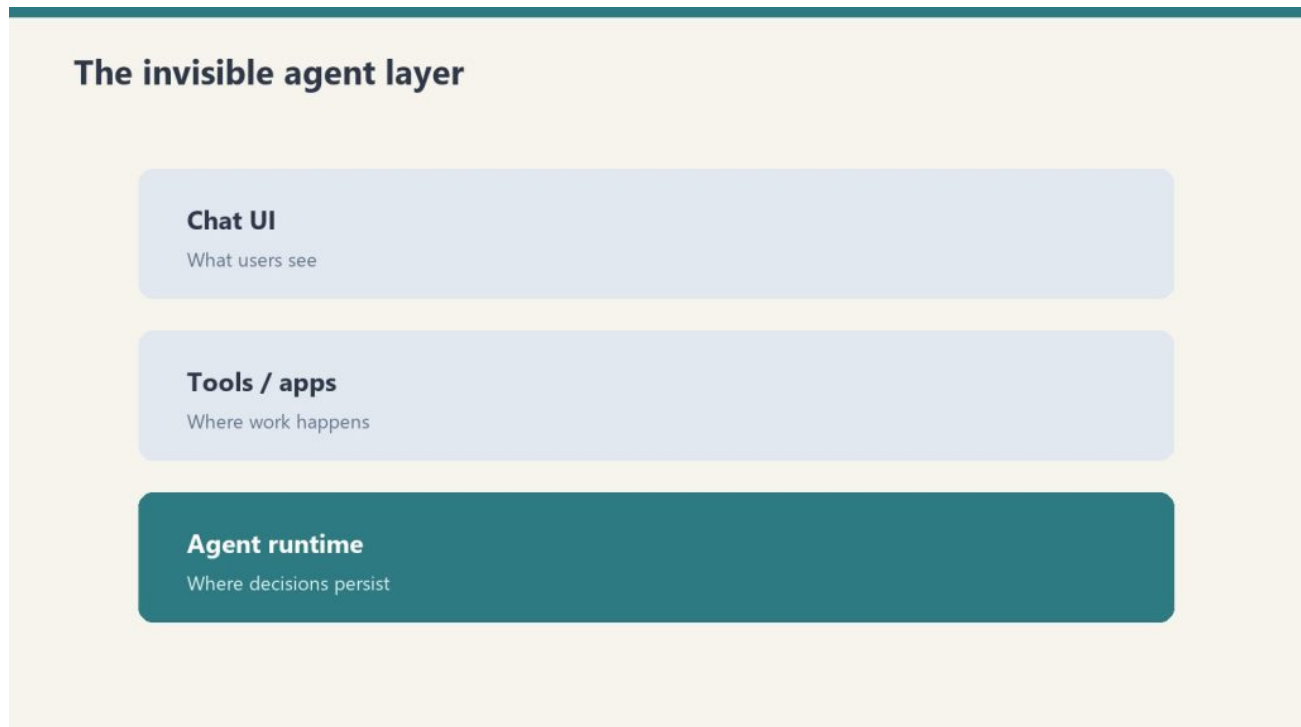


Restoration CRM Automation: GCP Architecture Guide

Who this is for: Your IT person, your developer, or a technical contractor. This brief describes a production-grade automation architecture for the restoration company CRM touch calendar using Google Cloud Platform (GCP). It assumes basic familiarity with cloud infrastructure, command line tools, and web APIs. It does not require deep expertise in any single area — the implementation is intentionally modular so that each component can be handed to a different person if needed.

The business strategy this automates is in [Your CRM Is Not a Lead Database](#). The manual version of this system is in the [Email Automation Setup Guide](#). This brief is for teams who want to reduce ongoing manual work and build a more robust, scalable version of the same workflow.

What This Architecture Automates



What this architecture automates.

The manual system requires a person to: export contacts from the CRM, validate emails, import to Mailchimp or Brevo, configure each campaign, schedule it, and log results back to Notion. For 4–6 campaigns per year, this is manageable manually. For a company running 10–15 campaigns across multiple divisions or service areas, or for an agency running this system for multiple restoration clients, a GCP automation layer eliminates the recurring labor.

