Building a GitHub App that responds to webhook events

In this article

Learn how to build a GitHub App that makes an API request in response to a webhook event.

Introduction

This tutorial demonstrates how to write code to create a GitHub App that makes an API request in response to a webhook event. Specifically, when a pull request is opened in a repository that the app was granted access to, the app will receive a pull request webhook event. Then, the app will use GitHub's API to add a comment to the pull request.

In this tutorial, you will use your computer or codespace as a server while you develop your app. Once the app is ready for production use, you should deploy your app to a dedicated server.

This tutorial uses JavaScript, but you can use any programming language that you can run on your server.

About webhooks

When you register a GitHub App, you can specify a webhook URL and subscribe to webhook events. When activity on GitHub triggers an event that your app is subscribed to, GitHub will send a webhook event to your app's webhook URL.

For example, you can subscribe your GitHub App to the pull request webhook event. When a pull request is opened in a repository that the app was granted access to, GitHub will send a pull request webhook event to your app's webhook URL. If multiple actions can trigger the event, the event payload will include an `action` field to indicate what type of action triggered the event. In this example, the value of `action` would be `opened` since the event was triggered because a pull request was opened.

If your app is running on a server that is listening for these webhook events, then your

app can take an action when it receives a webhook event. For example, your app can use the GitHub API to post a comment to the pull request when it receives a pull request webhook event.

For more information, see "Using webhooks with GitHub Apps." For information about the possible webhook events and actions, see "Webhook events and payloads."

Prerequisites

This tutorial requires your computer or codespace to run Node.js version 12 or greater and npm version 6.12.0 or greater. For more information, see Node.js.

This tutorial assumes you have a basic understanding of JavaScript and ES6 syntax.

Setup

The following sections will lead you through setting up the following components:

- a repository to store the code for your app
- a way to receive webhooks locally
- a GitHub App registration that is subscribed to "pull request" webhook events, has permission to add comments to pull requests, and uses a webhook URL that you can receive locally

Create a repository to store code for your app

1. Create a repository to store the code for your app. For more information, see "Creating a new repository."
2. Clone your repository from the previous step. For more information, see "Cloning a repository." You may use a local clone or GitHub Codespaces.
3. In a terminal, navigate to the directory where your clone is stored.
4. If the directory doesn't already include a `.gitignore` file, add a `.gitignore` file. You will add content to this file later. For more information about `.gitignore` files, see "Ignoring files."

You will add more code to this repository in later steps.

Get a webhook proxy URL

In order to develop your app locally, you can use a webhook proxy URL to forward webhooks from GitHub to your computer or codespace. This tutorial uses Smee.io to provide a webhook proxy URL and forward webhooks.

1. In your browser, navigate to https://smee.io/.
2. Click **Start a new channel**.
3. Copy the full URL under "Webhook Proxy URL". You will use this URL in a later step.

Register a GitHub App

For this tutorial, you must have a GitHub App registration that:

- Has webhooks active
- Uses a webhook URL that you can receive locally
- Has the "Pull request" repository permission
- Subscribes to the "Pull request" webhook event

The following steps will guide you through registering a GitHub App with these settings. For more information about GitHub App settings, see "Registering a GitHub App."

1. Navigate to your account settings.
    - For a GitHub App owned by a personal account, in the upper-right corner of any page, click your profile photo, then click **Settings**.
    - For a GitHub App owned by an organization, in the upper-right corner of any page, click your profile photo, then click **Your organizations**. Then, to the right of the organization, click **Settings**.
2. In the left sidebar, click **Developer settings**.
3. In the left sidebar, click **GitHub Apps**.
4. Click **New GitHub App**.
5. Under "GitHub App name", enter a name for your app. For example, `USERNAME-webhook-test-app` where `USERNAME` is your GitHub username.
6. Under "Homepage URL", enter a URL for your app. For example, you can use the URL of the repository that you created to store the code for your app.
7. Skip the "Identifying and authorizing users" and "Post installation" sections for this tutorial. For more information about these settings, see "Registering a GitHub App."
8. Make sure that **Active** is selected under "Webhooks."
9. Under "Webhook URL", enter your webhook proxy URL from earlier. For more information, see "Get a webhook proxy URL."
10. Under "Webhook secret", enter a random string. You will use this string later.
11. Under "Repository permissions", next to "Pull requests," select **Read & write**.
12. Under "Subscribe to events", select **Pull request**.
13. Under "Where can this GitHub App be installed?", select **Only on this account**. You can change this later if you want to publish your app.
14. Click **Create GitHub App**.

Write code for your app

The following sections will lead you through writing code to make your app respond to webhook events.

