Quarterly reconciliation pipeline for ENT practice data

TL;DR — the five decisions that matter

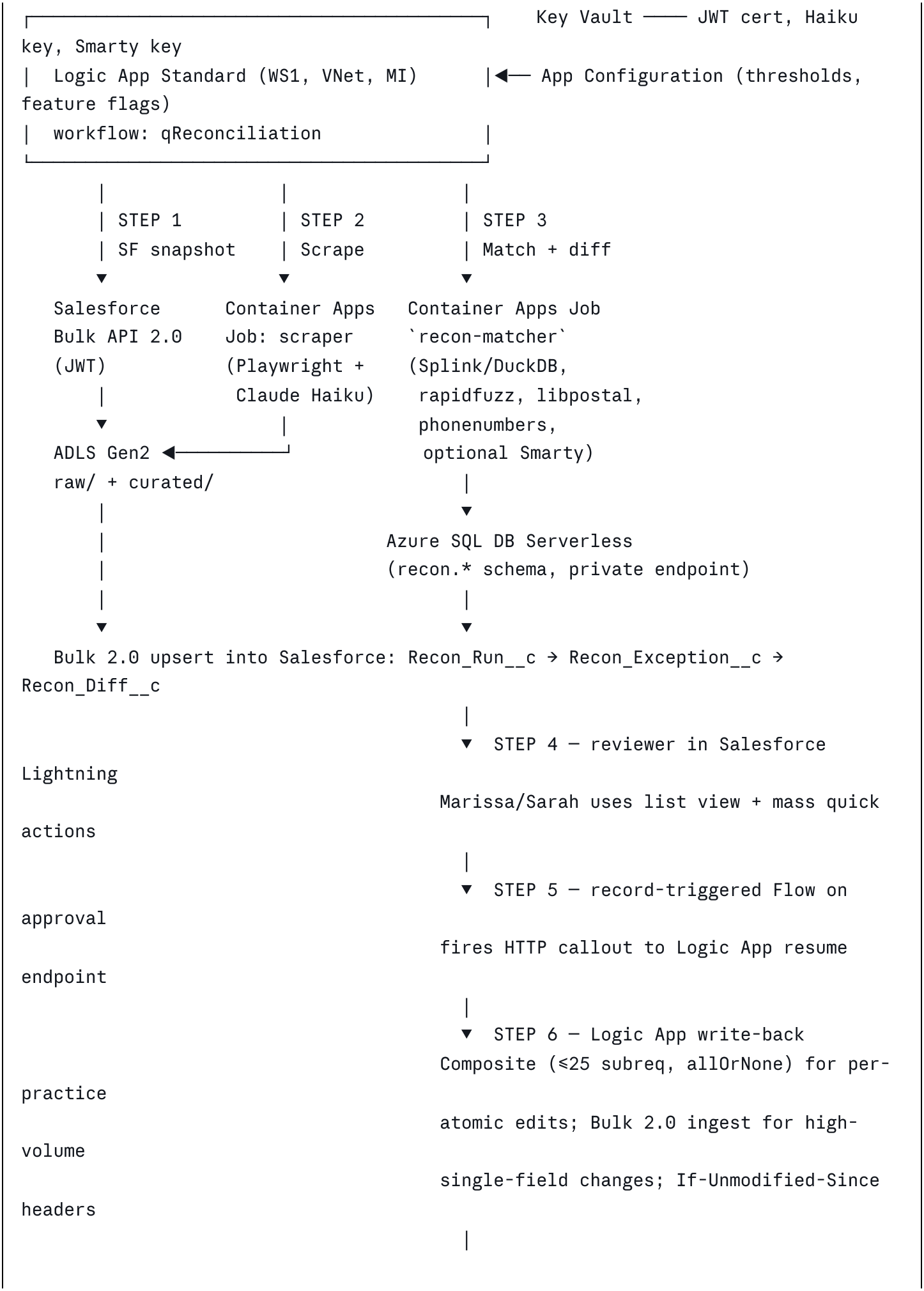
1. **Compute = Logic App Standard (orchestrator) + Container Apps Jobs (Python worker).** Re-use the existing WS1 plan as the durable, MI-authenticated orchestrator; offload Playwright + Claude Haiku + Splink/rapidfuzz to a containerised Job that scales to zero between quarters. Python ML work is wrong-shaped for Logic App inline code, and Container Apps Jobs' monthly free grant covers four ~30–90 min batches per year.
2. **Extract via Bulk API 2.0 Query — one job per object, not parent-child SOQL.** Bulk 2.0 query *does not support subqueries or aggregates*, so run five parallel jobs (Account, Contact, Practice__c, Prescriber__c, Location__c), land raw CSV in ADLS Gen2, then convert to Parquet for the matcher. Auth with **OAuth 2.0 JWT Bearer Flow** against a cert in Key Vault — no shared secrets, no IP allowlist needed.
3. **Match with Splink (DuckDB backend) for the fuzzy path; NPI-with-Luhn-and-NPPES as the deterministic anchor.** Probabilistic Fellegi-Sunter gives calibrated, explainable scores at 1,200×1,200 trivially. NPI is the anchor when present *and cross-validated* against NPPES name/taxonomy/ZIP; otherwise fall back to a weighted name+address+phone score with auto/review/reject thresholds at 0.95/0.50/0.50.
4. **Human review = Salesforce custom objects (Recon_Run__c → Recon_Exception__c → Recon_Diff__c), not a custom Azure web app.** The non-technical reviewer already lives in Salesforce, the volume (30–100 exceptions/quarter) is list-view-sized, and Option A's $160–$520/yr + 40–80 hr build cannot beat $0 + ~12 hr of clicks. Native list-view mass quick actions, inline edit, audit, and approval emails come for free.
5. **Write-back via Composite (atomic per-practice) + Bulk 2.0 (high-volume per-field), keyed on a dedicated `Idempotency_Key__c` external id and gated by `If-Unmodified-Since` against the snapshot's `SystemModstamp`.** Build a custom `Reconciliation_Audit__c` object (not Field History Tracking) for forensic before/after with source attribution.

