SPRINGBIG

STATE OF THE PLATFORM · AUGUST 2026

Twenty surfaces, measured against live production

The current state of the springbig platform, surface by surface, as of 16 August 2026. Every figure is a direct measurement against live production — 34.9 million web requests, 19 million background jobs, 369 deployed endpoints, 1,799 edge zones and the databases behind them. Where a trend matters, an earlier reading from 4 August is shown alongside.

Compiled 2026-08-16 springbig production · us-east-1 For engineering leadership

CONTENTS

PART I

Executive summary

The platform is stable and getting busier. What follows is not a list of outages — it is a set of places where the system is working exactly as configured, and the configuration is not what anyone intended.

Web requests	Production 5xx	Public API documented	Unpaid, still sending
34.9M	**0.10%**	**24.7%**	**$897k**
7 days · apdex 0.98	35,538 of 34.9M	23 of 93 endpoints	79 orgs · 26.8M messages/90d

Where it is strong

WHAT WORKS	EVIDENCE (MEASURED 16 AUGUST)
Server-side stability	35,538 server errors across 34.9M requests — 0.102%, with no incident in the window
Latency	p50 30ms, p95 422ms, apdex 0.98 across 34.9M transactions
Job fleet	19,071,497 executions over 3 days at 99.955% success
Ingestion	~341,000 visits/day across 36 vendors; every vendor delivering on the last full day
Polling hygiene	Wasted polling against dead accounts is now **zero** — a concern at the prior measurement, resolved
Growth without incident	Throughput +7% and compute +30% between readings with no stability change

Where it is not

Two revenue-assurance findings that no error dashboard can see. 31 of 233 merchants labelled messaging-only are running loyalty mechanics — Fire and Flower awarded 66.6M points in 90 days — because `platform_type` gates four call sites in the entire application, none of them on the API or the loyalty path. Separately, **79 billing organisations** sent **26.8M messages** in 90 days against **$897k** of collectible unpaid invoices — enterprise accounts billed off platform are excluded from that figure.

Observability remains configured to hide its own failures. New Relic records **zero error events across 345,845 HTTP 500s**. Rollbar discards `RecordNotFound` outright and files `ActionView::Template::Error` as a warning on the strength of a comment that is no longer true. The wallet project — a live consumer surface serving 1.98M requests a week — reports nothing at all.

The highest-volume failure is still unattributable, but the class around it is not. The transaction-date defect held at 23,112–27,550/day, unchanged across both measurements and still logged without merchant or POS identity. But duplicate-member errors — now the platform's

largest error class at 24,968/day — *are* attributable today via `enrollment_change_source`, which had been assumed impossible.

The public API contract is narrower than a raw endpoint count suggests, and sharper within its real scope. The documentation was enumerated completely. **pos** and **general** are the two gateways public by design; across their **93 endpoints only 23 are documented — 24.7%**. **35 undocumented endpoints mutate data** — twelve delete records or move member balances, including `DELETE /members/{}` and `PUT /members/{}/reset_balance`. Every one of the 24 API reference pages declares a **non-production host**, with a live "Try It" explorer pointed at it.

One partner is generating 350,000 rejected requests a day, and is now named. A single API key — `lmlp-leaflogix`, issued 2022 and never rotated — accounts for **100% of missing-header rejections** on the busiest POS endpoint, 29.3% of that partner's own calls to it. Seven other partners transact on the same endpoint with zero such failures. The traffic is rejected at the gateway, so it never reached an application metric and nobody reported it in over thirty days.

Two safety nets are not catching anything. A dead-letter queue holds **722 undelivered POS customer-sync messages** that nothing has ever consumed, its oldest message pinned at the four-day retention cap — so failures are being destroyed on a rolling basis. All 87 queues, and all 38 dead-letter queues, sit below the fourteen-day maximum. And there is **not a single CloudWatch alarm on any SQS queue**, which is the complete explanation for why nobody knew. Separately, 31 alarms have received no data at all, 25 of them for between 165 and 1,274 days.

Webhooks are noisier in reputation than in fact, and blinder than expected. Firing is guarded at three levels and volume is proportionate to business activity — the apparent excess is POS location fanout, and 73% of the delivery lambda's traffic turned out to be *inbound* callbacks rather than outbound webhooks. But **0 of 816 endpoints can receive a change-set**: the dirty state is discarded before the payload is built, so every receiver must blind-write the full record. That is the mechanism behind the known full-object clobber.

One recurring defect produced two fresh instances during the gap. Two audiences fail daily on `date_trunc('1', …)` — the exact misconfiguration diagnosed on 2026-08-08. The shape is still writable, so it recurs roughly every three to six days.

> **What two measurements establish that one cannot.** A single pass cannot separate a real defect from a transient. Re-measuring twelve days later can. Nothing measured in August turned out to be noise; one judgement expired because the fact underneath it changed; and three figures were overturned by better method. That last category is the most useful part of this document, and it is collected in Part III rather than buried.

What to do first

#	ACTION	WHY NOW
1	Raise the missing-header defect with Dutchie (key `lmlp-leaflogix`)	~350,000 rejections/day, 29.3% of their calls to that endpoint, 100% of all such failures. Named partner, evidence captured, no engineering work needed on our side.
2	Enforce `ConditionValueValidator` on the audience write path	Third occurrence of one defect; two new instances between readings
3	Turn on New Relic error collection; re-scope the two Rollbar filters	Zero error events across 345,845 HTTP 500s — no alert can fire from either source today
4	Reconcile the 31 messaging-only accounts running loyalty, then wire `require_platform!` into the API	Reconcile first — enforcing today changes live behaviour for merchants with real programmes
5	Work the 79 billing orgs with collectible open invoices	$896,551 the platform should have collected; enterprise off-platform contracts already excluded
6	Add `merchant_id` and `pos_type` to the transaction-date rescue	The one error class that genuinely cannot be attributed; ~25,000/day
7	Merge `observability-quick-fixes`	Already written; restores per-worker metrics and removes ~92% of log spend
8	Add SQS alarms and raise DLQ retention to 14 days	Zero SQS alarms exist; 722 messages expiring unreviewed. Configuration, not code.
9	Document or gate the 35 undocumented mutating endpoints	Twelve delete records or move balances. Publishing them without first adding per-method authorisation would remove the only thing currently limiting access.
10	Add a `changes` key to webhook payloads	0 of 816 endpoints can tell what changed — the root of the full-object clobber
11	Upload frontend source maps	Zero of the top 15 frontend errors are readable, and the backlog cannot be recovered retroactively
12	Resolve or disable cova location 26112	Billing-active, still polled, no data since 2026-05-20

01

The web tier

The strongest surface measured, and the one whose cost is most concentrated.

Requests (7d)	p50 / p95	Apdex	5xx
34.9M	**30 / 422**	**0.98**	**0.102%**
production app only	milliseconds	39,706 frustrated of 34.9M	35,538 requests

STATUS	REQUESTS (7D)	SHARE
200	28,150,388	80.548%
404	2,968,775	8.495%
401	2,046,372	5.855%
304	730,519	2.090%
422	628,587	1.799%
302	302,380	0.865%
400	40,920	0.117%
500	**35,538**	**0.102%**
403	13,876	0.040%

The 4xx are structural, not incidental

404 and 401 together are **14.35%** of all traffic, and neither is a defect. The 404s are overwhelmingly the POS **lookup-then-create** protocol: a partner asks whether a member exists, is told no, and creates one. The 401s are dominated by two endpoints known to be broken since the August endpoint census — `api/pos/v1/batches/create` (829,971 in 7 days) and `api/wallet/auth/v1/remember_me/authenticate` (320,896) — plus routine unauthenticated probing.

> **Never count POS 404s as errors.** They are the documented lookup-then-create handshake. A platform-wide error rate that includes them is inflated by roughly three million requests a week and will point investigations at healthy integrations.

Where the compute actually goes

ENDPOINT	REQUESTS	AVG	CPU HRS / 7D
`api/pos/v1/members/index`	4,116,835	295.8ms	**338.3**
`api/pos/v1/members/show`	7,487,489	35.3ms	73.4
`api/pos/v1/visits/create`	1,646,093	97.5ms	44.6
`api/native_app/v1/stashboard/show`	3,057,228	51.3ms	43.6
`api/wallet/v1/members/show`	661,965	214.7ms	39.5
`api/pos/v1/rewards/index`	3,069,181	46.1ms	39.3
`api/pos/v1/offers/index`	2,530,461	44.4ms	31.2
`api/pos/v1/members/update`	851,183	72.0ms	17.0

One endpoint is 59% of the top-ten compute. `api/pos/v1/members/index` is 11.8% of requests but **338 hours of compute a week** at 296ms average — an order of magnitude worse per call than its neighbours. It is the single largest optimisation target in the estate, and nothing about it is failing, so no alert will ever raise it.

02

The API contract

The deployed gateway and the Rails application still disagree, and the gap is wide, and one long-standing assumption about it turns out to be false.

Deployed endpoints	Rails routes	Proxying to Rails	404 into Rails
369	**3,497**	**305**	**29**
excluding OPTIONS	3,476 distinct verb+path	of 369 deployed	6 at the prior reading

Of the 369 deployed endpoints, **305 proxy to Rails** and 64 resolve to Lambda, SQS or mock integrations. Of those 305, **276 match a Rails route and 29 do not** — they are deployed, reachable, and land on nothing.

GATEWAY	DEAD ROUTES
brands	10
native_app	8
wallet	4
general	3
pos	3
callbacks	1

The growth from 6 to 29 is partly method. This pass compares the gateway's backend path *and* HTTP verb against the output of `bundle exec rails routes`, and two parser bugs were corrected during the count before it settled at 29. A sample was verified by hand — including `/api/pos/v1/reward_redemptions`, `/api/wallet/auth/v1/remember_me` and `/api/brands/v1/ads_configuration` — and each is genuinely absent from Rails or exists only for a different verb. The direction is real; the exact delta between readings should not be read as 23 new breakages.

A note on `POST /api/pos/v1/members`, sometimes cited as routing to a missing action. It does not. The action exists and is fully implemented at `members_controller.rb:31` — roughly thirty lines that set `pos_type` and `merchant_id`, apply double-opt-in configuration, call `MemberProfile.find_or_create_and_retry` and hydrate milestones. Either that reading was mistaken or this was fixed in the interval. Worth noting separately: the deployed `POST /v1/members` never reaches this action anyway, because the gateway routes it to Lambda.

