Engineer

webhook

📢 en HTTP-begäran som en tjänst skickar när något händer

Vad det är

En webhook är en HTTP-begäran som en tjänst skickar till en annan i samma ögonblick som något händer hos den: en betalning går igenom, en faktura blir betald, en lång uppgift blir klar. Den mottagande sidan registrerar en URL i förväg, och avsändaren skickar en beskrivning av varje händelse dit med POST. Specifikationen Standard Webhooks kallar webhooks ”a common name for HTTP callbacks” och beskriver dem som ett sätt för tjänster att meddela varandra om händelser. Ordet skrivs ibland som två ord, web hook, men Stripe, GitHub och specifikationen skriver det alla som ett ord.

Specifikationen beskriver också en webhook som en del av en tjänsts API, ett slags API som går baklänges. Normalt anropar klienten tjänsten när den vill ha något. Med en webhook anropar tjänsten klienten när den har något nytt att berätta. I en typisk API-integration kör båda sidor HTTP-servrar, den ena för att ta emot API-anrop och den andra för att ta emot webhooks. En tjänst kan också skicka webhooks utan att erbjuda ett eget API för förfrågningar. Specifikationen kallar avsändaren producer och mottagaren consumer, och den URL som mottagaren registrerar är dess webhook-slutpunkt.

Specifikationen rekommenderar en HTTP POST med en JSON-kropp men kräver inte något särskilt format eller någon särskild struktur för payloaden. Den föreslagna kroppen har en `type`, skriven med punkter och ordnad hierarkiskt, till exempel `invoice.paid`; en `timestamp` för när händelsen inträffade, i ISO 8601; och händelsedata antingen under `data` eller på översta nivån. Ytterligare metadata kan finnas på båda ställena. Leveransmetadata ligger i huvudfält: `webhook-id` identifierar händelsen, `webhook-timestamp` anger när just det här försöket gjordes, i Unix-sekunder, och `webhook-signature` bekräftar dess äkthet. Stripe och GitHub använder egna namn på huvudfälten, bland dem `Stripe-`

`Signature` och `X-Hub-Signature-256`, och deras verifieringsregler skiljer sig åt: Stripe signerar försökets tidsstämpel tillsammans med kroppen, medan GitHubs HMAC bara täcker kroppen.

En payload kan vara fullständig och innehålla hela det ändrade objektet, eller tunn och innehålla inte mycket mer än en identifierare, så att mottagaren själv får hämta resten via API:et. Specifikationen väger de två mot varandra. En fullständig payload besparar mottagaren extra anrop till API:et. En tunn payload innebär mindre data att generera och skicka, mer flexibilitet och mer utrymme för ändringar senare, eftersom en tunn payload alltid kan växa till en fullständig men inte tvärtom. Den gör också dataåtkomsten spårbar, eftersom mottagaren uttryckligen måste be om det den läser. Hur som helst rekommenderar specifikationen att payloads hålls små, oftast under 20 kB. Stripe erbjuder ett tunt format och anger det som det rekommenderade.

Använd begreppet i ett beslut

Använd ”Vad är en webhook?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

Utan webhooks måste ett system som behöver veta om en ändring någon annanstans polla: fråga om och om igen, för det mesta bara för att få höra att ingenting har hänt, och ändå inte upptäcka ändringen förrän vid nästa fråga. En webhook vänder på det. Tjänsten som känner till händelsen anropar en gång, i samma ögonblick som den inträffar. Det är så betalningsleverantörer, tjänster för kodhosting och e-postleverantörer rapporterar tillbaka till de system som är byggda ovanpå dem.

Den omvända riktningen har ett pris, och det mesta av det hamnar hos mottagaren. Dess slutpunkt är en offentlig URL som vem som helst på internet kan skicka en POST till, och ingenting i en inkommande begäran visar vem som skickade den, så specifikationen gör verifiering av äktheten till ett krav för varje säker implementation. Leveransen görs om tills mottagaren svarar med en 2xx-status eller leveranspolicyn stoppar fler försök, så samma händelse kan komma

mer än en gång; Stripe varnar för att slutpunkter ibland kan ta emot dubbletter. Ordningen är inte heller utlovad: Stripe skriver rakt ut att företaget inte garanterar att händelser kommer i den ordning de skapades.