What this architecture handles automatically:

- Scheduled contact export from ServiceTitan or Jobber via API
- Segmentation and deduplication logic
- Email validation pass before import
- Contact import to Mailchimp or Brevo
- Campaign creation from template stored in Cloud Storage
- Campaign scheduling per the calendar in Notion
- Results logging back to Notion after send

What still requires human review:

- Email copy review before scheduling (always — no automation should skip this)
 - Reply triage and qualitative logging
 - Warmth scoring and super-connector identification
-

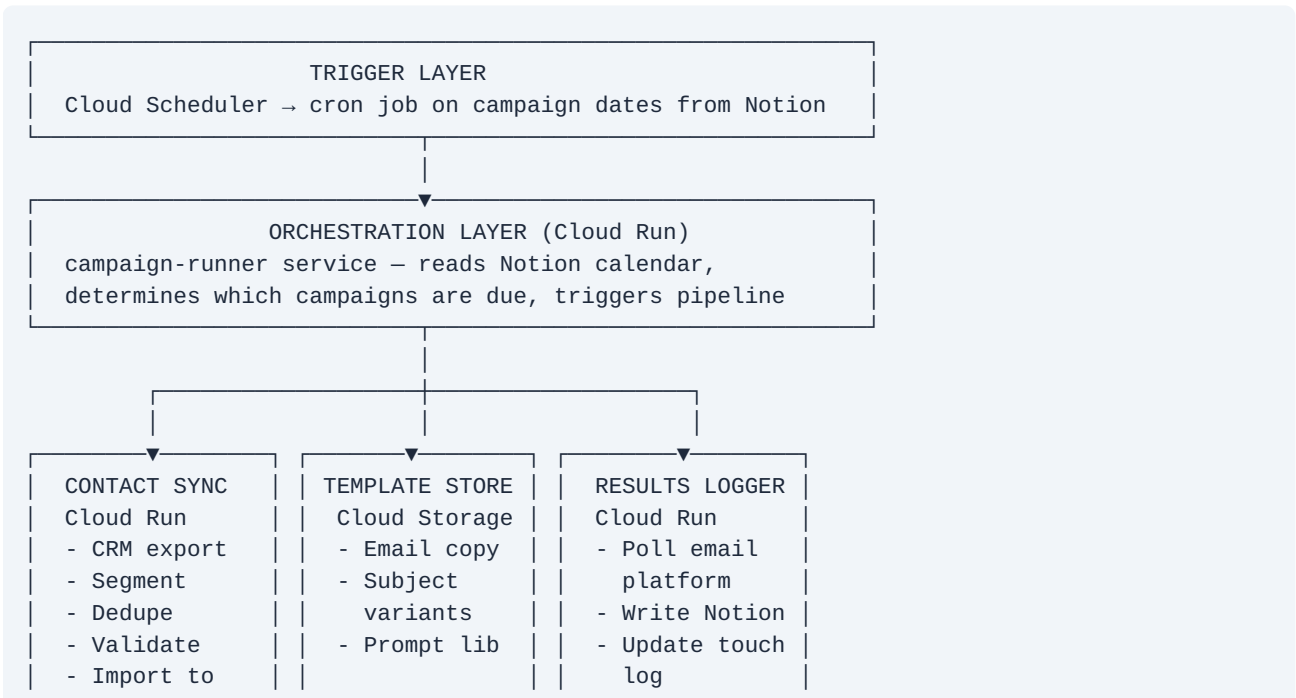
Prerequisites

- A Google Cloud Platform account with billing enabled (new GCP accounts include \$300 in free credits)
 - ServiceTitan or Jobber API access (ServiceTitan requires contacting their enterprise team; Jobber API is available on Connect plan and above at \$119–\$169/month)
 - Mailchimp account with API access (available on all paid plans) OR Brevo with API access (all plans)
 - Notion account with Notion API integration enabled (free at notion.com/my-integrations)
 - Basic Python familiarity (this implementation uses Python 3.11+)
-

Architecture Overview



Architecture overview.



email		
platform		

Component 1: GCP Project Setup

```
# Install gcloud CLI and authenticate
gcloud auth login
gcloud projects create restoration-crm-[yourcompany] --name="Restoration CRM Automation"
gcloud config set project restoration-crm-[yourcompany]

# Enable required APIs
gcloud services enable \
  run.googleapis.com \
  cloudscheduler.googleapis.com \
  secretmanager.googleapis.com \
  storage.googleapis.com

# Create service account for the automation
gcloud iam service-accounts create crm-automation-sa \
  --display-name="CRM Automation Service Account"

# Grant necessary permissions
gcloud projects add-iam-policy-binding restoration-crm-[yourcompany] \
  --member="serviceAccount:crm-automation-sa@restoration-crm-[yourcompany].iam.gserviceaccount.com" \
  --role="roles/run.invoker"
```

Component 2: Secret Manager for API Credentials

Claude enterprise security domains



Secret Manager for API credentials.