Total incremental Azure cost: ~$240–$340/year riding existing infra; ~$580–$1,005/year fully dedicated. Both fit the $1K budget — annual, not one-time.

---

1. Architecture (described)

```
[Quarterly timer, 02:00 on 1st of Jan/Apr/Jul/Oct]
                  |
                  ▼
```


Key Vault ——— JWT cert, Haiku key, Smarty key
Logic App Standard (WS1, VNet, MI)
workflow: qReconciliation
App Configuration (thresholds, feature flags)
STEP 1
SF snapshot
STEP 2
Scrape
STEP 3
Match + diff
Salesforce Bulk API 2.0 (JWT)
Container Apps Job: scraper (Playwright + Claude Haiku)
Container Apps Job `recon-matcher` (Splink/DuckDB, rapidfuzz, libpostal, phonenumbers, optional Smarty)
ADLS Gen2 raw/ + curated/
Azure SQL DB Serverless (recon.* schema, private endpoint)
Bulk 2.0 upsert into Salesforce: Recon_Run__c → Recon_Exception__c → Recon_Diff__c
STEP 4 — reviewer in Salesforce Lightning
Marissa/Sarah uses list view + mass quick actions
STEP 5 — record-triggered Flow on approval
fires HTTP callout to Logic App resume endpoint
STEP 6 — Logic App write-back
Composite (≤25 subreq, allOrNone) for per-practice
atomic edits; Bulk 2.0 ingest for high-volume
single-field changes; If-Unmodified-Since headers

Why this shape. The orchestrator owns durable state, retries, and approvals; the Python container owns CPU/IO-heavy work that needs arbitrary libs. Salesforce *itself* is the review UX, so we close the loop with a webhook back into the Logic App rather than building a parallel app. Everything traverses the existing VNet via Managed Identity and private endpoints.

---

2. Pulling Salesforce data

API choice: Bulk API 2.0 Query, one job per object. At 1,200 parents plus several thousand related rows, REST `/query` would work but Bulk 2.0 is async, returns CSV ready for ADLS, and is future-proof beyond the current scale. **Crucially, Bulk 2.0 query does not support parent-child subqueries or aggregates** — so the correct pattern is five parallel jobs (Account, Contact, Practice__c, Prescriber__c, Location__c) and a `JOIN` in Python. CDC and Platform Events are wrong tools because reconciliation needs a *full point-in-time snapshot*, not a delta stream with 72-hour retention.

Lifecycle: `POST /services/data/v61.0/jobs/query` → poll `GET .../jobs/query/{id}` until `JobComplete` → page results with `Sforce-Locator` header until `null`. No PK chunking knobs — v2 handles internal batches transparently (10K-row chunks). Limits worth knowing: 15 GB per query job, 25 concurrent jobs org-wide, 7-day result retention. Apex Hours + 2

Snapshot cadence: quarterly aligned to scrape, with a manual on-demand trigger for post-cleanup re-runs. Monthly is wasted work because the scraped reference doesn't change between quarters.

