Setting up your development environment to create a GitHub App

In this article

Learn the foundations for extending and building new GitHub Apps.

Introduction

This guide will walk through the steps needed to configure a GitHub App and run it on a server. GitHub Apps require some setup steps to manage webhook events and connect the app registration on GitHub to your code. The app in this guide serves as a foundation that you can use to extend and build new GitHub Apps.

By the end of this guide you'll have registered a GitHub App and set up a web server to receive webhook events. You'll learn how to use a tool called Smee to capture webhook payloads and forward them to your local development environment. The template app you'll configure in this section won't do anything special yet, but it will serve as a framework you can use to start writing app code using the API or complete other quickstart guides.

After completing this project you will understand how to authenticate as a GitHub App and an installation, and how those authentication methods are different.

Here are the steps you'll take to configure the template GitHub App:

1. Start a new Smee channel
2. Register a new GitHub App

3. Save your private key and App ID
4. Prepare the runtime environment
5. Review the GitHub App template code
6. Start the server
7. Install the app on your account

> **Note:** This guide demonstrates the app development process using the Ruby programming language. However, there are many flavors of Octokit. If you prefer JavaScript, you can use Probot and Node.js to develop GitHub Apps.

Prerequisites

You may find it helpful to have a basic understanding of the following:

- GitHub Apps
- Webhooks
- The Ruby programming language
- REST APIs
- Sinatra

But you can follow along at any experience level. We'll link out to information you need along the way!

Before you begin, you'll need to clone the repository with the template code used in this quickstart. Open your Terminal app and find a directory where you'd like to store the code. Run this command to clone the GitHub App template repository:

```
$ git clone https://github.com/github-developer/github-app-template.git
```

Step 1. Start a new Smee channel

To help GitHub send webhooks to your local machine without exposing it to the internet, you can use a tool called Smee. First, go to https://smee.io and click **Start a new channel**. If you're already comfortable with other tools that expose your local machine to the internet like `ngrok` or `localtunnel`, feel free to use those.

Starting a new Smee channel creates a unique domain where GitHub can send webhook payloads. Smee calls the URL for this unique domain the "Webhook Proxy URL." You'll need to know this URL for the next step.

Next, go back to the Terminal and follow these steps to run the Smee command-line interface (CLI) client:

> **Note:** The following steps are slightly different than the "Use the CLI" instructions you'll see in your Smee channel page. You do **not** need to follow the "Use the Node.js client" or "Using Probot's built-in support" instructions.

1. Install the client:

   ```
   $ npm install --global smee-client
   ```

2. Run the client (replacing `https://smee.io/qrfeVRbFbffd6vD` with your own domain):

```
$ smee --url https://smee.io/qrfeVRbFbffd6vD --path /event_handler --port 3000
```

You should see output like the following:

```
Forwarding https://smee.io/qrfeVRbFbffd6vD to http://127.0.0.1:3000/event_handl
Connected https://smee.io/qrfeVRbFbffd6vD
```

The `smee --url <unique_channel>` command tells Smee to forward all webhook events received by the Smee channel to the Smee client running on your computer. The `--path /event_handler` option forwards events to the `/event_handler` route, which we'll cover in a later section. The `--port 3000` option specifies port 3000, which is the port your server will be listening to. Using Smee, your machine does not need to be open to the public internet to receive webhooks from GitHub. You can also open that Smee URL in your browser to inspect webhook payloads as they come in.

We recommend leaving this Terminal window open and keeping Smee connected while you complete the rest of the steps in this guide. Although you *can* disconnect and reconnect the Smee client without losing your unique domain (unlike `ngrok`), you may find it easier to leave it connected and do other command-line tasks in a different Terminal window.

Step 2. Register a new GitHub App

If you don't yet have a GitHub account, now is a great time to join. Don't forget to verify your email before continuing! To register a new app, visit the app settings page in your GitHub profile, and click **New GitHub App**.

You'll see a form where you can enter details about your app. See "Registering a GitHub App" for general information about the fields on this page. For the purposes of this guide, you'll need to enter specific data in a few fields:

> **Note:** You can always update these settings later to point to a hosted server.