Destructive endpoints are still deployed and undocumented

`DELETE /members/{}` and `PUT /members/{}/reset_balance` both remain live on the pos gateway as proxies to Rails, and both have matching Rails routes. Both require an API key. That key is authentication, not authorisation: it establishes that the caller is a valid partner and then grants the whole stage. All 93 endpoints on the two public gateways carry `authType = NONE` — no Lambda authoriser, no IAM, no per-method scope — and none of the six usage plans throttles per method. **A destructive endpoint is configured identically to a read endpoint; the only thing separating them is knowing the path.**

> **The third layer is measured separately.** The published documentation was enumerated completely and is reported in section 02b. Two structural claims about the Rails side were not re-checked here — that 7 routes point at missing controller classes and 28 controllers are unrouted — and remain at their 4 August values.

02B

The published documentation

The published documentation was enumerated completely — 48 pages, all HTTP 200, via the readme.io site manifest and its embedded OpenAPI fragments. Every claim below comes from the published source text. The headline earlier estimate is overturned — not because it was too harsh, but because it was too generous.

Documented operations	External gateways	Reference pages on gamma	Public surface covered
27	**2**	**24/24**	**24.7%**
of 93 externally exposed	pos + general, by design	zero point at production	23 of 93 pos+general

The docs were enumerated completely rather than sampled: readme.io publishes a machine-readable site manifest at `llms.txt`, and each reference page embeds a full OpenAPI 3.1 fragment. All 48 pages returned HTTP 200 — no login wall, no rate limiting. One page (`docs/template-api`) is absent from the manifest and reachable only by following an inbound link.

The host claim, refuted in the wrong direction

DECLARED HOST	REFERENCES	PAGES
`gamma.api.springbig.technology`	26	24
`production.api.springbig.technology`	10	8 — all integration guides, no reference pages
`{environment}.api.springbig.technology`	7	4
`api.springbig.technology` (deprecated form)	1	1

An earlier estimate put this at two thirds of the docs. It is 100%. Every one of the 24 reference pages that declares a server declares `gamma`. Not one declares production. The only production hostnames in the entire documentation set appear in third-party integration guides — Shopify, Klaviyo and similar — never in the API reference itself. And because `x-readme.proxy-enabled` is set, the "Try It" explorer is live: every exploratory call a partner makes from the documentation hits a non-production environment. The Getting Started page does explain the `{environment}` substitution, so this is navigable — but the default is wrong everywhere it matters.

Scope: two gateways are public by design

pos and **general** are the externally documented gateways. The other ten are consumed by internal services — native app, wallet, brands, callbacks, tablet, join, reports, netsuite, analytics and shortlinks — and are either active internal surfaces or superseded. They are not expected to carry public documentation, and their absence from readme.io is correct, not a gap.

This scoping matters for how the numbers read. Counting all 369 deployed endpoints against 27 documented operations produces an alarming ratio that measures the wrong thing. The meaningful denominator is the **93 endpoints on the two public gateways**. The finding below is therefore narrower than a raw count suggests — and, within that scope, sharper.

The exact diff, for the two gateways that are documented

Matching was mechanical — lowercase, strip the version prefix, treat `{param}` names as wildcards, exclude OPTIONS. Zero of the 23 matches required judgement.

MEASURE	COUNT
Documented operations (pos + general)	27
Deployed endpoints (pos + general, excl. OPTIONS)	93
Matched	23
Documented but not deployed	4 — all typos with working counterparts
Deployed but undocumented	**70** (pos 32, general 38)
Documentation coverage	**24.7%**

> **70 of the 93 endpoints on the two public gateways are undocumented.** That is within the public scope alone — the ten internal gateways are excluded, since they are not expected to carry public documentation. Every quantity published on this surface previously was too small, though it was right that a problem existed and right about which two endpoints mattered most.

The four documented-but-not-deployed cases are all near-miss typos rather than missing functionality: `new-ten_dlc_submissions` (editing artifact), `/ten_dlc_submissions/status` (guide omits `{submission_id}`), `/attachment` versus the deployed `/attachments`, and the malformed `/{pos_user}/rewards`. Each has a working deployed counterpart at a slightly different path, so the functional documentation-only gap is **zero**.

Half the undocumented endpoints mutate data

35 of the 70 undocumented endpoints are POST, PUT, PATCH or DELETE. Twelve of those touch member balances or delete records outright:

METHOD	PATH	RISK
DELETE	`/v1/members/{pos_user}`	**Destructive** — member deletion
DELETE	`/v1/external_group_segments/{id}`	**Destructive** — segment deletion
PUT	`/v1/members/{pos_user}/reset_balance`	Balance-affecting
PUT	`/v1/members/{pos_user}`	Balance-affecting via full-object clobber
POST	`/v1/members/{}/cash_back/redeem`	Balance-affecting
POST	`/v1/members/{}/cash_back/{}/refund`	Financial reversal
POST	`/v1/members/{}/rewards/{}/redeem_points`	Point deduction
PATCH	`/v1/members/{}/reward_redemptions/{}`	Mutates a settled redemption
PATCH	`/v1/visits/{pos_id}`	Mutates a settled transaction
POST	`/v1/members/{}/send_message`	Sends real SMS — cost and compliance bearing

Two gateways expose parallel undocumented member-mutation surfaces — `pos` keyed on `{pos_user}` and `general` on `{member_id}`. Both can redeem, update and mutate the same members by different identifiers, and the documentation mentions neither. Thirty-five of the seventy mutate data.

These endpoints are not dormant — they are production traffic

Measured over 30 days from the POS gateway's own access log (`/aws/apigateway/pos-production`), by `resourcePath`:

ENDPOINT	CALLS (30D)	DOCUMENTED?
`POST /v1/visits`	7,566,312	guide only
`PUT /v1/members/{pos_user}`	3,595,269	**no**
`POST /v1/members`	1,649,282	yes
`POST /v1/members/{}/rewards/{}/redeem`	484,196	**no**
`PATCH /v1/members/{}/reward_redemptions/{}`	76,039	**no**
`POST /v1/members/{}/offers/{}/redeem`	61,584	**no**
`POST /v1/members/{}/rewards/{}/redeem_points`	3,046	**no**
`PATCH /v1/visits/{pos_id}`	7	**no**
`DELETE /v1/members/{pos_user}`	**1**	**no**

A member deletion was executed through the undocumented endpoint. On 31 July at 23:24:44 UTC, `DELETE /v1/members/{pos_user}` returned **200** with `integrationStatus 200` and 170ms of integration latency — Rails accepted and performed it. The caller was `34.96.44.59`, a Google Cloud address. This is the single strongest argument in the report against relying on obscurity: the endpoint has no per-method authorisation, no documentation, and one confirmed successful use.

The redemption endpoints are not edge cases either. Over the same window `POST .../rewards/{reward_id}/redeem` succeeded 482,691 times against 1,234 validation failures and 240 rejections — **core loyalty behaviour running entirely on undocumented paths**.

Reading these paths correctly requires the log group, not just the path. API Gateway's `resourcePath` field is relative to the gateway that logged it, with no prefix — so the same string means different endpoints in different logs:

LOG GROUP	RESOURCEPATH	ACTUAL PATH
`/aws/apigateway/pos-production`	`/v1/members/{pos_user}`	`/api/pos/v1/members/{pos_user}`
general gateway	`/v1/members/{member_id}`	`/api/general/v1/members/{member_id}`
— not gateway-exposed —	—	`/api/v1/members` (internal, `routes.rb:64`)

Every figure in this section comes from the POS gateway's own log, so all of it is the `api/pos/v1` namespace. The parameter name is a reliable tell: `{pos_user}` appears only on the POS gateway, `{member_id}` only on general. A bare `/v1/members` reads as though `/v1/` were the root, which it is not — and the internal `api/v1/members` namespace, which is not exposed through either public gateway, makes that ambiguity consequential.

The destructive endpoints are confirmed undocumented

All 48 pages were searched for `reset_balance`, `DELETE`, `redeem`, `deduct` and `adjust`:

`DELETE /members/{}` and `PUT /members/{}/reset_balance` appear nowhere in the documentation. Both are deployed and both proxy to Rails, behind an API key that grants the entire stage. These are the two most consequential omissions in the set. The only trace of deletion anywhere in the docs is the `member_deleted` webhook event in the Member Webhooks guide — which tells partners the event exists while never documenting the endpoint that fires it.

Which makes the documentation gap load-bearing in an unexpected way. Because authorisation is stage-wide, the practical control on these endpoints is that partners do not know they exist. That is security through obscurity, and it has two consequences worth stating plainly. First, it is not a control — a partner who guesses a path, reads a stack trace, or works from an old integration example has the same access as one who is entitled to it. Second, it inverts the obvious remedy: **publishing this set without first adding per-method authorisation would remove the only thing currently limiting it**. The order has to be authorise, then document.

Three POS write endpoints, three different documentation states

ENDPOINT	DOCUMENTED?	PROBLEM
`POST /pos/v1/members`	yes	Response body accurate; declares 2 status codes, returns 6, including 20.5% HTTP 500
`PUT /pos/v1/members/{pos_user}`	**no**	Entirely undocumented
`POST /pos/v1/visits`	partial	Appears only in the Shopify Flow guide — no reference page, no schema

The documented `POST /pos/v1/members` declares only 200 and 422 responses, and its 200 example returns a fully-populated member. That example is **accurate**: measured over 7 days, successful responses average 1,878 bytes (minimum 1,316) and integration latency is 96% of total response latency, so the Lambda returns synchronously with a real body. The read-after-write expectation is safe. What is *not* accurate is the response-code list — see below.

The undocumented PUT is the most consequential of the three. The update path overwrites the full member object — the behaviour behind the known Cova clobber — and partners have no published guidance that this is what a PUT does. An integration that sends a partial object will silently erase the fields it omitted.

Documented behaviour versus observed behaviour

The specification was compared against 30 days of POS gateway access logs — 98,961,673 requests, full-window aggregation, no sampling.

ENDPOINT	DECLARED	OBSERVED
`POST /v1/members`	200, 422	200, **500, 401, 400, 403**, 422, 404
`GET /v1/members`	200, 404	200, 404, **401, 400, 500, 403**
`GET /v1/rewards`	**200 only**	200, **401, 500, 422, 403, 400**
`GET /v1/members/{}/offers`	200, 404	200, 404, **401, 500**

Not one documented operation declares 401, 403 or 429 — yet 401 alone accounts for 3,513,324 responses in 30 days. A partner coding strictly to the published schema has no defined handling for authentication failure. Worse, `POST /v1/members` returns **HTTP 500 on 338,456 of 1,649,296 calls (20.5%)** with a **35-byte body** — undocumented, and too small to diagnose. One in five member creations fails in a way the specification says cannot happen.

A 400 that is 10.6% of all POS traffic

MEASURE	VALUE
Endpoint	`GET /v1/members/{pos_user}`
Error	`Missing required request parameters: [AUTH-TOKEN]`
Volume (30 days)	10,494,028
Daily rate	~350,000 (range 301,259–435,354)
Share of that endpoint	24.4%
Share of all POS traffic	10.6%
Distinct source IPs	417

These are rejected at the gateway before reaching Rails, so they consume no application capacity — but they are billed, logged, and invisible to every application-side metric.