Landing: ADLS Gen2, raw CSV in `raw/year=YYYY/quarter=Qx/object=Account/...`, converted to Parquet in `curated/` for the matcher. Parquet gives typed schemas, ~5–10× compression, and direct Power BI/Synapse consumption later; CSV alone forces every downstream consumer to re-infer types.

Auth: OAuth 2.0 JWT Bearer Flow. Generate an RS256 keypair, store the private key as a Key Vault certificate, upload the public cert to a Connected App with "Use digital signatures" enabled, pre-authorise a dedicated *Integration User License* via permission set. The Logic App pulls the cert via MI, signs a short-lived JWT, exchanges it at `/services/oauth2/token`. JWT beats Client Credentials Flow because a leaked Key Vault cert is materially harder than a leaked `client_secret` in a log, and rotation is automated by Key Vault. Username/password is non-negotiable: deprecated and incompatible with MFA-required orgs.

NPI deterministic anchor

NPIs are 10-digit numeric identifiers validated by the **ISO/IEC 7812 Luhn check digit with constant prefix "80840"** (the issuer identifier — "80" health, "840" US). The check digit algorithm CMS publishes is: Eclaims

```python
import re

def npi_is_valid(npi: str) -> bool:
    digits = re.sub(r'\D', '', npi)
    if len(digits) != 10 or digits[0] not in ('1', '2'):
        return False
    body, check = digits[:9], int(digits[9])
    total = 0
    for i, ch in enumerate(reversed(body)):
        d = int(ch)
        if i % 2 == 0:           # double the rightmost digit of the body first
            d *= 2
            if d > 9: d -= 9
        total += d
    total += 24                  # constant Luhn contribution of the "80840" prefix
    return ((10 - (total % 10)) % 10) == check
```

After format validation, **call NPPES** (`https://npiregistry.cms.hhs.gov/api/?version=2.1&number={npi}`, free, no auth, ~1 req/sec polite ceiling). Confirm `status == 'A'`, `enumeration_type` matches the expected scope (Type 1 prescriber vs Type 2 practice), and that the scraped name/ZIP overlaps the NPPES record. If NPPES says Dermatology in California but the scrape says Atlanta ENT, **the NPI is a hallucination** — quarantine, do not link.

Fuzzy fallback

Use Splink with the DuckDB backend as the primary library. Probabilistic Fellegi-Sunter gives EM-trained `m`/`u` weights per comparison field, exposes a match-weight waterfall per pair (perfect for the 0.50–0.95 review band UX), and handles blocking declaratively. At ~1.44M candidate pairs DuckDB runs in seconds on a laptop-sized container. `rapidfuzz` lives inside Splink ComparisonLevels for Jaro-Winkler / token_sort_ratio on practice names, and as a standalone utility for ad-hoc work. PyPI

Address standardisation: Smarty US Address Verification + libpostal preprocessing. Smarty is USPS CASS-certified, delivers canonical ZIP+4 (your strongest single non-NPI signal), and at

~4,800 lookups/year you can fit inside the free 250/month tier or buy the $20/month entry plan for one cycle (~$20–$80/year all-in). libpostal locally parses messy scraped strings into components *before* Smarty validation, improving match rate and avoiding wasted lookups. Lob and Melissa are equivalent vendors at this volume; libpostal alone doesn't give you CASS validation or DPV, so it's a complement, not a substitute.

Composite scoring (NPI-absent path)

```
S = 0.40·name_sim + 0.35·address_sim + 0.15·phone_sim + 0.10·city_state_sim
auto-accept ≥ 0.90 ; review 0.70–0.90 ; reject < 0.70
```

…or, using Splink's calibrated match probability: **auto ≥ 0.95, review 0.50–0.95, reject < 0.50**. Persist the top-3 candidates per scraped record so reviewers can pick when Splink is ambivalent.

