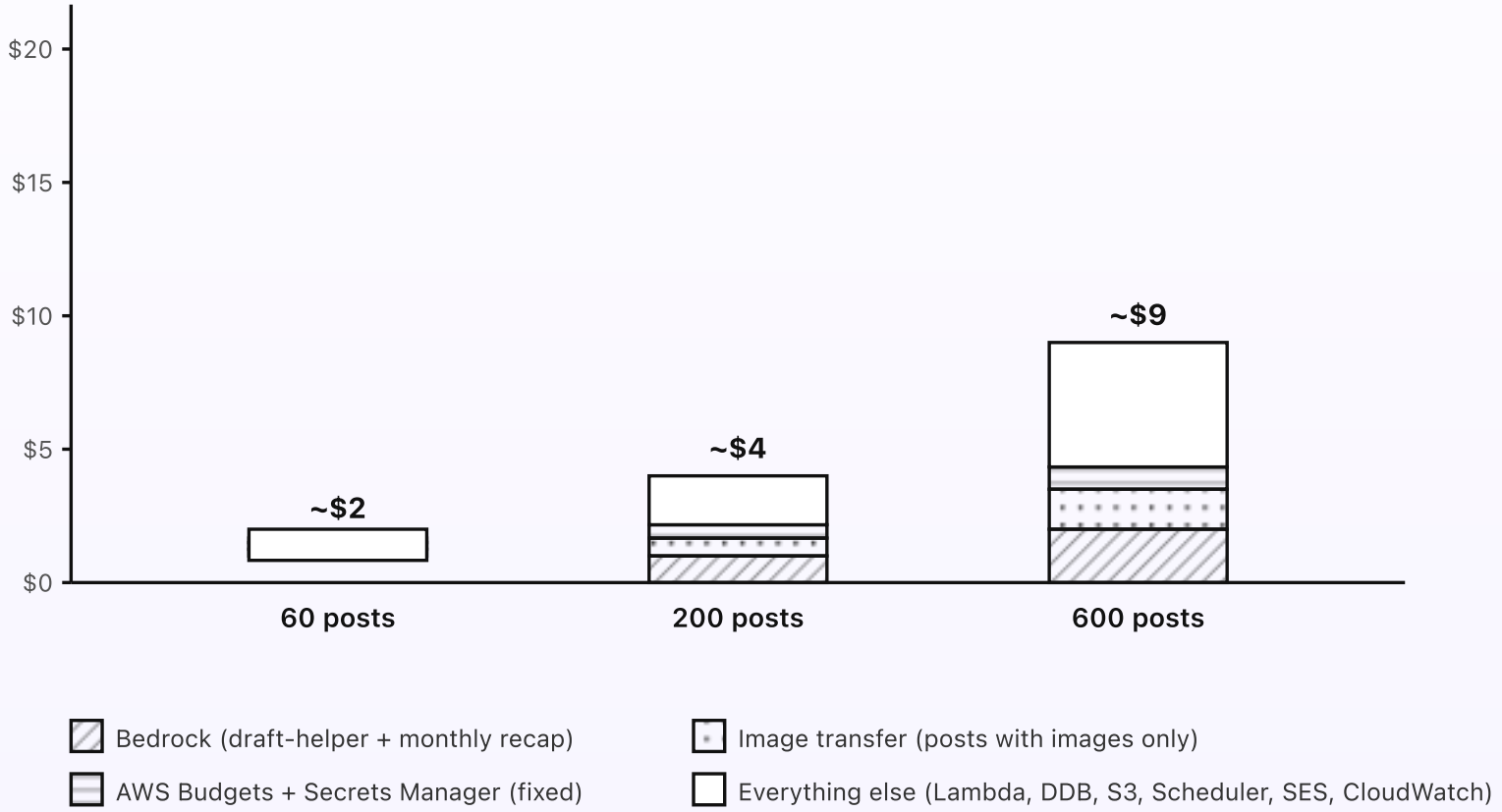
What the social scheduler costs

The scheduler is one of the cheapest systems in this whole series. The daily tick reads a CSV from S3, does some date arithmetic, writes a few rows to DynamoDB, and books a handful of sends. It calls no models on the tick. Bedrock fires only when you ask the draft-helper to rough out a post and once a month for the recap. At typical SMB volume, the bill is a couple of dollars a month, fixed cost essentially zero.

