Engineer

Webhook

📢 eine HTTP-Anfrage, die ein Dienst bei einem Ereignis sendet

Was es ist

Ein Webhook ist eine HTTP-Anfrage, die ein Dienst an einen anderen sendet, sobald bei ihm etwas passiert: Eine Zahlung wird abgewickelt, eine Rechnung bezahlt oder eine längere Aufgabe abgeschlossen. Der Empfänger hinterlegt vorher eine URL, an die der Absender eine Beschreibung jedes Ereignisses schickt. Die Spezifikation Standard Webhooks bezeichnet Webhooks als gängige Bezeichnung für HTTP-Callbacks und beschreibt sie als Kanal, über den ein Dienst einem anderen ein Ereignis meldet. Gelegentlich wird der Begriff getrennt als web hook geschrieben. Stripe, GitHub und die Spezifikation schreiben ihn jedoch als ein Wort.

Die Spezifikation beschreibt einen Webhook auch als Teil der API eines Dienstes, gewissermaßen als umgekehrt arbeitende API. Normalerweise ruft der Client den Dienst auf, wenn er etwas braucht. Bei einem Webhook ruft der Dienst den Client auf, wenn es Neuigkeiten gibt. Bei einer typischen API-Integration betreiben beide Seiten HTTP-Server: einer nimmt API-Aufrufe entgegen, der andere Webhooks. Ein Dienst kann aber auch Webhooks senden, ohne selbst eine Anfrage-API anzubieten. Der Absender heißt Producer, der Empfänger Consumer. Die vom Empfänger hinterlegte URL ist sein Webhook-Endpunkt.

Die Spezifikation empfiehlt HTTP POST mit einem JSON-Rumpf, schreibt aber weder Format noch Aufbau der Nutzdaten vor. Der vorgeschlagene Rumpf enthält einen `type` mit Punkten und hierarchischer Gliederung, etwa `invoice.paid`, einen `timestamp` für den Ereigniszeitpunkt in ISO 8601 und die Ereignisdaten entweder unter `data` oder auf oberster Ebene. Zusätzliche Metadaten können an beiden Stellen stehen. Zustellungsmetadaten gehören in die Header: `webhook-id` bezeichnet das Ereignis, `webhook-timestamp` hält den Zeitpunkt dieses Versuchs in Unix-Sekunden fest, und `webhook-signature` bestätigt seine Echtheit. Stripe und GitHub verwenden eigene Headernamen,

darunter `Stripe-Signature` und `X-Hub-Signature-256`. Auch ihre Prüfregeln unterscheiden sich: Stripe signiert einen Zeitstempel des Versuchs zusammen mit dem Rumpf, während GitHubs HMAC nur den Rumpf abdeckt.

Die Nutzdaten können vollständig sein und das gesamte geänderte Objekt enthalten oder schlank ausfallen, mit kaum mehr als einer Kennung. Im zweiten Fall ruft der Empfänger den Rest über die API ab. Die Spezifikation wägt beides ab. Vollständige Nutzdaten ersparen zusätzliche API-Aufrufe. Schlanke Nutzdaten bedeuten weniger zu erzeugende und zu sendende Daten, mehr Flexibilität und mehr Spielraum für spätere Änderungen: Sie lassen sich jederzeit zu vollständigen Nutzdaten erweitern, nicht aber umgekehrt. Zudem wird der Datenzugriff nachvollziehbar, weil der Empfänger explizit anfordern muss, was er liest. In beiden Fällen rät die Spezifikation zu kleinen Nutzdaten, gewöhnlich unter 20 kB. Stripe bietet ein schlankes Format an und kennzeichnet es als Empfehlung.

Das Konzept für eine Entscheidung nutzen

Nutzen Sie „Was ist ein Webhook?“ für eine konkrete Entscheidung. Wir können helfen, das Konzept mit dem System in Verbindung zu bringen, das Sie entwickeln oder einkaufen. Der Teil, den Nutzer nie sehen: wo die Daten liegen, welche Regeln gelten und wie andere Systeme darauf zugreifen dürfen.