Full matching pseudocode

python

```
def match(scraped, sf_records):
    s = normalize(scraped)   # libpostal + Smarty + phonenumbers E.164 + name canoni

    # --- NPI anchor path ---
    if s.npi and npi_is_valid(s.npi):
        nppes = nppes_lookup(s.npi)                 # cached
        if nppes and nppes['basic']['status'] == 'A':
            if name_sim(s, nppes) >= 0.80 and zip_overlap(s, nppes):
                sf_hit = sf_records.find_by_npi(s.npi)
                return Decision('AUTO_LINK_NPI', sf_hit, conf=1.0) if sf_hit \
                       else Decision('CREATE_FROM_NPPES', nppes, conf=1.0)
            return Decision('REVIEW_NPI_MISMATCH', nppes, conf=0.5)
    elif s.npi:
        flag_hallucination('invalid_npi_checkdigit', s)

    # --- Fuzzy fallback (Splink) ---
    cands = splink_score(s, sf_records, blocking=[
        block_on('state'),
        block_on('substr(zip5,1,3)'),
        block_on('first_name','last_name'),
        block_on('org_name_token1')])
    best = max(cands, key=lambda c: c.match_probability)
    if best.match_probability >= 0.95: return Decision('AUTO_LINK_FUZZY', best.sf, b
    if best.match_probability >= 0.50: return Decision('REVIEW', best.sf, best.match
    return Decision('CREATE_NEW', None, best.match_probability)
```

Multi-location and Type 1 vs Type 2

A 5-office ENT group typically has **one Type 2 NPI** (the legal entity) with primary + secondary practice locations exposed via the NPPES `practiceLocations` field and the *Practice Location Reference File* in CMS's monthly bulk dissemination, plus **N Type 1 NPIs** for prescribers. Salesforce models this as Account (Type 2 NPI) → Location__c children (ZIP+4 anchored) → Contacts (Type 1 NPI) with a junction object because providers rotate. Resolve Account first, then sub-match locations by ZIP+4, then attach prescribers. Never use a single phone number alone to identify a location — multi-location practices share switchboards.

Hallucination detection

A wrong-page scrape (same-name medical group, different city) is the most common LLM failure mode. The check is brutal and effective:

```
if jaro_winkler(scraped_name, sf_name) < 0.70
   AND address_components_differ(scraped_addr, sf_addr) in {street, city}:
       flag = "WRONG_PAGE_LIKELY"  → quarantine, never auto-apply
```

Have Claude Haiku return per-field `{value, confidence, evidence_span}` and drop fields below 0.6 confidence. Sample ~10% of HIGH-severity records for an LLM self-check ("given this HTML, is the phone X confirmed/refuted/unclear?"). Cross-validate NPI taxonomy against the expected ENT codes (`207Y00000X` and subspecialties; `231H*` audiology; `207K*` allergy). If a page lists >50 prescribers, the scraper hit a directory page — re-scrape.

4. Diffing and flagging

Normalize before compare, then diff only when `before_normalized != after_normalized`. This kills ~90% of cosmetic noise. Phones to E.164 via `phonenumbers`, addresses to USPS components via Smarty (compare per-component, not as a flattened string), names with credentials/honorifics stripped, URLs lowercased and de-tracked, hours parsed to structured JSON. Store both raw `before_value` and `before_normalized` — diff on normalized, display raw.

Severity rules (encoded as a YAML config in App Configuration):

Severity	Fields and triggers
HIGH	NPI change or mismatch, NPPES deactivation, street/ZIP change, practice closure detected, prescriber added/removed, any record with `hallucination_flag != None`
MEDIUM	Main phone, fax, business hours, taxonomy/specialty, suite/unit change
LOW	URL casing/scheme/trailing slash, name capitalisation/punctuation only, description whitespace, social handle query params

Do not auto-apply LOW in Q1, even though it's tempting. Run two quarters in shadow mode first, then enable auto-apply on a *narrow whitelist* (http→https, trailing slash removal, whitespace collapse) once the approval rate is >98%. The cost of one silent hallucination applied to 500 records is far higher than 80 extra checkboxes per quarter.

Diff schema

sql

```
CREATE TABLE recon.diff_queue (
  diff_id           BIGINT IDENTITY PRIMARY KEY,
  snapshot_date     DATE NOT NULL,
  match_id          BIGINT NOT NULL REFERENCES recon.match_results(match_id),
  sf_account_id     CHAR(18) NOT NULL,
  sf_object         NVARCHAR(40) NOT NULL,
  field_name        NVARCHAR(80) NOT NULL,
  before_value      NVARCHAR(MAX),  after_value     NVARCHAR(MAX),
  before_normalized NVARCHAR(MAX),  after_normalized NVARCHAR(MAX),
  change_kind       NVARCHAR(20),                        -- add|remove|modify|cosmet
  severity          TINYINT NOT NULL,                    -- 1 low … 3 high
  classifier_source NVARCHAR(10) NOT NULL,               -- rule|llm|manual
  confidence        DECIMAL(4,3),
  hallucination_flag NVARCHAR(60),
  evidence_url      NVARCHAR(1000),
  evidence_span     NVARCHAR(MAX),
  status            NVARCHAR(16) NOT NULL DEFAULT 'pending',
  reviewer_upn      NVARCHAR(128), reviewed_at DATETIME2(3),
  applied_at        DATETIME2(3),  sf_write_result NVARCHAR(MAX),
  created_at        DATETIME2(3) NOT NULL DEFAULT SYSUTCDATETIME()
);
CREATE INDEX ix_diff_status_sev  ON recon.diff_queue(status, severity DESC, created_
CREATE INDEX ix_diff_sfid_field  ON recon.diff_queue(sf_account_id, field_name);
```