KEY TAKEAWAYS

- Around $2/month at typical SMB volume (around 60 posts a month across a few channels).
- Fixed AWS cost is essentially zero. No always-on compute, no NAT Gateway, no API Gateway.
- The daily tick costs pennies — no model calls.
- Bedrock fires only on the optional draft-helper (a few times a month) and the monthly recap.
- At 200 posts a month the bill is around $4. At 600 posts it's around $9.

Cost at three volumes



The send and tick Lambdas are the dominant cost — and even that is fractions of a cent per post.

Fig 6. Monthly cost at three post volumes. Bedrock and image transfer are small slivers because Bedrock only fires on the draft-helper and the monthly recap. The dominant cost is the everything-else bucket: the daily tick and the per-post sends.

Where the dollars actually go

Lambda runtime (the bulk). The scheduler runs once a day. Each tick reads the posts CSV from S3, iterates the rows, computes when each is due, and books the day's sends. At 60 posts a month, that's a few hundred milliseconds a day. Add the Sender Lambda firing for each post (60 to 600 sends a month), the Function URL Lambda for approvals, the template-sync Lambda, and the drive-sync Lambda every five minutes — the Lambda total still lands under a couple of dollars at all three volumes.

DynamoDB on-demand. Three small tables: `ss-status`, `ss-sends`, `ss-audit`. Reads are dominant during the daily tick (one read per post per tick). Writes are send results and audit rows. Pennies a month at any of these volumes.

S3 + storage. The mirrored posts CSV plus the images for any posts that carry one. A few MB total at SMB volume. Effectively free, plus a small transfer cost when an image is fetched for a send.

EventBridge Scheduler. The daily tick rule plus one one-off rule per queued post for its exact send minute. A few dozen invocations a month at 60 posts. Pennies.

SES. Outbound for the review cards and the reports: $0.10 per thousand sent. A handful of emails a day, so a couple of cents a year. Negligible at this scale.

Bedrock (only when something fires it). The daily tick uses no Bedrock. The draft-helper fires Haiku 4.5 once per draft request: a short note in, a few hundred output tokens out, so a fraction of a cent per draft. At a few drafts a month, Bedrock costs cents. The monthly recap is one larger call: write a short paragraph summarizing the month's posts; a couple of cents.

Image transfer (only on posts with images). Fetching an image from S3 and handing it to each channel is a small data-transfer cost per post. A few cents a month at 60 posts; a bit more at 600. Posts without images skip it entirely.

What doesn't cost money

- **API Gateway.** Replaced by Lambda Function URLs for the approval endpoints.
- **NAT Gateway.** Nothing is in a VPC. No NAT, no $32/month minimum.
- **Always-on compute.** No EC2, no Fargate. The scheduler sleeps almost all day.
- **A per-seat subscription.** No monthly tool fee per user — you pay AWS for what you actually post, not a flat rate.
- **Models on the tick.** The daily decision is plain Python. Bedrock fires only on the draft-helper and the monthly recap.

How the cost scales

Lambda runtime grows roughly linearly with post count, because every post is read on every tick and every send is one Sender run. DynamoDB grows linearly too. Bedrock is uncorrelated with post count — it only fires when you ask the draft-helper or it's the first of the month. So the bill at 1,500 posts a month is around $20; at 3,000 it's around $38. Past those volumes you're posting far more than a small business usually does, and a few targeted optimizations (batching the tick, caching channel rules) keep it flat — those are tweaks, not redesigns.

Set an AWS Budgets alarm at $15/month so anything unusual pages you before the bill matters. The scheduler's 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, DynamoDB schemas, the channel-token setup, and EventBridge Scheduler config.

