PROJECT RECORD · FOR BRIAN'S REVIEW

Salesforce Write Access

From read-only API today to a guarded, tested write path. The full record of what we decided, why, and the two questions that need your call.

Prepared for Brian Di · Authored by Anthony + Claude · 2026-05-30 · Pairs with Jason's two playbooks

THE ONE-MINUTE VERSION

1. **Goal:** let a small set of operators write to Salesforce (Purchase Orders and B2C Products) safely, with everything logged and a Claude job auditing the logs so no human has to self-monitor.
2. **V1 is a deliberate test on the existing shared `apireadonly` user**, not per-person licenses. Prove the write path and guardrails first, decide the identity model after.
3. **The one accepted deviation from your framework:** a shared user can't attribute writes natively, so the Google Sheet audit log becomes the system of record for "who," with the real human's email on every write (your Option B).
4. **We need two things from you (Q1, Q2):** MCP vs. direct-API scripts, and your sign-off on the Option B deviation for a bounded V1.

01 Objective

What Anthony actually wants out of this.

- A repeatable, safe way to grant Emazing operators **write** access to Salesforce. Today only read-only API access exists (built by Jason, used with Will).
- Built on **your Meta write-access playbook as the template** ("copy that format, improve as needed"), adapted to Salesforce.
- End state Anthony cares about most: **Claude is the auditor.** Every grant and every write is logged, and a scheduled Claude job reviews the logs and flags anything off, so "nothing falls through the cracks from humans having to monitor themselves."
- **Working model:** Anthony's Claude and Jason's Claude each built a playbook off yours, swapped them, and reviewed each other's in a Slack DM. This doc is the synthesis, sent to you for the architecture sign-off before anyone builds.

02 Where Salesforce is today

Four pieces exist and are verified. Going to write adds exactly one more (a permission set), not a new license.

PIECE	WHAT IT IS
Integration user	`apireadonly@emazinglights.com`. The API identity. On a Salesforce Integration license (API-only, no human seat, no extra cost). Read-only today.
Read-only profile	`Minimum Access – API Read Only`. Edit / Create / Delete unchecked. This is the current lock.
Read permission set	`read-only field access`. Field-level Read on Purchase Order (header + lines) and Product objects.
External Client App	`SFOAuthEG`. Issues the OAuth credentials the scripts use. Revoking it kills every token at once.

It is a **direct API integration, not an MCP.** The path to write (per Jason's status doc): add one Edit-enabled permission set, assign it to the integration user, test, promote. Same user, same app, same credentials. The read-only side is untouched.

---

03 The V1 decision: ship on the shared user as a test

Anthony's explicit call. Prove the write path before committing to an identity model.

Your framework says writes should be per-actor. The clean way to do that here is one named Integration user per person (we have 4 free, see Section 09). **Anthony chose not to commit to that yet.** V1 ships on the existing shared `apireadonly` user with a small named operator set, to prove the write path and the guardrails work on real data. If it runs well, we revisit per-person from there.

The V1 operator set (4-5 people sharing one API)

PERSON	ROLE
Maryanne Daclan	Assistant Merchandise Manager
Arvhiane Valerio	Assistant Buyer
Will	Director of Finance
Jason	Salesforce admin (owns the integration)

The workflow V1 automates: today, when a cost or unit changes on a PO, the lines are edited by hand one at a time (a 5-line PO is 5 manual edits). V1 lets Claude make those edits through the guarded write path. Anthony also confirmed the operators need to write **B2C Products**, so that object is in V1 scope too (see Section 04).

04 The write scope

Exactly what becomes writable, and the landmines we mapped.

OBJECT	API NAME	WRITE
Purchase Order	`serp__Purchase_Order__c`	Edit + Create
PO Line	`serp__Purchase_Order_Line__c`	Edit + Create
B2C Product	`ECS__Product__c`	Edit + Create

Delete stays off on every object. "Removal" is a status / flag update, never a `delete()` call. Note the product object is `ECS__Product__c` (B2C Product), not the standard `Product2`.