5. Human review UX — pick Option B

	A: Azure web app	B: Salesforce custom objects	C: Email digest
Direct infra cost	$160–$520/yr	**$0**	$0
Kyle build hours	40–80	**~12**	4–8
Reviewer friction	Medium (context switch)	**Low (already in SF)**	Low at first, awful by item 30
Bulk actions / triage	Build it	**Native list views + mass quick actions**	Painful past 20
Audit trail	Build it	**Native (Field History, audit fields)**	Build it; weak
Partial-review resume	Build it	**Native (status filter)**	None

	A: Azure web app	B: Salesforce custom objects	C: Email digest
Fit for 30–100 items/qtr	Overkill	**Right-sized**	Breaks
Engineering risk for solo eng	High	**Low**	Low

Pick: Option B. The reviewer ergonomics dominate every other consideration. Marissa Hartman and Sarah Lee live in Salesforce all day — asking them to learn a new web app for a *quarterly* task is an adoption failure waiting to happen. At 30–100 exceptions, native list views with inline edit and mass quick actions (GA Spring '18) handle the volume trivially, Field History Tracking gives HIPAA-defensible audit for free, and Approval Processes / Flow Orchestrator come with email-with-approve-link as a built-in fallback — meaning you get Option C for free as the notification layer on top of Option B.

Option A would only win if the reviewer pool grows to 5+ non-SF-licensed users, if you need sophisticated interactive DOM diff with before/after screenshots and hover-zoom, if diff volume balloons past ~1,000/quarter, or if the pipeline expands beyond Salesforce to other MDM systems.

The Salesforce data model and UX

```
Recon_Run__c           one per quarterly batch (RUN-2026-Q1)
  ├─ roll-ups: Total_Diffs, High, Medium, Low
  └─ Status: Open | In Review | Complete

Recon_Exception__c     one per Salesforce record with ≥1 diff
  ├─ Master-Detail → Recon_Run__c
  ├─ Lookup → Account
  ├─ Max_Severity, Diff_Count (roll-up), Hallucination_Flag
  ├─ Evidence_URL, Status, Reviewer, Reviewed_At
  └─ Lightning Record Page with:
       • red banner LWC when Hallucination_Flag != None
       • left: Recon_Diff__c LWC table (inline-editable Decision picklist)
       • right: scraped-page screenshot (File) + live URL link
       • quick actions: Approve All / Approve LOW Only / Reject All / Mark Wrong
Page

Recon_Diff__c          one per field-level diff
  ├─ Master-Detail → Recon_Exception__c
  ├─ Field_Name, Before_Value, After_Value, Severity, Confidence
  └─ Decision (Pending|Approve|Reject), Apply_Result
```

Reviewer journey, target <60 min for 80 exceptions: quarterly approval email lands → click into `My Pending Exceptions This Quarter` list view sorted by Max Severity desc → cherry-pick HIGH items with side-by-side diff and evidence link → use `Bulk Approve LOW` mass quick action to clear cosmetic items → forward hallucination-flagged exceptions to Kyle via Chatter `@mention`. A record-triggered Flow on `Status → Approved` calls back to a Logic App resume endpoint to fire the write-back. HIC GLOBAL SOLUTIONS LinkedIn

The one real cost of Option B is pushing diffs *into* Salesforce. At 80 exceptions × ~3 diffs ≈ 240 Diff rows + 80 Exception rows per quarter, one Bulk API 2.0 insert job handles it trivially.

6. Write-back to Salesforce

Hybrid API strategy

Composite (`/composite`, allOrNone=true, ≤25 subrequests) for atomic multi-record commits within a single practice — when an approved address change must touch Practice + 2 child Prescribers + 1 Location together. Use `referenceId` chaining (`@{practice.id}`) to wire children to a newly-created parent in one call.

Bulk API 2.0 ingest (upsert) for high-volume, per-record-independent updates — e.g., 800 approved Prescriber phone changes across the org. Keyed on the External_Id, naturally idempotent. Per-record success/failure is fine here; you don't need cross-record atomicity.

