Engineer

webhook

📢 en HTTP-forespørsel en tjeneste sender når noe skjer

Hva det er

En webhook er en HTTP-forespørsel som én tjeneste sender til en annen i det øyeblikket noe skjer hos den: en betaling går gjennom, en faktura blir betalt, en langvarig oppgave blir ferdig. Mottakersiden registrerer en URL på forhånd, og avsenderen sender en beskrivelse av hver hendelse dit med POST. Standard Webhooks-spesifikasjonen kaller webhooks «a common name for HTTP callbacks» og beskriver dem som en måte tjenester kan varsle hverandre om hendelser på. Ordet skrives av og til som to ord, web hook, men Stripe, GitHub og spesifikasjonen skriver det alle i ett ord.

Spesifikasjonen beskriver også en webhook som en del av en tjenestes API, et slags API som går baklengs. Vanligvis kaller klienten tjenesten når den vil ha noe. Med en webhook kaller tjenesten klienten når den har noe nytt å fortelle. I en typisk API-integrasjon kjører begge sider HTTP-servere, den ene for å motta API-kall og den andre for å motta webhooks. En tjeneste kan også sende webhooks uten å tilby et eget forespørsels-API. Spesifikasjonen kaller avsenderen producer og mottakeren consumer, og URL-en mottakeren registrerer, er webhook-endepunktet dens.

Spesifikasjonen anbefaler en HTTP POST med en JSON-kropp, men krever ikke et bestemt format eller en bestemt struktur for payloaden. Kroppen den foreslår, har en `type`, skrevet med punktum og ordnet hierarkisk, for eksempel `invoice.paid`; et `timestamp` for når hendelsen skjedde, i ISO 8601; og hendelsesdataene enten under `data` eller på øverste nivå. Ytterligere metadata kan stå begge steder. Leveringsmetadata ligger i headere: `webhook-id` identifiserer hendelsen, `webhook-timestamp` angir når akkurat dette forsøket ble gjort, i Unix-sekunder, og `webhook-signature` bekrefter ektheten. Stripe og GitHub bruker egne headernavn, blant dem `Stripe-Signature` og `X-Hub-Signature-256`, og verifiseringsreglene deres er ulike: Stripe signerer et tidsstempel for forsøket sammen med kroppen, mens GitHubs HMAC bare dekker kroppen.

En payload kan være full og inneholde hele det endrede objektet, eller tynn og inneholde lite mer enn en identifikator, slik at mottakeren selv må hente resten gjennom API-et. Spesifikasjonen veier de to mot hverandre. En full payload sparer mottakeren for ekstra kall til API-et. En tynn payload betyr mindre data å generere og sende, mer fleksibilitet og mer rom for endringer senere, fordi en tynn payload alltid kan vokse til en full, men ikke omvendt. Den gjør også datatilgangen sporbar, fordi mottakeren uttrykkelig må be om det den leser. Uansett råder spesifikasjonen til å holde payloads små, vanligvis under 20 kB. Stripe tilbyr et tynt format og merker det som det anbefalte.

Bruk begrepet i en beslutning

Bruk «Hva er en webhook?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

Uten webhooks må et system som trenger å vite om en endring et annet sted, polle: spørre igjen og igjen, for det meste bare for å høre at ingenting har skjedd, og likevel først få vite om endringen ved neste runde. En webhook snur dette. Tjenesten som kjenner til hendelsen, kaller én gang, i det øyeblikket den skjer. Slik melder betalingsleverandører, tjenester for kodehosting og e-postleverandører tilbake til systemene som er bygget på dem.

Den omvendte retningen har en pris, og mesteparten av den havner hos mottakeren. Endepunktet er en offentlig URL som hvem som helst på internett kan sende en POST til, og ingenting ved en innkommende forespørsel viser hvem som sendte den, så spesifikasjonen gjør verifisering av ektheten til et krav for enhver sikker implementasjon. Leveringen prøves på nytt til mottakeren svarer med en 2xx-status eller leveringsreglene setter en stopper for flere forsøk, så den samme hendelsen kan komme mer enn én gang; Stripe advarer om at endepunkter av og til kan motta duplikater. Rekkefølgen er heller ikke lovet: Stripe skriver rett ut at den ikke garanterer at hendelser kommer i den rekkefølgen de ble generert i.