- For the "Homepage URL", use the domain issued by Smee.
- For the "Webhook URL", again use the domain issued by Smee.
- For the "Webhook secret", create a password to secure your webhook endpoints. This should be something that only you (and GitHub, via this form) know. The secret is important because you will be receiving payloads from the public internet, and you'll use this secret to verify the webhook sender. Note that the GitHub App settings say the webhook secret is optional, which is true in most cases, but for the template app code to work, you must set a webhook secret.
- On the Permissions & Webhooks page, you can specify a set of permissions for your app, which determines how much data your app has access to. Under the "Repository permissions" section, scroll down to "Metadata" and select `Access: Read-only`. If you decide to extend this template app, you can update these permissions later.
- At the bottom of the Permissions & Webhooks page, under "Where can this GitHub App be installed?", specify whether this is a private app or a public app.

This refers to who can install it: just you, or anyone in the world? For now, leave the app as private by selecting **Only on this account**.

Click **Create GitHub App** to create your app!

Step 3. Save your private key and App ID

After you create your app, you'll be taken back to the app settings page. You have two more things to do here:

- **Note the app ID GitHub has assigned your app**, which is displayed in the "About" section. You'll need this to prepare your runtime environment.
- **Generate a private key for your app**. This is necessary to authenticate your app later on. Scroll down to the "Private keys" section and click **Generate a private key**. Save the resulting `PEM` file (called something like *`app-name`*-*`date`*-`private-key.pem`) in a directory where you can find it again.

Step 4. Prepare the runtime environment

To keep your information secure, we recommend putting all your app-related secrets in your computer's memory where your app can find them, rather than putting them directly in your code. A handy development tool called dotenv loads project-specific environment variables from a `.env` file to `ENV`. Never check your `.env` file into GitHub. This is a local file that stores sensitive information that you don't want on the public internet. The `.env` file is already included in the repository's `.gitignore` file to prevent that.

The template code you downloaded in the Prerequisites section already has an example file called `.env-example`. Rename the example file from `.env-example` to `.env` or create a copy of the `.env-example` file called `.env`. You haven't installed dotenv yet, but you will install it later in this quickstart when you run `bundle install`. **Note:** Quickstarts that reference the steps in this guide may include additional environment variables in the `.env-example` file. Reference the quickstart guide for the project you've cloned on GitHub for guidance setting those additional environment variables.

You need to add these variables to the `.env` file:

- *`GITHUB_PRIVATE_KEY`*: Add the private key you generated and saved previously. Open the `.pem` file with a text editor or use the command line to display the contents of the file: `cat path/to/your/private-key.pem`. Copy the entire contents of the file as the value of `GITHUB_PRIVATE_KEY` in your `.env` file. **Note:** Because the PEM file is more than one line you'll need to add quotes around the value like the example below.
- *`GITHUB_APP_IDENTIFIER`*: Use the app ID you noted in the previous section.
- *`GITHUB_WEBHOOK_SECRET`*: Add your webhook secret.

Here is an example `.env` file:

```
GITHUB_PRIVATE_KEY="-----BEGIN RSA PRIVATE KEY-----
...
HkVN9...
...
-----END DSA PRIVATE KEY-----"
GITHUB_APP_IDENTIFIER=12345
GITHUB_WEBHOOK_SECRET=your webhook secret
```

Step 5. Review the GitHub App template code

The template app code already contains some code that every GitHub App will need. This sections walks you through the code that already exists in the GitHub App template. There aren't any steps that you need to complete in this section. If you're already familiar with the template code, you can skip ahead to "Step 6. Start the server."

Open up the `template_server.rb` file in your favorite text editor. You'll see comments throughout this file that provide additional context for the template code. We recommend reading those comments carefully and even adding your own comments to accompany new code you write.

At the top of the file you'll see `set :port 3000`, which sets the port used when starting the web server to match the port you redirected your webhook payloads to in "Step 1. Start a new Smee channel."

The next code you'll see is the `class GHApp < Sinatra::Application` declaration. You'll write all of the code for your GitHub App inside this class.