Install dependencies

This tutorial uses GitHub's `octokit` module to handle webhook events and make API requests. For more information about Octokit.js, see "Scripting with the REST API and JavaScript" and the Octokit.js README.

This tutorial uses the `dotenv` module to read information about your app from a `.env`

file. For more information, see dotenv.

This tutorial uses Smee.io to forward webhooks from GitHub to your local server. For more information, see smee-client.

1. In a terminal, navigate to the directory where your clone is stored.
2. Run `npm init --yes` to create a `package.json` file using the npm defaults.

   The `--type module` option will add `type: module` to the resulting `package.json` file to indicate that the project uses ES modules. The `--yes` option accepts all other defaults.
3. In the `package.json` file that was created in the previous step, add a top level key `type` with the value `module`. For example:

   ```
   {
    ...rest of the JSON object,
    "version": "1.0.0",
    "description": "",
    "type": "module",
    ...rest of the JSON object,
   }
   ```
4. Run `npm install octokit`.
5. Run `npm install dotenv`.
6. Run `npm install smee-client --save-dev`. Since you will only use Smee.io to forward webhooks while you are developing your app, this is a dev dependency.
7. Add `node_modules` to your `.gitignore` file.

Store your app's identifying information and credentials

This tutorial will show you how to store your app's credentials and identifying information as environment variables in a `.env` file. When you deploy your app, you will want to change how you store the credentials. For more information, see "Deploy your app."

Make sure that you are on a secure machine before performing these steps since you will store your credentials locally.

1. In your terminal, navigate to the directory where your clone is stored.
2. Create a file called `.env` at the top level of this directory.
3. Add `.env` to your `.gitignore` file. This will prevent you from accidentally committing your app's credentials.
4. Add the following contents to your `.env` file. Replace `YOUR_HOSTNAME` with the name of your GitHub Enterprise Server instance. You will update the other values in a later step.

   Text

   ```
   APP_ID="YOUR_APP_ID"
   WEBHOOK_SECRET="YOUR_WEBHOOK_SECRET"
   PRIVATE_KEY_PATH="YOUR_PRIVATE_KEY_PATH"
   ENTERPRISE_HOSTNAME="YOUR_HOSTNAME"
   ```

5. Navigate to the settings page for your app:

   a. Navigate to your account settings.

      - For a GitHub App owned by a personal account, in the upper-right corner of any page, click your profile photo, then click **Settings**.
      - For a GitHub App owned by an organization, in the upper-right corner of any page, click your profile photo, then click **Your organizations**. Then, to the right of the organization, click **Settings**.

   b. In the left sidebar, click **Developer settings**.

   c. In the left sidebar, click **GitHub Apps**.

   d. Next to your app's name, click **Edit**.

6. On your app's settings page, next to "App ID", find the app ID for your app.
7. In your `.env` file, replace `YOUR_APP_ID` with the app ID of your app.
8. On your app's settings page, under "Private keys", click **Generate a private key**. You will see a private key in PEM format downloaded to your computer. For more information, see "Managing private keys for GitHub Apps."
9. If you are using a codespace, move the downloaded PEM file into your codespace so that your codespace can access the file.
10. In your `.env` file, replace `YOUR_PRIVATE_KEY_PATH` with the full path to your private key, including the `.pem` extension.
11. In your `.env` file, replace `YOUR_WEBHOOK_SECRET` with the webhook secret for your app. If you have forgotten your webhook secret, under "Webhook secret (optional)", click **Change secret**. Enter a new secret, then click **Save changes**.

Add code to respond to webhook events

These steps lead you through writing code to make an API request in response to webhook events. To skip ahead to the final code, see "Full code example."

1. In your terminal, navigate to the directory where your clone is stored.
2. At the top level of this directory, create a JavaScript file to hold the code for your app. This tutorial will name the file `app.js`.
3. To the `scripts` object in your `package.json` file, add a script called `server` that runs `node app.js`. For example:

   JSON

   ```
   "scripts": {
     "server": "node app.js"
   }
   ```

   Your `package.json` file should look something like this. The `name` and version numbers under `dependencies` and `devDependencies` may differ for you.

```
 {
  "name": "github-app-webhook-tutorial",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "server": "node app.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "dotenv": "^16.0.3",
    "octokit": "^2.0.14"
  },
  "devDependencies": {
    "smee-client": "^1.2.3"
  }
}
```

4. At the top of `app.js`, add these dependencies:

JavaScript

```
import dotenv from "dotenv";
import {App} from "octokit";
import {createNodeMiddleware} from "@octokit/webhooks";
import fs from "fs";
import http from "http";
```

You installed the `dotenv` and `octokit` modules earlier. The `@octokit/webhooks` is a dependency of the `octokit` module, so you don't need to install it separately. The `fs` and `http` dependencies are built-in Node.js modules.

