Engineer

вебхук

📢 HTTP-запит, який сервіс надсилає, коли стається подія

Що це таке

Вебхук — це HTTP-запит, який один сервіс надсилає іншому тієї миті, коли в нього щось стається: проходить платіж, оплачено рахунок, завершилося довге завдання. Сторона-отримувач заздалегідь реєструє URL, а відправник надсилає туди опис кожної події методом POST. Специфікація Standard Webhooks називає вебхуки «a common name for HTTP callbacks» і описує їх як спосіб, у який сервіси повідомляють одне одного про події. Трапляється також написання через дефіс, веб-хук, а в англомовній документації Stripe, GitHub і специфікації слово пишуть латиницею й разом: webhook.

Специфікація також описує вебхук як частину API сервісу, такий собі API, що працює навпаки. Зазвичай клієнт звертається до сервісу, коли йому щось потрібно. З вебхуком сервіс звертається до клієнта, коли має новини. У типовій API-інтеграції обидві сторони запускають HTTP-сервери: одна для отримання викликів API, інша — для вебхуків. Сервіс може також надсилати вебхуки, не пропонуючи власного API для запитів. Відправника специфікація називає producer, отримувача — consumer, а URL, який реєструє отримувач, — це його кінцева точка для вебхуків (webhook endpoint).

Специфікація рекомендує HTTP POST із тілом JSON, але не вимагає певного формату чи структури payload. У запропонованому тілі є `type`, записаний через крапки й упорядкований ієрархічно, наприклад `invoice.paid`; `timestamp` — час, коли подія сталася, в ISO 8601; і дані події або в `data`, або на верхньому рівні. Додаткові метадані можуть бути в обох місцях. Метадані доставки йдуть у заголовках: `webhook-id` ідентифікує подію, `webhook-timestamp` фіксує час цієї спроби в секундах Unix, а `webhook-signature` підтверджує її автентичність. Stripe і GitHub мають власні назви заголовків, серед них `Stripe-Signature` і `X-Hub-Signature-256`, а правила перевірки в них різні: Stripe підписує мітку часу спроби разом із тілом, тоді як HMAC GitHub охоплює лише тіло.

Payload буває повним, коли містить увесь змінений об’єкт, або тонким, коли в ньому мало що є, крім ідентифікатора, і решту отримувач сам забирає через API. Специфікація зважує обидва варіанти. Повний payload заощаджує отримувачу додаткові виклики API. Тонкий означає менше даних для формування й надсилання, більше гнучкості й більше простору для змін надалі, бо тонкий payload завжди можна розширити до повного, а навпаки — ні. До того ж він робить доступ до даних придатним для аудиту, бо отримувач мусить явно запитати те, що читає. Хай там як, специфікація радить тримати payload невеликим, зазвичай до 20 кБ. Stripe пропонує тонкий формат і позначає його як рекомендований.

Застосуйте поняття до рішення

Застосуйте «Що таке вебхук?» до конкретного рішення. Ми можемо пов'язати поняття із системою, яку ви створюєте або купуєте. Та частина, якої користувач не бачить: де зберігаються дані, які діють правила і як інші системи можуть із нею взаємодіяти.

Переглянути наші послуги Зв'язатися з нами →

Перевірте припущення за терміном

Використайте пояснення та пов'язані роботи, щоб сформулювати запитання для проєкту. Ми можемо перевірити припущення разом до вибору інструмента. Спроєктувала й підтримувала 6 пайплайнів ETL/ELT — 6 пайплайнів ETL на трьох рушіях баз даних

Спроєктувала й підтримувала 6 пайплайнів ETL/ELT Зв'язатися з нами →

Чому це важливо

Без вебхуків система, якій треба знати про зміну деінде, мусить вдаватися до опитування (polling): питати знову й знову, здебільшого лише щоб почути, що нічого не сталося, і все одно дізнатися про зміну тільки під час наступного опитування. Вебхук розвертає це. Сервіс, що знає про подію, звертається один раз — тоді, коли вона стається. Саме так платіжні провайдери, сервіси хостингу коду й поштові провайдери повідомляють системи, побудовані на них.

Цей розворот має ціну, і здебільшого її платить сторона-отримувач. Її кінцева точка — публічний URL, на який будь-хто в інтернеті може надіслати POST, і ніщо у вхідному запиті не показує, хто його надіслав, тому специфікація робить перевірку автентичності обов’язковою для будь-якої безпечної реалізації. Доставку повторюють, доки отримувач не відповість статусом 2xx або політика доставки не припинить спроби, тож та сама подія може надійти більше ніж один раз; Stripe попереджає, що кінцеві точки зрідка можуть отримувати дублікати. Порядок теж не гарантовано: Stripe прямо пише, що не гарантує надходження подій у тому порядку, в якому їх згенеровано.