Out of the box, the class in the template does the following things:

- Read the environment variables
- Turn on logging
- Define a before filter
- Define the route handler
- Define the helper methods

Read the environment variables

The first thing that this class does is read the three environment variables you set in "Step 4. Prepare the runtime environment" and store them in variables to use later:

```
# Expects that the private key in PEM format. Converts the newlines
PRIVATE_KEY = OpenSSL::PKey::RSA.new(ENV['GITHUB_PRIVATE_KEY'].gsub('\n', "\n"))

# Your registered app must have a secret set. The secret is used to verify
# that webhooks are sent by GitHub.
WEBHOOK_SECRET = ENV['GITHUB_WEBHOOK_SECRET']

# The GitHub App's identifier (type integer) set when registering an app.
APP_IDENTIFIER = ENV['GITHUB_APP_IDENTIFIER']
```

Turn on logging

Next is a code block that enables logging during development, which is the default environment in Sinatra. This code turns on logging at the `DEBUG` level to show useful output in the Terminal while you are developing the app:

```
# Turn on Sinatra's verbose logging during development
configure :development do
  set :logging, Logger::DEBUG
end
```

Define a before filter

Sinatra uses before filters that allow you to execute code before the route handler. The `before` block in the template calls four helper methods. The template app defines those helper methods in a later section.

```
# Before each request to the `/event_handler` route
before '/event_handler' do
```

```
  get_payload_request(request)
  verify_webhook_signature
  authenticate_app
  # Authenticate the app installation in order to run API operations
  authenticate_installation(@payload)
end
```

Define a route handler

An empty route is included in the template code. This code handles all `POST` requests to the `/event_handler` route. You won't write this event handler in this quickstart, but see the other quickstart guides for examples of how to extend this template app.

```
post '/event_handler' do

end
```

Define the helper methods

The helper methods in this template do most of the heavy lifting. Four helper methods are defined in this section of the code.

Handling the webhook payload

The first method `get_payload_request` captures the webhook payload and converts it to JSON format, which makes accessing the payload's data much easier.

Verifying the webhook signature

The second method `verify_webhook_signature` performs verification of the webhook signature to ensure that GitHub generated the event. To learn more about the code in the `verify_webhook_signature` helper method, see "Securing your webhooks." If the webhooks are secure, this method will log all incoming payloads to your Terminal. The logger code is helpful in verifying your web server is working but you can always remove it later.

Authenticating as a GitHub App

To make API calls, you'll be using the Octokit library. Doing anything interesting with this library will require you, or rather your app, to authenticate. GitHub Apps have two methods of authentication:

- Authenticating as a GitHub App using a JSON Web Token (JWT).
- Authenticating as a specific installation of a GitHub App using an installation access token.

You'll learn about authenticating as an installation in the next section.

Authenticating as a GitHub App lets you do a couple of things:

- You can retrieve high-level management information about your GitHub App.
- You can request access tokens for an installation of the app.

For example, you would authenticate as a GitHub App to retrieve a list of the accounts (organization and personal) that have installed your app. But this authentication method doesn't allow you to do much with the API. To access a repository's data and perform operations on behalf of the installation, you need to authenticate as an installation. To do that, you'll need to authenticate as a GitHub App first to request an installation access token. For more information, see "About authentication with a GitHub App."

Before you can use the Octokit.rb library to make API calls, you'll need to initialize an Octokit client authenticated as a GitHub App. The `authenticate_app` helper method does just that!

```
# Instantiate an Octokit client authenticated as a GitHub App.
# GitHub App authentication requires that you construct a
# JWT (https://jwt.io/introduction/) signed with the app's private key,
# so GitHub can be sure that it came from the app an not altered by
# a malicious third party.
def authenticate_app
  payload = {
      # The time that this JWT was issued, _i.e._ now.
      iat: Time.now.to_i,

      # JWT expiration time (10 minute maximum)
      exp: Time.now.to_i + (10 * 60),

      # Your GitHub App's identifier number
      iss: APP_IDENTIFIER
  }

  # Cryptographically sign the JWT
  jwt = JWT.encode(payload, PRIVATE_KEY, 'RS256')

  # Create the Octokit client, using the JWT as the auth token.
  @app_client ||= Octokit::Client.new(bearer_token: jwt)
end
```

The code above generates a JSON Web Token (JWT) and uses it (along with your app's private key) to initialize the Octokit client. GitHub checks a request's authentication by verifying the token with the app's stored public key. To learn more about how this code works, see "Generating a JSON Web Token (JWT) for a GitHub App."