It is one caller with a broken code path, not a broken integration — and it is not us. Three measurements narrow this considerably. The 417 IPs are evenly distributed at roughly 8,500 requests each across AWS us-east-1 ranges, which is one horizontally-scaled fleet rather than many partners. The same IPs **succeed** on the same endpoint — one sampled IP shows 14,856 responses of 200 alongside 8,564 of 400 — so credentials and routing are fine and only one call path omits the header. And springbig's own VPC lambdas egress through a single NAT gateway (`54.243.149.25`) which appears **zero** times in seven days of this log, ruling out self-calls.

The caller is not directly identifiable, but the traffic can be fingerprinted. Because the rejection happens at parameter validation, the API key is never evaluated — `caller` and `user` are – on all 10.49M records, and four correlation routes dead-end: the execution log records only the validation failure; Rails logs the load-balancer address rather than the client; the gateway `requestId` is not propagated into Rails; and the IPs appear in `user_sessions` only as tablet records ending January 2022.

The caller is identified: `lmlp-leaflogix`. Access logs cannot attribute these requests, because rejection happens at parameter validation and the API key is never surfaced there. Enabling execution-level data tracing on this single method — briefly overnight, then for 18 minutes during business hours — resolved it. Across the business-hours capture, **696 of 696 attributable failures (100%) carried key `wd6n3j71og`, `lmlp-leaflogix`**, on `pos-partner-plan`, issued April 2022 and never rotated. An overnight capture returned the same result at smaller scale (194 of 194).

A complete failing request, verbatim from the execution log:

```
API Key ID: wd6n3j71og
HTTP Method: GET, Resource Path: /pos/v1/members/38685148
Method request headers: {x-datadog-sampling-priority=1, CloudFront-Viewer-C
  x-datadog-origin=rum, CloudFront-Forwarded-Proto=https, User-Agent=Amazon
Request parameter validation failed. Missing parameters: [AUTH-TOKEN]
Method completed with status: 400
Usage Plan check succeeded for API Key ********************************mm
```

The same key passes the usage-plan check on the same request. The credential is valid and present as an API key; what is absent is the `AUTH-TOKEN` header. That rules out an expired or revoked key, a wrong environment, and a misconfigured plan — the caller is authenticated at one layer and omits the header at another, on a subset of its calls.

Two alternative explanations were tested and eliminated. All 417 source addresses resolve to Amazon (`AMAZO-4`, `AT-88-Z`), so a vendor hosted elsewhere is not the origin. And although **1,048 of ~1,300** captured requests arrive via CloudFront — raising the possibility of a CDN stripping the header in transit — **300 of 300** captured header blocks that came through CloudFront carried the `AUTH-TOKEN` intact. CloudFront forwards it reliably; the omission originates upstream, at the caller.

The `lmlp-` prefix is a migration artifact, not an owner. 137 of the 217 API keys carry it, all created on 2026-04-08 and described "imported from lmlp account" — a bulk import, not an indication of who operates the key today. LMLP's own egress addresses (`34.204.210.35`, `54.175.157.22`) appear **zero** times in seven days of POS gateway traffic, and its four CloudFront distributions all serve S3 static origins rather than the API. The calls are not coming from that environment.

The failing key is the oldest of five that Dutchie holds, and all five are enabled.

KEY	CREATED	PLAN
`lmlp-leaflogix` — the failing one	2022-04-08	pos-partner-plan
`lmlp-dutchie`	2022-04-08	pos-partner-plan
`new-dutchie-pos`	2023-01-11	pos-partner-plan
`dutchie-ecom`	2025-03-13	pos-partner-plan
`dutchiepos-sms`	2025-04-18	pos-partner-message-plan

That a 2022-era key is still transacting alongside a key explicitly named `new-dutchie-pos` is itself worth raising: the failures may be an older integration path that was superseded but never retired. It also means the number of distinct credentials a single partner holds is not being managed — five enabled keys, none rotated since creation.

What this is worth to the vendor conversation. A named key, a complete request trace showing the header absent while the key validates, a 30-day flat rate of ~350,000 rejections a day, and a per-host failure ratio swinging 0%→80% as instances cycle. The partner cannot see this from their side — their requests are sent and their key works — which is why it has run for at least a month unreported.

No other partner fails at all. During the business-hours window, eight API keys transacted successfully on this endpoint — leaflogix 66.2%, cova 16.6%, waio-pos 7.4%, flowhub-pos 5.6%, digital_awesome 2.5%, plus blaze, greenline and alleaves. **Every one of the 696 missing-header rejections belonged to a single key.** Measured across leaflogix's own 3,132 requests in the window: **48.4% succeeded, 29.3% were rejected for the missing header**, 13.2% returned 404 and 9.1% returned 401. That 29.3% matches the ~30% per-host rate seen in the access logs, at full business-hours volume.

One hypothesis tested and not supported. Because Dutchie holds an older key alongside a newer one, the obvious explanation was a legacy integration path still running against the 2022 credential. The captured headers do not support it: **every request carries the same User-Agent** (`Amazon CloudFront`), and the 400s run at 29.1% within that single client signature. There is no second client fingerprint — same key, same client, same path, roughly three calls in ten arriving without the header.

Sampling and cleanup. Two captures were taken: ~25 minutes overnight and 18 minutes during business hours, the latter while the failure rate was climbing out of its overnight trough (67/hour at 09:00 UTC to 1,962/hour by 12:00). Data tracing was scoped to this one method and disabled immediately after each capture. The captured credentials remain in the execution log group for its 30-day retention and should be purged.

Rate limits: the published figure describes one plan of three

PLAN	RATE/S	BURST
pos-partner-message-plan	300	400
pos-partner-plan	500	500
pos-partner-high-limit-plan	5,000	800

The documented "300 requests per second, burst 400" matches exactly one of the three POS plans, presented as though it were universal — a **16.7× spread** between the slowest and fastest partner tier, undisclosed. A second page states 10,000 requests per minute, which matches no plan at all; the only quota in the estate is netsuite's 10,000 per *day*. **Zero 429s occurred in 30 days** across 99 million requests, so throttling is configured and never fires.

Defects in the published spec itself

- **A malformed path.** `GET /pos/v1/{pos_user}/rewards` is missing the `/members` prefix — verified verbatim in the source spec, and its sibling `GET /members/{pos_user}/offers` has it. As published, it would 404.
- **An editing artifact.** `GET /general/v1/new-ten_dlc_submissions/field_options` — the `new-` prefix has leaked into the published spec.
- **The guide contradicts the reference.** The 10DLC guide instructs partners to poll `/ten_dlc_submissions/status`; the reference documents `/ten_dlc_submissions/{submission_id}/status`. A partner following the narrative calls a path that does not exist.
- **A documented step with no endpoint.** `POST /ten_dlc_submissions/{submission_id}/attachment` is step 6 of the documented user flow and has no reference page or spec entry at all.
- **An incomplete lifecycle.** `GET /batches/{batch_id}` is documented; nothing documents how to *create* a batch. Partners are told how to check the status of something they have no documented way to produce.

Contradictions a partner would hit on day one

The reference specs uniformly declare `AUTH-TOKEN` in uppercase. The Shopify guide instructs lowercase `auth-token` and adds: *"It is very important that the headers have the exact syntax."* Across all pages the split is 50 uppercase to 16 lowercase. HTTP headers are case-insensitive, so

this is probably harmless in practice — but the documentation asserts a strict requirement that contradicts its own reference section.

Two further inconsistencies: the Authorization guide's example shows `AUTH-TOKEN: <partner_api_key>` while the surrounding prose defines AUTH-TOKEN as the *merchant* token; and the published rate limit is 300 requests/second on one page and 10,000 per minute (167/second) on another. Which the gateway actually enforces cannot be determined from the documentation.

> **What was not tested.** No live requests were issued against gamma or production — that needs partner credentials and would write to a real system. The malformed paths are therefore established as published defects, not as confirmed 404s.

03

The gateway estate

Twelve APIs, unchanged in a year of drift, still almost entirely uninstrumented — but the traffic they reject has fallen by nearly half.

API	ENDPOINTS	ACCESS LOG	METRICS
brands	67	no	off
native_app	63	no	off
wallet	52	**yes**	off
general	52	no	off
callbacks	46	no	off
pos	41	**yes**	off
tablet	15	no	off
join	13	no	off
reports	12	no	off
netsuite	4	no	off
analytics	2	no	off
shortlinks	2	no	off

The identifier map is unchanged since early August — no API added, none removed — and the orphaned `h2dn0titad` log group still exists with no API behind it. **Only 2 of 12 have access logging, and `metricsEnabled` is false on all 12**, so no per-route metric or alarm exists anywhere in the estate. That is unchanged.

> **There is no staging stage.** The gateways have previously been described as having identical staging and production deployments. Every one of the 12 APIs now has exactly **one** stage, named `production`. There is no staging stage left to compare against.

Requests rejected before they reach Rails

API	MEASURED (4H)	EXTRAPOLATED/DAY	04 AUG
pos — missing `AUTH-TOKEN`	98,056	~588,000	337,000
analytics — missing API key	52,533	~315,000	316,000
general — schema validation	2,425	~14,600	8,000
callbacks — timeouts	0	~0	447
Total	**153,014**	**~918,000**	1,640,000

Analytics rejections are entirely `POST /v1/native_app/events` with an API key not associated with a usage plan. The pos rejections are all `Missing parameters: [AUTH-TOKEN]`. **Callbacks has effectively stopped rejecting** — two rejections in a full 24-hour window, against 447 a day when last measured.

> **Treat the per-day column as order-of-magnitude.** These are a six-fold extrapolation from one four-hour afternoon window, and an attempt to check diurnal variation across 24 hours did not complete. The measured four-hour counts are solid; the daily projections are not, and the earlier 1.64M/day figure was not reproduced. Counts are deduplicated by request id — each rejection emits two or three log lines, so raw line counts overstate by that factor.

Where POS writes actually go

Three of the 41 pos endpoints bypass the monolith entirely:

METHOD	PATH	INTEGRATION
POST	`/v1/members`	Lambda — webhooks_delivery_service
PUT	`/v1/members/{pos_user}`	Lambda — webhooks_delivery_service
POST	`/v1/visits`	Lambda — webhooks_delivery_service

Unchanged, along with the two general endpoints that resolve to SQS (`auto_campaigns/trigger` and `message_template_logs`). A further 17 callbacks endpoints route to the same Lambda and 12 reports endpoints to an Athena handler.

> **A partner request may never touch code you are reading.** Anyone debugging POS member or visit writes by reading the Rails controllers is reading the wrong system — those three endpoints are served by a Lambda. This is the single most common source of wasted investigation time in this estate, and it is why the endpoint inventory is worth keeping current.

