Engineer

Webhook

📢 イベント発生時にサービスが送るHTTPリクエスト

どのようなものか

Webhookは、あるサービスで何かが起きたときに、そのサービスが別のサービスへ送るHTTPリクエストです。支払いが通った、請求書の代金が支払われた、時間のかかる処理が終わった、といった出来事を伝えます。受信側はあらかじめURLを登録し、送信側は各イベントの内容をそのURLに送信します。Standard Webhooks仕様は、Webhookを「HTTPコールバックの一般的な呼び名」とし、サービス同士がイベントを通知するための経路と説明しています。英語ではwebhookと2語で書かれることもありますが、Stripe、GitHub、この仕様はいずれも1語で書きます。

仕様はまた、WebhookをサービスのAPIの一部、いわば逆向きに動くAPIと説明しています。通常は、何かを必要とするクライアントがサービスを呼び出します。Webhookでは、知らせがあるサービスがクライアントを呼び出します。典型的なAPI連携では、片方がAPI呼び出しを、もう片方がWebhookを受け付けるため、双方がHTTPサーバーを動かします。ただし、独自のリクエストAPIを提供せずにWebhookだけを送るサービスもあります。送信側をプロデューサー、受信側をコンシューマーと呼び、受信側が登録するURLをWebhookエンドポイントと呼びます。

仕様はJSON本文を持つHTTP POSTを推奨していますが、特定のペイロード形式や構造を必須にはしていません。推奨する本文には、ピリオドで階層を区切る `type`（`invoice.paid` など）、イベント発生時刻をISO 8601で表す `timestamp`、`data` 配下または最上位に置くイベントデータが含まれます。追加のメタデータも、どちらに置いても構いません。配信のメタデータはヘッダーに置きます。`webhook-id` がイベントを識別し、`webhook-timestamp` が今回の配信試行の時刻をUnix秒で記録し、`webhook-signature` が真正性を確認するための署名になります。StripeやGitHubは、`Stripe-Signature` や `X-Hub-Signature-256` など、それぞれ独自のヘッダー名を使い、検証規則も異なります。Stripeは試行時刻と本文をまとめて署名しますが、GitHubのHMACが対象とするのは本文だけです。

ペイロードには、変更されたオブジェクト全体を含むfull形式と、識別子程度だけを含め、残りを受信側がAPIから取得するthin形式があります。仕様は両者の長短を比較しています。full形式なら、受信側は追加のAPI呼び出しを省けます。thin形式なら生成・送信するデータが少なく、柔軟で、後の変更にも余裕があります。thin形式をfull形式に広げることはできても、その逆はできないためです。また、必要なデータを受信側が明示的に要求するので、データへのアクセスを監査可能にできます。いずれの場合も、仕様はペイロードを小さく、通常は20 kB未満に抑えるよう勧めています。Stripeもthin形式を提供し、推奨としています。

概念を判断に生かす

「Webhookとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 データの保存先、処理のルール、他のシステムとの接続方法など、利用者の目に見えない部分をつくります。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 ETL/ELTパイプライン6本を設計・運用 — 6 3種類のDBエンジンにまたがるETLパイプライン

ETL/ELTパイプライン6本を設計・運用 お問い合わせ →

なぜ重要なのか

Webhookがなければ、別の場所での変更を知る必要があるシステムは、ポーリングで繰り返し問い合わせることになります。たいていは何も起きていないという答えを受け取り、変更があっても次の問い合わせまで分かりません。Webhookは、この向きを逆にします。イベントを知っているサービスが、その発生時に1回呼び出します。決済事業者、コードホスティングサービス、メール事業者は、この方法で、それらを利用するシステムに結果を通知します。

逆向きにすることには負担があり、その大半は受信側が負います。エンドポイントは、インターネット上の誰でもPOSTできる公開URLです。届いたリクエストだけでは送信者は分からないため、仕様は、安全な実装では真正性の検証を必須としています。配信は、受信側が2xxを返すか配信方針に従って断念するまで再試行されるので、同じイベントが複数回届くことがあります。Stripeも、エンドポイントに重複が届く場合があると警告しています。順序も保証されません。Stripeは、イベントが生成された順に届くことを保証しないと明記しています。

