

AWS Setup

AWS Setup

Anonymous · 10/10/2026

AWS Setup

Required AWS setup

This guide assumes the Lambda function itself already exists (python3.14 runtime), with the AWS-managed `AWSLambdaBasicExecutionRole` attached to its execution role — that part is generic Lambda setup, not specific to this project. Everything below covers wiring SES → S3 → this Lambda, using the AWS CLI. Replace every `<PLACEHOLDER>` with your own values, and every plain example value (account `123456789012`, region `us-east-2`, names) with your own; the commands below assume `aws configure` is already set up with credentials that can manage S3, SES, IAM and Lambda.

At a high level, you need:

- An S3 bucket that the SES service principal is allowed to write to.
- An SES receipt rule with two actions, in this order:
 1. **S3** — store the raw message in the bucket above.
 2. **Lambda** — invoke this function, invocation type `Event` (recommended; `RequestResponse` also works, the function returns a `disposition` for that case).
- The Lambda execution role needs `s3:GetObject` on that bucket (and prefix, if any).
- The Lambda function's own **Timeout** must be raised well above the runtime's default of 3 seconds — see "Lambda timeout" below.

The rest of this section walks through each of these with the actual CLI commands.

S3 bucket: creation and permissions

Namespace. When creating the bucket, prefer S3's **account regional namespace** over the shared global one (the console offers both, account regional is the one marked "recommended"). It only has to be unique within your own account and region, it can never be re-created by another AWS account

after you delete it, and it works exactly like any other general purpose bucket — no limitation on SES writing to it. The console (and the CLI, if you follow the same convention) appends `-<accountId>-<region>-an` to whatever prefix you choose, e.g. an `email-inbox` prefix in account `123456789012` and region `us-east-2` becomes `email-inbox-123456789012-us-east-2-an`.

Three separate permissions are involved, easy to confuse for a single “give access” step:

Who → who	Permission	Where it's configured
SES → S3	<code>s3:PutObject</code> (write the raw email)	bucket policy on the S3 bucket
SES → Lambda	<code>lambda:InvokeFunction</code>	resource policy on the Lambda function
Lambda → S3	<code>s3:GetObject</code> (read the raw email back)	execution role attached to the Lambda function

If you configure the S3 and Lambda actions through the SES console's receipt rule editor, it adds the first two automatically. The steps below use the CLI instead, so all three have to be set up explicitly.

1. Look up your account ID and create the bucket

```
$ aws sts get-caller-identity --query Account --output text
123456789012

$ aws s3api create-bucket \
  --bucket email-inbox-123456789012-us-east-2-an \
  --region us-east-2 \
  --create-bucket-configuration LocationConstraint=us-east-2
{
  "Location": "http://email-inbox-123456789012-us-east-2-an.s3.amazonaws.com/"
}
```

(`--create-bucket-configuration LocationConstraint=<region>` is required for every region except `us-east-1`, which is the one region `create-bucket` accepts with no location constraint at all — a historical quirk from before S3 supported multiple regions.)

2. Find (or create) the SES receipt rule set you'll use

```
$ aws ses describe-active-receipt-rule-set --query 'Metadata.Name' --output text
default-rule-set
```

If there's no active rule set yet:

```
$ aws ses create-receipt-rule-set --rule-set-name default-rule-set
$ aws ses set-active-receipt-rule-set --rule-set-name default-rule-set
```

3. Attach a bucket policy allowing SES to write into it. Save this as `bucket-policy.json`, replacing `<BUCKET_NAME>`, `<ACCOUNT_ID>`, `<REGION>`, `<RULE_SET_NAME>` and `<RECEIPT_RULE_NAME>` (the receipt rule is created in the next step, but its name must match exactly here):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowSESPuts",
      "Effect": "Allow",
      "Principal": {
        "Service": "ses.amazonaws.com"
      },
      "Action": "s3:PutObject",
      "Resource": "arn:aws:s3:::<BUCKET_NAME>/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "<ACCOUNT_ID>",
          "AWS:SourceArn": "arn:aws:ses:<REGION>:<ACCOUNT_ID>:receipt-rule-set/<RULE_SET_NAME>:receipt-rule/<RECEIPT_RULE_NAME>"
        }
      }
    }
  ]
}
```

Then apply it (no output on success):

```
$ aws s3api put-bucket-policy \
  --bucket email-inbox-123456789012-us-east-2-an \
  --policy file://bucket-policy.json
```

Notes on this policy:

- `Resource` ends in `/*` — the permission is on objects inside the bucket, not on the bucket itself.
- The `Condition` block locks this down to **this specific receipt rule** (`AWS:SourceArn`) inside **this specific account** (`AWS:SourceAccount`) — without it, any SES account writing through any rule could write to the bucket.
- If the rule set or the rule is ever renamed, `AWS:SourceArn` has to be updated to match, or SES puts will start failing with a “Could not write to bucket” error.
- `SES_S3_KEY_PREFIX` (see “Environment variables” below) is a separate, independent setting — it doesn’t appear in this policy at all, it only affects which key inside the bucket the object gets written to.

SES receipt rule and Lambda invoke permission

4. Create the receipt rule, with the S3 action before the Lambda action, replacing the recipient address, bucket name/prefix and Lambda function ARN with your own (no output on success):