PART 7 OF 7

MAY 13, 2026 PART 7 OF 7 · SOCIAL SCHEDULER SERIES ~8 MIN READ

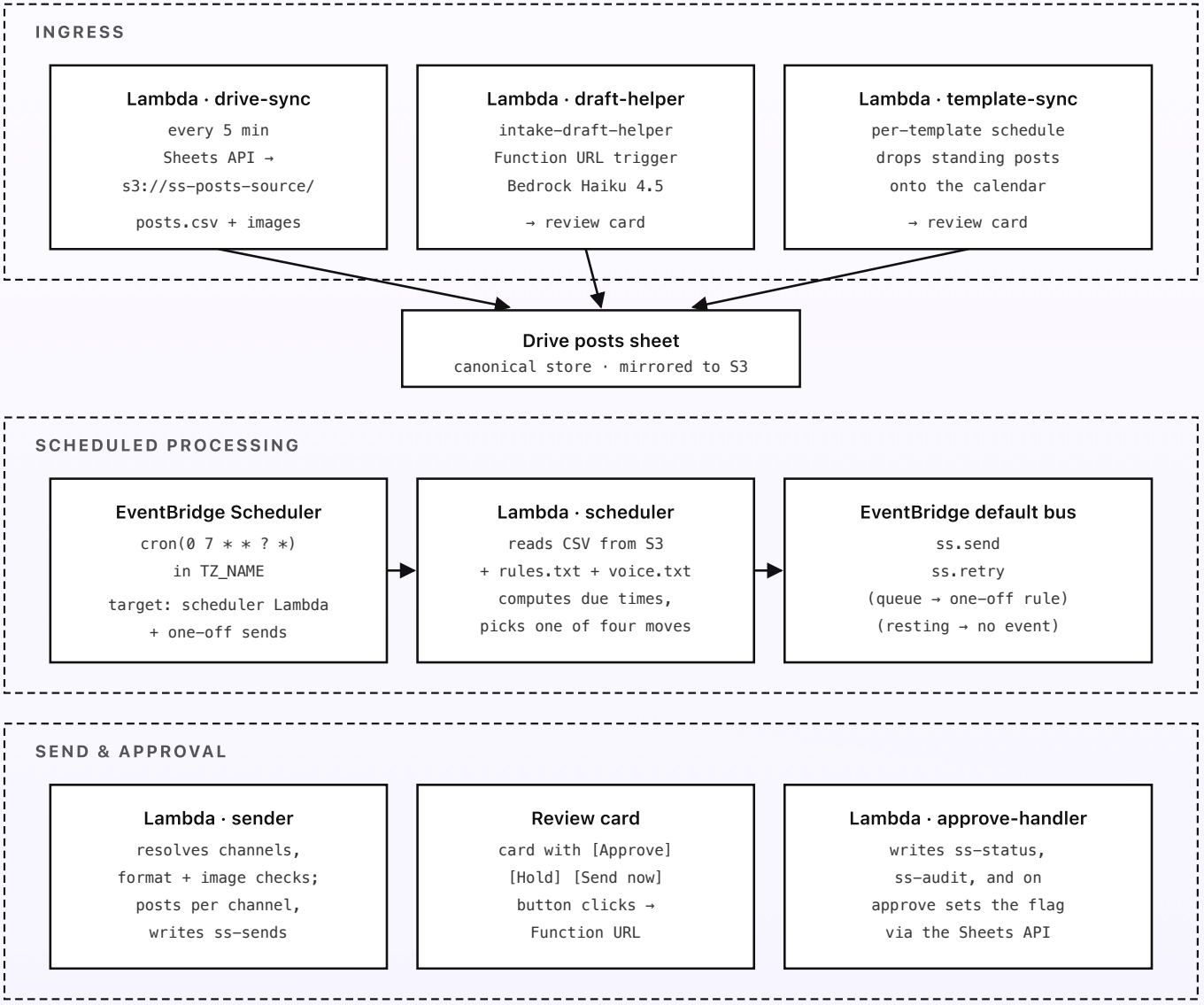
Engineering reference: the social scheduler architecture

Same system, drawn for engineers. Region, service names, resource identifiers, Bedrock model IDs, Lambda inventory, IAM scopes, the channel-token setup, EventBridge Scheduler config, the DynamoDB schemas, and the approval flow. Read alongside the previous six posts; this one's the build sheet.