Відправник, зі свого боку, мусить і далі пробувати. Специфікація рекомендує розклад повторних спроб на кілька днів з експоненційною затримкою і невеликим випадковим розкидом (jitter). У її прикладі десять спроб, остання — через 75 годин, 35 хвилин і 5 секунд після першої. Stripe повторює спроби до трьох днів у live mode. Отримувач, який повернувся в межах цього вікна, наздожене все сам; після нього Stripe дає змогу вручну повторно надіслати подію впродовж 15 днів після її створення або впродовж 30 днів через свій CLI. Якщо доставка довго не вдається, специфікація радить відправникові повідомити отримувача іншим каналом, наприклад електронною поштою, і вимкнути кінцеву точку.

Як це виглядає на практиці

Ось заголовки доставки у форматі специфікації. Ідентифікатор, мітка часу й підпис — це приклади значень із самої специфікації; значення `Host` і шлях вигадані:

```
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=
```

А ось тіло, тонкий payload з тієї ж специфікації, тут відформатований для читабельності. Під час реального надсилання специфікація радить передавати його мініфікованим:

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

Префікс `v1` позначає симетричний підпис: HMAC-SHA256 від ідентифікатора події, мітки часу й тіла, з’єднаних крапками, обчислений із секретним ключем, унікальним для кінцевої точки. Оскільки ідентифікатор і мітку часу підписують разом із тілом, жодне з них не можна змінити, не

зламавши підпис. Специфікація допускає й асиметричні підписи ed25519, позначені `v1a`, і рекомендує їх, коли ви не контролюєте безпеку і відправника, і отримувача.

GitHub теж підписує свої доставки HMAC-SHA256, кладе шістнадцятковий дайджест у `X-Hub-Signature-256` із префіксом `sha256=` і публікує секрет, payload і підпис, на яких можна перевірити верифікатор. Ось верифікатор на Python, перевірений саме на цьому прикладі:

```
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
```

Одного пробілу в кінці тіла достатньо, щоб перевірка не пройшла. `hmac.compare_digest` працює однаковий час, хоч рядки розходяться на початку, хоч у кінці; документація GitHub наголошує, що підписи ніколи не можна порівнювати звичайним `==`. Надійний обробник далі діє в чіткому порядку: прочитати сирі байти, перевірити підпис саме над цими байтами, переконатися, що мітка часу спроби не застара, якщо відправник її підписує, перевірити, чи не траплявся вже цей ідентифікатор події, поставити подію в чергу й відповісти 2xx. Справжня робота відбувається потім. Stripe наводить приклад: повернути 200, перш ніж позначати рахунок клієнта як оплачений у бухгалтерській системі.

Два проєкти з портфоліо цього сайту мають справу з вебхуками з боку отримувача. У проєкті про права доступу через Stripe та in-app підписка, серед іншого, мусить пережити вебхук, що прийшов двічі, і вебхук, що прийшов не в тому порядку. У поштовій системі з failover і журналом

доставки вхідні вебхуки повідомляють, що кожен провайдер зробив із листом, а проєкт фіксує і те, що надіслано, і те, що повернулося, — у журналі надсилань і в таблиці подій доставки.

Де виникають проблеми

Підпис ніколи не збігається. Типова причина: фреймворк отримувача розбирає JSON-тіло, а код знову серіалізує його, перш ніж обчислити HMAC. Специфікація називає це дуже поширеною помилкою: серіалізація JSON відрізняється в різних реалізаціях, і один зайвий пробіл робить підпис недійсним. Stripe потребує сирого тіла з тієї ж причини й попереджає, що будь-яка зміна в ньому зриває перевірку. GitHub додає, що проксі чи балансувальник навантаження перед кінцевою точкою не повинні змінювати payload або заголовки і що payload слід обробляти як UTF-8. Обчислюйте підпис над байтами точно такими, якими вони надійшли, ще до того, як щось їх розбере.

Та сама подія надходить двічі. Дублікати дають і повторні спроби, і збої мережі. За специфікацією `webhook-id` лишається тим самим під час кожної повторної спроби для події й слугує ключем ідемпотентності; її приклад — тримати ідентифікатори в Redis п'ять хвилин. Stripe радить записувати ідентифікатори оброблених подій і пропускати ті, що вже траплялися. Stripe також зазначає, що іноді створюються й надсилаються два окремі об'єкти Event; їх можна розпізнати за ідентифікатором об'єкта в `data.object` разом із типом події. Заголовок `X-GitHub-Delivery` від GitHub лишається тим самим, коли доставку надсилають повторно. Обробка має бути ідемпотентною, щоб друга доставка нічого не змінювала.