Avsändaren måste för sin del fortsätta försöka. Specifikationen rekommenderar en tidsplan för omförsök som sträcker sig över flera dagar, med exponentiell backoff och lite slumpmässig jitter. I dess exempel görs tio försök, det sista 75 timmar, 35 minuter och 5 sekunder efter det första. Stripe gör omförsök i upp till tre dagar i live mode. En mottagare som är tillbaka inom det fönstret kommer ikapp av sig själv; därefter kan man hos Stripe skicka om en händelse manuellt i upp till 15 dagar efter att den skapades, eller i upp till 30 dagar med Stripes CLI. När leveransen fortsätter att misslyckas under lång tid rekommenderar specifikationen att avsändaren meddelar mottagaren via en annan kanal, till exempel e-post, och inaktiverar slutpunkten.

Så ser det ut i praktiken

Det här är huvudfälten i en leverans i specifikationens format. Identifieraren, tidsstämpeln och signaturen är de exempelvärden som specifikationen själv använder; värdet för `Host` och sökvägen är påhittade:

```
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=
```

Och det här är kroppen, en tunn payload från samma specifikation, här med indrag för läsbarhetens skull. När den faktiskt skickas rekommenderar specifikationen att den är minifierad:

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

Prefixet `v1` markerar en symmetrisk signatur: HMAC-SHA256 över händelsens identifierare, tidsstämpeln och kroppen, sammanfogade med punkter och beräknade med en hemlig nyckel som är unik för slutpunkten. Eftersom identifieraren och tidsstämpeln signeras tillsammans med kroppen kan ingen av dem ändras utan att signaturen blir ogiltig. Specifikationen tillåter också asymmetriska ed25519-signaturer, markerade `v1a`, och rekommenderar dem när man inte själv kontrollerar säkerheten på både avsändar- och mottagarsidan.

GitHub signerar också sina leveranser med HMAC-SHA256, lägger hex-digesten i `X-Hub-Signature-256` med prefixet `sha256=` och publicerar en hemlighet, en payload och en signatur att testa en verifierare mot. Här är en verifierare i Python, testad mot just det exemplet:

```
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
```

Ett enda mellanslag sist i kroppen räcker för att verifieringen ska misslyckas. `hmac.compare_digest` tar lika lång tid oavsett om strängarna skiljer sig tidigt eller sent; GitHubs dokumentation säger att man aldrig ska jämföra signaturer med en vanlig `==`. En robust handler arbetar sedan i en fast ordning: läs de råa bytena, verifiera signaturen över exakt de bytena, kontrollera att försökets tidsstämpel är tillräckligt ny om avsändaren signerar en sådan, kontrollera om händelsens identifierare har setts förut, lägg händelsen i en kö och svara med en 2xx. Det egentliga arbetet görs efteråt. Stripe ger som exempel att svara med 200 innan en kunds faktura markeras som betald i bokföringssystemet.

Två projekt i portföljen här på webbplatsen handlar om webhooks sedda från mottagarsidan. I projektet med åtkomsträttigheter via Stripe och appköp måste en prenumeration bland annat klara en webhook som kommer två gånger och en webhook som kommer i fel ordning. I e-postsystemet med failover och leveransloggning rapporterar inkommande webhooks vad varje leverantör gjorde med ett meddelande, och projektet registrerar både det som skickades och det som kom tillbaka, i en utskickslogg och en tabell för leveranshändelser.

Var det går fel

Signaturen stämmer aldrig. En vanlig orsak är att mottagarens ramverk parsar JSON-kroppen och att koden serialiserar den på nytt innan HMAC:en beräknas. Specifikationen kallar det ett mycket vanligt fel: JSON-serialisering skiljer sig mellan implementationer, och ett enda mellanslag för mycket gör signaturen ogiltig. Stripe behöver den råa kroppen av samma skäl och varnar för att varje ändring av den får verifieringen att misslyckas. GitHub tillägger att en proxy eller lastbalanserare framför slutpunkten inte får ändra payloaden eller huvudfälten, och att payloaden bör hanteras som UTF-8. Beräkna signaturen över bytena exakt som de kom, innan något parsar dem.