What was verified in Setup (2026-05-29)

- **Field History Tracking is ON** for Purchase Order, PO Line, and B2C Product, so the auto-auditor has history to read. One gap to close: on PO Line, Purchase Price and quantity are tracked but **Cost is not**. If we write Cost, we enable history on it first.
- **Validation rules: zero** on all three objects. Nothing to trip there.
- **Apex triggers fire on every write and are the real landmine:** `ManagePurchaseOrder` on PO (~12KB), `ManagePOLines` on PO Line (~7.5KB), and 5 triggers on B2C Product including `UpdateMagentoInventory`, which syncs Product changes to live Magento. We map all trigger side-effects in sandbox before any live write.
- **Setup Audit Trail** retains roughly 180 days of config changes (6-month export). That covers the access-change half of the audit (who was granted write, when).

> **THE SINGLE BIGGEST RISK TO FLAG**
>
> Because B2C Product is in scope, the `UpdateMagentoInventory` trigger means a Product write pushes downstream to **live storefront inventory**. A bad write propagates to the store in seconds. This is why Product is riskier than POs and why it needs the extra controls in Section 06.

05 Identity and the compensating control

The one place V1 deviates from your framework, and how we keep it honest. This is Q2 for you.

On the shared `apireadonly` user, Salesforce's native history stamps every change as "apireadonly," not the actual human. That does not meet the per-actor bar on its own. We accept it **for V1 only** with a named compensating control (your framework's Section 03, applied here):

- **The Google Sheet audit log is the system of record for "who."** Every write records the real human's email, passed at call time.
- **No write path may skip the Sheet.** If a write can't be logged, the failure is surfaced and a human logs it. Never a silent gap.
- **Alert on any audit-append failure**, so a missing "who" is caught immediately.

HONEST FRAMING

This is a conscious, signed-off deviation (Option B), the same logic you use for a shared service account where native audit is weak. Acceptable for a bounded V1 on a named operator set. The path back to per-actor identities stays open. **We want your explicit yes / no on this.**

---

06 Security analysis

Anthony asked directly: what are the security concerns? Three, plus one caveat. None are blockers; each has a named control baked into the build.

1. Credential custody is the whole boundary (highest)

With one shared user, the OAuth credentials *are* the security perimeter. Two consequences:

- **Out-of-band writes are invisible.** If anyone uses the creds directly (a raw API call outside the guarded path), Salesforce logs "apireadonly," no Sheet row is written, and attribution silently fails. Per-user identities would catch this natively; shared + app-layer audit does not.
- **Control:** operators never hold the raw credentials. The write path runs from **one host** the operators trigger through Claude. The secret lives in one place, not on five laptops. Revocation stays clean (kill `SFOAuthEG`).

2. Magento blast radius on Product writes (second)

A `ECS__Product__c` write fires `UpdateMagentoInventory` → live storefront. The dry-run preview shows the *Salesforce* before/after, not necessarily the Magento consequence.

- **Control:** (a) map the Magento trigger in sandbox before any live Product write, non-negotiable; (b) go **field-selective on Product** (open only the merchandising fields the operators need, not the whole object), specifically so inventory-sensitive fields aren't casually writable. Keep open-all on PO + PO Line (contained); go selective on Product (Magento).

3. The audit Sheet must be append-only (third)

That Sheet is now the system of record for "who." If operators can edit it, they can edit the record of their own actions and the control is theater.

- **Control:** the integration writes rows via a service account; operators get read-only or no access; only Anthony and Jason admin it.

KEEP THIS HONEST

The auto-auditor is **detective, not preventive**. It catches a bad write on its next run (every 2 hours), after Magento already synced. The preventive controls below carry the real weight. The auditor is the backstop, not the seatbelt.

The full guardrail set (added after Anthony's review)

Grouped by job. Preventive controls stop a bad write at the door; detective controls catch and reverse what slips through.

Stop a bad write before it happens:

- **One chokepoint.** Every write goes through one centralized tool that holds the credentials. No operator keeps raw creds on a laptop. If a write does not go through the tool, it cannot happen, and the tool always writes the log. This is what makes attribution trustworthy on a shared user.
- **Dry-run / show-before-you-do.** Every change previews exactly what it will do; nothing happens until an explicit confirm.
- **Value sanity bounds.** Reject absurd values before the write (negative price, a change beyond X%, a 100x jump). Catches fat-finger and hallucinated numbers, the most common bad-instruction failure.
- **Per-person rate caps.** A ceiling on writes per operator per hour. A runaway agent is auto-paused after the cap instead of editing hundreds of records.
- **Field allow-list in the tool.** The tool only permits the specific agreed fields, independent of (and tighter than) the Salesforce permission set. This is where Product is locked to merchandising fields and away from the Magento-sensitive ones.
- **Two-person rule for high-risk writes.** Anything above a dollar threshold, any record creation, or any Product write needs a second named person's approval before it executes.
- **Read-back verification.** After each write, the tool re-reads the record and confirms it matches intent, catching silent trigger side-effects (Magento, Apex) in seconds.