04

Background jobs

The job fleet is reliable. The measurement of it needed correcting, and the corrected figures are below.

ENVIRONMENT	STARTS	DONES	FAILS	ERROR LINES
-secondary	17,569,012	17,566,128	2,890	186,498
-v2	1,502,485	1,496,842	5,640	4,608
Total (3 days)	**19,071,497**	**19,062,970**	**8,530**	**191,106**

Success rate **99.955%**, roughly 6.36M job starts a day, and the books balance (`start − done − fail = −3`). **-secondary carries 92% of all jobs**, which confirms the inverted environment naming: the environment that serves almost no web traffic runs nearly all the background work.

> **A note on how this was counted.** The earlier count used a substring match on `"fail"`. Validated against a 30-minute window, that pattern returns **105 matches where an exact anchor returns zero** — every one was the word *failed* inside `Validation failed:` error text. The same contamination inflates "start" by 7.5%. The 13,494 failures, 259,637 exceptions and 19.2:1 ratio derived from them should all be treated as unreliable. Recomputed with exact anchors the ratio is **22.4:1** — which must *not* be read as a rise, because the earlier figure was measured a different way.

Per-worker attribution is still impossible

Fail lines read, verbatim:

```
I, [2026-08-16T21:58:50.223153 #51824]  INFO -- : fail
```

No class, no JID, no elapsed time — only a PID. Per-worker execution counts, success rates and durations remain unobtainable from this source. The fix is written and waiting on the `observability-quick-fixes` branch, which sets a JSON formatter and `level: :info`; the latter also removes the ~92% DEBUG-SQL volume that makes a 30-day retention unaffordable today.

05

The transaction-date defect — and a class that is no longer blind

The platform's most persistent error is unchanged. But the error class around it turns out to be attributable today, which narrows the biggest standing recommendation considerably.

Steady since early August

WINDOW	COUNT	BASIS
14 August (full day, UTC)	27,550	measured
15 August (full day, UTC)	23,112	measured
04 August	26,845/day	—
May 2026 audit	47,901/day	—

Still logged as a bare string with no merchant, no location and no POS type:

```
[merchant_location] Validation failed: Transaction date can not be greater than created date
```

The rescue site (`pos_merchant_location_concern.rb`, ~line 175) logs only the exception object and is annotated `# do nothing`. The top standing recommendation — add `merchant_id` and `pos_type` here — remains unimplemented.

The error league table has changed at the top

ERROR (FULL DAY, 15 AUGUST)	NOW	04 AUG
Email has already been taken	**24,968**	7,709
Transaction date can not be greater than created date	23,112	26,845
Mobile subscriber has already been taken	6,837	3,800
Pos has already been taken	872	—
Phone number is invalid	14	—

Duplicate-email failures have **tripled** and are now the platform's largest single error class.

This class is attributable today, with no code change. The member-update retry lines carry `opt_event_data.enrollment_change_source`. Measured over four hours, the duplicate failures resolve cleanly to their originating integration:

SOURCE	COUNT (4H, MEASURED)
treez	1,018
hifyre	940
leaflogix	433
cova	176
janepos	105
flowhub_maui	94
blaze	63
shopify	52
web_wallet	37
zapier	23

No code change is required to produce this. Only the transaction-date rescue site genuinely lacks attribution — which makes that one fix both smaller and more clearly worth doing.

06

Swallowing rescues

The code census was run as an AST walk rather than a text search, so the counts are exact. What is new is the discovery that some of these rescues do not merely lose a log line — they can persist wrong data.

The census, re-measured

Re-run as a Ruby `Ripper.sexp` AST walk over all 2,350 `.rb` files in `app/` and `lib/`, rather than a text search. Zero parse failures.

LEVEL	04 AUG	NOW	
info	288	288	CONFIRMED
error	273	273	CONFIRMED
warn	56	56	CONFIRMED
debug	16	16	CONFIRMED
tagged	10	10	CONFIRMED
Total	**643**	**643**	CONFIRMED

A naive text search returns 644. The single difference is a **commented-out** logger call in `headset_integration/location_service.rb:43`, which the AST correctly excludes. Two directory-level movements since 4 August are both explained by git history: one worker logging call was removed in the Lightspeed token-refresh refactor, and one was added in the new `enroll/home_controller.rb`.

Swallowing rescues

METRIC	04 AUG	NOW
Log calls inside a `rescue` body	257	258
Of those, swallowing	201	208
Distinct swallowing rescue blocks that log	—	170

DIRECTORY	04 AUG	NOW
app/services	73	73
app/models	50	51
app/workers	40	45
app/controllers	32	33
app/helpers	4	4
app/policies	2	2

The rule, stated so the number is reproducible. For each logging call, the innermost enclosing `rescue` is located by line span; the site counts as swallowing if that rescue body contains no `raise`, `fail` or `throw` at any depth. Under progressively stricter variants the count moves 208 → 207 (no return-with-value) → 195 (no `retry`) → 191 (no `next`). The earlier figure of 201 sits between two of those variants, so the +7 is a rule-definition difference rather than seven new swallows. Known limitations: the rule tests lexical presence of `raise`, not reachability, and does not follow calls into helpers that may re-raise.

The structural consequence, now counted

When a rescue swallows inside a Sidekiq `perform` body, the job records as **succeeded** — no retry, no dead-letter queue, no alert. The subset can be counted precisely. **42 of the 208 swallowing sites sit directly inside a `perform` method**, across 33 worker files, with zero ambiguous attributions.

Thirteen of those 33 files are POS sync workers. Member profile sync, visit insert, catalogue and token workers all carry swallowing rescues inside `perform` — the exact path where a silently-succeeded job means member or visit data was never written. Combined with the log-formatter defect, which strips class and JID from every outcome line, such a loss is invisible from both directions: the job reports success, and the log cannot say which job it was.

FILE	SWALLOWING SITES
`app/models/member_profile.rb`	12
`member_import/mjfreeway_email_processor_service.rb`	11
`app/models/external/optimove.rb`	8
`services/ai/unified_chat_service.rb`	7
`services/shopify_service.rb`	7
`api/wallet/v1/members_controller.rb`	6
`concerns/pos_merchant_location_concern.rb`	6

The transaction-date rescue, quoted

The site behind the platform's most persistent error, verbatim:

```
    begin
      visit.save!
      if visit.persisted? && pos.try(:visit_details_included?, o)
        visit.visit_details = pos.create_visit_details(visit, o.dig("line_items"))
      end
    rescue ActiveRecord::RecordInvalid => e
      logger.tagged("merchant_location") do
        logger.info "trying to save a visit"
        logger.error e
      end
      # do nothing
      # its just sidekiq trying to save the same visit in multiple threads
```

It still swallows, and it logs **no merchant id, no location, no pos type and no order identifier** — only a literal tag and the exception. That is why ~25,000 failures a day cannot be attributed to anyone.

> **The inline justification is narrower than the code.** The comment assumes the only cause is a duplicate-write race between Sidekiq threads. But `RecordInvalid` is raised for *any* validation failure on the visit. A genuinely invalid order — bad total, missing required field — is discarded on the same path, and because `save!` failed, `visit.persisted?` is false, so the line items are never created either. **The order is dropped whole, silently.**

Swallowed exceptions that corrupt data

Twelve of the 208 sites sit in `app/models/member_profile.rb`. They look like safe retry loops and are not — the `retry` is guarded, so when the counter is exhausted control falls off the end of the rescue and the exception is absorbed:

```
      member_profile.save!
    rescue ActiveRecord::StatementInvalid => e
      logger.tagged('member_profile') do
        logger.error 'Trying to create a member_profile'
        logger.error e
      end
      if (retries -= 1).positive?
        sleep rand(0.1..0.8) # to avoid db lock due to sidekiq race conditions
        retry
      end
```

`save!` is the bang form, chosen precisely so it raises on failure. On the terminal path that raise is absorbed, the method returns normally, and **the caller cannot distinguish "profile created" from "profile never persisted."** The condition this code was written for — lock contention between concurrent Sidekiq jobs, per its own comment — is exactly the condition under which every retry fails and the loss is silent.

> **Two sites go further and mutate data on the error path.** Around lines 690 and 870 the rescue strips the fields that failed validation and retries the write: `error_attributes = member.errors.attribute_names` followed by removal of those attributes, and `found_member.errors.messages.each { |key| found_member[key[0]] = nil }`. A "successful" write can therefore persist a member record with silently dropped or nulled attributes. This is a concrete path from a swallowed exception to **wrong data**, not merely a lost log line — and it sits on the member-identity path, which is also where the platform's largest error class (24,968 duplicate-email failures a day) is generated.

07

POS vendor health

Strong across the board, with one location that has quietly gone dark.

Roughly **341,000 visits a day** across 36 active vendors. Every vendor with meaningful volume delivered on the last full day measured; none stopped.

POS	VISITS/DAY	WEEK OVER WEEK	MERCHANTS
dutchiepos	87,189	−0.7%	280
cova	45,508	−2.1%	75
hifyre	38,475	+28.3%	17
posabit	20,239	−1.8%	35
treez	18,594	+0.4%	48
flowhub_maui	14,553	−4.3%	42
blaze	13,183	−1.1%	42
lightspeed	9,982	+0.7%	20
greenline	7,990	−0.1%	30

Wasted polling is now zero. Lost merchants own 9,209 locations and **100% of them are disabled**. Not one lost merchant has an enabled location. greenbits is 0 of 334 enabled and treez 1 of 297 — which independently confirms the earlier judgement that their error volume was noise from departed tenants rather than an outage.

One location has quietly gone dark

cova location 26112 (Alpha Cannabis, merchant 11637) logged 17,795 failed 401s in seven days while visits landed normally, and was deliberately excluded from the 8 August cleanup on that basis.

Re-measured: **zero visits since 2026-05-20**, roughly three months, while the location remains `enabled` and actively scheduled for polling. The errors that were correctly dismissed now sit alongside no ingestion at all.

How a correct finding goes stale. That conclusion was not wrong when written — it rested on a condition ("visits are landing") that was true then and is false now. A finding that depends on a live condition needs to be restated as a monitor, not recorded as a verdict.

Two vendor-collapse alerts, both false

- **teamworks (−89.6%)** — merchant 36627 loaded 463,769 backfilled rows on 4 August whose `transaction_date` values span June and July. Measured by `transaction_date`, daily activity is flat.
- **import (−52.9%)** — a manual bulk-load channel that fires on isolated days. Its mean is meaningless, and it is not a vendor.

> **Trend on `transaction_date`, not `created_at`.** Backfills land tens of thousands of rows under a single insert date, which reads as a spike and then a collapse. Two screens, two false alarms.