Samma händelse kommer två gånger. Både omförsök och nätverksproblem ger dubbletter. Enligt specifikationen är `webhook-id` densamma vid varje omförsök av en händelse och fungerar som idempotensnyckel; dess exempel är att spara identifierarna i Redis i fem minuter. Stripe uppmanar dig att registrera de händelse-ID:n du har behandlat och hoppa över dem du redan har sett. Stripe påpekar också att det i vissa fall skapas och skickas två separata Event-objekt; dem känner man igen på objektets ID i `data.object` tillsammans med händelsetypen. GitHubs huvudfält `X-GitHub-Delivery` förblir detsamma när en leverans skickas om. Behandlingen bör vara idempotent, så att en andra leverans inte ändrar någonting.

En uppsnappad leverans spelas upp igen. En giltig signatur förblir giltig, så en angripare som snappar upp en leverans kan skicka den igen. Därför signeras tidsstämpeln: specifikationen säger att en `webhook-timestamp` utanför en tillåten tolerans ska avvisas. Stripes bibliotek använder fem minuter som standard, Stripe varnar för att en tolerans på 0 stänger av kontrollen helt och rekommenderar NTP för att hålla serverns klocka exakt. Både Stripe och specifikationen sätter en ny tidsstämpel på varje omförsök, så åldern mäts från det försöket och inte från den ursprungliga händelsen; ett försenat försök kan fortfarande hamna utanför toleransen. Spara de identifierare du har sett minst lika länge som toleransen, så fångas allt som spelas upp igen inom fönstret som en dubblett.

Händelser kommer i fel ordning. Stripes exempel: när en prenumeration skapas kan det ge `customer.subscription.created`, `invoice.created`, `invoice.paid` och, om det finns en debitering, `charge.created`, och ingenting garanterar att de kommer i den ordningen. Stripes snapshot-händelser anger `created` i hela sekunder, så olika händelser kan ha samma tidsstämpel, och Stripe säger att man inte ska använda den för att ordna händelser eller upptäcka dubbletter. Det robusta sättet behandlar en händelse som ett besked om att något har ändrats och hämtar objektets aktuella tillstånd från API:et innan det agerar på den.

Handlern är för långsam. GitHub förväntar sig en 2xx inom 10 sekunder; därefter avbryter GitHub anslutningen och räknar leveransen som misslyckad. Specifikationen rekommenderar en timeout för begäran någonstans mellan 15 och 30 sekunder. En handler som gör det egentliga arbetet innan den svarar råkar ut för en timeout, avsändaren gör ett omförsök och arbetet utförs två gånger. Stripe och GitHub rekommenderar båda en kö. Stripe pekar på toppen i början av månaden, när alla prenumerationer förnyas.

Bara en 2xx räknas. Allt annat är ett misslyckande, också en 3xx, för både specifikationen och Stripe, så en slutpunkt som har flyttats bakom en omdirigering misslyckas med varje leverans tills den registrerade URL:en uppdateras. Specifikationen ger några få koder en betydelse för avsändaren: `410 Gone` betyder att slutpunkten ska inaktiveras, en 429, 502 eller 504 betyder att avsändaren ska sakta ner, och ett `retry-after`-huvud bör styra nästa försök.

För avsändaren är URL:en fientlig indata. Kunder kan ange vilken URL de vill, och avsändarens servrar kommer att anropa den. Specifikationen kallar webhook-system särskilt sårbara för server-side request forgery (SSRF): en URL som pekar på en intern adress får avsändaren att anropa in i sitt eget nätverk. Rådet är att skicka alla webhook-begäranden genom en proxy som filtrerar bort interna IP-adresser, till exempel Stripes smokescreen, och att köra webhook-workers i ett privat subnät som inte når interna tjänster.

Läs mer

- Standard Webhooks specification — version 1.0.0: payloadens uppbyggnad, signaturmetoden, huvudfält, tidsplanen för omförsök, hantering av statuskoder och SSRF.
- Receive Stripe events in your webhook endpoint — tunna payloads och snapshot-payloads, omförsök, dubbletter, händelser i fel ordning och skydd mot replay-attacker.

- Best practices for using webhooks — GitHubs lista: få händelser, en webhook-hemlighet, HTTPS, ett svar inom 10 sekunder, ny leverans och `X-GitHub-Delivery` .
- Validating webhook deliveries — `X-Hub-Signature-256` , jämförelse i konstant tid och ett testexempel att kontrollera en verifierare mot.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är en webhook?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

welcome@engineer.company +45 50 10 61 84 Köpenhamn, Danmark

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/webhook/