API Trigger: Stripe Webhook for email composition

API triggers are the most flexible: Any system that can send a POST request to a URL can trigger your routine.

When should you choose an API trigger?

Scheduled routines run according to the clock. GitHub-event routines react to code changes. Everything else is in the trigger API.

API triggers are the most flexible: Any system that can send a POST request to a URL can trigger your routine. That includes:

1. Stripe webhook - customer just paid, upgraded their subscription, and is requesting a refund.
2. Sentry Warning - New Error, Threshold Exceeded
3. Monitoring systems - Datadog, PagerDuty, Honeycomb - are activated based on a single indicator.
4. Customer support tools - new tickets, SLA violations, escalation
5. Your own backend system - any events you've announced.

In this lesson, we'll build an example from Anthropic's launch blog post: A Stripe webhook triggers a routine, reads new customer details, and composes a personalized welcome email that's saved to your Gmail drafts folder—ready for the user to review and send.

The structure of the trigger API

When you create a routine that's triggered by the API, Anthropic generates a unique endpoint URL. It looks like this:

```
https://api.claude.com/v1/routines/{routine-id}/fire
```

This is the URL your source system sends the POST request to. Authentication takes place via headers—usually the API key generated when you set up your routine. For example:

```
Authorization: Bearer ccr_xxxxxxxxxxxxxxxxxx
```

The request body should be JSON only. Anything you send will be passed to the routine prompt as input data. This is the minimum curl statement required:

```
curl -X POST  "https://api.claude.com/v1/routines/{routine-id}/fire"  -H "Author
```

The routine will be executed. Your prompt can reference any field from that JSON body.

Step 1: Prompt

Start with the prompt, as the prompt identifies the data you need from Stripe. This is the prompt we will use:

```
B?n ?ang vi?t m?t email chào m?ng ???c cá nhân hóa cho m?t khách hàng m?i v
?a ??ng ký. B?n s? nh?n ???c s? ki?n `subscription.created` c?
a Stripe làm d? li?u ??u vào, bao g?
m: email khách hàng, tên khách hàng và gói d?ch v? h? ?ã ch?n. S? d?
ng Gmail MCP connector ?? t?o b?n NHÁP (không g?i) trong th? m?c nháp c?
a Gmail v?i c?u trúc sau: To: {email khách hàng} From: tôi Subject: Chào m?
ng b?n ??n v?i {tên s?n ph?m} - hãy cùng thi?t l?p nhé N?i dung (d??i 150 t
?): - L?i chào thân thi?n ng?n g?n g?i ??n khách hàng b?ng tên - ?? c?p c?
 th? ??n gói d?ch v? h? ?ã ch?n và nh?ng l?i ích mà gói d?ch v? mang l?
i - Ba hành ??ng c? th? h? có th? th?c hi?n ngay hôm nay, ???c li?t kê b?
ng d?u ch?m ??u dòng - L?i k?t "hãy tr? l?i email này n?u b?n c?n b?t c? ?i
?u gì" t? tôi Quy t?c: - Luôn t?o d??i d?ng b?n nháp, không bao gi? g?
i - Không bao gi? ?? c?p ??n giá c?, hoàn ti?n ho?c nâng c?p - Không bao gi
? s? d?ng bi?u t??ng c?m xúc - N?u thi?u tên khách hàng, hãy s? d?
ng "there" làm l?i chào - N?u s? ki?n không ph?
i là subscription.created, thì không làm gì c? và thoát
```

Note the defensive patterns:

1. The DRAFT-ON instructions are clear - reiterated for safety reasons, although we will also be limiting the token scope.
2. Handle empty state - if the event is not as expected, exit.
3. Strict word limit - 150 words
4. The correct email format and structure.

Step 2: Set up Gmail MCP connector (Minimum scope)

This is where security becomes crucial.

Go to **Settings ? Connectors ? Gmail** and start the OAuth routine. When Gmail asks which areas you want to grant permission to, select carefully:

Scope	Is permission granted?	Reason
`gmail.drafts.create`	? Yes	Essential - this routine generates drafts
`gmail.send`	? NO	We absolutely do not want Claude to send
`gmail.readonly`	? NO	This routine does not need to read existing emails.
`gmail.modify`	? NO	Don't let routine edit or delete.
`gmail.labels`	? NO	Not required for the draft.

If the connector's user interface doesn't allow you to select individual scopes, use a Google Workspace admin policy or an OAuth application with limited scope. The rule is: if the task is "compose emails," the connector should only be able to compose emails. Not send. Not read. Not edit.

This is the second most common security error in routines (after unfiltered connectors): Granting too much scope to a connector at the time of OAuth.

Step 3: Configure the trigger API

Return to Routines UI:

1. **Similar routine**
2. **Trigger type** : API
3. **Routine name:** `stripe-onboarding-draft`
4. **Generate an API key** - this is the Bearer token that the Stripe webhook handler will use. Copy it and store it in your secret manager. You will no longer be able to see it.
5. **Connectors** : Gmail only (deselect everything else - see Lesson 2)
6. **Paste the prompt from step 1**
7. **Save**

After saving, you will see the routine's trigger URL. Copy that URL—it will be added to the Stripe webhook configuration.

