Engineer

webhook

📢 petición HTTP que envía un servicio cuando ocurre un evento

Qué es

Un webhook es una petición HTTP que un servicio envía a otro en el momento en que ocurre algo en su lado: se procesa un pago, se abona una factura o termina una tarea larga. El receptor registra de antemano una URL y el emisor publica en ella una descripción de cada evento. La especificación Standard Webhooks presenta «webhook» como «un nombre común para las llamadas de retorno HTTP» y describe estos mecanismos como un canal por el que un servicio anuncia un evento a otro. A veces el término se escribe en dos palabras, web hook, pero Stripe, GitHub y la especificación lo escriben en una sola.

La especificación también describe un webhook como parte de la API de un servicio, una especie de API al revés. Normalmente el cliente llama al servicio cuando quiere algo. Con un webhook, el servicio llama al cliente cuando tiene novedades. En una integración habitual de API, ambos lados ejecutan servidores HTTP: uno para recibir llamadas de API y otro para recibir webhooks. Un servicio también puede enviar webhooks sin ofrecer su propia API de peticiones. El emisor se llama productor, el receptor consumidor, y la URL que registra el receptor es su punto de recepción de webhooks.

La especificación recomienda un POST HTTP con un cuerpo JSON, pero no exige un formato ni una estructura concretos para la carga útil. El cuerpo que propone incluye un `type`, escrito con puntos y agrupado jerárquicamente, como `invoice.paid`; un `timestamp` del momento en que ocurrió el evento, en ISO 8601; y los datos del evento bajo `data` o en el nivel superior. Puede haber metadatos adicionales en ambos lugares. Los metadatos de entrega van en las cabeceras: `webhook-id` identifica el evento, `webhook-timestamp` registra cuándo se hizo este intento, en segundos Unix, y `webhook-signature` lo autentica. Stripe y GitHub usan sus propios nombres de cabecera, entre ellos `Stripe-Signature` y `X-Hub-Signature-`

256 , y sus reglas de verificación son distintas: Stripe firma una marca temporal del intento junto con el cuerpo, mientras que el HMAC de GitHub abarca solo el cuerpo.

La carga útil puede ser completa e incluir todo el objeto modificado, o reducida y llevar poco más que un identificador, dejando que el receptor obtenga el resto mediante la API. La especificación sopesa ambas opciones. Una carga completa ahorra llamadas adicionales a la API. Una reducida supone menos datos que generar y enviar, más flexibilidad y más margen para cambios posteriores: siempre puede ampliarse hasta ser completa, pero no al revés. También permite auditar el acceso a los datos, porque el receptor debe pedir expresamente lo que lee. En cualquier caso, la especificación aconseja cargas pequeñas, normalmente inferiores a 20 kB. Stripe ofrece un formato reducido y lo señala como el recomendado.

Utilice el concepto en una decisión

Aplique «¿Qué es un webhook?» a una decisión real. Podemos ayudarle a relacionar el concepto con el sistema que está construyendo o comprando. La parte que el usuario no ve: dónde se guardan los datos, qué reglas se aplican y cómo pueden comunicarse otros sistemas con ella.

Explore nuestros servicios Póngase en contacto →

Compruebe los supuestos detrás del término

Utilice la explicación y los trabajos enlazados para formular preguntas sobre su proyecto. Podemos examinar los supuestos con usted antes de elegir una herramienta. Diseño y operación de 6 flujos ETL/ELT — 6 flujos ETL en tres motores de bases de datos

Diseño y operación de 6 flujos ETL/ELT Póngase en contacto →

Por qué importa

Sin webhooks, un sistema que necesita conocer un cambio en otro lugar tiene que realizar sondeos periódicos: preguntar una y otra vez, casi siempre para oír que no ha ocurrido nada, y aun así enterarse del cambio solo en el siguiente sondeo. Un webhook invierte el proceso. El servicio que conoce el evento llama una vez, cuando ocurre. Así informan los proveedores de pagos, alojamiento de código y correo electrónico a los sistemas construidos sobre ellos.

Invertir la dirección tiene costes, y la mayoría recaen en el receptor. Su punto de recepción es una URL pública a la que cualquiera en internet puede enviar un POST; nada en una petición entrante demuestra quién la envió, de modo que la especificación exige verificar su autenticidad en cualquier implementación segura. La entrega se reintenta hasta que el receptor responde con un estado 2xx o la política de entrega deja de intentarlo. Por eso, el mismo evento puede llegar más de una vez; Stripe advierte de que los receptores pueden recibir

duplicados ocasionalmente. Tampoco se promete el orden: Stripe indica expresamente que no garantiza que los eventos lleguen en el orden en que se generaron.