一方、送信側は試行を続けなければなりません。仕様は、指数バックオフとランダムなジッターを使い、数日にわたって再試行する日程を推奨しています。例では10回試行し、最後は最初の試行から75時間35分5秒後です。Stripeは、本番モードで最大3日間再試行します。その期間内に受信側が復旧すれば、自動的に追

い付きます。期間を過ぎた場合でも、Stripeではイベント作成後15日間は手動で、CLIなら30日間は再送できます。長期間失敗が続く場合、仕様は、メールなど別の経路で受信側に知らせ、エンドポイントを無効にするよう勧めています。

実際の活用例

次は、仕様の形式に従った配信ヘッダーです。識別子、タイムスタンプ、署名は仕様自身の例の値ですが、`Host` の値とパスは架空のものです。

```
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=
```

本文は次のとおりです。同じ仕様のthin形式のペイロードを、ここでは読みやすく整形しています。仕様は、実際に送る際には余分な空白を省くよう推奨しています。

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

`v1` という接頭辞は、共通鍵を使う署名を表します。イベント識別子、タイムスタンプ、本文をピリオドでつなぎ、エンドポイント固有のシークレットを使ってHMAC-SHA256を計算します。識別子とタイムスタンプも本文と一緒に署名されるため、どちらも署名を無効にせずに変更することはできません。仕様は `v1a` で示す非対称鍵のed25519署名も認めており、送信側と受信側の安全管理を別々の主体が担う場合に推奨しています。

GitHubも配信にHMAC-SHA256で署名し、`X-Hub-Signature-256` に、`sha256=` を先頭に付けた16進表記のダイジェストを入れます。また、検証器をテストするためのシークレット、ペイロード、署名を公開しています。次のPythonの検証器は、その例との一致を確認したものです。

```
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
```

本文の末尾に空白が1つ増えるだけで検証に失敗します。`hmac.compare_digest`は、文字列の不一致が先頭付近にあっても末尾付近にあっても、同じ時間で比較します。GitHubの文書は、単純な`==`で署名を比較しないよう求めています。その後の適切なハンドラーは、決まった順序で処理します。生のバイト列を読み、そのバイト列そのものに対する署名を検証し、送信側が試行時刻も署名しているなら、それが十分に新しいことを確認します。次に、このイベント識別子を受信済みか確認し、イベントをキューに入れ、2xxを返します。本来の処理は、その後に行います。Stripeは、会計システムで顧客の請求書を支払い済みにする前に200を返す例を挙げています。

このサイトの実績には、受信側としてWebhookを扱うプロジェクトが2つあります。Stripeとアプリ内購入の利用権限管理では、サブスクリプションは、同じWebhookが2回届くことや、順序が入れ替わって届くことにも耐えなければなりません。自動切り替えと配信記録を備えたメールシステムでは、受信Webhookが、各メール事業者がメッセージをどう扱ったかを通知します。送った内容と返ってきた通知の両方を、送信ログと配信イベントのテーブルに記録します。

問題が起きるところ

署名が一度も一致しない。 よくある原因は、受信側のフレームワークがJSON本文を解析し、コードがHMACを計算する前に再びシリアライズすることです。仕様も、これを非常によくある失敗としています。JSONのシリアライズ方法は実装ごとに異なり、余計な空白が1つあるだけで署名が無効になります。同じ理由で、Stripeも生の本文を必要とし、手を加えると検証に失敗すると警告しています。GitHubはさらに、エンドポイントの前にあるプロキシやロードバランサー

も、ペイロードやヘッダーを変更してはならず、ペイロードはUTF-8として扱うべきだとしています。何かが解析する前に、届いたバイト列そのものから署名を計算します。

同じイベントが2回届く。 再試行でもネットワークの障害でも重複は起きます。仕様では、同じイベントの再試行では `webhook-id` を変えず、冪等キーとして使います。例では、識別子をRedisに5分間保存しています。Stripeは、処理したイベントIDを記録し、処理済みなら再び処理しないよう求めています。また、1つの出来事から別々のEventオブジェクトが2つ生成されることもあり、その場合は `data.object` のオブジェクトIDとイベントの種類を組み合わせて識別できるとしています。GitHubの `X-GitHub-Delivery` ヘッダーも、再配信時に同じ値を保ちます。2回目の配信では何も変わらない、冪等な処理にします。