Avsenderen må på sin side fortsette å prøve. Spesifikasjonen anbefaler en tidsplan for nye forsøk som strekker seg over flere dager, med eksponentiell backoff og litt tilfeldig jitter. I eksempelet gjøres det ti forsøk, det siste 75 timer, 35 minutter og 5 sekunder etter det første. Stripe prøver på nytt i opptil tre dager i live mode. En mottaker som er tilbake innenfor det vinduet, tar igjen det tapte av seg selv; etter det kan man hos Stripe sende en hendelse på nytt manuelt i opptil 15 dager etter at den ble opprettet, eller i opptil 30 dager med Stripes CLI. Når leveringen fortsetter å feile over lang tid, anbefaler spesifikasjonen at avsenderen gir mottakeren beskjed gjennom en annen kanal, for eksempel e-post, og deaktiverer endepunktet.

Slik ser det ut i praksis

Dette er headerne i en levering i spesifikasjonens format. Identifikatoren, tidsstempelet og signaturen er eksempelverdiene spesifikasjonen selv bruker; verdien av `Host` og stien er oppdiktet:

```
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=
```

Og dette er kroppen, en tynn payload fra den samme spesifikasjonen, her vist med innrykk. Når den faktisk sendes, anbefaler spesifikasjonen at den er minifisert:

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

Prefikset `v1` markerer en symmetrisk signatur: HMAC-SHA256 over hendelsens identifikator, tidsstempelet og kroppen, satt sammen med punktum og beregnet med en hemmelig nøkkel som er unik for endepunktet. Fordi identifikatoren og tidsstempelet signeres sammen med kroppen, kan ingen av dem endres uten at

signaturen blir ugyldig. Spesifikasjonen tillater også asymmetriske ed25519-signaturer, merket `v1a`, og anbefaler dem når man ikke selv kontrollerer sikkerheten på både avsender- og mottakersiden.