Merchant status and unprocessed churn

lost 3,863 · active 732 · pending 158 · suspended 56 · onboarding 12. Stable. **Lost outnumbers active better than five to one**, so any platform-wide error count that does not filter on merchant status is dominated by dead accounts.

129 billing-active merchants have sent zero visits in 60 days (245 silent, less the 116 that never transacted); roughly 30 have been silent for over two years. Four dormant accounts flagged earlier have not recovered: merchants 10370 (last visit 2024-12-16), 13977 (2025-02-01), 12143 (2026-01-09) and 17351 (2026-04-01).

One cluster is worth a targeted check: four dutchiepos merchants — The Vault Spokane (2066), Silvana (2358), Lake Stevens (2853) and Herbn Elements (3650) — went silent on exactly **2026-05-21**, each retaining one enabled location. dutchiepos overall is healthy, so this is per-account rather than a vendor problem.

08

Partners: is it them or us?

This is answerable for one of the two largest error classes and not the other. One of the two can now be answered — and the answer points at us more than at them.

The question matters commercially: when a POS integration produces hundreds of thousands of errors a week, the response differs entirely depending on whether the partner is sending bad data or springbig is mishandling good data. That reading concluded honestly that an honest split was **not computable**. That is now half true.

What can be attributed today

Duplicate-member failures — the platform's largest error class at **24,968 a day** — carry `opt_event_data.enrollment_change_source` on the retry line. Over a four-hour window they resolve cleanly:

SOURCE	FAILURES (4H)	SHARE
treez	1,018	33.6%
hifyre	940	31.0%
leaflogix	433	14.3%
cova	176	5.8%
janepos	105	3.5%
flowhub_maui	94	3.1%
blaze	63	2.1%
shopify	52	1.7%
web_wallet	37	1.2%
zapier	23	0.8%
posabit	22	0.7%
dutchie_ecomm	16	0.5%

A straightforward correction, in the useful direction. The class previously recorded as unattributable is attributable with a log query and no code change. Two integrations — treez and hifyre — generate roughly two thirds of all duplicate-member failures. That is now a specific, ownable conversation rather than an aggregate complaint.

But attribution is not the same as fault

A duplicate-member error means the platform tried to create a member who already exists. That can be the partner re-sending, or it can be springbig's own matching logic failing to recognise an existing member. The **member-identity code path is the one carrying twelve swallowing rescues that null out failed fields and retry** — so a member record whose email or mobile was silently dropped on a previous write will not match on the next one, and will present as a duplicate.

> **The two findings may be the same finding.** The largest error class sits on the code path with the most data-corrupting rescues. This report does not claim to have proved a causal link — that needs a targeted trace of individual member records across successive writes. But it is the single most promising thread in this document, because if the link holds, a partner-facing problem is actually ours, and the fix is a rescue block rather than twelve integration conversations.

What still cannot be attributed

The transaction-date defect — roughly 25,000 failures a day — remains genuinely blind. Its rescue site logs a literal tag and an exception object with no merchant, location, POS type or order identifier. It is now the **only** major error class in that state, which makes the instrumentation recommendation both smaller in scope and harder to defer.

> **The 404s are not part of this.** POS partners generate roughly 3 million 404s a week through the documented lookup-then-create handshake — asking whether a member exists before creating one. They are protocol, not failure, and any them-or-us split that counts them will blame partners for behaving correctly.

09

APM: what New Relic sees, and what it is configured not to

Every transaction is instrumented. No error is. This is a settings problem, not a tooling problem, and it is the cheapest item on the list.

METRIC (7 DAYS)	04 AUG	NOW	
Transactions	52,226,146	55,670,502	+6.6%
Background jobs	16,358,570	17,503,582	+7.0%
Background hours	735	959	+30.5%
Web hours	794	1,033	+30.2%
`TransactionError` events	0	0	UNCHANGED

Zero error events across 345,845 HTTP 500s. The `error` attribute is false on every transaction, including those returning 500. New Relic therefore reports a 0% error rate while the platform serves hundreds of thousands of server errors a week. No alert can fire from this source. Enabling error collection is a configuration change, not a project.

Compute grew about 30% while request volume grew about 7%. Work per request is rising faster than traffic — not yet a problem, but the trend is worth watching, and `api/pos/v1/members/index` at 338 hours a week is where it concentrates.

A retention limit that will mislead you. This account retains `Transaction` events for **eight days** — the earliest available event is 2026-08-08. Any `TIMESERIES` query reaching further back returns zeros that look exactly like a clean onset date. During this investigation that artifact very nearly produced a false attribution of a defect to a specific deploy. Always probe `earliest(timestamp)` before claiming an onset from this source.

10

Error reporting

Rollbar's active item count more than doubled in twelve days. That is not necessarily more breakage — but nobody is triaging it, and the highest-volume items carry no stack trace.

METRIC	04 AUG	NOW	
rails active items	69	166	+141%
frontend active items	91	115	+26%
frontend total occurrences	265,777	266,521	+744
frontend oldest item	2,028d	2,040d	CONFIRMED
frontend items > 1yr / > 3yr	12 / 11	19 / 16	WORSE
wallet items	0	0	STILL SILENT

84 of the 166 rails items (51%) are new since 4 August, and **118 of 166 (71%) fired within the last three days**. This is a live surface, not a stale backlog. The new items are numerous but individually low-volume — a broad front rather than one runaway error.

The wallet project's silence is now proven rather than assumed: its token authenticates normally and returns zero for active, resolved and muted alike. A live consumer surface serving 1.98M requests a week reports nothing at all. The frontend project shows 0 resolved and 0 muted against 115 active — nobody is working it.

> **An earlier claim corrected: "the stack traces are gone" is not right as stated.** 16 of the top 20 rails items carry full stack traces, 56 to 146 frames each. The four without are `Rollbar.error("string")` calls — the code never sent a trace in the first place. But those four are the largest by volume: **100,486 of 130,855 occurrences (77%)** carry no trace. The symptom reported earlier was real; the cause was misdiagnosed, so the fix is different — change the call sites to report exception objects, rather than adjusting Rollbar's configuration.

Suppression is in code, not in the console

`config/initializers/rollbar.rb` discards or downgrades whole exception classes:

```
'ActiveRecord::RecordNotFound' => 'ignore',
'ActionController::RoutingError' => 'ignore',
'ActionView::Template::Error' => 'warning', # this is just CS people / ActiveAdmin
```

Both rules are individually defensible and collectively blinding. The `RecordNotFound` rule cannot distinguish a bot probing a URL from a member whose link stopped working — both raise the

same class and both are dropped. The `ActionView::Template::Error` comment is an assumption that has outlived its context: the same class now fires on customer-facing enrol pages and on the API. The Reward 48176 deletion produced 194 occurrences of it between 13 and 14 August, all filed as warnings, resolving to `app/views/api/web/v1/extension_rewards/_reward.jbuilder:1`.

New clusters since 4 August

- `PG::InvalidTextRepresentation` — 1,355 occurrences in a tight burst on 10 August that started and stopped the same day.
- `PG::QueryCanceled` and `ConnectionNotEstablished` — ~220 occurrences 12–14 August against a single host, the known Aurora replica-conflict signature.
- `ActionView::Template::Error` — ~194 occurrences from 13 August, the Reward 48176 deletion, still filed as warnings.
- Frontend: `vue-query hooks can only be used inside setup()` accounts for 397 of 464 new occurrences.

> **One caveat on the counts.** The Rollbar API returns `total_count = 1000` at every limit, so resolved and unique-item counts are a floor rather than a confirmed total. The active counts (166 / 115 / 0) are fully paginated and reliable.

11

Native apps, and why frontend errors are unreadable

The sharpest earlier claim on this surface is confirmed emphatically. Two earlier counts are not reproducible from the underlying data and are withdrawn. And one dataset previously recorded as unavailable turns out to hold 410 million rows.

Not one readable stack trace

The top 15 frontend items account for 264,516 occurrences — 99.2% of all active frontend volume. One instance was fetched from each and every frame examined.

VERDICT	ITEMS	OCCURRENCES
Symbolicated — real file and function	0 / 15	0
Minified only	12 / 15	195,694
No usable trace at all	3 / 15	68,822

Every frame resolves to a content-hashed bundle and a mangled identifier:

```
assets/index-mm3f9i1-.js :28:1996  [XMLHttpRequest.S]
assets/index.afe282a1.js :16:1994  [RB]
assets/index-BjBsRhPE.js :14:1401  [Y4]
```

Three independent signals of source-map resolution were checked across all frames — `.vue`/`.ts` extensions, `webpack:/` or `/src/` paths, and Rollbar's `code` field, which populates only when a map resolves. **Zero hits on all three.** Rollbar is receiving raw minified frames and resolving nothing.

Uploading source maps now would not recover the backlog. Every deploy produces a different bundle hash, and Rollbar resolves maps against the hash recorded at capture time. Historical items stay unreadable permanently. The fix has to ship before the errors you want to read, not after.

This is not a noise problem

CLASS	ITEMS	OCCURRENCES	SHARE
Genuine application errors	112	264,319	99.2%
Browser noise (ResizeObserver etc.)	3	2,202	0.8%
Third-party scripts	0	0	0.0%
Browser extensions	0	0	0.0%

Classification was done on **stack-frame filenames** rather than titles — extension protocol prefixes and known third-party CDN hosts — so it does not depend on error text. Of the top 45 items by volume, 35 originate on `springbig.io` itself and none on an extension or a third party. The inbox is almost entirely first-party signal that cannot be read.

ERROR FAMILY	ITEMS	OCCURRENCES	SHARE
HTTP 401 from the API (axios)	2	178,423	66.9%
Unhandled rejection with no reason	1	66,788	25.1%
HTTP 404 from the API	2	8,381	3.1%
vue-query used outside `setup()`	40	4,868	1.8%
HTTP 422 from the API	1	3,182	1.2%

> **Two structural distortions in these counts.** The 40 vue-query items are almost certainly **one bug** fragmented across 40 item ids by per-deploy bundle hashes — which inflates the item count and splits the occurrence total. Separately, two active items are Rollbar's own rate-limit warnings (`Your rate limit has been reached`, `item per minute limit reached, ignoring errors`), which means **266,521 is a floor**, not a true total. Errors are being dropped before they are recorded.

Two counts that cannot be reproduced

Figures of **198 mobile app projects** and **318 cloud projects** have circulated internally. Neither is reproducible from the underlying data.

CANDIDATE MEASURE	VALUE
merchant_organizations with a native app name	1,909 of 1,910
Distinct native app names	1,877
Distinct `native_app_firebase_project` values	272
Distinct app names in telemetry (all time / 30d)	194 / 124

The provisioning column cannot support a project count. `native_app_firebase_project` is an **integer, not a GCP project identifier**. 1,467 organisations share the value `1` and 128 share `100` — 1,595 of 1,910 sit on two sentinel values. Any figure derived from it as a project id is unsound. No GCP credential is reachable from this environment, so "318 cloud projects" is recorded as **not measured** rather than refuted.

