Engineer

webhook

an HTTP request a service sends when an event happens

What it is

A webhook is an HTTP request that one service sends to another the moment something happens on its side: a payment goes through, an invoice is paid, a long task finishes. The receiving side registers a URL in advance, and the sender posts a description of each event to it. The Standard Webhooks specification calls webhooks "a common name for HTTP callbacks" and describes them as a channel through which one service announces an event to another. The term is sometimes written as two words, web hook, but Stripe, GitHub and the specification all write it as one.

The specification also describes a webhook as part of a service's API, a kind of API running in reverse. Normally the client calls the service when it wants something. With a webhook, the service calls the client when it has news. In a typical API integration both sides run HTTP servers, one to receive API calls and the other to receive webhooks. A service may also send webhooks without offering a request API of its own. The sender is called the producer, the receiver the consumer, and the URL the receiver registers is its webhook endpoint.

The specification recommends an HTTP POST with a JSON body, but does not require a particular payload format or shape. Its suggested body has a `type`, written with full stops and grouped hierarchically, such as `invoice.paid`; a `timestamp` for when the event occurred, in ISO 8601; and the event data either under `data` or at the top level. Additional metadata may appear in either place. Delivery metadata goes in headers: `webhook-id` identifies the event, `webhook-timestamp` records when this attempt was made, in Unix seconds, and `webhook-signature` authenticates it. Stripe and GitHub use header names of their own, `Stripe-Signature` and `X-Hub-Signature-256` among them, and their verification rules differ: Stripe signs an attempt timestamp with the body, while GitHub's HMAC covers the body alone.

A payload can be full, carrying the whole changed object, or thin, carrying little more than an identifier and leaving the receiver to fetch the rest through the API. The specification weighs the two. A full payload saves the receiver extra API calls. A thin one means less data to generate and send, more flexibility and more room to change later, since a thin payload can always grow into a full one but not the other way round. It also makes data access auditable, because the receiver has to ask explicitly for what it reads. Either way the specification advises keeping payloads small, usually under 20 kB. Stripe offers a thin format and marks it as the recommended one.

Use the concept in a decision

Put “What is a webhook?” to work in a real decision. We can help relate the concept to the system you are building or buying. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

Without webhooks, a system that needs to know about a change somewhere else has to poll: ask again and again, mostly to hear that nothing has happened, and still learn about the change only at the next poll. A webhook turns that around. The service that knows about the event calls once, when it happens. That is how payment providers, code hosting services and email providers report back to the systems built on them.

The reversal has costs, and most of them land on the receiving end. Its endpoint is a public URL that anyone on the internet can POST to, and nothing about an incoming request shows who sent it, so the specification makes verifying authenticity a requirement for any secure implementation. Delivery is retried until the receiver answers with a 2xx status or the delivery policy gives up, so the same event can arrive more than once; Stripe warns that endpoints might occasionally receive duplicates. Order is not promised either: Stripe states plainly that it does not guarantee events arrive in the order they were generated.

The producer, for its part, has to keep trying. The specification recommends a retry schedule spanning several days, with exponential backoff and some random jitter. Its example makes ten attempts, the last one 75 hours, 35 minutes and 5

seconds after the first. Stripe retries for up to three days in live mode. A receiver that comes back within that window catches up on its own; after it, Stripe lets you resend an event by hand for up to 15 days after it was created, or up to 30 days with its CLI. When delivery keeps failing for a long time, the specification recommends that the producer tell the consumer through another channel, such as email, and disable the endpoint.

What it looks like in practice

These are the headers of a delivery in the specification's format. The identifier, timestamp and signature are the example values the specification itself uses; the `Host` value and the path are made up:

```
POST /webhooks/billing HTTP/1.1
Host: example.com
Content-Type: application/json
webhook-id: msg_2KWPBgLlAfxdpx2AI54pPJ85f4W
webhook-timestamp: 1674087231
webhook-signature: v1,K5oZfzN95Z9UVu1EsfQmfVNQhnkZ2pj9o9NDN/H/pI4=
```

And this is the body, a thin payload from the same specification, pretty-printed here. On the wire the specification recommends sending it minified:

```
{
  "type": "contact.created",
  "timestamp": "2022-11-03T20:26:10.344522Z",
  "data": {
    "id": "1f81eb52-5198-4599-803e-771906343485"
  }
}
```

The `v1` prefix marks a symmetric signature: HMAC-SHA256 over the event identifier, the timestamp and the body joined by full stops, keyed with a secret unique to the endpoint. Because the identifier and timestamp are signed along with the body, neither can be altered without breaking the signature. The specification also allows asymmetric ed25519 signatures, marked `v1a`, and recommends them when different parties are responsible for securing the producer and the consumer.