Region and account shape

Default region: **ap-southeast-1** (Singapore). Bedrock cross-Region inference, EventBridge Scheduler, and SES outbound are all in good shape there. A second region for multi-region resilience isn't worth the extra setup work at SMB volume — the failure mode for an SMB is one post going out a few minutes late, not a regional outage. One AWS account dedicated to the scheduler (separate from your other workloads) keeps the IAM blast radius small and lets a single AWS Budgets alarm cover the whole system.

Topology



Nothing posts without approval — and every interaction is logged to ss-audit.

Fig 7. AWS topology, in three regions of the diagram: ingress (three lanes into the sheet), scheduled processing (the daily scheduler tick booking sends and emitting events), send and approval (the post ships and the owner's response is recorded). Every Lambda is event- or schedule-driven; nothing is synchronous-chained.

Lambda functions

All Lambdas use the `arm64` architecture, the smallest memory size that meets latency targets (typically 256 MB), Python 3.14 runtime, and CloudWatch Logs at 7-day retention. Each function has its own least-privilege IAM role. None run inside a VPC.

- **`drive-sync`** — EventBridge Scheduler target, fires every 5 minutes. Uses the Google Drive API + Sheets API (service-account credentials in Secrets Manager under `ss/drive/sa`) to export the posts sheet as CSV and mirror new images from the Drive folder to `s3://ss-posts-source/` only if the sheet or folder has changed since the last sync. Same pattern syncs the rules and voice docs to `s3://ss-rules-source/`. Memory: 256 MB. Timeout: 30 s.
- **`intake-draft-helper`** — Lambda Function URL. Triggered when the owner submits a short note from the draft-helper. Reads the brand tone from `s3://ss-rules-source/voice.txt` and calls Bedrock Haiku 4.5 (`anthropic.claude-haiku-4-5-20251001-v1:0` via `global.anthropic.claude-haiku-4-5-20251001-v1:0`) to propose post text per chosen channel, with the per-channel character limit injected into the prompt so the draft fits on the first try. Posts the proposal to the review card with Approve/Edit/Discard. On approve, writes the row to the Drive sheet via the Sheets API with the approved flag unset. Memory: 512 MB. Timeout: 30 s.

- **`template-sync`** — EventBridge Scheduler target, one schedule per recurring template (e.g. `cron(0 9 ? * FRI *)` for a weekly Friday post). Reads the matching template from the rules doc, drops a fresh row onto the calendar a few days ahead, and surfaces it in the same review card as the draft-helper. Memory: 256 MB. Timeout: 30 s.
- **`scheduler`** — EventBridge Scheduler target, daily at 7am local time (the schedule expression runs in `TZ_NAME` set to the SMB's timezone, e.g. `Asia/Singapore`). Reads `s3://ss-posts-source/posts.csv` and the rules and voice docs. For each row, computes `minutes_to_send`, reads state from `ss-status` and `ss-sends`, and decides on a move. For queued posts it creates a one-off Scheduler rule for the exact send minute; for due or retry posts it emits `ss.send` or `ss.retry` with the post context as the event payload. Resting posts emit nothing. Memory: 512 MB. Timeout: 60 s. *No Bedrock calls.*
- **`sender`** — EventBridge rule on the `ss.send` and `ss.retry` events. Resolves channels, runs the format check (character limit, link count) and the image check (size, type) against the per-channel rules, fetches the image from `s3://ss-posts-source/`, refreshes the channel token from Secrets Manager if it's near expiry, then posts to each channel through its posting endpoint. On a per-channel failure it records the error and lets the scheduler queue a retry; on a format or image failure it flags the post back to the owner and drops it to draft. Writes a row to `ss-sends` per channel after each attempt. Memory: 512 MB. Timeout: 60 s.
- **`approve-handler`** — Lambda Function URL, public with `AuthType: NONE`; verifies a signed token on the request body. Triggered by review-card button clicks (Approve/Hold/Send-now). Writes to `ss-status` and `ss-audit`; on

