Deploying to PythonAnywhere via GitHub

Aadi Bajpai Following 5 min read · Jun 20, 2019

Everyone can do this:

local → github

With ssh access (paid) you can do this:

local → pythonanywhere

What I do here is (free):

local → github → pythonanywhere

I first list why you might wanna do this and then go to the how. If you want to directly jump to the how then feel free to take a leap.

PythonAnywhere is awesome. It's free, good resources, even a database and you can have a dynamic website running in minutes. It's a very good resource if you're starting out and just want to play around or even if you want to host an API or something of your own.

But it has its flaws, what if you want to open-source what you're working on? Do you maintain two separate places and keep committing twice? Once to production at PythonAnywhere and once again to GitHub for people to check your code out. What if you merge a Pull Request or want to integrate CI? Doing the same thing over and over again sucks.

GitHub is great for collaboration and even viewing the code, they have a way better UI than PythonAnywhere and I mean, editing code directly on PythonAnywhere leaves something to be desired. What if there was a way to link all of it?

You push to GitHub and it updates your running web app at PythonAnywhere and reloads it as well. You can merge a PR, use Issues, browse your code from anywhere without even signing in and opening the file in an editor and well everything GitHub has to offer.

I hear you, "Okay Aadi, all of this sounds well and good but how?"

Say no more.

We use GitHub webhooks to tell our application that it has been updated so it does a pull and then reloads itself.

I'll take the example of my application SwagLyrics whose backend is hosted on PythonAnywhere. I'm using Flask so you might want to adjust accordingly if you're not. This is my repo so you can use it as a reference:

SwagLyrics/swaglyrics-issue-maker

remote server code to make an issue on the main repo for unsupported songs. — SwagLyrics/swaglyrics-issue-maker

github.com

First off, make sure your repository on GitHub and PythonAnywhere are in sync, the GitHub one should be the origin and if you haven't made a repo on PythonAnywhere yet, you could initialise one or clone into PythonAnywhere from GitHub.

