Automatically redelivering failed deliveries for a GitHub App webhook

You can write a script to handle failed deliveries of a GitHub App webhook.

In this article

About automatically redelivering failed deliveries

This article describes how to write a script to find and redeliver failed deliveries for a GitHub App webhook. For more information about failed deliveries, see "Handling failed webhook deliveries."

This example shows you:

- A script that will find and redeliver failed deliveries for a GitHub App webhook
- What credentials your script will need, and how to store the credentials securely as GitHub Actions secrets
- A GitHub Actions workflow that can securely access your credentials and run the script periodically

This example uses GitHub Actions, but you can also run this script on your server that handles webhook deliveries. For more information, see "Alternative methods."

Storing credentials for the script

The endpoints to find and redeliver failed webhooks require a JSON web token, which is generated from the app ID and private key for your app.

The endpoints to fetch and update the value of environment variables require a personal access token, GitHub App installation access token, or GitHub App user access token. This example uses a personal access token. If your GitHub App is installed on the repository where this workflow will run and has permission to write repository variables, you can modify this example to create an installation access token during the GitHub Actions workflow instead of using a personal access token. For more information, see "Making authenticated API requests with a GitHub App in a GitHub Actions workflow."

1. Find the app ID for your GitHub App. You can find the app ID on the settings

page for your app. The app ID is different from the client ID. For more information about navigating to the settings page for your GitHub App, see "Modifying a GitHub App registration."

2. Store the app ID from the previous step as a GitHub Actions secret in the repository where you want the workflow to run. For more information about storing secrets, see "Using secrets in GitHub Actions."

3. Generate a private key for your app. For more information about generating a private key, see "Managing private keys for GitHub Apps."

4. Store the private key, including `-----BEGIN RSA PRIVATE KEY-----` and `-----END RSA PRIVATE KEY-----`, from the previous step as a GitHub Actions secret in the repository where you want the workflow to run.

5. Create a personal access token with the following access. For more information, see "Managing your personal access tokens."

   - For a fine-grained personal access token, grant the token:
     - Write access to the repository variables permission
     - Access to the repository where this workflow will run
   - For a personal access token (classic), grant the token the `repo` scope.

6. Store your personal access token from the previous step as a GitHub Actions secret in the repository where you want the workflow to run.

Adding a workflow that will run the script

This section demonstrates how you can use a GitHub Actions workflow to securely access the credentials that you stored in the previous section, set environment variables, and periodically run a script to find and redeliver failed deliveries.

Copy this GitHub Actions workflow into a YAML file in the `.github/workflows` directory in the repository where you want the workflow to run. Replace the placeholders in the `Run script` step as described below.

YAML Beside Inline

```yaml
name: Redeliver failed webhook deliveries
```

```yaml
on:
  schedule:
    - cron: '40 */6 * * *'
  workflow_dispatch:
```

This workflow runs every 6 hours or when manually triggered.

```yaml
permissions:
  contents: read
```

This workflow will use the built in `GITHUB_TOKEN` to check out the repository contents. This grants `GITHUB_TOKEN` permission to do that.

```yaml
jobs:
  redeliver-failed-deliveries:
    name: Redeliver failed deliveries
    runs-on: ubuntu-latest
    steps:
```

```yaml
      - name: Check out repo content
        uses: actions/checkout@v4
```

This workflow will run a script that is stored in the repository. This step checks out the repository contents so that the workflow can access the script.

```yaml
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18.x'
```

This step sets up Node.js. The script that this workflow will run uses Node.js.

```yaml
      - name: Install dependencies
        run: npm install octokit
```

This step installs the octokit library. The script that this workflow will run uses the octokit library.

```yaml
      - name: Run script
        env:
          APP_ID: ${{ secrets.YOUR_APP_ID_SECRET_NAME }}
          PRIVATE_KEY: ${{ secrets.YOUR_PRIVATE_KEY_SECRET_NAME }}
          TOKEN: ${{ secrets.YOUR_TOKEN_SECRET_NAME }}
          LAST_REDELIVERY_VARIABLE_NAME: 'YOUR_LAST_REDELIVERY_VARIABLE_NAME'
          WORKFLOW_REPO: ${{ github.event.repository.name }}
          WORKFLOW_REPO_OWNER: ${{ github.repository_owner }}
        run: |
          node .github/workflows/scripts/redeliver-failed-deliveries.js
```

This step sets some environment variables, then runs a script to find and redeliver failed webhook deliveries.