Authenticating as an installation

An *installation* refers to any user or organization account that has installed the app. Even if someone installs the app on more than one repository, it only counts as one installation because it's within the same account. The last helper method `authenticate_installation` initializes an Octokit client authenticated as an installation. This Octokit client is what you'd use to make authenticated API calls.

```
# Instantiate an Octokit client authenticated as an installation of a
# GitHub App to run API operations.
def authenticate_installation(payload)
  installation_id = payload['installation']['id']
  installation_token = @app_client.create_app_installation_access_token(installatio
  @installation_client = Octokit::Client.new(bearer_token: installation_token)
end
```

The `create_app_installation_access_token` Octokit method creates an installation token. This method accepts two arguments:

- Installation (integer): The ID of a GitHub App installation
- Options (hash, defaults to `{}`): A customizable set of options

Any time a GitHub App receives a webhook, it includes an `installation` object with an `id`. Using the client authenticated as a GitHub App, you pass this ID to the `create_app_installation_access_token` method to generate an access token for each installation. Since you're not passing any options to the method, the options default to an empty hash. The response for `create_app_installation_access_token` includes two fields: `token` and `expired_at`. The template code selects the token in the response and initializes an installation client.

With this method in place, each time your app receives a new webhook payload, it creates a client for the installation that triggered the event. This authentication process enables your GitHub App to work for all installations on any account.

Now you're ready to start making API calls!

Step 6. Start the server

Your app doesn't *do* anything yet, but at this point, you can get it running on the server.

Keep Smee running in the current tab in your Terminal. Open a new tab and `cd` into the directory where you cloned the template app code. The Ruby code in this repository will start up a Sinatra web server. This code has a few dependencies. You can install these by running:

```
$ gem install bundler
```

Followed by:

```
$ bundle install
```

With the dependencies installed, you can start the server:

```
$ bundle exec ruby template_server.rb
```

You should see a response like:

```
> == Sinatra (v2.0.3) has taken the stage on 3000 for development with backup from P
> Puma starting in single mode...
> * Version 3.11.2 (ruby 2.4.0-p0), codename: Love Song
> * Min threads: 0, max threads: 16
> * Environment: development
> * Listening on tcp://localhost:3000
> Use Ctrl-C to stop
```

If you see an error, make sure you've created the `.env` file in the directory that contains `template_server.rb`.

Once the server is running, you can test it by going to `http://localhost:3000` in your browser. If the app works as expected, you'll see a helpful error page that says, "Sinatra doesn't know this ditty."

This is good! Even though it's an error page, it's a *Sinatra* error page, which means your app is connected to the server as expected. You're seeing this message because you haven't given the app anything else to show.

Step 7. Install the app on your account

You can test that the server is listening to your app by triggering an event for it to receive. A simple event you can test is installing the app on your GitHub account, which should send the `installation` event. If the app receives it, you should see some output in the Terminal tab where you started `template_server.rb`.

To install the app, visit the app settings page, choose your app, and click **Install App** in the sidebar. Next to your username, click **Install**.

You'll be asked whether to install the app on all repositories or selected repositories. If

you don't want to install the app on *all* of your repositories, that's okay! You may want to create a sandbox repository for testing purposes and install your app there.

After you click **Install**, look at the output in your Terminal. You should see something like this:

```
> D, [2018-06-29T15:45:43.773077 #30488] DEBUG -- : ---- received event integration_
> D, [2018-06-29T15:45:43.773141 #30488] DEBUG -- : ----          action created
> 192.30.252.44 - - [29/Jun/2018:15:45:43 -0400] "POST / HTTP/2" 200 2 0.0067
> D, [2018-06-29T15:45:43.833016 #30488] DEBUG -- : ---- received event installatio
> D, [2018-06-29T15:45:43.833062 #30488] DEBUG -- : ----          action created
> 192.30.252.39 - - [29/Jun/2018:15:45:43 -0400] "POST / HTTP/2" 200 2 0.0019
```

This is good news! It means your app received a notification that it was installed on your GitHub account. If you see something like this, your app is running on the server as expected.