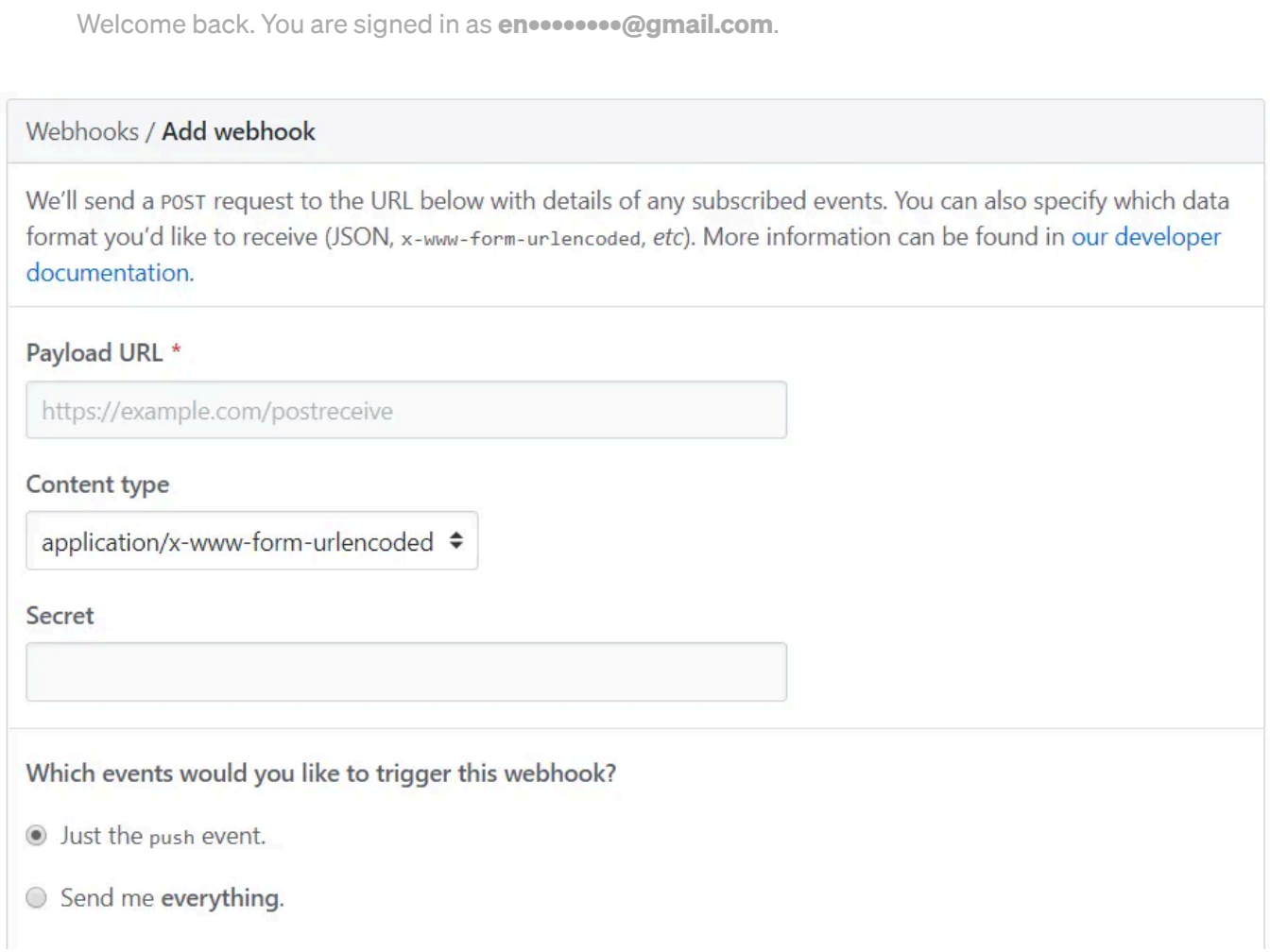
Something like

```
git init
```

```
git remote add origin https://github.com/yourusername/yourreponame.git
```

Now go to your repository on GitHub → Settings → Webhooks → Add webhook.

This is what you see:



In the Payload URL add https://your_domain/route_to_update

Eg. my repo has the webhook sent to https://aadibajpai.pythonanywhere.com/update_server

Then change the Content type from `application/x-www-form-urlencoded` to `application/json` (I'll tell you why a bit later).

We'll come to the secret part a bit later as well.

Make sure it's the push event that triggers the webhook and click on Add webhook.

Now, open up your flask app and we'll configure a route which receives the in

the same as the one we used in the Payload URL. We do not explicitly checkout master since I assume that's the only branch here.

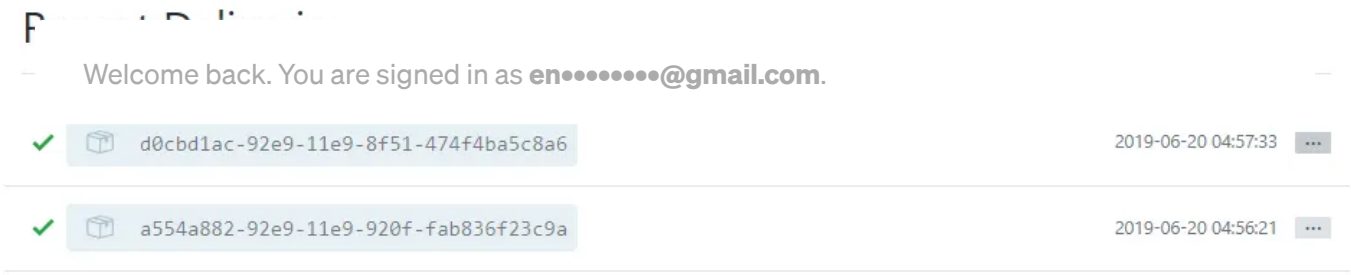
A barebones simple setup would look something like this:

```
from flask import Flask, request
import git

app = Flask(__name__)

@app.route('/update_server', methods=['POST'])
    def webhook():
        if request.method == 'POST':
            repo = git.Repo('path/to/git_repo')
            origin = repo.remotes.origin

origin.pull()

return 'Updated PythonAnywhere successfully', 200
        else:
            return 'Wrong event type', 400
```

This is a very rudimentary version and there's a more complete version below.

