Setting Up a Webhook for App Notifications

When a new app notification is created by Replyke, developers can configure their server to be notified about the event. This is achieved by setting up a webhook endpoint on the developer's server. A webhook is essentially an HTTP endpoint that is triggered by Replyke whenever a notification is generated. The webhook will receive a payload containing all the relevant details about the notification, enabling developers to send corresponding push notifications or perform other actions.

Notification Payload Details

The payload sent to the webhook contains the following details:

Field	Description
`projectId`	The ID of the project that generated the notification.
`userId`	The ID of the user who is the recipient of the notification.
`type`	A string representing the event type (e.g., `entity-comment`, `comment-reply`, `entity-mention`, etc.).
`action`	A recommended action to take, such as `open-entity`, `open-comment`, or `open-profile`.
`metadata`	A payload containing additional details relevant to the notification. Refer to the notification types table for specifics in the "Notification Templates" chapter.

Securing Your Webhook

Verifying the integrity of a notification - covered in more detail below - is similar to verifying other webhooks, such as those triggered when an entity or user is created or updated, as discussed in the security section of this documentation.

To ensure the integrity of the data received by your webhook, Replyke uses a secure signing mechanism:

1. **Generate a Secret Key:** Developers must generate a secret key in their project's dashboard and store it securely as an environment variable on their server.
2. **Payload Signing:** Before sending the notification details to the webhook, Replyke signs the payload using the HMAC-SHA256 algorithm. This process uses the secret key to create a signature.
3. **Verification:** The webhook server must verify the payload using the same secret key to confirm that the request originated from Replyke. Any request with an invalid signature should be rejected.

Example: Verifying a Webhook in Node.js + Express

Here is an example implementation of a webhook in a Node.js + Express environment:

Webhook Controller

```
import { Request, Response } from "express";
import { validateIncomingHmac } from "./utility-functions";

export default async (req: Request, res: Response) => {
  const sharedSecret = process.env.REPLYKE_PROJECT_SECRET;
  try {
    // Step 1: Validate the incoming HMAC signature
    validateIncomingHmac(req, sharedSecret!);

    const newNotificationData = req.body;

    // Step 2: Process the data as needed. There is no need to send a response.
    console.log({ newNotificationData });

  } catch (err: any) {
    console.error("Error validating webhook: ", err);
    // Respond with an error status
    res.status(400).send({ error: "Invalid webhook request" });
  }
};
```

Utility Function for Validation

```
import crypto from "crypto";
import { Request } from "express";

/**
 * Validate HMAC signature of an incoming request.
 */
export function validateIncomingHmac(req: Request, secret: string): void {
  const signature = req.headers["x-signature"] as string;
  const timestamp = req.headers["x-timestamp"] as string;

  if (!signature || !timestamp) {
    throw new Error("Missing HMAC signature or timestamp in headers");
  }

  // Reject requests older than 5 minutes to prevent replay attacks
  if (Date.now() - parseInt(timestamp, 10) > 5 * 60 * 1000) {
    throw new Error("Request timestamp expired");
  }

  // Compute the expected signature
  const payload = JSON.stringify(req.body);
  const expectedSignature = crypto
    .createHmac("sha256", secret)
    .update(`${timestamp}.${payload}`)
    .digest("hex");

  if (signature !== expectedSignature) {
    throw new Error("Invalid HMAC signature");
  }
}
```

Steps to Integrate the Webhook

1. **Set Up Your Webhook Endpoint:** Create an HTTP endpoint on your server to receive the notification payloads.
2. **Generate and Store Your Secret Key:** Go to the Replyke project dashboard to generate a secret key and store it securely as an environment variable.
3. **Implement Payload Validation:** Use the provided example to validate incoming requests using the secret key, or implement a similar HMAC validation logic in your preferred programming language if you are not using Node.js.
4. **Handle the Notification Data:** Use the payload details to trigger appropriate actions, such as sending push notifications to users.

5. **Monitor and Debug** (*Optional*): Log incoming requests and errors to ensure the webhook is functioning correctly.

By following these steps, you can reliably integrate a secure webhook into your system, ensuring trusted communication between your server and Replyke.

Last updated on May 7, 2025