approve, sets the approved flag on the Drive sheet via the Sheets API and books the one-off send; on hold, cancels any queued one-off Scheduler rule and drops the status to draft; on send-now, emits an immediate `ss.send`. Memory: 256 MB. Timeout: 15 s.

- **`digest`** — EventBridge Scheduler target, weekly Sunday 6pm. Reads `ss-sends` for the past week and the calendar; sends a digest email summarizing what posted and what's coming up. No Bedrock; the message is a plain summary table. Memory: 256 MB.
- **`recap`** — EventBridge Scheduler target, monthly on the first Monday at 9am. Reads the past month's `ss-sends` and `ss-audit`; calls Bedrock Haiku 4.5 to write a one-paragraph recap narrative; emails it via SES to the configured stakeholder list. Memory: 512 MB.

Storage

- **DynamoDB · `ss-status`** — one row per post. PK `post_id`; attributes: `state` (draft/queued/sending/sent/failed), `queued_for`, `approved_by`, `scheduled_at`. On-demand. No TTL.
- **DynamoDB · `ss-sends`** — one row per send attempt. PK `(post_id, channel)`; sort key `attempt`; attributes: `result` (ok/fail), `posted_url`, `error`, `sent_at`. On-demand.
- **DynamoDB · `ss-audit`** — one row per write action of any kind. PK `(post_id, ts)`; attributes: `action` (approve/hold/send-now/draft-helper), `by_user`, `before`, `after`. On-demand. No TTL — this is the long-term audit trail.

- **S3 · `ss-posts-source`** — mirrored CSV from the Drive posts sheet plus the images from the Drive folder. Versioning enabled. Lifecycle to Glacier at 90 days for old images; CSV kept hot.
- **S3 · `ss-rules-source`** — mirrored rules and voice docs as plain text. Versioning enabled.
- **S3 · `ss-render-cache`** — resized image variants per channel, cached so a re-send doesn't re-process the same image. Lifecycle expiry at 30 days.

Bedrock

- **Foundation model.** `anthropic.claude-haiku-4-5-20251001-v1:0` via the Global cross-Region inference profile `global.anthropic.claude-haiku-4-5-20251001-v1:0`. Two callsites: `intake-draft-helper` for roughing out post text, and `recap` for the monthly narrative. A heavier model (`anthropic.claude-sonnet-4-6`) is wired but unused by default; it's only worth switching the draft-helper to Sonnet if a business wants longer-form, multi-paragraph posts where the extra reasoning earns its cost.
- **Embeddings.** Not used. The posts are structured rows; deterministic lookup and date arithmetic beat vector retrieval here. No Knowledge Base, no S3 Vectors.
- **Quotas.** Default account quotas are more than enough at SMB volume. The scheduler itself doesn't call Bedrock; the draft-helper fires a few times a month at most.

EventBridge Scheduler config

- **`ss-daily-tick`** — `cron(0 7 * * ? *)` in the SMB's timezone. Target: `scheduler` Lambda.
- **`ss-drive-sync`** — `rate(5 minutes)`. Target: `drive-sync` Lambda.
- **`ss-template-*`** — one rule per recurring template, with the template's own cron. Target: `template-sync` Lambda.
- **`ss-weekly-digest`** — `cron(0 18 ? * SUN *)` in TZ. Target: `digest` Lambda.
- **`ss-monthly-recap`** — `cron(0 9 ? * 2#1 *)` (first Monday at 9am) in TZ. Target: `recap` Lambda.
- **One-off send rules** — created on the fly by `scheduler` and `approve-handler` when a post is queued for its exact minute. Use `at(YYYY-MM-DDTHH:MM:SS)` expressions with `--action-after-completion DELETE` so the rule self-cleans. Hold cancels the matching rule by name.

