Engineer

Webhook

📢 une requête HTTP envoyée par un service lors d'un événement

Ce que c’est

Un webhook est une requête HTTP qu’un service envoie à un autre dès qu’un événement se produit de son côté : un paiement aboutit, une facture est réglée, une longue tâche se termine. Le destinataire enregistre une URL à l’avance, et l’expéditeur y envoie une description de chaque événement. La spécification Standard Webhooks explique que webhook est un « nom courant pour les rappels HTTP » et décrit les webhooks comme un canal par lequel un service annonce un événement à un autre. Le terme s’écrit parfois en deux mots, web hook, mais Stripe, GitHub et la spécification l’écrivent tous en un seul.

La spécification décrit aussi un webhook comme une partie de l’API d’un service, une sorte d’API fonctionnant à l’envers. Habituellement, le client appelle le service lorsqu’il veut quelque chose. Avec un webhook, le service appelle le client lorsqu’il a une nouvelle à annoncer. Dans une intégration API typique, les deux côtés exploitent des serveurs HTTP, l’un pour recevoir les appels API, l’autre pour recevoir les webhooks. Un service peut aussi envoyer des webhooks sans proposer sa propre API de requêtes. L’expéditeur est appelé producteur, le destinataire consommateur, et l’URL enregistrée par ce dernier est son point de terminaison de webhook.

La spécification recommande une requête HTTP POST avec un corps JSON, sans imposer de format ni de structure particuliers pour la charge utile. Le corps suggéré comporte un `type`, écrit avec des points et organisé hiérarchiquement, par exemple `invoice.paid` ; un `timestamp` indiquant quand l’événement s’est produit, au format ISO 8601 ; et les données de l’événement, soit sous `data`, soit au premier niveau. Des métadonnées supplémentaires peuvent figurer aux deux endroits. Les métadonnées de livraison vont dans les en-têtes : `webhook-id` identifie l’événement, `webhook-timestamp` indique quand cette tentative a été faite, en secondes Unix, et `webhook-signature` l’authentifie. Stripe et GitHub

utilisent leurs propres noms d'en-têtes, dont `Stripe-Signature` et `X-Hub-Signature-256` . Leurs règles de vérification diffèrent : Stripe signe l'horodatage d'une tentative avec le corps, tandis que le HMAC de GitHub ne couvre que le corps.

Une charge utile peut être complète, contenant tout l'objet modifié, ou allégée, ne contenant guère plus qu'un identifiant et laissant au destinataire le soin de récupérer le reste par l'API. La spécification compare les deux. Une charge complète évite au destinataire des appels API supplémentaires. Une charge allégée signifie moins de données à générer et à envoyer, davantage de souplesse et de possibilités d'évolution : elle peut toujours devenir complète, mais pas l'inverse. Elle rend aussi l'accès aux données auditable, car le destinataire doit demander explicitement ce qu'il lit. Dans les deux cas, la spécification conseille de limiter la taille des charges utiles, généralement inférieures à 20 ko. Stripe propose un format allégé et le désigne comme recommandé.

Utilisez le concept pour prendre une décision

Utilisez « Qu'est-ce qu'un webhook ? » pour éclairer une décision réelle. Nous pouvons vous aider à relier ce concept au système que vous construisez ou achetez. La partie invisible pour les utilisateurs : où résident les données, quelles règles s'appliquent et comment les autres systèmes peuvent y accéder.

Découvrir nos prestations Contactez-nous →

Vérifiez les hypothèses derrière le terme

Appuyez-vous sur l'explication et les réalisations liées pour formuler les questions de votre projet. Nous pouvons examiner les hypothèses avec vous avant le choix d'un outil. 6 pipelines ETL/ELT conçus et exploités — 6 pipelines ETL sur trois moteurs de bases de données

6 pipelines ETL/ELT conçus et exploités Contactez-nous →

Pourquoi c'est important

Sans webhooks, un système qui doit connaître une modification ailleurs doit interroger régulièrement sa source : demander encore et encore, le plus souvent pour apprendre que rien ne s'est passé, puis n'apprendre le changement qu'à l'interrogation suivante. Un webhook inverse cette logique. Le service qui connaît l'événement appelle une fois, au moment où il se produit. C'est ainsi que les prestataires de paiement, services d'hébergement de code et fournisseurs de messagerie informent les systèmes construits sur leurs services.

Cette inversion a un coût, principalement supporté par le destinataire. Son point de terminaison est une URL publique à laquelle n'importe qui sur Internet peut envoyer un POST. Rien dans une requête entrante ne prouve qui l'a envoyée ; la spécification exige donc la vérification de son authenticité pour toute

implémentation sûre. La livraison est retentée jusqu'à ce que le destinataire réponde avec un statut 2xx ou que la politique de livraison abandonne. Le même événement peut ainsi arriver plusieurs fois ; Stripe avertit que les points de terminaison peuvent occasionnellement recevoir des doublons. L'ordre n'est pas non plus promis : Stripe précise qu'il ne garantit pas l'arrivée des événements dans l'ordre de leur génération.