One related figure is reconciled rather than disputed: **568** active non-demo merchants sit under an organisation with a native app; the 594 quoted elsewhere in this session is the same measure *including* 26 demo accounts.

Mobile is instrumented for behaviour and not at all for failure

Mobile telemetry has been described internally as not retained anywhere queryable. It is: `springbig_production_logs.native_app_events` holds **410,602,054 rows** spanning January 2023 to today, live. It appears in Postgres as a foreign table and is unreadable there, but queries directly against Redshift.

MEASURE	VALUE
Total rows	410,602,054
Events, last 7 days	148,457
Distinct devices, last 7 days	7,324
Distinct devices, all time	1,129,437
Distinct event names	82

None of it is crash data. All 82 event names were searched for crash, error, exception, fatal and ANR signatures; the only match is `NativeApp::view-cart-failed` — a business-flow failure, not a crash. All three Rollbar projects were swept for mobile crash signatures (`java.lang`, `NSException`, `EXC_BAD_ACCESS`, `SIGSEGV`, Crashlytics, React Native, Flutter) with **zero matches**; the only frameworks reporting are browser-js, rails and sidekiq. The apps are richly instrumented for behaviour and carry **no failure telemetry at all**. The earlier claim that the crash gap "is narrower, and it is iOS" cannot be evaluated — there is no crash data in reach to size it with.

The wallet reports nothing, and it is a live consumer surface

The wallet Rollbar project is **enabled**, its token is **valid**, and it returns **zero items in every status** — active, resolved, muted and suspended alike. It has eight registered environments including production, and Rollbar creates an environment only when events arrive, so **it reported at some point and stopped**.

Compounding, not isolated. The wallet serves roughly 1.98M requests a week and is the surface most likely to carry the in-app consumer experience. Between this and the absence of crash telemetry, the consumer-facing mobile experience has **neither error reporting nor crash reporting**. Whether the SDK was removed, misconfigured or its token rotated is not determinable from the API alone — that needs the wallet app's deploy configuration.

The edge

A full sweep of all 1,799 zones — not a sample — gives the first complete picture of edge traffic. Nearly a quarter of it is a 4xx, and the plan tier prevents us from saying why.

CLASS	REQUESTS (7D)	SHARE
2xx	10,980,751	60.15%
3xx	2,601,639	14.25%
4xx	**4,195,308**	**22.98%**
5xx	**477,539**	**2.62%**

The 4xx figure is dominated by **3,653,786 403s — 20.02% of all edge traffic**. That should not be read as 3.65 million security events: a 403 at the edge is frequently a member-not-found rather than a genuine authorization failure. Distinguishing the two requires firewall event data, which is **denied on the Free plan** — re-tested at both zone and account level, still `code=authz`.

> **This is the concrete cost of the plan tier.** The single largest anomaly at the edge — one request in five — is uninterpretable with the data available. Firewall events, bot scores and origin-latency percentiles are all unavailable at any granularity. Whether to upgrade is now a question with a specific number attached to it.

METRIC	04 AUG	NOW
Zones	1,860	1,799
Zones with traffic	1,858	1,798
Cache hit rate	3.31%	4.15%
Worker invocations (7d)	15,996,051	12,221,543
Firewall / bot / origin latency	not authorized	not authorized

Cache remains effectively unused at **4.15%** of requests, against a 59.3% byte hit rate — meaning a few large static assets cache well while nearly all dynamic and API traffic does not. Workers remain the well-instrumented layer at a 0.015% error rate, with two exceptions: `wallet-3-testing-amplify` fails **100%** of its 85 invocations and `wallet-3-testing-api` 69.6% of 125. Volume is negligible, so these read as broken test workers rather than customer impact — but they fail silently.

Several low-volume zones fail on nearly every request and nobody is watching them: `springbig.ai` (99.3% 4xx), `sbtest101.com` (99.4% 5xx), `stash-board.com` (88.3% 5xx), `springbig-wallet-alpha.click` (72.1% 5xx).

Still no POS traffic at the edge, and one number withheld. A keyword sweep of the top 120 hosts and paths found two apparent POS matches, both false positives — `posnrgy.com` is a merchant SMS domain whose brand name contains "pos", and `/apple-touch-icon-precomposed.png` contains it inside "precomposed". POS traffic bypasses Cloudflare entirely, so edge data will never help diagnose a POS incident. Separately, the account-level rollup undercounts by 40% against the full per-zone sweep for reasons that could not be established — so no traffic delta between readings is published here, because the two baselines may not be derived the same way.

13

The lambda estate and its alarms

The estate is healthy on invocations. What matters here is quieter: a large fraction of the alarm estate is not protecting anything, and one whole service has no alarms at all.

METRIC	04 AUG	NOW
Lambda functions	—	49
`/aws/lambda/*` log groups	51	56
Active (≥1 invocation in 7 days)	28	37
Silent	23	12
Orphan log groups (function deleted)	—	7

The nine-day outage is resolved. `pos_service_request` crashed 100% of its 1,440 daily invocations from 21 to 29 July, exactly as described earlier. It **recovered on 30 July** and is healthy now, with two brief relapses around 11.9–13.3% on 10 and 11 August that have since cleared. Because the function is cron-driven at precisely one invocation a minute, its failure never varied with load — which is why nothing else in the estate reflected it.

Error rates, measured as ERROR lines per invocation

FUNCTION	ERROR LINES / INVOCATION	INVOCATIONS (24H)
analytics_process_incoming_events	789%	1,782
web_request_runner	215%	67,663
messaging_send_push	178%	1,757
quickbooks_online	61%	723
messaging_send_sms	52%	6,863
pos_response_runner	48%	38,564
webhooks_delivery_service	27%	1,029,485

Ratios exceed 100% because batch handlers emit many error lines per invocation — this is a lines-per-invocation measure, not a failure rate, and it is not comparable to the earlier 58.7%. `analytics_process_incoming_events` remains rank one with the identical error string quoted earlier.

Its CloudWatch `Errors` metric is zero. The analytics handler catches and logs every rejection, so the invocation succeeds. The platform metric — and every alarm built on it — sees a perfectly healthy function while roughly 14,000 events a day are discarded. This is the lambda-tier version of the swallowing-rescue problem documented in section 06.

The alarm estate

STATE	COUNT
OK	93
ALARM	1
INSUFFICIENT_DATA	**31**
Alarms with no action configured	**24**
Total metric alarms	125

31 alarms have no data, and 25 have had none for 165 to 1,274 days. An alarm receiving no data is not protection — it is the appearance of protection. Twelve of them are a complete data-freshness suite (`*_lag-alarm` on visits, transactions, members, merchants, campaigns, SMS logs and more) that has reported nothing for roughly **2.8 years**. A further 24 alarms across the estate have **zero actions** configured, including the single alarm currently firing.

The structural point — CloudWatch notifies on state *transition*, so a stuck ALARM is invisible — remains architecturally true. But the specific instance it cited is not reproducible today: only one alarm is in ALARM and it tripped eleven minutes before measurement.

Provenance

Only 2 of 49 functions run a deprecated runtime (both python3.9, both silent CDK helpers), and seven have been unmodified for over a year — including `test_lambda`, a test function deployed to production, silent for 370 days and still provisioned.

The "one repo hosts four lambdas" finding is **confirmed**: the `pos-request-service` repository deploys `pos_request_runner`, `pos_cache_runner`, `pos_response_runner` and `pos_service_request` as four separate CloudFormation stacks. Nothing in the function list reveals this — only the stack-name prefix does.

A telling detail about the July outage. The alarm named `pos_request_service errors` — named for the *repository* rather than any of its four functions — has been in INSUFFICIENT_DATA for **1,006 days**. The one alarm whose name suggests it would have caught a nine-day outage in that service has been dead for nearly three years. Four further functions carry no stack tag at all, so there is no automated path back to their source, including `Aurora_Parameter_Compliance`, which fails 75% of its invocations on a missing `configurationItem` key and has no alarm.

14

Dead-letter queues

This was called "a delete timer wearing a safety-net label" and found one instance. It is not one instance. It is every queue in the account.

Queues	Undelivered messages	Queues at 14-day max	SQS alarms
87	**742**	**0**	**0**
38 of them dead-letter queues	in two DLQs	of 87	across the entire estate

RETENTION	QUEUES	OF WHICH DLQ
1 day	19	0
4 days	68	38
14 days (the AWS maximum)	0	0

Every dead-letter queue in this account is a four-day delete timer. Not one queue uses the maximum retention, and the nineteen send-request queues at one day are tighter still.

722 messages are being destroyed on a rolling basis

QUEUE	MESSAGES	RETENTION	OLDEST MESSAGE
`pos_customer_sync_payloads_dlq`	722	4 days	4.00 days — at the cap
`pos_order_requests_dlq.fifo`	20	4 days	2.78 days

Fourteen days of metrics on the larger queue show `NumberOfMessagesReceived = 0` and `NumberOfMessagesDeleted = 0` every single day — **nothing has ever consumed it**. Yet its depth oscillates between 571 and 722 while the oldest message sits pinned at exactly the four-day retention cap.

Depth that moves with zero deletes means messages are ageing out. This is unreviewed POS customer-sync failure data being destroyed on a rolling four-day cycle, and it has been happening for at least fourteen days. Raising retention to fourteen days is immediate and reversible, and should happen before anyone tries to drain it.

> **There is not a single alarm on any SQS queue.** The estate carries 125 metric alarms across Lambda, Route53, RDS, S3, DynamoDB, WAF and more — and **zero** on `AWS/SQS`. No `ApproximateNumberOfMessagesVisible`, no `ApproximateAgeOfOldestMessage`, nothing on any of the 38 dead-letter queues. That is the direct and complete explanation for why a 722-message backlog has sat in production unnoticed. It is also the highest-value gap in this report to close, because it is configuration rather than code.

The "dead queue still provisioned" finding is confirmed literally: `dead_queue` exists with zero messages, alongside `shoryuken_legacy_jobs` and its DLQ.

Sending identity: 10DLC registration

The registration chain is in good shape, and the gap that exists is specific and actionable.

LAYER	ACTIVE	INACTIVE	TOTAL
Brands	740	172	912
Campaigns	753	326	1,079

Of 652 active non-demo merchants, **499 are flagged `ten_dlc_enabled`** and expected to send SMS. Tracing the full brand → campaign → number chain for those 499:

GAP	MERCHANTS
No brand at all	0
Brand present but inactive	0
No active campaign	2
No active phone number	**36**
Any gap in the chain	**36 (7.2%)**

464 of 499 hold the complete chain. The 36 that do not have a valid brand and campaign and are missing only the number — a single, specific, fixable link rather than a registration failure.