Unsere Leistungen entdecken Kontakt aufnehmen →

Die Annahmen hinter dem Begriff prüfen

Nutzen Sie die Erklärung und die verlinkten Arbeiten, um Fragen für Ihr Projekt zu formulieren. Wir können die Annahmen gemeinsam mit Ihnen prüfen, bevor ein Werkzeug ausgewählt wird. 6 ETL/ELT-Pipelines entworfen und betrieben — 6 ETL-Pipelines über drei Datenbankengines hinweg

6 ETL/ELT-Pipelines entworfen und betrieben Kontakt aufnehmen →

Warum es wichtig ist

Ohne Webhooks muss ein System immer wieder bei einem anderen System nachfragen, ob sich etwas geändert hat. Meist lautet die Antwort, dass nichts passiert ist. Von einer tatsächlichen Änderung erfährt es trotzdem erst beim nächsten Abruf. Ein Webhook kehrt das um: Der Dienst, der von dem Ereignis weiß, ruft den Empfänger einmal auf, sobald es eintritt. So melden Zahlungsanbieter, Codehosting-Dienste und E-Mail-Anbieter Ergebnisse an die Systeme zurück, die auf ihnen aufbauen.

Diese Umkehr hat ihren Preis, überwiegend auf der Empfangsseite. Ihr Endpunkt ist eine öffentliche URL, an die jeder im Internet POST-Anfragen senden kann. Einer eingehenden Anfrage ist nicht anzusehen, wer sie geschickt hat. Deshalb verlangt die Spezifikation für jede sichere Implementierung eine Echtheitsprüfung. Die Zustellung wird wiederholt, bis der Empfänger mit einem

2xx-Status antwortet oder die Zustellungsrichtlinie aufgibt. Dasselbe Ereignis kann daher mehrfach ankommen; Stripe warnt ausdrücklich vor gelegentlichen Duplikaten. Auch die Reihenfolge ist nicht zugesichert: Stripe garantiert nicht, dass Ereignisse in der Reihenfolge ihres Entstehens eintreffen.

Der Producer muss seinerseits weitere Zustellversuche unternehmen. Die Spezifikation empfiehlt einen über mehrere Tage verteilten Wiederholungsplan mit exponentiell wachsenden Wartezeiten und zufälligen Abweichungen. Ihr Beispiel umfasst zehn Versuche; der letzte erfolgt 75 Stunden, 35 Minuten und 5 Sekunden nach dem ersten. Stripe wiederholt im Livebetrieb bis zu drei Tage lang. Ein Empfänger, der in diesem Zeitraum wieder erreichbar ist, erhält die ausstehenden Zustellungen automatisch. Danach lässt Stripe Ereignisse bis zu 15 Tage nach ihrer Erstellung manuell erneut senden, mit seiner CLI bis zu 30 Tage. Scheitert die Zustellung über längere Zeit weiter, empfiehlt die Spezifikation, den Consumer über einen anderen Kanal, etwa per E-Mail, zu informieren und den Endpunkt zu deaktivieren.

Wie es in der Praxis aussieht

Dies sind die Header einer Zustellung im Format der Spezifikation. Kennung, Zeitstempel und Signatur sind deren eigene Beispielwerte; der `Host`-Wert und der Pfad sind erfunden:

```
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=
```

Dazu gehört dieser Rumpf: schlanke Nutzdaten aus derselben Spezifikation, hier lesbar eingerückt. Für die Übertragung empfiehlt sie eine minimierte Darstellung:

```
{
  "type": "contact.created",
  "timestamp": "2022-11-03T20:26:10.344522Z",
  "data": {
```

```
    "id": "1f81eb52-5198-4599-803e-771906343485"
  }
}
```

Das Präfix `v1` kennzeichnet eine symmetrische Signatur: HMAC-SHA256 über die mit Punkten verbundenen Werte für Ereigniskennung, Zeitstempel und Rumpf, mit einem nur diesem Endpunkt zugeordneten Geheimnis als Schlüssel. Da Kennung und Zeitstempel zusammen mit dem Rumpf signiert sind, lässt sich keiner dieser Werte ändern, ohne die Signatur ungültig zu machen. Die Spezifikation erlaubt außerdem asymmetrische ed25519-Signaturen mit der Kennzeichnung `v1a`. Sie empfiehlt sie, wenn unterschiedliche Parteien für die Sicherheit von Producer und Consumer verantwortlich sind.