Le producteur doit, de son côté, continuer à essayer. La spécification recommande un calendrier de nouvelles tentatives sur plusieurs jours, avec une temporisation exponentielle et un peu de variation aléatoire. Son exemple prévoit dix tentatives, la dernière 75 heures, 35 minutes et 5 secondes après la première. Stripe réessaie pendant au plus trois jours en mode réel. Un destinataire qui redevient accessible pendant cette période reçoit automatiquement les livraisons en attente. Au-delà, Stripe permet de renvoyer manuellement un événement jusqu'à 15 jours après sa création, ou jusqu'à 30 jours avec son CLI. Si la livraison échoue durablement, la spécification recommande au producteur d'en informer le consommateur par un autre canal, comme le courriel, et de désactiver le point de terminaison.

À quoi cela ressemble en pratique

Voici les en-têtes d'une livraison au format de la spécification. L'identifiant, l'horodatage et la signature sont les valeurs d'exemple de la spécification elle-même ; la valeur de `Host` et le chemin sont inventés :

```
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=
```

Voici le corps, une charge utile allégée tirée de la même spécification, présentée ici avec une mise en forme lisible. Pour la transmission, la spécification recommande de la minifier :

```
{
  "type": "contact.created",
  "timestamp": "2022-11-03T20:26:10.344522Z",
  "data": {
```

```
    "id": "1f81eb52-5198-4599-803e-771906343485"
  }
}
```

Le préfixe `v1` désigne une signature symétrique : un HMAC-SHA256 calculé sur l'identifiant de l'événement, l'horodatage et le corps joints par des points, avec un secret propre au point de terminaison. Puisque l'identifiant et l'horodatage sont signés avec le corps, aucun des deux ne peut être modifié sans invalider la signature. La spécification autorise aussi des signatures asymétriques ed25519, marquées `v1a`, et les recommande lorsque des parties différentes sont responsables de la sécurité du producteur et du consommateur.

GitHub signe aussi ses livraisons avec HMAC-SHA256, place l'empreinte hexadécimale dans `X-Hub-Signature-256` avec un préfixe `sha256=` et publie un secret, une charge utile et une signature pour tester un vérificateur. Voici un vérificateur en Python, contrôlé avec cet exemple :

```
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
```

Un seul espace final dans le corps suffit à faire échouer la vérification. `hmac.compare_digest` prend le même temps, que les chaînes diffèrent au début ou à la fin ; la documentation de GitHub demande de ne jamais comparer des signatures avec un simple `==`. Un bon gestionnaire suit ensuite un ordre fixe : lire les octets bruts, vérifier la signature sur ces octets exacts, vérifier que l'horodatage de la tentative est récent si le producteur en signe un, vérifier si cet identifiant d'événement a déjà été vu, placer l'événement dans une file d'attente

et répondre par un 2xx. Le véritable travail se fait ensuite. Stripe donne l'exemple d'un retour 200 avant de marquer la facture d'un client comme réglée dans le système comptable.

Deux projets du portfolio de ce site traitent les webhooks du côté réception. Dans la gestion des droits liés à Stripe et aux achats intégrés, un abonnement doit résister, entre autres, à un webhook reçu deux fois et à un webhook reçu dans le désordre. Dans le système de messagerie avec basculement et journalisation des livraisons, les webhooks entrants indiquent ce que chaque fournisseur a fait d'un message. Le projet consigne ce qui a été envoyé comme ce qui est revenu, dans un journal d'envoi et une table d'événements de livraison.

Où les problèmes surviennent

La signature ne correspond jamais. Une cause courante est que le framework du destinataire analyse le corps JSON, puis que le code le sérialise à nouveau avant de calculer le HMAC. La spécification décrit ce cas comme très fréquent : la sérialisation JSON varie selon les implémentations, et un seul espace parasite invalide la signature. Stripe exige le corps brut pour la même raison et avertit que toute manipulation fait échouer la vérification. GitHub ajoute qu'un proxy ou répartiteur de charge devant le point de terminaison ne doit modifier ni la charge utile ni les en-têtes, et que la charge doit être traitée comme UTF-8. Calculez la signature sur les octets tels qu'ils sont arrivés, avant toute analyse.

Le même événement arrive deux fois. Les nouvelles tentatives et les incidents réseau produisent tous deux des doublons. Selon la spécification, `webhook-id` reste identique pour toutes les nouvelles tentatives d'un événement et sert de clé d'idempotence ; son exemple conserve les identifiants dans Redis pendant cinq minutes. Stripe demande de journaliser les identifiants des événements traités et d'ignorer ceux déjà vus. Il précise aussi qu'une même occurrence sous-jacente peut produire deux objets Event distincts ; on les reconnaît par l'identifiant d'objet dans `data.object` associé au type d'événement. L'en-tête `X-GitHub-Delivery` de GitHub reste identique lorsqu'une livraison est renvoyée. Le traitement doit être idempotent : une seconde livraison ne change rien.

