

Introduction

AWS Lambda function that receives email through Amazon SES and forwards it as a signed HTTP webhook to another system

Anonymous · 10/10/2026

AWS Lambda Function for Email2Webhook

github derafu/aws-lambda-email2webhook  passing

AWS Lambda Function that receives inbound email through Amazon SES and forwards it as a signed HTTP webhook, so a downstream system (for example, a support ticketing workflow) can process it.

How it works

Amazon SES invokes this function directly through a receipt rule's **Lambda action** (no SNS topic involved). The Lambda action never delivers the email body, only metadata, so the receipt rule must also have an **S3 action placed before the Lambda action** to store the raw MIME message. This function:

1. Reads the `messageId` from the SES event.
2. Downloads the raw email from the S3 bucket (object key = `messageId`, optionally prefixed).
3. Builds an envelope: `{"meta": {"source": "aws-ses", "format": "...", "version": "1.0.0"}, "data": {...}}`. `meta.format (WEBHOOK_FORMAT)` selects which builder produces data, so the shape of data is fully determined by format. See "Output formats" below.
4. Signs the envelope (`WEBHOOK_SIGNATURE`) and sends it as a POST request to the configured webhook.

Output formats (`WEBHOOK_FORMAT`)

- `postal` (default) — data matches the JSON shape Postal sends for HTTP endpoints configured with `Encoding=BodyAsJSON` and `Format=Hash` (verified against Postal's own source, `app/senders/http_sender.rb`), so an existing Postal-based integration can receive it without changes to its parsing logic. Two fields have no real SES equivalent and are approximated:
 - `id / token` are Postal's internal sequential integer ID and per-message secret; both are filled with SES's `messageId` (a string, not an integer) since nothing downstream is known to rely on `id` being numeric.
 - `spam_status` is derived from SES's `spamVerdict.status` via the `SPAM_STATUS_MAP` dict in `lambda_function.py` (edit it directly if the mapping needs to change) — `GRAY` and

`PROCESSING_FAILED` are mapped to "Spam" on purpose: an uncertain or failed scan shouldn't be trusted as much as a `PASS`, so it gets the same downstream priority as a confirmed `FAIL` instead of being silently treated as clean.

- `received_with_ssl` is always `null` — SES doesn't report whether the original SMTP session used TLS.
- `bounce` is always `false` — an inbound SES message is never a bounce notification.
- `ses` — `data` is the envelope's own native representation, no MIME parsing performed by this function: `{"ses": {"mail": {...}, "receipt": {...}}, "email_base64": "<raw .eml, base64>"}`. Use this for a receiver that wants to parse the MIME message itself.

Adding a new output format means adding one function to `DATA_BUILDERS` in `lambda_function.py` — nothing else in the function needs to change.

Signing (`WEBHOOK_SIGNATURE`)

- `hmac` (default) — HMAC-SHA256 of the exact request body, sent as `X-Webhook-Signature-256: <hex digest>`, with no prefix by default. `WEBHOOK_SIGNATURE_PREFIX` prepends any literal string to the digest if the receiving end expects one (e.g. set it to `sha256=` for the convention GitHub webhooks use) — see “Verifying the webhook signature” below.
- `none` — no signature header sent.

There is intentionally no `rsa` option that mimics Postal's own signature scheme (`X-Postal-Signature-256`, RSA-SHA256 signed with Postal's private key): that private key lives on the Postal server, and reusing it here would mean sharing one signing key across two independent systems, which is the wrong tradeoff for what it saves. If your receiving system already authenticates some other source using its own signature scheme, add a **separate branch or webhook trigger** for this Lambda's `hmac` signature instead of trying to make this function reproduce that other scheme — the `format: "postal"` envelope already gets the `data` shape right, only the authentication step needs its own path.

Last updated on 10/10/2026