- Replace `YOUR_APP_ID_SECRET_NAME` with the name of the secret where you stored your app ID.
- Replace `YOUR_PRIVATE_KEY_SECRET_NAME` with the name of the secret where you stored your private key.
- Replace `YOUR_TOKEN_SECRET_NAME` with the name of the secret where you stored your personal access token.
- Replace `YOUR_LAST_REDELIVERY_VARIABLE_NAME` with the name that you want to use for a configuration variable that will be stored in the repository where this workflow is stored. The name can be any string that contains only alphanumeric characters and `_`, and does not start with `GITHUB_` or a number. For more information, see "Variables."

Adding the script

This section demonstrates how you can write a script to find and redeliver failed deliveries.

Copy this script into a file called `.github/workflows/scripts/redeliver-failed-deliveries.js` in the same repository where you saved the GitHub Actions workflow file above.

JavaScript   Beside   Inline

```javascript
const { App, Octokit } = require("octokit");
```

This script uses GitHub's Octokit SDK to make API requests. For more information, see "Scripting with the REST API and JavaScript."

```javascript
async function checkAndRedeliverWebhooks() {
```

```javascript
  const APP_ID = process.env.APP_ID;
  const PRIVATE_KEY = process.env.PRIVATE_KEY;
  const TOKEN = process.env.TOKEN;
  const LAST_REDELIVERY_VARIABLE_NAME =
process.env.LAST_REDELIVERY_VARIABLE_NAME;
  const WORKFLOW_REPO_NAME = process.env.WORKFLOW_REPO;
  const WORKFLOW_REPO_OWNER = process.env.WORKFLOW_REPO_OWNER;
```

Get the values of environment variables that were set by the GitHub Actions workflow.

```javascript
  const app = new App({
    appId: APP_ID,
    privateKey: PRIVATE_KEY,
  });
```

Create an instance of the octokit `App` using the app ID and private key values that

were set in the GitHub Actions workflow.

This will be used to make API requests to the webhook-related endpoints.

```javascript
  const octokit = new Octokit({
    auth: TOKEN,
  });
  try {
```

Create an instance of `Octokit` using the token values that were set in the GitHub Actions workflow.

This will be used to update the configuration variable that stores the last time that this script ran.

```javascript
    const lastStoredRedeliveryTime = await getVariable({
      variableName: LAST_REDELIVERY_VARIABLE_NAME,
      repoOwner: WORKFLOW_REPO_OWNER,
      repoName: WORKFLOW_REPO_NAME,
      octokit,
    });
    const lastWebhookRedeliveryTime = lastStoredRedeliveryTime || (Date.now()
- (24 * 60 * 60 * 1000)).toString();
```

Get the last time that this script ran from the configuration variable. If the variable is not defined, use the current time minus 24 hours.

```
    const newWebhookRedeliveryTime = Date.now().toString();
```

Record the time that this script started redelivering webhooks.

```
    const deliveries = await
fetchWebhookDeliveriesSince({lastWebhookRedeliveryTime, app});
```

Get the webhook deliveries that were delivered after `lastWebhookRedeliveryTime`.

```
    let deliveriesByGuid = {};
    for (const delivery of deliveries) {
      deliveriesByGuid[delivery.guid]
        ? deliveriesByGuid[delivery.guid].push(delivery)
        : (deliveriesByGuid[delivery.guid] = [delivery]);
    }
```

Consolidate deliveries that have the same globally unique identifier (GUID). The GUID is constant across redeliveries of the same delivery.

```
    let failedDeliveryIDs = [];
    for (const guid in deliveriesByGuid) {
      const deliveries = deliveriesByGuid[guid];
      const anySucceeded = deliveries.some(
        (delivery) => delivery.status === "OK"
      );
      if (!anySucceeded) {
        failedDeliveryIDs.push(deliveries[0].id);
      }
    }
```

For each GUID value, if no deliveries for that GUID have been successfully delivered within the time frame, get the delivery ID of one of the deliveries with that GUID.