> **A join key that produces a false crisis.** `ten_dlc_brands.merchant_id` is populated on only **5 of 912** rows; the real link is `merchants.ten_dlc_brand_id`. Joining the obvious direction yields "3 merchants have an active brand, 496 broken" — a catastrophic-looking number that is entirely an artifact. Anyone auditing this subsystem should verify the direction of the join before reporting anything.

Live numbers held by accounts that should not have them

MERCHANT STATUS	ACTIVE NUMBERS	MERCHANTS
suspended	1,769	41
pending	588	12
lost	341	7
onboarding	294	6
demo accounts	297	9
Total on non-active accounts	**3,289**	**68**

The 341 numbers held by seven **lost** merchants are the clearest reclaim candidates — a departed account holding live 10DLC numbers is both a recurring cost and unnecessary carrier-registration surface.

> **Two figures deliberately not asserted.** The earlier "438 registered messaging campaigns" cannot be reconciled against any column combination here — the nearest anchors are 1,079 total, 753 active and 148 submitted to Telnyx — so it is recorded as not reproducible rather than replaced. And **931 of 1,079 campaigns (86.3%) carry `submitted_to_telnyx = false`**, which is either a large real registration gap or a stale flag. Distinguishing the two requires the carrier-side registry, which is not in this database. It is flagged here because if it is real it is significant, and it is cheap to settle.

Data-integrity notes from the same pass: 39 brands are orphaned (no merchant points at them), 2 active campaigns hang off an inactive brand, and all 652 active merchants carry a non-null legacy `dlc_campaigns` JSON column running parallel to the normalised tables — a possible dual source of truth that was not audited.

Revenue assurance

Two findings that belong together. Both describe a commercial arrangement and a running system that disagree, and neither appears on any monitoring surface — because in both cases every request succeeds.

Loyalty mechanics on messaging-only accounts

Of 233 active, non-demo merchants labelled `platform_type = messaging_only`, **31 are running loyalty mechanics** in the last 90 days.

SIGNAL (90 DAYS)	MERCHANTS
Awarding points (> 0)	17
15 or more redemptions	4
15 or more reward grants issued	16
Any of the above	**31**

What was deliberately excluded, and why. POS integrations and visit ingestion are **legitimate** on a messaging-only account — that is how member contact details, consent values and segmentation facets arrive. A merchant with visits landing and zero points awarded is correctly configured, not broken. Only loyalty *mechanics* — points actually awarded, rewards redeemed, grants issued — count toward the 31.

MERCHANT	POINTS (90D)	REDEMPTIONS	GRANTS
Fire and Flower	66,595,545	0	0
Bud Supply Group	5,942,818	0	0
Lucid	4,976,389	0	0
Prairie Records	2,951,955	0	0
Garden Variety	2,550,491	0	0
Food 4 Thought	195,010	1	19,501
Electric Ave	112,800	282	7,960
High Ties	0	1	908,706
Natural Remedies	0	19	35,614
American Cannabis	0	15	15,841

These are not recent reclassifications. Read against PaperTrail — up to 2,393 versions per merchant — **every one has held `messaging_only` with `loyalty = true` for its entire recorded history**. One merchant, High Ties, did move from `loyalty_and_messaging` on 2026-03-18, after which its points programme genuinely stopped (redemptions fell from 46,296 to 1) while its offers programme continued. **231 of the 233** messaging-only merchants carry `loyalty = true`.

Why it is possible: the enforcement gap

The capability boundary exists in code and is almost entirely unwired. `require_platform!` is defined twice and called **four times**, all in two legacy server-rendered controllers. The `api/web/v1` concern defines its own copy that **nothing calls**.

PATH	PLATFORM-GATED?
`rewards_controller` / `merchants_controller` (legacy)	YES — 4 CALL SITES
`api/web/v1/*` — the dashboard API	DEFINED, NO CALLERS
Visit ingestion and point accrual (`visit.rb:413`)	CHECKS `MERCHANT.LOYALTY` ONLY
Grant issuance and redemption	UNGATED

> **Sequence matters more than the fix.** Switching enforcement on today would change live behaviour for the 31 merchants running real programmes. The order has to be reconcile, then enforce: decide per account whether the label or the behaviour is correct, correct the mismatched field, and only then wire `require_platform!` into the API and loyalty paths so the drift cannot recur. The mechanism already exists — this is wiring and data reconciliation, not new development.

Messages sent against unpaid invoices

Measured at billing-organisation level over twelve months, excluding zero-amount invoices:

COHORT	ORGS	UNPAID	READ
Collectible — payment method on file, or a small org billed on platform	79	$896,551	Open invoices the platform should have collected. This is the actionable figure.
Billed out of band — no payment method and 3+ merchants	2	$459,094	**Excluded from the unpaid total.** Enterprise contracts settled off platform; the invoice record simply is not visible here.
No open invoices	308	—	Current

The collectible cohort sent **26,805,228 messages** in 90 days. Largest exposures: Story Cannabis $180,460 (7.7M messages), Beyond Hello $135,047 (11.8M messages), Spiritleaf $133,509, Lume Co. $96,621, FIKA $75,010 (2.4M messages), Sessions Cannabis $51,454 (1.2M messages).

Enterprise accounts are deliberately excluded, not overlooked. A billing organisation with no payment method on file *and* three or more merchants under it is settling on a negotiated contract, not defaulting — the platform simply has no visibility into that settlement. Applying that rule moves two organisations and **$459,094** out of the unpaid figure. The rule validates itself on inspection: the organisations it selects are Trulieve (68 merchants), Curaleaf (24), GTI (15) and Ethan Consulting (13), and they sent **zero** messages in the window. Reporting them as delinquent would have been wrong.

The backlog is **persistent rather than recent** — unpaid invoices appear in every one of the last twelve months, between 33 and 114 per month.

Two checks that changed these numbers. A per-merchant rollup first reported **$6.4M**. That was wrong: organisation-level invoices are attributed to every child merchant, so seven sibling accounts each showed the same $450,783. Rolled up at the billing organisation the figure is **$1.36M** — a 4.7× difference — of which $897k is collectible once enterprise off-platform contracts are set aside. Separately, `status = 0` was validated as genuinely unpaid before use: such invoices carry a settled payment 10.9% of the time, against 83.3% for `status = 1`. 278 zero-amount invoices were excluded.

Why these two sit together. One is revenue delivered but not contracted; the other is revenue contracted but not collected. Both are invisible to every monitoring surface in this report, because nothing is failing — the platform is doing exactly what it was configured to do. Both are closed by the same thing: a view that puts an account's commercial state beside its running state, so the difference is reported rather than discovered.

17

Audiences

Two audiences are failing right now, both with the defect closed on 8 August, because the configuration shape that causes it is still writable.

STATE	COUNT
completed	3,356
disabled	1,684
in progress	9
queued	7
failed	**2**

Both carry `failure_message = NULL`, which is itself diagnostic: the Node Lambda writes `state` directly to Postgres and never runs the Rails setter that populates the message. A null message with state 3 means the query died in Redshift.

71620 — merchant 38389

```
variant: greater_than
value: 1
secondary_variant: month     # datepart in the WRONG key
secondary_value: '1'
```

71378 — merchant 3092

```
variant: greater_than
value: 1
secondary_value: '1'         # datepart MISSING entirely
```

Working audiences on the same event type put the datepart in `secondary_value`. Because `ParseQueryService#render` substitutes an empty string for any variable it cannot resolve, the misconfiguration does not raise — it renders structurally broken SQL that travels through SQS and dies far from its cause. Captured from `/aws/lambda/audience_build_members`:

```
date_trunc('1', transaction_details.transaction_date AT TIME ZONE 'UTC' AT TIME ZONE 'America
ERROR: Invalid datetime part for DATE_TRUNC()
```

Both audiences fail **every day**: 71378 on eight of the last seven days' runs (10, 11, 12, 13, 14, 15 and twice on 10 August), 71620 on four. Twelve `failedBuildsForDlq` records over seven days, every one the same signature.

> **Third occurrence of one defect.** Audience 69762 was diagnosed with this exact signature on 8 August. Two more have appeared since, both healthy within the last week. The negative specs in `condition_value_validator` already say this shape must be rejected — the validator simply is not enforced on the write path that created these rows. Until it is, expect roughly one new instance every three to six days.

18

Webhooks: firing frequency and payload completeness

Two hypotheses were tested — that webhooks fire too often, and that they carry incomplete data. The second is confirmed. The first is largely refuted, and the reason it looked true is itself the more useful finding.

Endpoints	Outbound webhooks	Log group is inbound	Payloads with a change-set
816	**~474k**	**73%**	**0**
756 merchants · 100% on member_updated	per day, extrapolated	POS callbacks, not webhooks	of 816 endpoints

"Fires too often" — refuted as stated

The emission path is guarded at three independent levels, and the measured rate is close to the floor. `webhook_member_concern.rb:32–34` checks an attribute allowlist, so a bare `touch` or an `updated_at`-only save emits nothing:

```
changed_values = previous_changes.keys.map(&:to_sym)
deliver_webhook(:updated) if saved_changes? && (WEBHOOK_PERMITTED_ATTRIBUTES & changed_value
```

A second gate at enqueue time returns early unless a subscribed endpoint exists, and `EventWorker` carries `unique_for: 60.seconds`. Measured over two hours against real business activity:

DRIVER (2H, MEASURED)	COUNT
Member rows updated at webhook merchants	22,031
Members created	1,235
Visits	26,049
Outbound webhooks emitted	**39,527**
Ratio per member-change	1.70
Deliveries per distinct member	1.30 avg (35 max)

The amplification is fanout, not re-firing. A single invocation contacts 1.49 POS locations on average and up to 17. The outlier of 35 deliveries for one member resolves entirely to Lume Co., which has 43 leaflogix locations behind 2 endpoints — that is one event fanning out across locations, not the same event firing 35 times. Volume is proportionate to underlying business activity.

The headline number was a misattribution, and it was mine. The `webhooks_delivery_service` log group records **1.03M invocations a day** — but **73% of that traffic is inbound POS callbacks**, not outbound webhooks. Splitting the two by request path: outbound ~474,000/day (26.9%), inbound ~1,292,000/day (73.1%). Attributing the whole log group to webhook firing overstates it by nearly four times, and this report did exactly that before the split was measured.

"Carries incomplete data" — confirmed, but not by being sparse

Payloads are *large*: 29 to 32 fields, including points balance (`ghetto_balance`), consent state (`allowed_email`, `allowed_sms`, `allowed_loyalty`) and tier (`current_milestone_tiers`). What is missing is not bulk. It is **meaning**.