Auch GitHub signiert Zustellungen mit HMAC-SHA256. Es schreibt den hexadezimalen Prüfwert in `X-Hub-Signature-256` mit dem Präfix `sha256=` und veröffentlicht ein Geheimnis, Nutzdaten und eine Signatur zum Testen einer Prüffunktion. Diese Python-Prüffunktion wurde gegen das Beispiel geprüft:

```
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
```

Ein einziges zusätzliches Leerzeichen am Ende des Rumpfs lässt die Prüfung scheitern. `hmac.compare_digest` benötigt dieselbe Zeit, ob sich die Zeichenketten früh oder spät unterscheiden. GitHubs Dokumentation warnt davor, Signaturen mit einem einfachen `==` zu vergleichen. Ein solider Handler arbeitet dann in fester Reihenfolge: die unveränderten Bytes lesen, die Signatur über genau diese Bytes prüfen, gegebenenfalls die Aktualität eines vom Producer signierten Versuchszeitstempels prüfen, nachsehen, ob die Ereigniskennung schon bekannt

ist, das Ereignis in eine Warteschlange stellen und mit 2xx antworten. Erst danach beginnt die eigentliche Arbeit. Stripe nennt als Beispiel, mit 200 zu antworten, bevor die Rechnung eines Kunden im Buchhaltungssystem als bezahlt markiert wird.

Zwei Projekte im Portfolio dieser Website behandeln Webhooks aus Sicht des Empfängers. Bei den Berechtigungen aus Stripe und In-App-Käufen muss ein Abonnement unter anderem einen doppelt eintreffenden Webhook und eine vertauschte Reihenfolge verkraften. Beim E-Mail-System mit Ausfallsicherung und Zustellungsprotokollierung melden eingehende Webhooks, was die einzelnen Anbieter mit einer Nachricht getan haben. Das Projekt erfasst sowohl das Gesendete als auch die Rückmeldungen in einem Sendeprotokoll und einer Tabelle der Zustellungsereignisse.

Wo es schiefgeht

Die Signatur stimmt nie. Häufig zerlegt das Framework des Empfängers den JSON-Rumpf, und der Code serialisiert ihn erneut, bevor er den HMAC berechnet. Die Spezifikation nennt das einen sehr häufigen Fehler: JSON-Serialisierung unterscheidet sich zwischen Implementierungen, und schon ein zusätzliches Leerzeichen macht die Signatur ungültig. Stripe verlangt aus demselben Grund den unveränderten Rumpf und warnt, dass jede Bearbeitung die Prüfung scheitern lässt. GitHub ergänzt, dass ein vorgeschalteter Proxy oder Lastverteiler weder Nutzdaten noch Header verändern darf und die Nutzdaten als UTF-8 behandelt werden sollen. Berechnen Sie die Signatur über die Bytes genau in der empfangenen Form, bevor diese geparst werden.

Dasselbe Ereignis kommt zweimal an. Sowohl Wiederholungen als auch Netzprobleme erzeugen Duplikate. Nach der Spezifikation bleibt `webhook-id` bei jedem erneuten Versuch desselben Ereignisses gleich und dient als Idempotenzschlüssel. Ihr Beispiel speichert die Kennungen fünf Minuten in Redis. Stripe empfiehlt, bereits verarbeitete Ereignis-IDs zu protokollieren und bekannte zu überspringen. Es weist auch darauf hin, dass ein einzelner zugrunde liegender Vorgang zwei unterschiedliche Event-Objekte erzeugen kann. Erkennbar sind sie an der Objekt-ID in `data.object` zusammen mit dem Ereignistyp. GitHubs Header `X-GitHub-Delivery` bleibt bei einer erneuten Zustellung gleich. Die Verarbeitung soll idempotent sein, sodass eine zweite Zustellung nichts ändert.

Eine abgefangene Zustellung wird erneut abgespielt. Eine gültige Signatur bleibt gültig. Wer eine Zustellung abfängt, kann sie daher noch einmal senden. Deshalb wird der Zeitstempel mit signiert: Die Spezifikation verlangt, einen `webhook-timestamp` außerhalb der zulässigen Toleranz zurückzuweisen. Stripes Bibliotheken verwenden standardmäßig fünf Minuten. Stripe warnt, dass die Toleranz 0 die Prüfung vollständig ausschaltet, und empfiehlt NTP für eine genaue Serveruhr. Sowohl Stripe als auch die Spezifikation versehen jeden erneuten Versuch mit einem neuen Zeitstempel. Das Alter wird damit ab diesem Versuch und nicht ab dem ursprünglichen Ereignis gemessen; ein verzögerter Versuch kann trotzdem außerhalb der Toleranz liegen. Bewahren Sie bekannte Kennungen mindestens so lange wie die Toleranz auf. Dann wird jede Wiederholung innerhalb dieses Zeitfensters als Duplikat erkannt.