If you don't see the output, make sure Smee is running correctly in another Terminal tab. If you need to restart Smee, note that you'll also need to *uninstall* and *reinstall* the app to send the `installation` event to your app again and see the output in Terminal. If Smee isn't the problem, see the "Troubleshooting" section for other ideas.

If you're wondering where the Terminal output above is coming from, it's written in the app template code in `template_server.rb`.

Troubleshooting

Here are a few common problems and some suggested solutions. If you run into any other trouble, you can ask for help or advice in the APIs and Integrations discussions on GitHub Community.

- **Q:** When I try to install the Smee command-line client, I get the following error:

  ```
  > npm: command not found
  ```

  A: Looks like you don't have npm installed. The best way to install it is to download the Node.js package at https://nodejs.org and follow the installation instructions for your system. npm will be installed alongside Node.js.

- **Q:** When I run the server, I get the following error:

  ```
  > server.rb:38:in `initialize': Neither PUB key nor PRIV key: header too long (0
  ```

  A: You probably haven't set up your private key environment variable quite right. Your `GITHUB_PRIVATE_KEY` variable should look like this:

  ```
  GITHUB_PRIVATE_KEY="-----BEGIN RSA PRIVATE KEY-----
  ...
  HkVN9...
  ...
  -----END RSA PRIVATE KEY-----"
  ```

  Double-check that you've copied the correct public key into your `.env` file.

- **Q:** When I run the server, it crashes with this error:

  ```
  > Octokit::Unauthorized ... 401 - Bad credentials`
  ```

A: You may be authenticated as a GitHub App but not as an installation. Make sure you follow all the steps under "Authenticate as an installation," and use the `@installation_client` instance variable (authenticated with an installation access token) for your API operations, not the `@app_client` instance variable (authenticated with a JWT). The `@app_client` can only retrieve high-level information about your app and obtain installation access tokens. It can't do much else in the API.

- **Q:** My server isn't listening to events! The Smee client is running in a Terminal window, and I'm installing the app on a repository on GitHub, but I don't see any output in the Terminal window where I'm running the server.

  A: You may not be running the Smee client, running the Smee command with the wrong parameters or you may not have the correct Smee domain in your GitHub App settings. First check to make sure the Smee client is running in a Terminal tab. If that's not the problem, visit your app settings page and check the fields shown in "Step 2. Register a new GitHub App." Make sure the domain in those fields matches the domain you used in your `smee -u <unique_channel>` command in "Step 1. Start a new Smee channel." If none of the above work, check that you are running the full Smee command including the `--path` and `--port` options, for example: `smee --url https://smee.io/qrfeVRbFbffd6vD --path /event_handler --port 3000` (replacing `https://smee.io/qrfeVRbFbffd6vD` with your own Smee domain).

- **Q:** I'm getting an `Octokit::NotFound` 404 error in my debug output:

  ```
  2018-12-06 15:00:56 - Octokit::NotFound - POST http(s)://HOSTNAME/api/v3/app/ins
  ```

  A: Ensure the variables in your `.env` file are correct. Make sure that you have not set identical variables in any other environment variable files like `bash_profile`. You can check the environment variables your app is using by adding `puts` statements to your app code and re-running the code. For example, to ensure you have the correct private key set, you could add `puts PRIVATE_KEY` to your app code:

  ```
  PRIVATE_KEY = OpenSSL::PKey::RSA.new(ENV['GITHUB_PRIVATE_KEY'].gsub('\n', "\n"))
  puts PRIVATE_KEY
  ```

Conclusion

After walking through this guide, you've learned the basic building blocks for developing GitHub Apps! To review, you:

- Registered a new GitHub App
- Used Smee to receive webhook payloads
- Ran a simple web server via Sinatra
- Authenticated as a GitHub App
- Authenticated as an installation

Next steps

You now have a GitHub App running on a server. It doesn't do anything special yet, but check out some of the ways you can customize your GitHub App template in the other quickstart guides.