GitHub signerer også leveringene sine med HMAC-SHA256, legger hex-digesten i `X-Hub-Signature-256` med prefikset `sha256=` og publiserer en hemmelighet, en payload og en signatur som man kan teste en verifikator mot. Her er en verifikator i Python, testet mot nettopp det eksempelet:

```
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 eneste mellomrom til slutt i kroppen er nok til at verifiseringen feiler. `hmac.compare_digest` bruker like lang tid uansett om strengene skiller seg tidlig eller sent; GitHubs dokumentasjon sier at man aldri skal sammenligne signaturer med en vanlig `==`. En solid handler jobber deretter i en fast rekkefølge: les de rå bytene, verifiser signaturen over nøyaktig de bytene, sjekk at tidsstempelet for forsøket er nytt nok hvis avsenderen signerer et, sjekk om hendelsens identifikator er sett før, legg hendelsen i en kø og svar med en 2xx. Det egentlige arbeidet skjer etterpå. Stripe gir som eksempel å svare med 200 før en kundes faktura markeres som betalt i regnskapssystemet.

To prosjekter i porteføljen på dette nettstedet handler om webhooks sett fra mottakersiden. I prosjektet med tilgangsrettigheter for Stripe og appkjøp må et abonnement blant annet tåle en webhook som kommer to ganger, og en webhook som kommer i feil rekkefølge. I e-postsystemet med failover og

leveringslogging rapporterer innkommende webhooks hva hver leverandør gjorde med en melding, og prosjektet registrerer både det som ble sendt, og det som kom tilbake, i en sendelogg og en tabell for leveringshendelser.

Hvor det går galt

Signaturen stemmer aldri. En vanlig årsak er at mottakerens rammeverk parser JSON-kroppen, og at koden serialiserer den på nytt før HMAC-en beregnes. Spesifikasjonen kaller dette en svært vanlig feil: JSON-serialisering varierer mellom implementasjoner, og ett eneste mellomrom for mye gjør signaturen ugyldig. Stripe trenger den rå kroppen av samme grunn og advarer om at enhver endring av den får verifiseringen til å feile. GitHub legger til at en proxy eller lastbalanserer foran endepunktet ikke må endre payloaden eller headerne, og at payloaden bør håndteres som UTF-8. Beregn signaturen over bytene akkurat slik de kom, før noe parser dem.

Den samme hendelsen kommer to ganger. Både nye forsøk og nettverksproblemer gir duplikater. Etter spesifikasjonen er `webhook-id` den samme ved hvert nytt forsøk med en hendelse og fungerer som idempotensnøkkel; eksempelet der er å ta vare på identifikatorene i Redis i fem minutter. Stripe ber deg registrere hendelses-ID-ene du har behandlet, og hoppe over dem du allerede har sett. Stripe påpeker også at det i noen tilfeller opprettes og sendes to separate Event-objekter; dem kjenner man igjen på objektets ID i `data.object` sammen med hendelsestypen. GitHubs `X-GitHub-Delivery` -header forblir den samme når en levering sendes på nytt. Behandlingen bør være idempotent, slik at en ny levering ikke endrer noe.

En oppfanget levering spilles av på nytt. En gyldig signatur forblir gyldig, så en angriper som fanger opp én levering, kan sende den igjen. Derfor signeres tidsstempelet: spesifikasjonen sier at et `webhook-timestamp` utenfor en tillatt toleranse skal avvises. Stripes biblioteker bruker fem minutter som standard, Stripe advarer om at en toleranse på 0 slår av kontrollen helt, og anbefaler NTP for å holde serverens klokke nøyaktig. Både Stripe og spesifikasjonen setter et nytt tidsstempel på hvert nytt forsøk, så alderen måles fra det forsøket og ikke fra den opprinnelige hendelsen; et forsinket forsøk kan fortsatt falle utenfor toleransen. Ta vare på identifikatorene du har sett, minst like lenge som toleransen, så blir alt som spilles av på nytt innenfor vinduet, fanget opp som et duplikat.

Hendelsene kommer i feil rekkefølge. Stripes eksempel: når et abonnement opprettes, kan det gi `customer.subscription.created`, `invoice.created`, `invoice.paid` og, hvis det er en belastning, `charge.created`, og ingenting garanterer at de kommer i den rekkefølgen. Stripes snapshot-hendelser angir `created` i hele sekunder, så ulike hendelser kan ha samme tidsstempel, og Stripe sier at man ikke skal bruke det til å sortere hendelser eller oppdage duplikater. Den robuste tilnærmingen behandler en hendelse som en beskjed om at noe er endret, og henter objektets nåværende tilstand fra API-et før den handler på den.

Handleren er for treg. GitHub forventer en 2xx innen 10 sekunder; etter det bryter GitHub forbindelsen og regner leveringen som mislykket. Spesifikasjonen anbefaler et tidsavbrudd for forespørselen på et sted mellom 15 og 30 sekunder. En handler som gjør det egentlige arbeidet før den svarer, får tidsavbrudd, avsenderen prøver på nytt, og arbeidet blir gjort to ganger. Stripe og GitHub anbefaler begge en kø. Stripe peker på toppen i begynnelsen av måneden, når alle abonnementer fornyes.

Bare en 2xx teller. Alt annet er en feil, også en 3xx, for både spesifikasjonen og Stripe, så et endepunkt som er flyttet bak en omdirigering, feiler ved hver levering til den registrerte URL-en blir oppdatert. Spesifikasjonen gir noen få koder en betydning for avsenderen: `410 Gone` betyr at endepunktet skal deaktiveres, en 429, 502 eller 504 betyr at avsenderen skal sette ned tempoet, og en `retry-after`-header bør styre neste forsøk.

For avsenderen er URL-en fiendtlig input. Kunder kan oppgi hvilken URL de vil, og avsenderens servere kommer til å kalle den. Spesifikasjonen kaller webhook-systemer spesielt sårbare for server-side request forgery (SSRF): en URL som peker på en intern adresse, får avsenderen til å kalle inn i sitt eget nettverk. Rådet er å sende alle webhook-forespørsler gjennom en proxy som filtrerer bort interne IP-adresser, for eksempel Stripes smokescreen, og å kjøre webhook-workerne i et privat subnett som ikke når interne tjenester.

Les mer

- Standard Webhooks specification — versjon 1.0.0: oppbygningen av payloaden, signaturmetoden, headere, tidsplanen for nye forsøk, håndtering av statuskoder og SSRF.

- Receive Stripe events in your webhook endpoint — tynne payloads og snapshot-payloads, nye forsøk, duplikater, hendelser i feil rekkefølge og beskyttelse mot replay-angrep.
- Best practices for using webhooks — GitHubs liste: få hendelser, en webhook-hemmelighet, HTTPS, et svar innen 10 sekunder, ny levering og `X-GitHub-Delivery` .
- Validating webhook deliveries — `X-Hub-Signature-256` , sammenligning i konstant tid og et testeksempel å kontrollere en verifikator mot.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er en webhook?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/webhook/