Standard REST `PATCH /sobjects/{Object}/{ExternalIdField}/{value}` for one-offs. Append `?updateOnly=true` (Summer '24+) to prevent accidental record *creation* — vital for a reconciliation pipeline that should never invent phantom Prescribers.

Apex REST endpoint only when justified — i.e., cross-object atomicity beyond Composite's 25 limit, or guaranteed same-transaction audit writes. Start with the standard endpoints; introduce `/services/apexrest/recon/v1/applyChange` only when you hit a concrete wall.

External_Id pattern

Object	External_Id field	Why
Prescriber (Contact or custom)	`NPI__c` Text(10), External Id, Unique	CMS guarantees global uniqueness for individuals
Practice (Account or custom)	`Reconciliation_Key__c` Text(64) = SHA-256 of `(group_NPI ‖ tax_id ‖ normalized_address)`	Many practices have no NPI; need a synthetic stable key

Object	External_Id field	Why
Location__c	`Reconciliation_Key__c` = `practice_key + zip4_hash`	Multiple locations per practice

Audit object — `Reconciliation_Audit__c` (not Field History Tracking)

Field History Tracking caps at 20 fields per object, 18 months retention (UI) / 24 months (API), no source attribution, no structured before/after for long text. For NPI/prescriber data that's a regulatory weakness. Build the custom object with: `Run_Id`, `Snapshot_Id`, `Target_Object`, `Target_Record_Id`, `External_Key`, `Field_API_Name`, `Old_Value`, `New_Value`, `Source_System` (Scrape:NPPES / Manual / Salesforce), `Change_Type`, `Approval_Status`, `Approved_By`, `Approved_At`, `Applied_At`, `Apply_Status` (Success/Failed/Skipped-Conflict/Rolled-Back), `Apply_Error`, `Snapshot_SystemModstamp`, `Pipeline_Version`, and crucially `Idempotency_Key` **(Text(64), External Id, Unique) = SHA-256(run_id + target_record_id + field + new_value)`** so retries are inherently safe. Gearset CRS Info Solutions

Idempotency and optimistic concurrency

Every write-back call: (1) upsert the audit row by `Idempotency_Key__c`; (2) if `Apply_Status == 'Success'`, skip the PATCH entirely (replay-safe); (3) otherwise PATCH with `If-Unmodified-Since: <snapshot SystemModstamp, RFC 1123 GMT>`; on `412 Precondition Failed`, mark audit `Skipped-Conflict` and surface for re-review (a Salesforce admin edited the record between snapshot and write-back). Persist an append-only transaction log in ADLS (`recon_transactions/run_id=.../` Parquet) so you survive any Salesforce-side audit purge. Salesforce

Rollback strategy by failure scale

1. **Single practice (≤25 subrequests):** Composite `allOrNone=true` — Salesforce rolls back server-side. Free, cheapest, use first.
2. **Cross-practice batch (Bulk 2.0):** no native rollback; compensating transactions. For each failure, mark audit `Failed`; for each success in a logically-grouped batch where peers failed, optionally issue reverse PATCH using `Old_Value__c` from the audit row.
3. **High-stakes two-phase pattern:** Bulk 2.0 upsert into a staging custom object `Reconciliation_Stage__c` (never touches real records); a scheduled Apex / invocable Flow promotes staged rows to production in chunks of 200 with `allOrNone=true` per chunk; failures stay in staging for human review.

Sample Composite payload (atomic per-practice)

```
json
```

```
POST /services/data/v61.0/composite
{
  "allOrNone": true,
  "compositeRequest": [
    {"method":"PATCH","referenceId":"practice",
     "url":"/services/data/v61.0/sobjects/Practice__c/Reconciliation_Key__c/a1b2c3d4
     "body":{"Phone__c":"+15125550199","Billing_Street__c":"100 Oak Ridge Pkwy",
             "Expected_LastModifiedDate__c":"2026-05-01T14:22:11.000Z"}},
    {"method":"PATCH","referenceId":"presc1",
     "url":"/services/data/v61.0/sobjects/Prescriber__c/NPI__c/1234567890",
     "body":{"Phone__c":"+15125550199","Practice__c":"@{practice.id}"}},
    {"method":"POST","referenceId":"audit",
     "url":"/services/data/v61.0/sobjects/Reconciliation_Audit__c",
     "body":{"Run_Id__c":"f3d2-9e8a","Target_Object__c":"Practice__c",
             "Field_API_Name__c":"Phone__c","Old_Value__c":"+15125550100",
             "New_Value__c":"+15125550199","Source_System__c":"Scrape:NPPES",
             "Apply_Status__c":"Success","Idempotency_Key__c":"sha256:abcd..."}}
  ]
}
```