WHAT A RECEIVER DOES NOT GET	CONSEQUENCE
No change-set — no `saved_changes`, no previous values	Cannot tell what changed; must diff or blind-write all ~30 fields
No visit reference, though a visit is the trigger	Cannot tell a purchase from a profile edit
No timestamp of the change	Cannot order events or detect staleness
No `event_name` at all on the pos PUT path	42.5% of outbound volume arrives unlabelled

A change-set is not merely absent — it is structurally impossible to deliver. `WebhookMemberConcern` computes `previous_changes` at line 32 and uses it *only* to decide whether to fire. It is then discarded. The enqueued job carries just `{id, klass, event}`, and `EventWorker` re-fetches the record from a **read replica** in a different process, minutes later. By serialization time the dirty state no longer exists. **0 of 816 endpoints can receive a change-set**, and 100% of ~474,000 daily webhooks therefore require full-record reconciliation.

This is the documented mechanism behind the known Cova full-object-PUT clobber: a receiver that cannot tell what changed must write everything back, and a partial write erases what it omits.

Sampled production payloads

POS TYPE	EVENT_NAME	FIELDS	NULL
treez	member_updated	29	14%
leaflogix	member_updated	30	20%
greenline	member_updated	30	30%
lightspeed	member_updated	30	33%
cova	member_updated	32	38%
klaviyo	member_created	32	56%
pos (PUT)	**absent**	8	25%

Every sampled payload carried `current_milestone_tiers: {spend: null, point: null, visit: null}` — tier data is structurally present and empty in practice. None contained any change indicator, confirming the code reading empirically.

> **Incidental but serious: live POS credentials are written to CloudWatch in plaintext.** Sampled payloads embed `pos_token`, `api_key` and `access_token` values, including `Authorization: Bearer lsxs_pt_…`. This is a credential-exposure issue independent of the webhook question, and it compounds the plaintext token already noted in the POS request runner.

Ten declared events have no subscribers

Of 13 declared event types, only three have any subscriber: `member_updated` (816 endpoints), `member_created` (811) and `member_deleted` (**1**). The remaining ten — campaign, message template, template image and reward grant events — still execute their callbacks and a database existence check on every relevant save, for zero recipients. `visit_created` and `mailchimp_created` are declared and never emitted at all.

Endpoint hygiene

MERCHANT STATUS	ENDPOINTS	MERCHANTS
lost	**395**	364
active	389	361
suspended	25	24
pending	7	7

More endpoints belong to lost merchants than to active ones, and there is no merchant-status filter anywhere in the delivery path — `webhook_scope.for_event` selects purely on the subscribed event. 134 of those lost merchants still recorded **1,011,596 visits** in 30 days, so they are actively generating deliveries. There is also **no `active` column** on `webhook_endpoints`: an endpoint cannot be disabled, only deleted. And **no deliveries table exists** — there is no delivery audit trail in the database at all, so a partner's "we never received it" can only be checked against a 207 GB log group.

Four endpoints point at `production.api.sringbig.technology` — a misspelled domain that cannot resolve.

What to change

#	CHANGE	WHY
1	Pass `previous_changes` into the payload as a `changes` key	Addresses the confirmed half at its root; requires serializing at event time rather than delivery time
2	Include the triggering visit id on visit-driven `member_updated`	Lets a receiver distinguish a purchase from a profile edit
3	Add `event_name` to the pos PUT path	42.5% of volume currently arrives unlabelled
4	Filter delivery on merchant status	Half of all endpoints belong to departed accounts
5	Stop logging POS credentials to CloudWatch	Live tokens in plaintext
6	Add an `active` flag and a delivery-audit table	Endpoint health is currently unobservable
7	Prune the ten zero-subscriber event types	Callbacks and DB checks executing for no recipient

What could not be measured. 91 of the 816 endpoints are non-callback URLs that bypass this Lambda via `LambdaWebRequest::Async`, so their delivery success rate is unmeasured here. Because there is no `active` column and no delivery history, how many of the 816 endpoints are effectively dead also cannot be determined. And only leaflogix logs a parseable `memberId`, so the duplicate rate for other POS types is unknown.

PART III

Confidence and known limits

Every figure in this document is a measurement, and measurements have error bars. This section records where the numbers are firm, where they are estimates, and the specific ways this investigation got things wrong before catching them — because a reader deciding what to act on needs to know which is which.

Corrections made during the investigation

Seven working conclusions were wrong and were caught by a second check. They are listed because each one nearly reached this document as fact, and because they share a pattern worth naming.

- **A controller label was read as a page identity.** New Relic groups by controller action, so `/favicon.ico` — swallowed by a catch-all route — was counted as `enroll/members/edit`. A 99.6% failure rate looked like members locked out of their wallets. The nginx access log, which records the requested URL, showed 85.6% were browser icon requests and **zero** were member wallet pages.
- **A deploy was nearly blamed for it.** Those failures appeared to begin on 11 August, matching a commit that touched that controller. New Relic retains eight days; the apparent onset was the retention edge.
- **A malformed query returned a confident zero.** `http.statusCode` is numeric in this account, so `= '500'` matched nothing.
- **A live table was declared dead.** `invoice_items` was filtered on `transaction_date`, which is null table-wide; the live column is `created_at`.
- **An unpaid total was inflated 4.7×.** Organisation-level invoices are attributed to every child merchant, so seven sibling accounts each showed the same $450,783.
- **A YAML parser was pointed at JSON.** `versions.object` is JSON; the first pass returned null for every field across 9,771 rows and would have read as "no history".
- **An index was assumed absent.** `reward_grants` has a `merchant_id` index, so a workaround built to avoid a sequential scan was unnecessary.

> **The shape they share.** Every one is a proxy signal mistaken for the thing itself — a timestamp for a cause, an error count for customer impact, a null result for an absence, a controller name for a page. The reliable check never changes: ask whether the desired outcome occurred. Did the page render, did the visit land, did the build succeed, did the points get debited. This is the single most transferable finding in the document.

Figures that are firm

Counts drawn directly from a full population, with no sampling or extrapolation: the endpoint and route inventories; the code census (an AST walk over 2,350 files, zero parse failures); merchant, audience, webhook-endpoint and billing counts from Postgres; the Cloudflare zone sweep across all 1,799 zones; Rollbar active-item counts, fully paginated.

Figures that carry caveats

FIGURE	CAVEAT
Gateway rejections, ~918,000/day	Six-fold extrapolation from one four-hour afternoon window. Order-of-magnitude only; the four-hour counts themselves are solid.
Webhook volume, ~474,000/day	Twelve-fold extrapolation from a two-hour window.
Lambda error rates	ERROR lines per invocation, not a failure rate — batch handlers emit many lines per call, so ratios exceed 100%.
Rollbar resolved and unique counts	The API caps `total_count` at 1,000; these are floors. Rate-limit warnings mean the frontend occurrence total is also a floor.
Reward-grant and messaging volumes	Denormalised counters recorded at send time, not delivery-confirmed.
Cloudflare traffic totals	The account rollup undercounts the per-zone sweep by 40% for reasons not established. The per-zone sweep is used throughout.

What could not be measured

- **No GCP credential is reachable** from this environment, so the cloud-project estate is not measurable here.
- **Mobile crash telemetry does not exist** in any reachable source — not in Rollbar, not in the 410.6M-row mobile event table. The apps are instrumented for behaviour and not for failure.
- **Carrier-side 10DLC registration** is not in this database, so the 931 campaigns flagged not-submitted cannot be adjudicated from here.
- **Delivery success for 91 webhook endpoints** that bypass the delivery Lambda is unmeasured.
- **Two Rails-side structural claims** — 7 routes at missing controller classes, 28 unrouted controllers — were not re-checked and are carried from the earlier measurement.
- **Whether the member-identity rescues cause the duplicate-member errors** is a hypothesis, not a finding. It needs a targeted trace of individual records across successive writes.

On the second measurement. Several figures here differ from an internal reading taken on 4 August. Where both exist, the 16 August measurement supersedes it — in most cases because this pass used a stricter method (exact log anchors rather than substring matching, an AST walk rather than a text search, full enumeration rather than estimation). The earlier reading is useful as a second observation of a moving system, not as a set of conclusions requiring adjudication.

PART IV

Method

SURFACE	SOURCE	WINDOW
Web tier, APM, 5xx attribution	New Relic NRQL via GraphQL	7 days (retention limit: 8)
Background jobs, transaction-date, gateway rejections	CloudWatch Logs Insights	full days and 2–4h windows, stated per figure
Error reporting, symbolication	Rollbar API, 3 projects, fully paginated	all active items
POS health, merchants, audiences, billing	Production Postgres replica, read-only	30–90 days
Mobile telemetry	Redshift `native_app_events`	all time (410.6M rows)
Edge	Cloudflare GraphQL, all 1,799 zones	2026-08-09 → 08-15
API contract	`rails routes` + `apigateway get-resources`	repo at `3e50175`, 2026-08-13
Code census	Ruby `Ripper` AST over 2,350 files	same commit

Constraints that shape these numbers

- **New Relic retains eight days.** No onset earlier than 2026-08-08 can be established from it, and per-day `TIMESERIES` results proved unreliable in this account — discrete windowed counts were used instead and cross-checked against aggregates.
- **Non-production applications must be excluded.** 90% of raw 5xx volume is Beta-environment health checks that return 500 by design. Every figure here filters to `appName = 'springBIG'`.
- **Rollbar caps `total_count` at 1,000**, so resolved and unique counts are floors. Rate-limit warnings in the frontend project mean its occurrence total is also a floor.
- **Cloudflare's account rollup undercounts by 40%** against a full per-zone sweep, for reasons that could not be established. The per-zone sweep is used throughout, and no traffic delta between readings is published because the baselines may differ in derivation.
- **Firewall, bot-score and origin-latency data remain unavailable** on the Cloudflare Free plan — re-tested and still denied at both zone and account level.
- **No GCP credential is reachable** from this environment, so the cloud-project estate is not measurable here.
- **The published documentation was enumerated completely** via the readme.io site manifest and its embedded OpenAPI fragments — 48 pages, all HTTP 200. Two Rails-side structural claims (7 routes at missing controller classes, 28 unrouted controllers) were not re-checked and remain at their 4 August values.
- **Sunday effect.** 16 August was a Sunday. Where a figure could be distorted by weekend seasonality it was measured across full days and normalised against total log volume; the transaction-date rate is flat at ~19,500–20,600 errors per million log lines across the window.

Measured versus inferred. Every number in this report is a direct measurement unless labelled otherwise. Extrapolations state their window and multiplier. Where an earlier figure could not be reproduced it is marked not-reproducible rather than replaced with a guess, and where a surface could not be reached it is marked not-measured rather than omitted — because an unmeasured surface that looks absent is exactly how the findings in Part III went unseen the first time.