Feature: EventSub forwarding

Summary

As a TAU user, it would be great if I could add a webhook URL into a dashboard to forward specific webhooks I’m subscribed to.

Github URL:

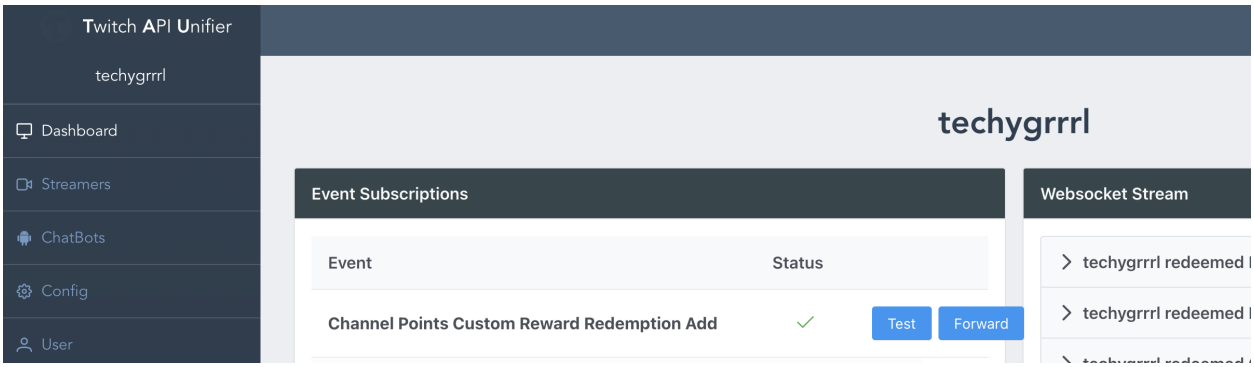
Why is this useful

This feature would be useful in the following cases:

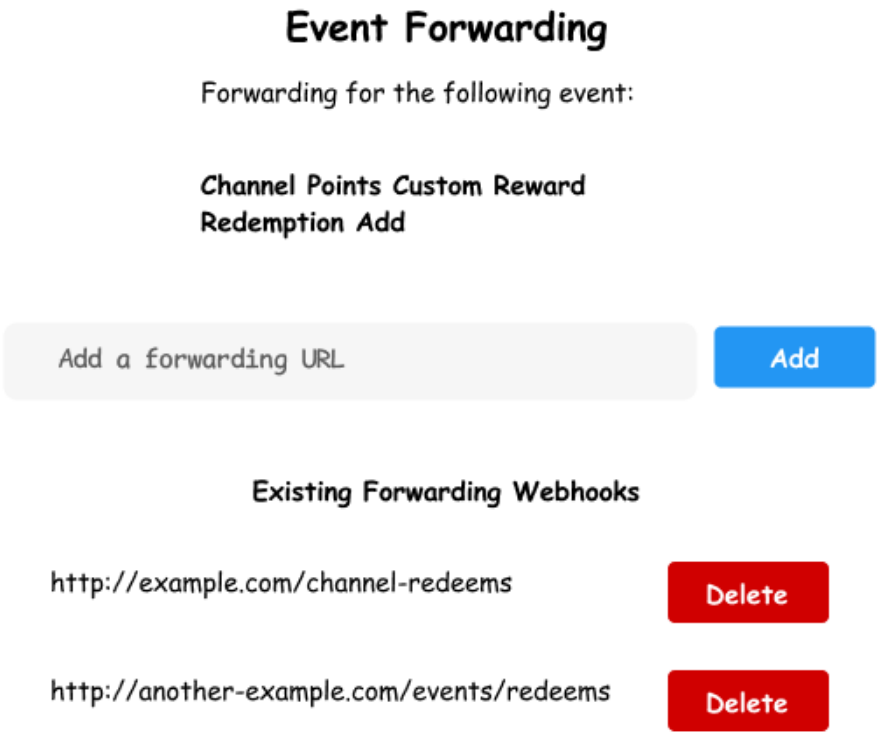
- Web sockets is not possible
- Devs want to leverage always-on, cost-saving infrastructure like cloud functions to handle events
    - Cloudflare Workers, Vercel, Netlify, etc. allow you to deploy cloud functions for cheap or free

Design

It could be added to the current Event Subscriptions section as a “Forward” button.



The Forward button could pop open a modal:



The modal could contain:

- Field to Add a new URL
- A list of URLs for that subscription, with a delete button

Data Modelling

Assuming the table where we store the canonical reference to the webhooks is `twitch_twitcheventsubsubscription`, we can create a separate table that references this

with a foreign key.

> ❗ **Note**: It looks like the Event Subscriptions data is partially dynamic from Twitch so an alternative solution may be required.

A new table, e.g. `event_forwarding` could be added, where we reference `twitch_twitcheventsubsubscription.id` as the foreign key to the event, and a new field `url` as the URL to forward to:

- `event_id` → UUID (foreign key to the existing local events table, e.g. `twitch_twitcheventsubsubscription.id`)
- `url` → URL to post to

Auth

Endpoints need to trust a webhook event from TAU and should verify. We can do this with JWT verification where the user auth token is the secret used for signature verification.

```
tau_header = JWT.encode(
  { service: 'tau' }, # the contents of this token data is not important
  secret: EnvironmentVariables.get("user_auth_token")
)
```

On New Event

When a new event is received from Twitch, TAU could go through all the events. For example, in pseudo-code/ActiveRecord-like syntax:

```
# triggered_event would be the local webhook event stored in the TAU DB

forwarding_urls = EventForwarding.where(event_id: triggered_event.id)

forwarding_urls.each do |url|
  RestClient.post(url,
```

```
    data: triggered_event.event_data,
    headers: {
      "X-Tau-Token": tau_header # see Auth above
    }
  )
end
```

The consumer, e.g. a cloud function, would also have the auth token stored in an environment variable, and would do signature verification on the provided token:

```
const tauToken = request.headers['X-Tau-Token']
if (!tauToken) {
  return res.status(401).json({ error: 'unauthorized' })
}

try {
  const tokenData = jwt.verify(tauToken, { secret: process.env.TAU_USER_TOKEN })
  // at this point, tokenData.service should be "tau"

  // Do whatever needs to be done with this event
  console.log('event forwarded from TAU', request.body)

} catch (e) {
  // check the type of exception
  return res.status(401).json({ error: 'unauthorized' })
}
```

Flow

1. TAU receives an event from Twitch
2. TAU queries the new `event_forwarding` table to find all forwarding URLs for the specific event
3. TAU iterates through each event in order to forward the event
    a. TAU creates a JWT token signed with the user auth token and puts it in a custom header, e.g. `X-Tau-Token`
    b. TAU makes a post request with the data to the URL and includes the JWT token
4. The registered URL receives the request, e.g. a cloud function.

5. The cloud function verifies the request is authentic from TAU by using the user auth token it has stored in its environment variables to do a JWT signature verification on the header.
6. The cloud function can now safely use the data that was forwarded