El productor, por su parte, debe seguir intentándolo. La especificación recomienda un calendario de reintentos que abarque varios días, con espera exponencial y una pequeña variación aleatoria. Su ejemplo hace diez intentos; el último se realiza 75 horas, 35 minutos y 5 segundos después del primero. Stripe reintenta durante un máximo de tres días en modo real. Un receptor que vuelve a estar disponible dentro de ese plazo recibe automáticamente las entregas pendientes; después, Stripe permite reenviar un evento manualmente hasta 15 días después de su creación, o hasta 30 días mediante su CLI. Cuando la entrega falla durante mucho tiempo, la especificación recomienda que el productor avise al consumidor por otro canal, como el correo electrónico, y desactive el punto de recepción.

Cómo se aplica en la práctica

Estas son las cabeceras de una entrega con el formato de la especificación. El identificador, la marca temporal y la firma son los valores de ejemplo que usa la propia especificación; el valor de `Host` y la ruta son inventados:

```
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=
```

Este es el cuerpo, una carga reducida de la misma especificación, presentada aquí con sangría para facilitar la lectura. Para transmitirla, la especificación recomienda minificarla:

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

El prefijo `v1` indica una firma simétrica: HMAC-SHA256 sobre el identificador del evento, la marca temporal y el cuerpo unidos por puntos, con una clave secreta exclusiva del punto de recepción. Como el identificador y la marca temporal se firman junto con el cuerpo, ninguno puede cambiarse sin invalidar la firma. La especificación también permite firmas asimétricas ed25519, marcadas con `v1a`, y las recomienda cuando distintas partes se encargan de proteger al productor y al consumidor.

GitHub también firma sus entregas con HMAC-SHA256, coloca el resumen hexadecimal en `X-Hub-Signature-256` con el prefijo `sha256=` y publica un secreto, una carga útil y una firma para probar un verificador. Este verificador en Python se ha comprobado con ese ejemplo:

```
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
```

Basta un espacio al final del cuerpo para que falle. `hmac.compare_digest` tarda lo mismo tanto si las cadenas difieren al principio como al final; la documentación de GitHub indica que nunca se deben comparar firmas con un simple `==`. Un manejador sólido sigue después un orden fijo: leer los bytes sin procesar, verificar la firma sobre esos mismos bytes, comprobar que la marca temporal del intento es reciente si el productor firma una, comprobar si ya se ha visto ese identificador de evento, poner el evento en una cola y responder con un 2xx. El trabajo real se hace después. Stripe pone el ejemplo de devolver un 200 antes de marcar como pagada una factura del cliente en el sistema contable.

Dos proyectos del portafolio de este sitio trabajan con webhooks desde el lado receptor. En las autorizaciones para Stripe y compras dentro de aplicaciones, una suscripción debe soportar, entre otras cosas, un webhook que llega dos veces y otro que llega fuera de orden. En el sistema de correo con conmutación y registro de entregas, los webhooks entrantes informan de lo que cada proveedor hizo con un mensaje; el proyecto registra tanto lo enviado como lo recibido en un registro de envíos y una tabla de eventos de entrega.

Dónde surgen los problemas

La firma nunca coincide. Una causa habitual es que el framework receptor analice el cuerpo JSON y el código lo serialice otra vez antes de calcular el HMAC. La especificación lo considera un fallo muy común: la serialización JSON varía entre implementaciones, y un solo espacio de más invalida la firma. Stripe necesita el cuerpo sin procesar por la misma razón y advierte de que cualquier manipulación hace fallar la verificación. GitHub añade que un proxy o balanceador situado delante del receptor no debe modificar la carga útil ni las cabeceras, y que la carga debe tratarse como UTF-8. Calcule la firma sobre los bytes exactamente como llegaron, antes de que nada los analice.

El mismo evento llega dos veces. Tanto los reintentos como los problemas de red generan duplicados. Según la especificación, `webhook-id` no cambia entre los reintentos de un evento y sirve como clave de idempotencia; su ejemplo consiste en guardar los identificadores en Redis durante cinco minutos. Stripe indica que se registren los identificadores de eventos procesados y se omitan los ya vistos. También señala que un mismo hecho puede producir dos objetos Event distintos; se reconocen mediante el identificador del objeto en `data.object` junto con el tipo de evento. La cabecera `X-GitHub-Delivery` de GitHub no cambia cuando se repite una entrega. El procesamiento debe ser idempotente para que una segunda entrega no cambie nada.