Now whenever there's a push event, this route will receive the info and update itself by doing a pull.

Hopefully, this is what it looks like when you make a commit



fingers crossed

Before I tell you how to secure your webhook, I'll tell you how to make sure your app reloads after a pull so you don't have to manually do that.

Auto-reloading your webapp

We're going to use git hooks to do it. These are shell commands executed after a git event takes place. There's no post-pull hook, but...

We take advantage of the fact that a `git pull` is nothing but a `git fetch` → `git merge` and there is a post-merge hook. It executes if the pull completed successfully.

In your git repo over at PythonAnywhere go to .git/hooks/

There will be a bunch of pre-existing ones but make a new file called post-merge.

Put the following code there:

```
#!/bin/sh
touch /path/to/username_pythonanywhere_com_wsgi.py
```

Use the path to your wsgi file which when touched, reloads your webapp.

Now to make this executable, open a bash console there and run


```
chmod +x post-merge
```

That's it, now you can test that it works by making a sample commit.

Most importantly, now we move on to securing your webhook.

Securing your webhook

This is necessary so that someone else cannot spam your server with requests to update it.

We follow this guide for it https://developer.github.com/webhooks/securing/

First off, export that secret token and add it to PythonAnywhere as an environment variable as well as in the Secret field in your GitHub webhook settings. This might be helpful https://help.pythonanywhere.com/pages/environment-variables-for-web-apps

Now, GitHub lists their method in Ruby but we use Python so while the method is same, here is the comparison function:

Now, modify your update_server route to check if signature is valid by adding these lines before the updation part of your route:

```
x_hub_signature = request.headers.get('X-Hub-Signature')

if not is_valid_signature(x_hub_signature, request.data, w_secret):
```

w_secret is supposed to be your webhooks secret that you set as an

en

Now it doesn't make much sense to have no logging as well as no other security checks to verify that the webhook is in fact from GitHub or that the pull event actually contained data, so you can go and copy paste the route on my repo that contains all that into your own and modify it for yourself where necessary, you already know the important parts by now 😀

I hope this was helpful, I know it's a bit detailed but I wanted you to know what's happening. If you found it too long, you could get away with copy pasting a lot of it after configuring the webhook over at GitHub and making the git hook.

I obviously didn't come up with it myself, rather I'd say I just cherry picked the right parts from the right places and experimented in the hope that this can be a complete resource.

Here are the references so you can know and do more with it.

1. https://stackoverflow.com/a/54268132/9044659 (basic setup)
2. https://developer.github.com/webhooks/ (GitHub webhooks documentation)
3. https://github.com/CCExtractor/sample-platform/blob/master/mod_deploy/controllers.py (detailed implementation with better checks, in case you want to go all in)

4. https://github.com/SwagLyrics/swaglyrics-issue-

#L268 (my own route to update PythonAnywhere)

5. https://github.com/SwagLyrics/SwagLyrics-For-Spotify (this is where I use it, shameless repo plug to get stars ❤️)

Please let me know if there is any part that can be improved or made easier to understand.

Thanks a lot for reading!

P. S. I did stay up way late doing this, the things we do for science *sigh*

Github Python Pythonanywhere Flask Git

Written by Aadi Bajpai

36 followers · 20 following

Following

design some, code some // clashkahznlvpwfg.onion // I don't write here anymore
go to https://aadibajpai.com/blog/

Responses (8)

Eng Omar Essam