5. Add the following code to `app.js`. This will read your `.env` file and add the variables from that file to the `process.env` object in Node.js.

JavaScript

```
dotenv.config();
```

6. Add the following code to `app.js` to read the values of your environment variables.

JavaScript

```
const appId = process.env.APP_ID;
const webhookSecret = process.env.WEBHOOK_SECRET;
const privateKeyPath = process.env.PRIVATE_KEY_PATH;
const enterpriseHostname = process.env.ENTERPRISE_HOSTNAME;
```

7. Add the following code to `app.js` to read the contents of your private key file.

JavaScript

```
const privateKey = fs.readFileSync(privateKeyPath, "utf8");
```

8. Add the following code to `app.js` to create a new instance of the Octokit App class.

```JavaScript
const app = new App({
  appId: appId,
  privateKey: privateKey,
  webhooks: {
    secret: webhookSecret
  },
  Octokit: Octokit.defaults({
    baseUrl: `https://${enterpriseHostname}/api/v3`,
  }),
});
```

9. Optionally, check your progress:

   a. Add the following code to `app.js` to make an API request and log the app's name:

```JavaScript
try {
  const {data} = await app.octokit.request("/app");
  console.log(`Authenticated as '${data.name}'`);
} catch (error) {
  if (error.response) {
    console.error(`Error! Status: ${error.response.status}. Message: ${err
  }
  console.error(error)
}
```

   b. Verify that `app.js` now looks like this:

```JavaScript
import dotenv from "dotenv";
import {App} from "octokit";
import {createNodeMiddleware} from "@octokit/webhooks";
import fs from "fs";
import http from "http";

dotenv.config();

const appId = process.env.APP_ID;
const webhookSecret = process.env.WEBHOOK_SECRET;
const privateKeyPath = process.env.PRIVATE_KEY_PATH;
const enterpriseHostname = process.env.ENTERPRISE_HOSTNAME;

const privateKey = fs.readFileSync(privateKeyPath, "utf8");

const app = new App({
  appId: appId,
  privateKey: privateKey,
  webhooks: {
    secret: webhookSecret
  },
  Octokit: Octokit.defaults({
    baseUrl: `https://${enterpriseHostname}/api/v3`,
  }),
```

```
});

try {
  const {data} = await app.octokit.request("/app");
  console.log(`Authenticated as '${data.name}'`);
} catch (error) {
  if (error.response) {
    console.error(`Error! Status: ${error.response.status}. Message: ${err
  }
  console.error(error)
}
```

c. In your terminal, from the directory where `app.js` is stored, run `npm run server`. This will execute the `server` script that you added to your `package.json` file earlier.

d. In your terminal, you should see output like this, where `YOUR-APP-NAME` is the name of your app.

```
Authenticated as 'YOUR-APP-NAME'
```

e. Delete the following code from `app.js`, which you added for testing purposes:

JavaScript

```
try {
  const {data} = await app.octokit.request("/app");
  console.log(`Authenticated as '${data.name}'`);
} catch (error) {
  if (error.response) {
    console.error(`Error! Status: ${error.response.status}. Message: ${err
  }
  console.error(error)
}
```

10. Add the following code to `app.js` to define the message that your app will post to pull requests.

JavaScript

```
const messageForNewPRs = "Thanks for opening a new PR! Please follow our contri
```

11. Add the following code to `app.js`.

This code adds an event handler that you will call later. When this event handler is called, it will log the event to the console. Then, it will use GitHub's REST API to add a comment to the pull request that triggered the event.

JavaScript

```
async function handlePullRequestOpened({octokit, payload}) {
  console.log(`Received a pull request event for #${payload.pull_request.number

  try {
```

```
    await octokit.request("POST /repos/{owner}/{repo}/issues/{issue_number}/com
      owner: payload.repository.owner.login,
      repo: payload.repository.name,
      issue_number: payload.pull_request.number,
      body: messageForNewPRs,
    });
  } catch (error) {
    if (error.response) {
      console.error(`Error! Status: ${error.response.status}. Message: ${error.
    }
    console.error(error)
  }
};
```

12. Add the following code to `app.js`.

    This code sets up a webhook event listener. When your app receives a webhook event from GitHub with a `X-GitHub-Event` header value of `pull_request` and an `action` payload value of `opened`, it calls the `handlePullRequestOpened` event handler that you added in the previous step.