Store all API credentials in GCP Secret Manager. Never hardcode credentials in source code.

```
# Store each credential as a separate secret
echo -n "your-servicetitan-api-key" | gcloud secrets create servicetitan-api-key \
  --data-file=-

echo -n "your-jobber-api-key" | gcloud secrets create jobber-api-key \
  --data-file=-

echo -n "your-mailchimp-api-key" | gcloud secrets create mailchimp-api-key \
  --data-file=-

echo -n "your-notion-token" | gcloud secrets create notion-token \
  --data-file=-

# In Python, access secrets like this:
# from google.cloud import secretmanager
# client = secretmanager.SecretManagerServiceClient()
# name = f"projects/{project_id}/secrets/{secret_id}/versions/latest"
# response = client.access_secret_version(request={"name": name})
# secret_value = response.payload.data.decode("UTF-8")
```

Component 3: Contact Sync Service

This Cloud Run service handles the contact export → segment → validate → import pipeline. Deploy as a container triggered by the orchestration layer.

```
# contact_sync/main.py

import os
import json
import requests
from google.cloud import secretmanager

def get_secret(secret_id):
    client = secretmanager.SecretManagerServiceClient()
    project_id = os.environ.get("GCP_PROJECT_ID")
    name = f"projects/{project_id}/secrets/{secret_id}/versions/latest"
    response = client.access_secret_version(request={"name": name})
    return response.payload.data.decode("UTF-8")

def export_jobber_contacts():
    """Export residential clients from Jobber API"""
    api_key = get_secret("jobber-api-key")

    # Jobber uses GraphQL API
    query = """
    query GetClients($after: String) {
      clients(first: 100, after: $after) {
        nodes {
          id
          firstName
          lastName
          emails { address isPrimary }
          tags { label }
          jobs(first: 1) {
            nodes { jobType completedAt }
          }
        }
      }
      pageInfo { hasNextPage endCursor }
    }
    """

    headers = {
        "Authorization": f"Bearer {api_key}",
        "Content-Type": "application/json"
    }

    contacts = []
    cursor = None

    while True:
        variables = {"after": cursor} if cursor else {}
```

```

response = requests.post(
    "https://api.getjobber.com/api/graphql",
    headers=headers,
    json={"query": query, "variables": variables}
)
data = response.json()
clients = data["data"]["clients"]

for client in clients["nodes"]:
    email = next(
        (e["address"] for e in client["emails"] if e["isPrimary"]),
        client["emails"][0]["address"] if client["emails"] else None
    )
    if not email:
        continue

    # Determine segment based on tags
    tags = [t["label"].lower() for t in client["tags"]]
    if "residential" in tags or not any(t in tags for t in ["commercial",
"adjuster", "vendor"]):
        segment = "Homeowner"
    elif any(t in tags for t in ["adjuster", "agent", "insurance"]):
        segment = "Industry"
    else:
        segment = "Trade"

    # Get most recent job type
    job_type = None
    if client["jobs"]["nodes"]:
        job_type = client["jobs"]["nodes"][0].get("jobType", "")

    contacts.append({
        "first_name": client["firstName"],
        "last_name": client["lastName"],
        "email": email.lower().strip(),
        "segment": segment,
        "job_type": job_type or ""
    })

    if not clients["pageInfo"]["hasNextPage"]:
        break
    cursor = clients["pageInfo"]["endCursor"]

return contacts

def deduplicate_contacts(contacts):
    """Remove duplicate emails, keep most recent record"""
    seen = {}
    for contact in contacts:
        email = contact["email"]
        if email not in seen:
            seen[email] = contact
    return list(seen.values())

def segment_contacts(contacts):

```

```

    """Split into three segment lists"""
    segments = {"Homeowner": [], "Industry": [], "Trade": []}
    for contact in contacts:
        seg = contact.get("segment", "Homeowner")
        if seg in segments:
            segments[seg].append(contact)
    return segments

def import_to_mailchimp(contacts, tag, api_key, list_id):
    """Batch import contacts to Mailchimp with tag"""

    # Mailchimp batch operations (max 500 per call)
    batch_size = 500

    for i in range(0, len(contacts), batch_size):
        batch = contacts[i:i+batch_size]

        operations = []
        for contact in batch:
            operations.append({
                "method": "PUT",
                "path": f"/lists/{list_id}/members/{contact['email'].encode().hex()}",
                "body": json.dumps({
                    "email_address": contact["email"],
                    "status_if_new": "subscribed",
                    "merge_fields": {
                        "FNAME": contact.get("first_name", ""),
                        "LNAME": contact.get("last_name", ""),
                        "JOB_TYPE": contact.get("job_type", "")
                    },
                    "tags": [tag]
                })
            })

        response = requests.post(
            "https://us1.api.mailchimp.com/3.0/batches",
            auth=("anystring", api_key),
            json={"operations": operations}
        )

        if response.status_code not in [200, 201]:
            raise Exception(f"Mailchimp batch import failed: {response.text}")

    return len(contacts)

def run_contact_sync(request):
    """Main Cloud Run handler"""
    mailchimp_api_key = get_secret("mailchimp-api-key")
    mailchimp_list_id = os.environ.get("MAILCHIMP_LIST_ID")

    contacts = export_jobber_contacts()
    contacts = deduplicate_contacts(contacts)
    segments = segment_contacts(contacts)

    results = {}

