

COPY

14 / 21

Daily Enrichment Skill via API

Turn the one-off spreadsheet job into a daily system that finds new subscribers, writes their fuzzy variables, and sends them the welcome email without you touching it.

01 When to use this

Use this once the sheet version works and you have judged the output with your own eyes. This is the rung where the work stops being a task and becomes a system. Do not climb here first, because you will be automating a process you have not proven.

02 The prompt

I want to turn the enrichment work we just did into a daily skill that runs against `<EMAIL_PLATFORM>` directly instead of against a spreadsheet.

Before you write anything, research the `<EMAIL_PLATFORM>` API and tell me whether it exposes the endpoints needed to do all three of these:

1. Every 24 hours, list subscribers who joined since the last run.
2. For each new subscriber, generate the fuzzy variables we defined and write them into that subscriber's custom fields.
3. Add that subscriber to the welcome broadcast and send it, without ever sending twice to the same person.

If any of the three is not possible with the available endpoints, say so plainly and propose the closest workable design. Do not pretend an endpoint exists.

Once you have confirmed the endpoints, design the flow and show it to me before building.

Constraints:

- The fuzzy variable rules are unchanged: `<PASTE_YOUR_LOCKED_SLOT_RULES>`.
- Double sending is the failure I care about most. Design explicitly against it.
- Credentials: use `<PLATFORM>_API_KEY` from my environment. You cannot read the value, so to prove it works, make one harmless read-only call, for example fetch my account details, and show me the response.
- Log what you did each run: how many new subscribers, how many enriched, how many sent.

03 What it is actually doing

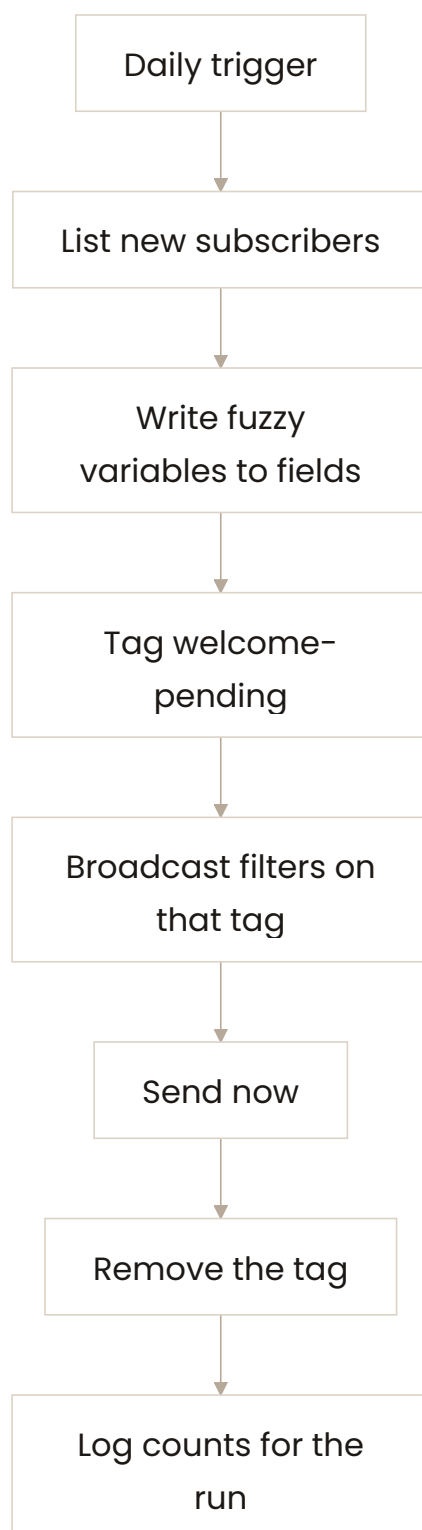
The sheet version proves the idea works. It does not scale, because you have to be there.