    JavaScript

    ```
    app.webhooks.on("pull_request.opened", handlePullRequestOpened);
    ```

13. Add the following code to `app.js` to handle errors.

    JavaScript

    ```
    app.webhooks.onError((error) => {
      if (error.name === "AggregateError") {
        console.error(`Error processing request: ${error.event}`);
      } else {
        console.error(error);
      }
    });
    ```

14. Add the following code to `app.js` to determine where your server will listen:

    JavaScript

    ```
    const port = 3000;
    const host = 'localhost';
    const path = "/api/webhook";
    const localWebhookUrl = `http://${host}:${port}${path}`;
    ```

    For local development, your server will listen to port 3000 on `localhost`. When you deploy your app, you will change these values. For more information, see "Deploy your app."

15. Add the following code to `app.js` to set up a middleware function to handle incoming webhook events:

    JavaScript

    ```
    const middleware = createNodeMiddleware(app.webhooks, {path});
    ```

Octokit's `createNodeMiddleware` function takes care of generating this middleware function for you. The resulting middleware function will:

- Check the signature of the incoming webhook event to make sure that it matches your webhook secret. This verifies that the incoming webhook event is a valid GitHub event.
- Parse the webhook event payload and identify the type of event.
- Trigger the corresponding webhook event handler.

16. Add the following code to `app.js`:

```
JavaScript
```

```
http.createServer(middleware).listen(port, () => {
  console.log(`Server is listening for events at: ${localWebhookUrl}`);
  console.log('Press Ctrl + C to quit.')
});
```

This code creates a Node.js server that listens for incoming HTTP requests (including webhook payloads from GitHub) on the specified port. When the server receives a request, it executes the `middleware` function that you defined earlier. Once the server is running, it logs messages to the console to indicate that it is listening.

17. Check your code against the full code example in the next section. You can test your code by following the steps outlined in the "Testing" section below the full code example.

Full code example

This is the full code example that was outlined in the previous section. In addition to this code, you must also create a `.env` file with your app's credentials. For more information, see "Store your app's identifying information and credentials."

```
JavaScript
```

```
import dotenv from "dotenv";
import {App} from "octokit";
import {createNodeMiddleware} from "@octokit/webhooks";
import fs from "fs";
import http from "http";

dotenv.config();

const appId = process.env.APP_ID;
const webhookSecret = process.env.WEBHOOK_SECRET;
const privateKeyPath = process.env.PRIVATE_KEY_PATH;
const enterpriseHostname = process.env.ENTERPRISE_HOSTNAME;

const privateKey = fs.readFileSync(privateKeyPath, "utf8");

const app = new App({
  appId: appId,
  privateKey: privateKey,
  webhooks: {
    secret: webhookSecret
  },
  Octokit: Octokit.defaults({
    baseUrl: `https://${enterpriseHostname}/api/v3`,
  }),
```

```
});

const messageForNewPRs = "Thanks for opening a new PR! Please follow our contributi

async function handlePullRequestOpened({octokit, payload}) {
  console.log(`Received a pull request event for #${payload.pull_request.number}`);

  try {
    await octokit.request("POST /repos/{owner}/{repo}/issues/{issue_number}/comments
      owner: payload.repository.owner.login,
      repo: payload.repository.name,
      issue_number: payload.pull_request.number,
      body: messageForNewPRs,
    });
  } catch (error) {
    if (error.response) {
      console.error(`Error! Status: ${error.response.status}. Message: ${error.respo
    }
    console.error(error)
  }
};

app.webhooks.on("pull_request.opened", handlePullRequestOpened);

app.webhooks.onError((error) => {
  if (error.name === "AggregateError") {
    console.error(`Error processing request: ${error.event}`);
  } else {
    console.error(error);
  }
});

const port = 3000;
const host = 'localhost';
const path = "/api/webhook";
const localWebhookUrl = `http://${host}:${port}${path}`;

const middleware = createNodeMiddleware(app.webhooks, {path});

http.createServer(middleware).listen(port, () => {
  console.log(`Server is listening for events at: ${localWebhookUrl}`);
  console.log('Press Ctrl + C to quit.')
});
```

Testing

Follow these steps to test the app that you created above.

Install your app

In order for your app to leave a comment on pull requests in a repository, it must be installed on the account that owns the repository and granted access to that repository. Since your app is private, it can only be installed on the account that owns the app.