```

```

for segment_name, segment_contacts in segments.items():
    count = import_to_mailchimp(
        segment_contacts,
        tag=segment_name,
        api_key=mailchimp_api_key,
        list_id=mailchimp_list_id
    )
    results[segment_name] = count

return json.dumps({"status": "success", "imported": results})

```

Component 4: Cloud Scheduler Trigger

Cloud Scheduler runs the orchestration service on the campaign dates stored in your Notion calendar. The scheduler checks Notion weekly for upcoming campaigns and pre-triggers the contact sync 7 days before each scheduled send.

```

# Create weekly scheduler job
gcloud scheduler jobs create http crm-weekly-check \
  --schedule="0 9 * * 1" \
  --uri="https://[cloud-run-url]/check-upcoming-campaigns" \
  --oidc-service-account-email="crm-automation-sa@[project].iam.gserviceaccount.com" \
  --time-zone="America/Los_Angeles" \
  --location="us-west1"

```

The orchestration service reads your Notion Campaign Calendar database, finds any campaigns with a send date within the next 7 days and a Status of “Scheduled,” and triggers the contact sync and campaign creation pipeline for each one.

Component 5: Results Logger

After each campaign sends, this service polls the Mailchimp or Brevo API for campaign analytics and writes them back to your Notion Campaign Calendar database.

```

# results_logger/main.py (simplified)

def log_campaign_results(campaign_id, notion_page_id):
    mailchimp_api_key = get_secret("mailchimp-api-key")
    notion_token = get_secret("notion-token")

    # Get Mailchimp campaign report
    response = requests.get(
        f"https://us1.api.mailchimp.com/3.0/reports/{campaign_id}",
        auth=("anystring", mailchimp_api_key)
    )

```

```

)
report = response.json()

open_rate = report.get("opens", {}).get("open_rate", 0)

# Update Notion page
notion_headers = {
    "Authorization": f"Bearer {notion_token}",
    "Content-Type": "application/json",
    "Notion-Version": "2022-06-28"
}

requests.patch(
    f"https://api.notion.com/v1/pages/{notion_page_id}",
    headers=notion_headers,
    json={
        "properties": {
            "Status": {"select": {"name": "Sent"}},
            "Open Rate": {"number": round(open_rate * 100, 1)}
        }
    }
)

```

Estimated Monthly GCP Costs

For a single restoration company running 6 campaigns per year:

Service	Usage	Monthly Cost
Cloud Run (contact sync)	6 invocations/year, <5min each	<\$1
Cloud Scheduler	52 weekly checks/year	\$0.10
Cloud Storage (templates)	Minimal storage	<\$0.01
Secret Manager	4 secrets, <1000 accesses/month	<\$0.10
Total		<\$2/month

For an agency running this system for 10 restoration clients simultaneously, the cost scales linearly — approximately \$15–20/month in GCP costs for the full multi-client operation. The manual labor savings at that scale are significant: an estimated 8–12 hours per month of manual campaign setup eliminated.

Deployment Checklist

- GCP project created and APIs enabled
- Service account created with appropriate permissions

- All API credentials stored in Secret Manager
- Contact sync service containerized and deployed to Cloud Run
- Cloud Scheduler job created and tested
- Notion Campaign Calendar database connected
- Results logger deployed and tested with a historical campaign
- Full end-to-end test run on a staging contact list before live deployment

Full documentation for each GCP service referenced here: cloud.google.com/run/docs, cloud.google.com/scheduler/docs, cloud.google.com/secret-manager/docs.

About this image. This image was generated by Google's Vertex AI Imagen 4 model as the featured visual for The 4% Problem: Why Almost Nobody in Restoration Is Using the AI That's Already in Their CRM. Every image in the Tygart Media visual library is AI-generated, converted to WebP for optimal performance, and enriched with IPTC/XMP metadata for search engine discoverability.

- Model: Vertex AI Imagen 4 Standard (imagen-4.0-generate-001)
- Format: WebP (quality 90)
- Aspect Ratio: 16:9
- Metadata: IPTC/XMP injected (title, description, creator, rights, location)
- Creator: Tygart Media — tygartmedia.com

Part of The Studio collection — AI-generated visuals powering the Tygart Media content engine.