Перехоплену доставку відтворюють повторно. Дійсний підпис лишається дійсним, тож зловмисник, який перехопив одну доставку, може надіслати її знову. Саме тому мітку часу підписують: специфікація каже відхиляти `webhook-timestamp`, що виходить за межі допустимого відхилення. Бібліотеки Stripe за замовчуванням допускають п'ять хвилин, Stripe попереджає, що відхилення 0 повністю вимикає перевірку, і рекомендує NTP, щоб годинник сервера йшов точно. І Stripe, і специфікація ставлять нову мітку часу на кожну повторну спробу, тож вік відлічують від цієї спроби, а не від початкової події; затримана спроба все ще може вийти за межі допустимого відхилення. Тримайте вже бачені ідентифікатори щонайменше стільки, скільки триває допустиме відхилення, — і все, що відтворять повторно в межах цього вікна, буде виявлено як дублікат.

Події надходять не в тому порядку. Приклад від Stripe: створення підписки може породити `customer.subscription.created`, `invoice.created`, `invoice.paid` і, якщо є списання, `charge.created`, і ніщо не гарантує, що вони надійдуть саме в такому порядку. Знімкові події (snapshot events) Stripe записують `created` у цілих секундах, тож різні події можуть мати однакову мітку часу, і Stripe радить не використовувати її, щоб упорядковувати події чи виявляти дублікати. Надійний підхід — сприймати подію як сповіщення, що щось змінилося, і перед тим як діяти, отримувати поточний стан об'єкта через API.

Обробник надто повільний. GitHub очікує 2xx упродовж 10 секунд; після цього GitHub розриває з'єднання й зараховує доставку як невдалу. Специфікація рекомендує тайм-аут запиту десь від 15 до 30 секунд. Обробник, який виконує справжню роботу, перш ніж відповісти, натрапляє на тайм-аут, відправник повторює спробу, і робота виконується двічі. І Stripe, і GitHub рекомендують чергу. Stripe вказує на пік на початку місяця, коли поновлюються всі підписки.

Зараховується лише 2xx. Усе інше вважається невдачею, зокрема й 3xx, і для специфікації, і для Stripe, тож кінцева точка, яку перенесли за переспрямування, провалює кожну доставку, доки зареєстрований URL не оновлять. Кільком кодам специфікація надає значення для відправника: `410 Gone` означає, що кінцеву точку треба вимкнути, 429, 502 чи 504 — що слід сповільнитися, а заголовок `retry-after` має визначати наступну спробу.

Для відправника URL — ворожі вхідні дані. Клієнти можуть указати будь-який URL, і сервери відправника його викличуть. Специфікація називає системи вебхуків особливо вразливими до підробки запитів на боці сервера (server-side request forgery, SSRF): URL, що вказує на внутрішню адресу, змушує відправника звертатися до власної мережі. Її порада — пропускати всі запити вебхуків через проксі, який відфільтровує внутрішні IP-адреси, як-от smokescreen від Stripe, і запускати воркери вебхуків у приватній підмережі, яка не має доступу до внутрішніх сервісів.

Що почитати далі

- Standard Webhooks specification — версія 1.0.0: структура payload, алгоритм підпису, заголовки, розклад повторних спроб, обробка кодів стану й SSRF.

- Receive Stripe events in your webhook endpoint — тонкі та знімкові payload, повторні спроби, дублікати, події не в тому порядку й захист від повторного відтворення.
- Best practices for using webhooks — список GitHub: підписка на мінімум подій, секрет вебхука, HTTPS, відповідь упродовж 10 секунд, повторна доставка й `X-GitHub-Delivery`.
- Validating webhook deliveries — `X-Hub-Signature-256`, порівняння за сталий час і тестовий приклад, на якому можна перевірити верифікатор.

Словник

Обирайте за фактичними умовами

Опишіть робочий процес, наявні інтерфейси та користувачів, яким потрібна зміна. Ми можемо визначити перевірюваний результат і спосіб передачі, який дасть вашій команді змогу підтримувати його.

Дивитися, скільки це коштує Зв'язатися з нами →

Зробіть термінологію практичною

Запитайте, як «Що таке вебхук?» стосується вашої організації. Конкретного прикладу з поточної системи достатньо для початку розмови.

Переглянути наші послуги

welcome@engineer.company +45 50 10 61 84 Копенгаген, Данія

Візьміть цю сторінку із собою

Візитівка

Дбайливо зроблено з любов'ю — D.

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

Авторське право © 2025 – дотепер · Engineer ApS · Усі права захищено.

Надруковано з https://engineer.company/uk/glossary/webhook/