Catch and reverse what slips through:

- **The log is also the version history.** Every row stores the before value (captured before the write) and the after value. The same logbook that proves who did it is the snapshot that lets us undo it. Salesforce has no one-click restore, so "undo" means writing the old value back through the guarded path (which on a Product re-syncs Magento, as intended).
- **Bypass detection.** The auditor cross-checks our log against Salesforce's own Field History / Setup Audit Trail. A Salesforce change with no matching log row means someone bypassed the tool, and it alarms. This closes the shared-user attribution gap.
- **AI QC every 2 hours, and it acts.** On a serious flag (a delete, an off-scope write, an unapproved actor, or an unlogged change) it does not just alert. It auto-pulls the write permission set, freezing all writes until a human looks.
- **Team-wide freeze command.** Any operator can trigger an instant freeze of all writes if something looks wrong. A human circuit breaker on top of the AI one.

---

07 The five guardrails, as wired

Same five as your framework. Here is the V1 wiring.

1. **Dry-run by default.** Every write previews the exact records and fields (count + before/after) and changes nothing until a second explicit confirm.
2. **Threshold starts at 0%.** Because POs are financial, every write requires a second explicit override to start. Relax toward your 20% only if 0% proves too noisy.
3. **Audit Sheet, one row per write:** timestamp (PT), real human's email, object, record ID, action, before, after, API response, notes.
4. **Slack ping per write** to a dedicated channel where the operator and a reviewer both see before/after. Best-effort; a failed ping never rolls back a real change.
5. **Identity = shared `apireadonly` + the compensating control** from Section 05.

> **BELT AND SUSPENDERS**
>
> If the audit row fails to write, do NOT fake a rollback. Surface the failure so a human logs it. A visible gap beats an invented recovery.

08 The auto-auditor

A log nobody reads is not a control. A scheduled Claude job reads it for us. This is Anthony's signature requirement.

A scheduled job (launchd cron running `claude -p`, matching existing infra) runs **every 2 hours** over the new rows since its last run, pulling from the write log, Setup Audit Trail, and Field History. It flags: a write with no matching audit row (bypass), any threshold override and whether it looked justified, any delete (should never happen), a write outside the approved scope, a write by someone not on the approved-access list, off-hours or volume spikes, and audit-append failures. It posts a green "clean" check to Slack, or a flagged list with record links. On a serious flag it does not just alert, it **auto-pulls the write permission set** to freeze all writes until a human looks.

> LOAD-BEARING
>
> The auditor is only as good as the logs feeding it. That is why Field History on the written fields (including PO Line Cost) and Setup Audit Trail retention are not optional. **Not built yet** — described here as the target; building the scheduled job is fast-follow work.

09 Licenses (so the identity options are grounded)

Verified from Setup screenshots, 2026-05-29.

LICENSE TYPE	REALITY
`SF Integration (API-only)`	5 total, 1 used by `apireadonly`, so **4 free**, then $10/user/month. This is what write integration users would use.
`API Integration PSL`	5 available, 0 used. Expands an Integration user's object/API access (how we grant write scope).
`SF full (UI login)`	4 total, 1 free, effectively maxed. The costly per-seat licenses, not the API path.
`API call volume`	3.45M/month allowance, ~1% used. Not a constraint.

Why this matters: per-person identity (the framework-ideal) is **free for up to 4 operators**. Anthony still chose shared-for-V1 to keep the test small and prove the path first, not because per-person costs money. The door to per-person stays open at no license cost.

10 Testing

Two options, both on the table. Jason leans controlled-live; Anthony's earlier instinct was sandbox-first.

Option A — Controlled live test (Jason's lean): read + dry-run scope checks on production are 100% safe (no writes). Then real writes only on a single disposable test PO we create and own, all guardrails on. This is your Meta approach (real account, control record): faster, real data, no stale-sandbox friction.

Option B — Salesforce sandbox: stand up the write permission set in `emazing--uat` and run the full seven-test cycle there first. Fully isolated, zero risk to live records. Trade-off: sandbox data is stale and test-record setup is UI-heavy.