記録された配信が再生される。 有効な署名は有効なままなので、1回の配信を傍受した攻撃者は、それを再送できます。そのため、タイムスタンプにも署名します。仕様は、許容範囲を外れた `webhook-timestamp` を拒否するよう求めています。Stripeのライブラリーでは、設定を変えなければ許容範囲は5分です。Stripeは、0を指定すると検査自体が無効になると警告し、サーバーの時計を正確に保つためにNTPを推奨しています。Stripeも仕様も、再試行のたびに新しいタイムスタンプを付けるため、古さを測る起点は元のイベントではなく、その試行です。遅れた試行なら、それでも許容範囲を外れることがあります。受信済み識別子を少なくとも許容期間と同じ長さだけ保管すれば、期間内の再生は重複として検出できます。

イベントの到着順が入れ替わる。 Stripeの例では、サブスクリプションの作成によって `customer.subscription.created` 、 `invoice.created` 、 `invoice.paid` が生成され、課金があれば `charge.created` も生成されますが、この順で届く保証はありません。Stripeのスナップショットイベントの `created` フィールドは秒単位で、複数イベントが同じタイムスタンプを持つことがあります。Stripeは、これを順序の判定や重複検出に使わないよう求めています。堅牢な方法は、イベントを何かが変わったという通知として扱い、動作する前にAPIからオブジェクトの現在の状態を取得することです。

ハンドラーが遅すぎる。 GitHubは10秒以内の2xxを要求し、それを過ぎると接続を終了して配信失敗と数えます。仕様が推奨するリクエストのタイムアウトは、15秒から30秒の間です。本来の処理をハンドラー内でそのまま行うと、タイムアウトし、送信側が再試行して、処理が2回走ります。StripeもGitHubもキューを推奨しています。Stripeは、すべてのサブスクリプションが更新される月初の集中を例に挙げています。

成功は2xxだけ。 仕様でもStripeでも、それ以外は3xxも含めて失敗です。そのため、エンドポイントをリダイレクト先に移しただけでは、登録URLを更新するまで毎回失敗します。仕様は、一部のコードごとに送信側が取るべき対応を定めています。`410 Gone` はエンドポイントの無効化、429、502、504は送信を遅くする合図で、`retry-after` ヘッダーも次の試行時刻に反映すべきものです。

送信側にとってURLは敵対的な入力になる。 顧客は好きなURLを入力でき、送信側のサーバーがそこを呼び出します。仕様は、Webhookシステムは特にサーバーサイドリクエストフォージェリ（SSRF）の影響を受けやすいとしています。内部アドレスを指すURLによって、送信側に自身のネットワーク内を呼び出させる攻撃です。対策として、Stripeのsmokescreenのように内部IPアドレスを除外するプロキシにすべてのWebhookリクエストを通し、Webhookワーカーを内部サービスに到達できないプライベートサブネットで動かすよう勧めています。

参考資料

- Standard Webhooks仕様 — バージョン1.0.0。ペイロードの構造、署名方式、ヘッダー、再試行の日程、ステータスコードの扱い、SSRF。
- WebhookエンドポイントでStripeイベントを受信する — thin形式とスナップショット形式のペイロード、再試行、重複と順序の入れ替わり、再生への対策。
- Webhook利用のベストプラクティス — GitHubの推奨事項。イベントを絞ること、Webhookのシークレット、HTTPS、10秒以内の応答、再配信、`X-GitHub-Delivery`。
- Webhook配信の検証 — `X-Hub-Signature-256`、定時間比較、検証器を照合するためのテスト例。

用語集

実際の制約に合わせて選ぶ

業務の流れ、既存のインターフェース、変更を必要とする利用者をお知らせください。検証可能な納品内容と、お客様のチームが保守を続けられる引き継ぎ内容を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「Webhookとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/webhook/