1. In the account that owns the app you created, create a new repository to install the app on. For more information, see "Creating a new repository."
2. Navigate to the settings page for your app:
    a. Navigate to your account settings.
        - For a GitHub App owned by a personal account, in the upper-right corner of any page, click your profile photo, then click **Settings**.
        - For a GitHub App owned by an organization, in the upper-right corner of

any page, click your profile photo, then click **Your organizations**. Then, to the right of the organization, click **Settings**.

b. In the left sidebar, click **Developer settings**.

c. In the left sidebar, click **GitHub Apps**.

d. Next to your app's name, click **Edit**.

3. Click **Public page**.
4. Click **Install**.
5. Select **Only select repositories**.
6. Select the **Select repositories** dropdown menu and click the repository that you chose at the start of this section.
7. Click **Install**.

Start your server

For testing, you will use your computer or codespace as a server. Your app will only respond to webhooks when your server is running.

1. In a terminal, navigate to the directory where your app's code is stored.
2. To receive forwarded webhooks from Smee.io, run `npx smee -u WEBHOOK_PROXY_URL -t http://localhost:3000/api/webhook`. Replace `WEBHOOK_PROXY_URL` with your webhook proxy URL from earlier. If you forgot your URL, you can find it in the "webhook URL" field on your app's settings page.

   You should see output that looks like this, where `WEBHOOK_PROXY_URL` is your webhook proxy URL:

   ```
   Forwarding WEBHOOK_PROXY_URL to http://localhost:3000/api/webhook
   Connected WEBHOOK_PROXY_URL
   ```

3. In a second terminal window, navigate to the directory where your app's code is stored.
4. Run `npm run server`. Your terminal should say, `Server is listening for events at: http://localhost:3000/api/webhook`.

Test your app

Now that your server is running and receiving forwarded webhooks events, test your app by opening a pull request on the repository that you selected when you installed your app.

1. Open a pull request on the repository that you selected when you installed your app. For more information, see "Creating a pull request."

   Make sure to use the repository that you selected when you installed your app, not the repository where your app's code is stored. For more information, see "Install your app."

2. Navigate to your webhook proxy URL on smee.io. You should see a `pull_request` event. This indicates that GitHub successfully sent a pull request event when you created a pull request.
3. In the terminal where you ran `npm run server`, you should see something like "Received a pull request event for #1" where the integer after the `#` is the number of the pull request that you opened.
4. In the timeline of your pull request, you should see a comment from your app.
5. In both terminal windows, enter `Ctrl` + `C` to stop your server and stop listening for forwarded webhooks.

Next steps

Now that you have an app that responds to webhook events, you might want to expand your app's code, deploy your app, and make your app public.

Modify the app code

This tutorial demonstrated how to post a comment on a pull request in when a pull request was opened. You can update the code to respond to different types of webhook events or to do something different in response to the webhook event.

Remember to update your app's permissions if your app needs additional permissions for the API requests that you want to make or the webhook events you want to receive. For more information, see "Choosing permissions for a GitHub App."

This tutorial stored all of the code into a single file, but you may want to move functions and components into separate files.

Deploy your app

This tutorial demonstrated how to develop your app locally. When you are ready to deploy your app, you need to make changes to serve your app and keep your app's credential secure. The steps you take depend on the server that you use, but the following sections offer general guidance.

Host your app on a server

This tutorial used your computer or codespace as a server. Once the app is ready for production use, you should deploy your app to a dedicated server. For example, you can use Azure App Service.

Update the webhook URL

Once you have a server that is set up to receive webhook traffic from GitHub, update the webhook URL in your app settings. You should not use Smee.io to forward your webhooks in production.

Update the `port` and `host` constants

When you deploy your app, you will want to change the host and port where your server is listening.

For example, you can set a `PORT` environment variable on your server to indicate the

port where your server should listen. You can set a `NODE_ENV` environment variable on your server to `production`. Then, you can update the place where your code defines the `port` and `host` constants so that your server listens to all available network interfaces (`0.0.0.0`) instead of the local network interface (`localhost`) on your deployment port:

JavaScript

```
const port = process.env.PORT || 3000;
const host = process.env.NODE_ENV === 'production' ? '0.0.0.0' : 'localhost';
```

Secure your app's credentials

You should never publicize your app's private key or webhook secret. This tutorial stored your app's credentials in a gitignored `.env` file. When you deploy your app, you should choose a secure way to store the credentials and update your code to get the value accordingly. For example, you can store the credentials with a secret management service like Azure Key Vault. When your app runs, it can retrieve the credentials and store them in environment variables on the server where your app is deployed.

For more information, see "Best practices for creating a GitHub App."

Share your app

If you want to share your app with other users and organizations, make your app public. For more information, see "Making a GitHub App public or private."

Follow best practices

You should aim to follow best practices with your GitHub App. For more information, see "Best practices for creating a GitHub App."