7. Azure compute, storage, and cost

Compute — why hybrid

	Logic App Standard WS1	Functions Premium EP1	Container Apps Job
Pricing	~$150–$185/mo plan DEV Community GitHub	~$146–$180/mo base DEV Community	$0.000024/vCPU-s, $0.000003/GiB-s, **free grant 180K vCPU-s + 360K GiB-s + 2M req/mo per sub**; Azure Docs Microsoft Learn scales to zero Azure Docs Microsoft Azure
Python ML stack	Poor (Inline Code limited)	Native, ≤1.5 GB image	**Excellent — any container**
Playwright + Chromium	Not viable	Heavy on cold-start	**Canonical Azure home**
VNet	Built-in	Premium-only	Built-in (workload-profile env, /27)

	Logic App Standard WS1	Functions Premium EP1	Container Apps Job
Long-running batch	Good (durable history)	EP1 host-bound, restart risk	**Best — Jobs run to completion**
Fit for user's pattern	Existing	New, same price as WS1	New, dramatically cheaper

Logic App Standard orchestrates (timer trigger, durable history, Salesforce connector glue, approval webhook receiver). **Container Apps Jobs do the Python work** (`recon-scraper` Playwright+Haiku; `recon-matcher` Splink+rapidfuzz+phonenumbers+Smarty). Four ~30–90 min runs/year stay inside the free grant. Functions Premium EP1 has no scenario where it wins here — it costs the same as WS1 without being the user's existing pattern.

Storage — Azure SQL DB Serverless + ADLS Gen2

Dedicated SQL MI would single-handedly blow the budget ($700+/mo). Sharing the existing SQL MI is free but couples lifecycles. The pragmatic middle is **Azure SQL DB Serverless (GP Gen5, 0.5–2 vCore, auto-pause 15 min)**: paused 99% of the time, ~$5–$30/month realistic, same T-SQL and Entra/MI auth as MI, private endpoint, 30–60s resume latency (fine for a quarterly batch and a reviewer who's already in Salesforce). Pair with **ADLS Gen2** hot for raw scrape HTML/PDFs and canonical Parquet snapshots — at ~$0.018/GB-month, years of snapshots cost pennies. OneUptime Oreate AI

Schema — six tables

`recon.salesforce_snapshot` (partitioned by `snapshot_date`, indexes on `(snapshot_date, sf_account_id)`, `(snapshot_date, website)`, `(snapshot_date, state, city)`), `recon.scraped_raw` (partitioned by `snapshot_date`, captures Haiku prompt hash + token counts for reproducibility), `recon.match_results` (top-N candidates per scrape with `signals_json`, `match_method`, `confidence`), `recon.diff_queue` (above), `recon.approval_log` (append-only, reviewer UPN + decision + note + session), `recon.write_back_log` (request body, HTTP status, response body, attempt count, retry chain via `retry_of_wb_id`).

ADLS layout:

```
recon/
  salesforce_snapshot/year=YYYY/quarter=Qx/{accounts,contacts,...}.parquet
  scraped_raw/year=YYYY/quarter=Qx/seed=ENT-
NNNN/{page.html,page.pdf,extracted.json}
  matching_artifacts/year=YYYY/quarter=Qx/{match_signals,diffs}.parquet
  exports/year=YYYY/quarter=Qx/review_report.xlsx
```

Lifecycle: snapshots → Cool @ 90d, Archive @ 365d, delete @ 7yr. Raw scrapes → Cool @ 30d, Archive @ 180d, delete @ 2yr.

Security

Key Vault for JWT cert + Haiku key + Smarty key (existing KV, prefix new secrets `recon-`). Managed Identity → SQL via `CREATE USER FROM EXTERNAL PROVIDER` + `db_datareader/db_datawriter`. MI → KV via `Key Vault Secrets User` scoped per-secret. MI → ADLS via `Storage Blob Data Contributor` scoped to the `recon` container. App Configuration holds non-secret feature flags (`scrape_concurrency`, `match_confidence_auto_accept_threshold`, `enabled_field_writebacks[]`, `severity_rules.yaml` reference) with sentinel-based refresh. Private endpoints on SQL, Storage, Key Vault, App Config follow the user's existing pattern. Append-only enforcement on `approval_log` and `write_back_log` via permissions (deny UPDATE/DELETE to MI).