Step 4: Connect to Stripe

Now connect to Stripe. This is a one-time setup in your Stripe control panel:

1. **Developers ? Webhooks ? Add endpoint**
2. **Endpoint URL** : Activation URL from the routine
3. **Events to listen for:** `customer.subscription.created` (and only this event)
4. **Save**

Here's something to note: Stripe sends POST requests directly to your URL with Stripe's event signature in the header, but your routine won't know how to verify that signature itself. You have two options:

Option A : Place a proxy server in between (recommended for production environments). Your server receives the Stripe webhook, verifies the Stripe signature, and then forwards the payload to the routine's trigger URL with your Bearer token. Your server is the trusted boundary.

Option B : Accept that anyone who finds your trigger URL can activate the routine. This is only acceptable for demos, internal testing, or non-sensitive work. If someone guesses or steals the URL + Bearer token, they can activate the routine and consume your limit.

For this lesson, option B would be fine – we're building an email sending routine just for drafting. For a production environment (especially if the routine can send actual messages or modify actual data), always use option A.

This is the minimum Node proxy you can place in front of it:

```
// Cloudflare Worker, hàm Vercel ho?c b?t k?
 endpoint serverless nào export async function POST(request) { // 1. Xác minh sig
?c theo s? ki?
n chúng ta quan tâm if (event.type !== 'customer.subscription.created') { return
?n ti?p ??
n hàm await fetch(process.env.CCR_TRIGGER_URL, { method: 'POST', headers: { 'Autl
```

This is approximately 40 connection lines, and it's a safe pattern for any API-triggered routine in a production environment.

Step 5: Test with a fake payload

Before waiting for an actual registration, run the routine manually:

```
curl -X POST  "$CCR_TRIGGER_URL"  -H "Authorization: Bearer $CCR_API_KEY"  -H "Co
```

Open the routine's run history. You will see:

1. The request has been received.
2. I used the Gmail MCP connector.
3. The draft has been created.
4. The process is complete.

Open your Gmail drafts folder. There will be a draft addressed to test@example.com, greeting Alex by name, mentioning the Pro Monthly plan, with the next three actions and your signature.

If everything looks good, flip the switch on Stripe. Your next real subscriber will receive a personalized draft in your inbox in just seconds.

Common mistake: Using the same template for "Send actual message"

The question is: "Why didn't I let Claude send the email?"

Don't do that. Not in your initial routines, and not without thorough testing.

Reason:

1. Drafts may be reviewed. Actual emails will not.
2. Drafts create a natural barrier for users. You see the message before the customer sees it.
3. Drafts vary in scale. One wrong draft could send an email to 1,000 customers. A wrong draft only causes embarrassment in the worst-case scenario.

If you need to automate sending later, justify it. Start with drafts, monitor your routine to ensure output quality for 30 days, then consider granting scoped sending permissions. Even then—keep the draft priority option for the most sensitive messages.

Key points to remember

1. Trigger APIs unlock any event source, not just GitHub.
2. Event data becomes input that your prompt can reference.
3. Always set up a proxy server in advance to verify the source signature for the production environment.
4. Grant minimum scope OAuth permissions to connectors - drafts.create, not full send.
5. Start all new automation routines in draft-only mode, then upgrade to send only after reliability has been proven.

1. Question 1:

   What is the MINIMUM permission range you should grant to the routine token for the Gmail MCP connector?

   1. A. Full account access – easier to configure – this might seem logical at first glance, but research points in a different direction.
   2. B. Only include the scope necessary for the task (e.g., only drafts.create, not send).
   3. C. Read-only mode on all directories.
   4. D. Admin scope in the workspace

   EXPLAIN:

   The principle of least privilege prevails. If the task is 'drafting emails for others to review,' the routine only needs the drafts.create permission—no send, no read, no admin. A violated routine can only do what its scope allows.

2. Question 2:

   When Stripe triggers a webhook at your routine endpoint, where is the event payload sent?

   1. A. It's ignored - only the URL matters.
   2. B. It is passed to the routine as prompt input (accessible as a variable).
   3. C. It was saved to a separate file that Claude couldn't see.
   4. D. It's encrypted so Claude can't read it.

   EXPLAIN:

   The payload data becomes input to the routine—you reference it in your prompt just like any other variable. Here's the whole point: The MAIN event data is the work Claude needs to perform.

3. Question 3:

   What should you do before connecting any third-party services to a routine's trigger API?

   1. A. Nothing needs to be done - Anthropic handles security automatically - this is a common belief but doesn't hold up on closer examination.
   2. B. Set up a routine that only runs on weekends.
   3. C. Verify the webhook signature to ensure only your source can trigger the routine.

4. D. Disable all MCP connectors.

EXPLAIN:

API-triggered routines are publicly accessible endpoints. Without signature verification, anyone who discovers the URL can trigger the routine and consume your limits (or worse, steal connector data). Always verify webhook signatures in your source system or at your proxy server.

Submit your work

Training results

You have completed **0** questions.

-- / --

Review the lesson