This will prevent duplicate redeliveries if a delivery has failed multiple times. This will also prevent redelivery of failed deliveries that have already been successfully redelivered.

```
    for (const deliveryId of failedDeliveryIDs) {
      await redeliverWebhook({deliveryId, app});
    }
```

Redeliver any failed deliveries.

```
    await updateVariable({
      variableName: LAST_REDELIVERY_VARIABLE_NAME,
      value: newWebhookRedeliveryTime,
      variableExists: Boolean(lastStoredRedeliveryTime),
      repoOwner: WORKFLOW_REPO_OWNER,
      repoName: WORKFLOW_REPO_NAME,
      octokit,
      });
```

Update the configuration variable (or create the variable if it doesn't already exist) to store the time that this script started. This value will be used next time this script runs.

```
    console.log(
      `Redelivered ${
        failedDeliveryIDs.length
      } failed webhook deliveries out of ${
        deliveries.length
      } total deliveries since ${Date(lastWebhookRedeliveryTime)}.`
    );
  } catch (error) {
```

Log the number of redeliveries.

```
    if (error.response) {
      console.error(
        `Failed to check and redeliver webhooks:
${error.response.data.message}`
      );
    }
    console.error(error);
    throw(error);
  }
}
```

If there was an error, log the error so that it appears in the workflow run log, then throw the error so that the workflow run registers as a failure.

```
async function fetchWebhookDeliveriesSince({lastWebhookRedeliveryTime, app})
{
  const iterator = app.octokit.paginate.iterator(
    "GET /app/hook/deliveries",
    {
      per_page: 100,
      headers: {
        "x-github-api-version": "2022-11-28",
      },
    }
  );
  const deliveries = [];
  for await (const { data } of iterator) {
    const oldestDeliveryTimestamp = new Date(
      data[data.length - 1].delivered_at
    ).getTime();
    if (oldestDeliveryTimestamp < lastWebhookRedeliveryTime) {
      for (const delivery of data) {
        if (
          new Date(delivery.delivered_at).getTime() >
lastWebhookRedeliveryTime
        ) {
          deliveries.push(delivery);
        } else {
          break;
        }
      }
      break;
    } else {
      deliveries.push(...data);
    }
  }
  return deliveries;
}
```

This function will fetch all of the webhook deliveries that were delivered since `lastWebhookRedeliveryTime`. It uses the `octokit.paginate.iterator()` method to iterate through paginated results. For more information, see "[Scripting with the REST API and JavaScript](#)."

If a page of results includes deliveries that occurred before `lastWebhookRedeliveryTime`, it will store only the deliveries that occurred after `lastWebhookRedeliveryTime` and then stop. Otherwise, it will store all of the deliveries from the page and request the next page.

```
async function redeliverWebhook({deliveryId, app}) {
  await app.octokit.request("POST
/app/hook/deliveries/{delivery_id}/attempts", {
    delivery_id: deliveryId,
  });
}
```

This function will redeliver a failed webhook delivery.

```
async function getVariable({ variableName, repoOwner, repoName, octokit }) {
  try {
    const {
      data: { value },
    } = await octokit.request(
      "GET /repos/{owner}/{repo}/actions/variables/{name}",
      {
        owner: repoOwner,
        repo: repoName,
        name: variableName,
      }
    );
    return value;
  } catch (error) {
    if (error.status === 404) {
      return undefined;
    } else {
      throw error;
    }
  }
}
```

This function gets the value of a configuration variable. If the variable does not exist, the endpoint returns a 404 response and this function returns `undefined`.

```
async function updateVariable({
  variableName,
  value,
  variableExists,
  repoOwner,
  repoName,
  octokit,
}) {
  if (variableExists) {
    await octokit.request(
      "PATCH /repos/{owner}/{repo}/actions/variables/{name}",
      {
        owner: repoOwner,
        repo: repoName,
        name: variableName,
        value: value,
      }
    );
  } else {
      await octokit.request("POST /repos/{owner}/{repo}/actions/variables", {
        owner: repoOwner,
        repo: repoName,
        name: variableName,
        value: value,
      });
    }
  }
```

This function will update a configuration variable (or create the variable if it doesn't already exist). For more information, see "Variables."

```
(async () => {
  await checkAndRedeliverWebhooks();
})();
```

This will execute the `checkAndRedeliverWebhooks` function.

Testing the script

You can manually trigger your workflow to test the script. For more information, see "Manually running a workflow" and "Using workflow run logs."

Alternative methods

This example used GitHub Actions to securely store credentials and to run the script on a schedule. However, if you prefer to run this script on your server than handles webhook deliveries, you can:

- Store the credentials in another secure manner, such as a secret manager like Azure key vault. You will also need to update the script to access the credentials from their new location.
- Run the script on a schedule on your server, for example by using a cron job or task scheduler.
- Update the script to store the last run time somewhere that your server can access and update. If you choose not to store the last run time as a GitHub Actions secret, you do not need to use a personal access token, and you can remove the API calls to access and update the configuration variable.