Channels and tokens

- Each channel is registered in the rules doc with its name, posting endpoint, and the per-channel format rules (character limit, accepted image types and sizes, link count).
- Each channel's long-lived token and refresh credential live in Secrets Manager under `ss/channels/<channel-name>`. The `sender` reads the token at send time and refreshes it via the channel's token endpoint if it's within the refresh window.
- SES outbound for the review cards, the weekly digest, and the monthly recap: verify a sender identity at `scheduler@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:

- **scheduler role:** `s3:GetObject` on the posts, rules, and voice keys; `dynamodb:Query` + `GetItem` on `ss-status`, `ss-sends`; `events:PutEvents` on the default bus; `scheduler:CreateSchedule` for the one-off sends. *No* `bedrock:*`.
- **sender role:** `s3:GetObject` on the image keys and `s3:PutObject` on `ss-render-cache`; `secretsmanager:GetSecretValue` on the channel-token secrets; `dynamodb:PutItem` on `ss-sends`; outbound network access to each channel's posting host.
- **approve-handler role:** `dynamodb:PutItem` on `ss-status` and `ss-audit`; `secretsmanager:GetSecretValue` on the Sheets-API service-account secret; outbound network access to `sheets.googleapis.com`; `scheduler:CreateSchedule` + `DeleteSchedule` for booking and cancelling one-off sends.
- **intake-draft-helper role:** `s3:GetObject` on `ss-rules-source`; `bedrock:InvokeModel` on the Haiku ARN; `secretsmanager:GetSecretValue` on the Sheets-API secret.
- **drive-sync and template-sync roles:** `secretsmanager:GetSecretValue` on the Google service-account secret; `s3:PutObject` on the posts and rules buckets; outbound network to `www.googleapis.com`.

Approval flow

The review card is delivered as an HTML email via SES (and, if a business uses Slack, the same card can be posted with Block Kit blocks). Each button carries a signed token in its link that encodes the `post_id` and the action (`approve`, `hold`, `send_now`). Clicks land on the `approve-handler` Function URL, which verifies the signature, applies the action, and returns a small confirmation page. Approve and Hold are one-click; Send-now is one-click but still routes the post through the full format check in `sender` before anything posts. The approval flag on the Drive sheet is the single gate every post passes — a post with the flag unset is never sent, no matter what state the rest of the system is in.

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:** scheduler Lambda failures > 0 in a day (the daily tick is the one piece that has to run); sender failure rate > 1% in 24h; approve-handler signature-verification failures > 5/hour (might mean the signing 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 `ss-cost-alarm` subscribed to the on-call admin's email.

Config and secrets

Service-account credentials for Drive and Sheets APIs live in Secrets Manager under `ss/drive/sa` (one service account with scopes for both APIs). Channel

tokens and refresh credentials live under `ss/channels/*`. The approval-link signing secret is `ss/approval/signing-secret`. SES sender identity lives in IAM and the verified-domain config. The configured timezone, posting-window hours, blackout days, and account-default channel list all live in Parameter Store under `/ss/config/`. Lambdas fetch config on cold start and cache for the lifetime of the execution environment.

Deploy

GitHub Actions with OIDC and AWS SAM — no long-lived keys. The opinionated bits: turn on S3 versioning for both `ss-posts-source` and `ss-rules-source` so a bad Drive edit can be rolled back in one click; version the EventBridge Scheduler timezone setting so you don't accidentally start running the daily tick in UTC after a CI rotation; and keep the channel-token secrets in a separate stack so a token rotation doesn't require redeploying the whole system. Total deployable surface: around eight Lambdas, three DDB tables, three S3 buckets, one EventBridge rule on the default bus (plus the Scheduler rules), 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.