Cost — both variants under $1K

Variant A — max-shared (recommended start):

Line	Annual
Logic App Standard WS1 (shared with existing)	$0
Logic App connector calls (~5K/qtr × $0.000125)	<$3
Container Apps Jobs (inside free grant)	$0
ADLS Gen2 hot (~1 GB + txns)	~$2
Shared SQL MI schema	$0
Key Vault, App Config (existing)	<$1
1 new private endpoint (if needed)	~$88
Claude Haiku API (1,200 × 6K in + 1K out tokens × 4 runs)	~$60
Smarty (free tier or smallest paid)	$0–$80
Buffer	$60
Total Variant A	**~$240–$340/yr**

Variant B — dedicated (no shared infra):

Line	Annual
Container Apps Jobs only (orchestrate via KEDA cron)	~$0
Azure SQL DB Serverless GP, auto-pause	$120
SQL storage	$12
ADLS Gen2 hot	$2
Key Vault	$1
App Configuration (Free tier; Standard adds $432)	$0
Private endpoints × 4 (SQL, Storage, KV, App Config)	$350
Claude Haiku	$60
Smarty	$24
Log Analytics	$3
Total Variant B (Free App Config + 4 PEs)	**~$580/yr**
Total Variant B (Standard App Config + 4 PEs)	~$1,005/yr

The largest single line in either variant is private endpoints (~$88/yr each). If existing VNet policy mandates PEs everywhere, that's a sunk cost; otherwise use Service Endpoints for KV and App Config to save ~$175/yr. Pump

8. Phased build plan

Phase 1 — MVP (2–3 weeks, ~$15/mo incremental). Add `qReconciliation` workflow to the existing Logic App Standard. Add `recon` schema to the existing SQL MI logical DB. Stand up one combined `recon-scraper-matcher` Container Apps Job with user-assigned MI. JWT-bearer auth to Salesforce via Bulk 2.0 Query for the snapshot. Matching v1 = NPI deterministic + exact website + phone E.164 + name+state Jaccard ⩾ 0.85, top-3 fallback candidates. Push exceptions into Salesforce custom objects, reviewer uses native list views. **Write-back disabled (dry-run)**: log to `write_back_log` with `success=NULL` so the first quarter is shadow mode.

Phase 2 — Polish (3–4 weeks, +~$10/mo). Enable write-back with field allowlist and Composite-per-practice atomic commits. Replace fuzzy matcher with Splink trained on Phase 1's accepted matches. Add libpostal + Smarty + phonenumbers normalisation. Stand up the LWC `diffTable`

and mass quick actions on `Recon_Exception__c`. Turn on hallucination detection rules and `WRONG_PAGE_LIKELY` quarantine. ADLS lifecycle policies. SQL Audit to existing Log Analytics.

Phase 3 — Scale and hardening (ongoing, +~$5/mo). Partition `salesforce_snapshot` and `scraped_raw` by `snapshot_date`; archive partitions older than 8 quarters. Add a KEDA cron trigger on Container Apps Jobs as a defense-in-depth backup to the Logic App schedule. Quarterly drift report Parquet → Excel → `exports/`. App Insights SLO dashboard (`% diffs auto-resolved`, `% write-backs succeeded`, `time-to-review`). Whitelist narrow LOW-severity changes for auto-apply once the approval rate clears 98% over two quarters. If scraping grows past 10K sites, increase Job `parallelism` and split scrape from match.

Conclusion — what changed in the design

Three things are non-obvious until you do the research. First, **Bulk API 2.0 query doesn't support parent-child SOQL**, so the obvious one-query approach is impossible and you must run parallel jobs per object — this is a constraint that drives the whole snapshot shape. Second, the temptation to build a custom Azure web app for human review is real but wrong: a non-technical Salesforce admin reviewing 30–100 items per quarter is in the platform that already gives you list views, audit, and approval email for free, and Option A costs you $160–$520/yr plus 40–80 hours of build that buys nothing meaningful at this scale. Third, **the Container Apps Jobs free grant essentially makes the heavy Python work free** at four quarterly runs, which inverts the usual assumption that "real compute" needs a real bill — you spend your money on private endpoints, not CPU.

The pipeline is fundamentally cheap because the data is small, the cadence is slow, and Salesforce itself is the most expensive piece of infrastructure already paid for. The work is in matching quality (Splink + NPPES cross-validation), hallucination detection (wrong-page quarantine, LLM evidence spans, NPI taxonomy sanity), and idempotency (deterministic `Idempotency_Key__c`, `If-Unmodified-Since` optimistic concurrency, two-phase staging for high-stakes commits) — not in compute.