```
$ aws ses create-receipt-rule \
  --rule-set-name default-rule-set \
  --rule '{
    "Name": "email2webhook",
    "Enabled": true,
    "TlsPolicy": "Optional",
    "ScanEnabled": true,
    "Recipients": ["inbox@example.com"],
    "Actions": [
      {
        "S3Action": {
          "BucketName": "email-inbox-123456789012-us-east-2-an",
          "ObjectKeyPrefix": "inbound/"
        }
      },
      {
        "LambdaAction": {
          "FunctionArn": "arn:aws:lambda:us-
east-2:123456789012:function:email2webhook",
          "InvocationType": "Event"
        }
      }
    ]
  }'
```

5. Allow SES to invoke the Lambda function (this is the resource policy from the permissions table above; the CLI doesn't add it for you the way the console's receipt rule editor does):

```
$ aws lambda add-permission \
  --function-name email2webhook \
  --statement-id AllowSESInvoke \
  --action lambda:InvokeFunction \
  --principal ses.amazonaws.com \
  --source-account 123456789012
{
  "Statement":
  "{ \"Sid\": \"AllowSESInvoke\", \"Effect\": \"Allow\", \"Principal\": { \"Service\": \"ses.amazonaws.com\" }, \"Action\": \"lambda:InvokeFunction\", \"Resource\": \"arn:aws:lambda:us-east-2:123456789012:function:email2webhook\", \"Condition\": { \"StringEquals\": { \"AWS:SourceAccount\": \"123456789012\" } } }"
```

6. Attach s3:GetObject to the Lambda's execution role (the third permission from the table — never added automatically). Save this as `s3-read-policy.json`, replacing `<BUCKET_NAME>` and, if you set `SES_S3_KEY_PREFIX`, restricting `Resource` to that prefix:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowReadEmailFromS3",
      "Effect": "Allow",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::<BUCKET_NAME>/inbound/*"
    }
  ]
}
```

Then attach it to your function's execution role (no output on success):

```
$ aws iam put-role-policy \
  --role-name email2webhook-role \
  --policy-name AllowReadEmailFromS3 \
  --policy-document file://s3-read-policy.json
```

If this permission is missing, invocations fail with an `AccessDenied` error on `s3:GetObject`, visible in the function's CloudWatch Logs (`/aws/lambda/<function-name>`).

Object retention (lifecycle rule)

Raw emails accumulate in the bucket forever unless something expires them — nothing in this function ever deletes an object (deliberately: it's what makes “re-run this email through the webhook manually” possible after a failure). Deletion is handled entirely by an **S3 Lifecycle rule** on the bucket, not by this function or its IAM role — it needs no permission on the Lambda's execution role at all.

7. Add a lifecycle rule that expires objects after some number of days. Save this as `lifecycle.json`, adjusting the prefix and retention period:

```
{
  "Rules": [
    {
      "ID": "Delete after 30 days",
      "Filter": { "Prefix": "inbound/" },
      "Status": "Enabled",
      "Expiration": { "Days": 30 }
    }
  ]
}
```

Then apply it (no output on success):

```
$ aws s3api put-bucket-lifecycle-configuration \
  --bucket email-inbox-123456789012-us-east-2-an \
  --lifecycle-configuration file://lifecycle.json
```

Pitfall to avoid if you configure this by hand in the console: use the **Expiration** action (deletes the current, only version of an object after N days), not **“Permanently delete noncurrent versions”** (NoncurrentVersionExpiration). The latter only deletes *old* versions of an object after it’s been overwritten, which requires bucket versioning to be enabled *and* the same key to be written more than once — neither is typically true here (versioning off, every email gets a unique key), so a rule using it silently deletes nothing, forever, with no error to notice.

AMAZON_SES_SETUP_NOTIFICATION is a one-time test object SES writes into the bucket (respecting the configured prefix) when the S3 receipt rule action is first configured, to confirm it can write there — it plays no role in actually receiving email and is safe to delete manually at any time. Left alone, the lifecycle rule above expires it automatically along with everything else, no special handling needed.

Lambda timeout

The function’s **Timeout** setting needs enough room for one S3 `GetObject` call plus up to two sequential HTTP POSTs to the webhook — `send_webhook` first POSTs to `WEBHOOK_DEBUG_URL` (if set) and then to `WEBHOOK_URL`, each with its own 10-second internal timeout in the code (`requests.post(..., timeout=10)`). The runtime default of 3 seconds is easy to exceed once S3 latency and one webhook round-trip are added together; with both `WEBHOOK_URL` and `WEBHOOK_DEBUG_URL` set, that budget roughly doubles. Set this to **60 seconds** for a comfortable margin — there’s no cost downside to a larger timeout, Lambda only bills for actual execution time:

```
$ aws lambda update-function-configuration \
  --function-name email2webhook \
  --timeout 60
```

Last updated on 10/10/2026