Une livraison interceptée est rejouée. Une signature valide reste valide. Un attaquant qui intercepte une livraison peut donc la renvoyer. C'est pourquoi l'horodatage est signé : la spécification demande de rejeter un `webhook-timestamp` hors d'une tolérance admise. Les bibliothèques Stripe utilisent cinq minutes par défaut ; Stripe avertit qu'une tolérance de 0 désactive entièrement le contrôle et

recommande NTP pour maintenir l'horloge du serveur à l'heure. Stripe comme la spécification attribuent un nouvel horodatage à chaque nouvelle tentative. L'ancienneté est donc mesurée depuis cette tentative, pas depuis l'événement initial ; une tentative retardée peut encore dépasser la tolérance. Conservez les identifiants déjà vus au moins pendant la durée de la tolérance ; tout rejeu dans cette fenêtre est alors reconnu comme un doublon.

Les événements arrivent dans le désordre. Exemple de Stripe : la création d'un abonnement peut produire `customer.subscription.created`, `invoice.created`, `invoice.paid` et, s'il y a un paiement, `charge.created`, sans aucune garantie d'arrivée dans cet ordre. Le champ `created` des événements Stripe contenant un instantané a une résolution d'une seconde, ce qui permet à plusieurs événements de porter le même horodatage. Stripe demande de ne pas l'utiliser pour ordonner les événements ni détecter les doublons. L'approche robuste traite un événement comme l'annonce d'un changement et récupère l'état actuel de l'objet par l'API avant d'agir.

Le gestionnaire est trop lent. GitHub attend un 2xx dans les 10 secondes ; au-delà, il termine la connexion et considère la livraison comme échouée. La spécification recommande un délai maximal de requête compris entre 15 et 30 secondes. Un gestionnaire qui fait son véritable travail pendant la requête dépasse ce délai ; le producteur réessaie, et le travail est exécuté deux fois. Stripe et GitHub recommandent tous deux une file d'attente. Stripe cite le pic du début du mois, lorsque tous les abonnements se renouvellent.

Seul un 2xx compte. Tout autre statut est un échec, 3xx compris, pour la spécification comme pour Stripe. Un point de terminaison passé derrière une redirection échoue donc à chaque livraison jusqu'à la mise à jour de son URL enregistrée. La spécification donne un sens à certains codes pour le producteur : `410 Gone` signifie désactiver le point de terminaison ; 429, 502 ou 504 signifient ralentir ; un en-tête `retry-after` doit guider la tentative suivante.

Pour le producteur, l'URL est une entrée hostile. Les clients peuvent saisir l'URL de leur choix, et les serveurs du producteur l'appelleront. La spécification décrit les systèmes de webhooks comme particulièrement vulnérables à la falsification de requêtes côté serveur (SSRF) : une URL pointant vers une adresse interne amène le producteur à appeler son propre réseau. Elle conseille de faire passer chaque requête de webhook par un proxy filtrant les adresses IP internes, comme smokescreen de Stripe, et d'exécuter les processus de travail des webhooks dans un sous-réseau privé qui ne peut atteindre les services internes.

Pour aller plus loin

- Spécification Standard Webhooks — version 1.0.0 : structure de la charge utile, schéma de signature, en-têtes, calendrier des nouvelles tentatives, traitement des codes de statut et SSRF.
- Recevoir les événements Stripe dans un point de terminaison de webhook — charges utiles allégées et charges contenant un instantané, nouvelles tentatives, événements en double ou dans le désordre et protection contre le rejeu.
- Bonnes pratiques d'utilisation des webhooks — la liste de GitHub : peu d'événements, un secret de webhook, HTTPS, une réponse dans les 10 secondes, le renvoi et `X-GitHub-Delivery` .
- Valider les livraisons de webhooks — `X-Hub-Signature-256` , comparaison en temps constant et exemple de test pour vérifier un vérificateur.

Glossaire

Choisissez selon vos contraintes réelles

Présentez le processus de travail, les interfaces existantes et les utilisateurs qui ont besoin du changement. Nous pouvons définir une livraison testable et une transmission permettant à votre équipe d'assurer la maintenance des éléments livrés.

Consulter les tarifs Contactez-nous →

Rendez la terminologie concrète

Demandez comment « Qu'est-ce qu'un webhook ? » s'applique à votre organisation. Un exemple concret tiré de votre système actuel suffit pour engager la discussion.

Découvrir nos prestations

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

Emportez cette page avec vous

Fiche de contact

Créé avec soin et amour par D.

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

Droit d'auteur © 2025 – aujourd'hui · Engineer ApS · Tous droits réservés.

Imprimé depuis https://engineer.company/fr/glossary/webhook/