Se reproduce una entrega interceptada. Una firma válida sigue siendo válida, así que un atacante que intercepte una entrega puede enviarla otra vez. Por eso se firma la marca temporal: la especificación indica que se rechace un `webhook-timestamp` fuera de la tolerancia permitida. Las bibliotecas de Stripe usan cinco minutos de forma predeterminada; Stripe advierte de que una tolerancia de 0 desactiva totalmente la comprobación y recomienda NTP para mantener preciso el reloj del servidor. Tanto Stripe como la especificación ponen una marca temporal nueva en cada reintento: la antigüedad se mide desde ese intento, no desde el evento original. Un intento retrasado puede quedar fuera de la

tolerancia. Conserve los identificadores vistos al menos durante el plazo de tolerancia, y cualquier reproducción dentro de ese plazo se detecta como duplicado.

Los eventos llegan fuera de orden. El ejemplo de Stripe: crear una suscripción puede producir `customer.subscription.created`, `invoice.created`, `invoice.paid` y, si hay un cargo, `charge.created`, sin garantía de que lleguen en ese orden. El campo `created` de los eventos de instantánea de Stripe tiene resolución de un segundo, por lo que varios eventos pueden compartir marca temporal; Stripe indica que no se use para ordenar eventos ni detectar duplicados. El enfoque sólido trata un evento como aviso de que algo cambió y obtiene el estado actual del objeto mediante la API antes de actuar.

El manejador es demasiado lento. GitHub espera un 2xx en 10 segundos; después cierra la conexión y considera fallida la entrega. La especificación recomienda un tiempo de espera de entre 15 y 30 segundos para la petición. Un manejador que hace el trabajo real antes de responder agota ese plazo, el productor reintenta y el trabajo se hace dos veces. Stripe y GitHub recomiendan una cola. Stripe señala el pico de principios de mes, cuando se renuevan todas las suscripciones.

Solo cuenta un 2xx. Tanto para la especificación como para Stripe, cualquier otro estado es un fallo, incluido un 3xx. Un punto de recepción que se ha trasladado detrás de una redirección falla en todas las entregas hasta que se actualiza su URL registrada. La especificación asigna significado a algunos códigos para el productor: `410 Gone` indica que debe desactivar el punto de recepción; un 429, 502 o 504, que debe reducir el ritmo; y una cabecera `retry-after` debe orientar el siguiente intento.

Para el productor, la URL es una entrada hostil. Los clientes pueden introducir cualquier URL y los servidores del productor la llamarán. La especificación considera los sistemas de webhooks especialmente vulnerables a la falsificación de peticiones del lado del servidor, SSRF: una URL que apunta a una dirección interna hace que el productor llame a su propia red. Su consejo es enviar todas las peticiones de webhook a través de un proxy que filtre las direcciones IP internas, como smokescreen de Stripe, y ejecutar los trabajadores de webhooks en una subred privada que no pueda alcanzar los servicios internos.

Lecturas adicionales

- Standard Webhooks specification — versión 1.0.0: estructura de la carga útil, método de firma, cabeceras, calendario de reintentos, estados HTTP y SSRF.
- Receive Stripe events in your webhook endpoint — cargas reducidas y de instantánea, reintentos, eventos duplicados o desordenados y protección frente a reproducciones.
- Best practices for using webhooks — lista de GitHub: pocos eventos, un secreto de webhook, HTTPS, respuesta en 10 segundos, repetición de entregas y `X-GitHub-Delivery`.
- Validating webhook deliveries — `X-Hub-Signature-256`, comparación en tiempo constante y un ejemplo para comprobar el verificador.

Glosario

Elija según sus restricciones reales

Describa el flujo de trabajo, las interfaces existentes y los usuarios que necesitan el cambio. Podemos definir una entrega comprobable y un traspaso que permita a su equipo mantener lo entregado.

Vea cuánto cuesta Póngase en contacto →

Haga práctica la terminología

Pregunte cómo se aplica «¿Qué es un webhook?» a su organización. Basta un ejemplo concreto de su sistema actual para iniciar la conversación.

Explore nuestros servicios

welcome@engineer.company +45 50 10 61 84 Copenhague, Dinamarca

Llévese esta página

Tarjeta de contacto

Creado con esmero y cariño por D.

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

Derechos de autor © 2025 – actualidad · Engineer ApS · Todos los derechos reservados.

Impreso desde https://engineer.company/es/glossary/webhook/