GitHub signs its deliveries with HMAC-SHA256 too, puts the hex digest in `X-Hub-Signature-256` with a `sha256=` prefix, and publishes a secret, a payload and a signature for testing a verifier. This is a verifier in Python, checked against that example:

```
import hashlib
import hmac


def verify(body: bytes, secret: str, header: str) -> bool:
    mac = hmac.new(secret.encode("utf-8"), body, hashlib.sha256)
    expected = "sha256=" + mac.hexdigest()
    return hmac.compare_digest(expected, header)


secret = "It's a Secret to Everybody"
header = (
    "sha256=757107ea0eb2509fc211221cce984b8a"
    "37570b6d7586c22c46f4379c8b043e17"
)
print(verify(b"Hello, World!", secret, header))  # True
print(verify(b"Hello, World! ", secret, header))  # False
```

One trailing space in the body is enough to fail. `hmac.compare_digest` takes the same time whether the strings differ early or late; GitHub's documentation tells you never to compare signatures with a plain `==`. A sound handler then works in a fixed order: read the raw bytes, verify the signature over exactly those bytes, check that the attempt timestamp is recent if the producer signs one, check whether this event identifier has been seen before, put the event on a queue and answer with a 2xx. The real work happens afterwards. Stripe gives the example of returning a 200 before marking a customer's invoice as paid in the accounting system.

Two projects in this site's portfolio deal with webhooks from the receiving end. In the Stripe and in-app purchase entitlements, a subscription has to survive, among other things, a webhook that arrives twice and a webhook that arrives out of order. In the email system with failover and delivery logging, inbound webhooks report what each provider did with a message, and the project records both what was sent and what came back, in a send log and a delivery event table.

Where it goes wrong

The signature never matches. A common cause is that the receiver's framework parses the JSON body and the code serialises it again before computing the HMAC. The specification calls this a very common failure mode: JSON serialisation differs between implementations, and a single stray space invalidates the signature. Stripe needs the raw body for the same reason and warns that any manipulation of it makes verification fail. GitHub adds that a proxy or load balancer in front of the endpoint must not modify the payload or the headers, and that the payload should be handled as UTF-8. Compute the signature over the bytes exactly as they arrived, before anything parses them.

The same event arrives twice. Retries and network trouble both produce duplicates. Under the specification, `webhook-id` stays the same across every retry of an event and serves as an idempotency key; its example is to keep the identifiers in Redis for five minutes. Stripe tells you to log the event IDs you have processed and skip ones you have seen. It also notes that one underlying occurrence can produce two distinct Event objects; you can recognise those by the object ID in `data.object` together with the event type. GitHub's `X-GitHub-Delivery` header stays the same when a delivery is redelivered. Processing should be idempotent, so a second delivery changes nothing.

A captured delivery is played back. A valid signature stays valid, so an attacker who intercepts one delivery can send it again. That is why the timestamp is signed: the specification says to reject a `webhook-timestamp` outside an allowed tolerance. Stripe's libraries default to five minutes, Stripe warns that a tolerance of 0 switches the check off entirely, and it recommends NTP to keep the server's clock accurate. Both Stripe and the specification put a fresh timestamp on every retry, so age is measured from that attempt, not from the original event; a delayed attempt can still fall outside the tolerance. Keep the seen identifiers for at least as long as the tolerance, and anything replayed inside the window is caught as a duplicate.

Events arrive out of order. Stripe's example: creating a subscription might produce `customer.subscription.created`, `invoice.created`, `invoice.paid` and, if there is a charge, `charge.created`, and nothing guarantees they arrive in that order. The `created` field on Stripe snapshot events has one-second resolution, allowing several events to carry the same timestamp, and Stripe says not to use it to order events or detect duplicates. The robust approach treats an event as a notice that something changed and fetches the object's current state from the API before acting on it.

The handler is too slow. GitHub expects a 2xx within 10 seconds; after that it terminates the connection and counts the delivery as failed. The specification recommends a request timeout of somewhere between 15 and 30 seconds. A handler that does its real work inline times out, the producer retries, and the work runs twice. Stripe and GitHub both recommend a queue. Stripe points to the spike at the start of the month, when all subscriptions renew.

Only a 2xx counts. Anything else is a failure, a 3xx included, to the specification and to Stripe alike, so an endpoint that moved behind a redirect fails every delivery until its registered URL is updated. The specification gives a few codes a meaning for the producer: a `410 Gone` means disable the endpoint, a 429, 502 or 504 means slow down, and a `retry-after` header should shape the next attempt.

For the producer, the URL is hostile input. Customers can enter any URL they like, and the producer's servers will call it. The specification calls webhook systems especially vulnerable to server-side request forgery (SSRF): a URL that points at an internal address makes the producer call into its own network. Its advice is to send every webhook request through a proxy that filters internal IP addresses, such as Stripe's smokescreen, and to run the webhook workers in a private subnet that cannot reach internal services.

Further reading

- Standard Webhooks specification — version 1.0.0: payload structure, the signature scheme, headers, the retry schedule, status-code handling and SSRF.
- Receive Stripe events in your webhook endpoint — thin and snapshot payloads, retries, duplicate and out-of-order events, and replay protection.
- Best practices for using webhooks — GitHub's list: few events, a webhook secret, HTTPS, an answer within 10 seconds, redelivery and `X-GitHub-Delivery`.
- Validating webhook deliveries — `X-Hub-Signature-256`, constant-time comparison and a test example to check a verifier against.

Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is a webhook?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/webhook/