> THE INTERACTION THAT MATTERS
>
> Controlled-live is fine for a throwaway PO (contained). It is NOT safe for Product, because there is no throwaway Magento sync. So: use the sandbox once to watch the `ManagePurchaseOrder` / `ManagePOLines` / Magento triggers, then controlled-live for POs. Do not skip the threshold-breach test (test 6). Run the audit-integrity check before declaring go-live.

---

11 Build sequence and revocation

Build sequence (admin view)

1. Create a permission set `PO Write (Integration)`: check Edit + Create on PO + PO Line (and the selected Product fields). Leave Delete unchecked.
2. Enable Field History on the PO Line Cost field (and any written field not yet tracked).
3. Stand up the audit Google Sheet (append-only) + the Slack channel for write pings.
4. Wire the guarded write path (dry-run / confirm, 0% threshold, audit-Sheet write with the human email, Slack ping). REST sObject calls.
5. Run the seven smoke tests per Section 10.
6. Assign the permission set to `apireadonly`, add a rollout decision-log row (date, operators, objects/fields in scope, reason), go live for the V1 operator set.

Revocation (the kill switch)

1. **Stop writes now:** remove the `PO Write (Integration)` assignment. All write capability dies instantly; read access stays.
2. **Kill all API access:** revoke / disable `SFOAuthEG`. Every OAuth token dies at once.
3. **Remove one operator** (once per-person): freeze that integration user; others unaffected.
4. **Confirm:** any credentials on a laptop are now inert. The lock lived in Salesforce, not the token.

---

12 Jason's answers (the SF facts, verified)

All ten submitted through the playbook page and the Feedback sheet. Condensed.

Q1 — Licenses

4 free Integration licenses (5 total, 1 used), $10/user/mo beyond. API Integration PSL: 5 free. Per-person (Option A) free for up to 4 operators.

Q2 — Write scope

PO + PO Line, Edit + Create, Delete off. Product read-only at first, later expanded to B2C Product writes (`ECS__Product__c` , not Product2) per Anthony's call.

Q3 — Sandbox

Yes: `emazing--uat` (https://emazing--uat.sandbox.my.salesforce.com). Creds out-of-band.

Q4 — History / retention

Field History ON for PO, PO Line, B2C Product. Gap: PO Line Cost not tracked (enable before writing it). Setup Audit Trail ~180 days.

Q5 — Landmines

Zero validation rules. Triggers DO fire: `ManagePurchaseOrder` , `ManagePOLines` , and 5 Product triggers incl. `UpdateMagentoInventory` (Magento sync). REST sObject. Required fields / flows / approvals confirmed in sandbox.

Q6 — Threshold

Start 0% (every PO write needs the override), relax to 20% later if too noisy.

Q7 — Permission precedence

An additive Edit permission set grants write over the read-only profile (most-permissive wins). Proven in the org, re-checked in sandbox.

Q8 / Q9 — for you (Jason's lean)

Scripts first, move to MCP after the seven tests pass; `preview_then_execute()` ports either way. No-DELETE is a hard rule for a system of record. Option B acceptable only with a hardened app-layer audit.

---

13 What we need from you

Two questions. Answer them with your Claude and send back; the high-level HTML has inline forms, or just reply.

Q1 BRIAN **Integration shape**

Build a Salesforce MCP with the dry-run tools, or keep the direct-API scripts and bolt the guardrails on? Does your `preview_then_execute()` helper port cleanly? Our lean: scripts a human runs to start, MCP after the seven tests pass.

Q2 BRIAN **Sign off Option B**

Is the shared `apireadonly` + hardened app-layer audit (Option B) acceptable for a bounded V1, or a hard no? And is there anything in the five-guardrail model you would change for a system-of-record database (Salesforce) versus an ad platform (Meta), especially around the shared-user audit deviation?

---

14 Current state

- Anthony's playbook built and live; Jason's two playbooks (framework + build-out) built and shared. Both off your Meta template.
- Jason answered all 10 SF-fact questions; Anthony and Jason are aligned on V1-as-a-test on the shared user, with B2C Product in scope.
- **Open:** your Q1 / Q2. Once you weigh in, we stand up the test (permission set → sandbox trigger-mapping → controlled-live on a throwaway PO → go-live for the operator set).