Ereignisse kommen in vertauschter Reihenfolge an. Stripes Beispiel: Beim Anlegen eines Abonnements können `customer.subscription.created`, `invoice.created`, `invoice.paid` und, falls eine Belastung erfolgt, `charge.created` entstehen. Nichts garantiert diese Reihenfolge bei der Zustellung. Das Feld `created` von Stripes Snapshot-Ereignissen hat eine Auflösung von einer Sekunde. Mehrere Ereignisse können daher denselben Zeitstempel tragen. Stripe rät davon ab, ihn zum Sortieren oder Erkennen von Duplikaten zu verwenden. Ein robuster Ansatz nimmt ein Ereignis als Hinweis auf eine Änderung und ruft vor jeder weiteren Handlung den aktuellen Zustand des Objekts über die API ab.

Der Handler ist zu langsam. GitHub erwartet innerhalb von 10 Sekunden eine 2xx-Antwort. Danach beendet es die Verbindung und wertet die Zustellung als gescheitert. Die Spezifikation empfiehlt ein Anfragezeitlimit zwischen 15 und 30 Sekunden. Erledigt der Handler die eigentliche Arbeit sofort im Anfrageablauf, überschreitet er das Zeitlimit; der Producer versucht es erneut, und die Arbeit läuft zweimal. Stripe und GitHub empfehlen beide eine Warteschlange. Stripe nennt die Lastspitze am Monatsanfang, wenn alle Abonnements verlängert werden.

Nur 2xx zählt. Alles andere gilt sowohl nach der Spezifikation als auch bei Stripe als Fehler, einschließlich 3xx. Bei einem Endpunkt hinter einer Weiterleitung scheitert deshalb jeder Zustellversuch, bis seine hinterlegte URL aktualisiert wird. Einigen Statuscodes gibt die Spezifikation eine Bedeutung für den Producer: `410 Gone` heißt, den Endpunkt zu deaktivieren; 429, 502 und 504 bedeuten, langsamer zu werden. Ein `retry-after`-Header soll den nächsten Versuch beeinflussen.

Für den Producer ist die URL eine potenziell feindliche Eingabe. Kunden können jede beliebige URL angeben, die die Server des Producers dann aufrufen. Die Spezifikation bezeichnet Webhook-Systeme als besonders anfällig für Server-Side Request Forgery (SSRF): Eine URL mit interner Adresse bringt den Producer dazu, sein eigenes Netz aufzurufen. Sie empfiehlt, jede Webhook-Anfrage über einen Proxy zu senden, der interne IP-Adressen filtert, etwa Stripes smokescreen. Die Webhook-Worker sollen in einem privaten Subnetz laufen, das keine internen Dienste erreichen kann.

Weiterführende Lektüre

- Standard Webhooks specification — Version 1.0.0: Aufbau der Nutzdaten, Signaturverfahren, Header, Wiederholungsplan, Statuscodes und SSRF.
- Receive Stripe events in your webhook endpoint — schlanke und Snapshot-Nutzdaten, Wiederholungen, doppelte und vertauschte Ereignisse sowie Schutz vor dem erneuten Abspielen.
- Best practices for using webhooks — GitHubs Empfehlungen: wenige Ereignisse, ein Webhook-Geheimnis, HTTPS, Antwort innerhalb von 10 Sekunden, erneute Zustellung und `X-GitHub-Delivery`.
- Validating webhook deliveries — `X-Hub-Signature-256`, Vergleich mit konstanter Laufzeit und ein Testbeispiel für eine Prüffunktion.

Glossar

Für Ihre tatsächlichen Rahmenbedingungen auswählen

Beschreiben Sie den Arbeitsablauf, die vorhandenen Schnittstellen und die Nutzer, die die Veränderung benötigen. Wir können eine prüfbare Lieferung definieren und die Übergabe so gestalten, dass Ihr Team die gelieferten Bestandteile warten kann.

Preise ansehen Kontakt aufnehmen →

Die Begriffe praktisch nutzbar machen

Fragen Sie, wie „Was ist ein Webhook?“ auf Ihre Organisation zutrifft. Ein konkretes Beispiel aus Ihrem bestehenden System reicht aus, um das Gespräch zu beginnen.

Unsere Leistungen entdecken

welcome@engineer.company +45 50 10 61 84 Kopenhagen, Dänemark

Nehmen Sie diese Seite mit

Kontaktkarte

Mit Sorgfalt und Liebe gestaltet von D.

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

Copyright © 2025 – heute · Engineer ApS · Alle Rechte vorbehalten.

Gedruckt von https://engineer.company/de/glossary/webhook/