So you point the same job at the live platform. The prompt does not tell Claude how to do it. It tells Claude what has to be true, then asks it to go read the API docs and report whether the platform can actually do it. That ordering matters. If you assume the endpoints exist, Claude will happily build against an endpoint it invented.

The design Claude came back with is worth learning on its own, because it solves the hard part. Create a tag like `welcome-pending`. Tag each new subscriber. Update the draft broadcast so it filters on that tag. Set the send time to now. Send. Then remove the tag from everyone who just got it. That tag dance is the whole double-send guard: only tagged people receive the broadcast, and nobody stays tagged after they receive it.

The credentials part is a small trick that saves real time. Claude cannot see the value of an API key in your environment, so it cannot check the key by looking at it. Instead you tell it to make one harmless read-only call with the key. If the call returns your account, the key works. If it errors, you fix the key before you build anything on top of it.

04 How it flows



05 Step by step

01 Prove the manual version first

Run the sheet enrichment, read the output, fix the values. Only automate a process whose output you have already judged as good.

02 Ask about endpoints before building

Make Claude research the platform API and confirm all three capabilities exist. Tell it to say so plainly if one does not, rather than inventing an endpoint.

03 Pick the API key over the connector

Nick found the platform's MCP connector did not cover what he needed, so he fell back to an API key stored in the local environment. Check the connector first, fall back second.

04 Disconnect the connector if you go API

If you use the API-key route, disconnect the MCP connector for that platform. Two access paths to the same service will conflict and produce confusing failures.

05 Prove the key with a read-only call

In a fresh chat, ask Claude to make one harmless read call with the key and show the response. That is how you verify a secret Claude cannot see.

06 Review the tag dance before you run it

Have Claude explain the tag, filter, send, untag sequence back to you. This is the part that prevents double sends, so understand it before it fires.

07

Carry context into a new chat

Long sessions get sloppy. Paste the prior conversation into a fresh thread so the new session starts with the decisions already made instead of relearning them.

08

Watch the first live run

Sit through one real 24 hour cycle. Check the platform's sent log against your own count. Do not take the run summary as proof.

06 Words explained

Endpoint

One specific thing a service lets outside software ask it to do, like list subscribers or update a field. If there is no endpoint for it, no amount of prompting makes it possible.

MCP connector

A prebuilt bridge that lets Claude use a service directly. Convenient, but each one only exposes the actions its author chose to expose.

Custom field

An extra slot on a contact record in your email platform where you can store anything you want, including a fuzzy variable.

Broadcast

A one-off email sent to a chosen group, as opposed to an ongoing automated sequence.

Tag

A label you stick on a contact so you can filter for them later. Here it acts as a temporary flag meaning this person has not had the welcome yet.

Environment variable

A secret like an API key stored outside your code so tools can use it without anyone reading it in a chat.

07 Watch out for

- The platform's MCP connector may not expose what you need. Check its actual actions before you design around it.
 - Running the connector and an API key at the same time causes conflicts. Disconnect the connector if you go the key route.
 - Claude cannot read your API key, so it cannot confirm the key works by inspection. Force a read-only test call instead.
 - Without the untag step at the end, the next run sends the welcome email to the same people again.
 - If you skip the endpoint research, Claude will build against an endpoint that does not exist and only fail at run time.
-
-

08 How it connects

Newsletter Fuzzy Variable Enrichment

This is the same enrichment logic, moved off the spreadsheet and pointed at the live platform.

Self-Healing and Error Logging Clauses

A daily job that nobody watches needs both clauses appended, or it will fail silently the first time the API changes.

Automation Ladder – Prompt Skill Loop Routine

This note is the prompt to skill jump for the copy build, and the same ladder rules apply to every other build in the course.

Fuzzy Variables for Personalized Copy

The framework note that records the tag dance as the standard guard against double sends.

Credit. Framework and approach by Nick Saraev, from the daily enrichment skill segment of the copy personalization build in his Zero to One course on Claude Code for marketing. Written up for the Kape vault by BrewedOps.