LaTeX Font Warning: Size substitutions with differences (Font) up to 117.74374pt have occurred.

Build a Serverless Github Bot in GCP

Franklin E. Diaz

frank378@gmail.com

December 1, 2022

Abstract

Did you ever wonder how the cool kids get their bots going to manage pull requests in Github? The bots that can comment on Pull Requests, label things, perform other actions that are helpful to human developers? Well so did I, so I assembled one a while back that. Read on for more details.

Contents

List of Figures

List of Tables

Part I
Introduction

1 Build a Serverless Github Bot in GCP

This workshop is meant to be a fun way to learn more about some modern software development technologies. You don't need to have a deep understanding of all parts. Rather, you can follow along from end to end and choose to focus more deeply on any part that holds your attention.

Yes, there are easier ways to do many of the things in this document. It's OK to use those easier ways. Doing things "the hard way" may give deeper understanding and valuable insight. You have to show up to the gym every day and put in the work if you want the results. Same with this.

Note that this document has many clickable links included. You may miss some of these should you choose to print a copy of this document.

1.1 Outline

So what are we trying to do here, exactly? The idea is that we will run some Python that will monitor a Github repository for new pull requests. When a new PR comes in, we can define a set of actions to be taken that can help us manage that pull request and subsequent commits.

add a diagram here that shows the overall workflow
A high level overview of the learning path is as follows:

- Prerequisites
- Github setup.
- Set up a development environment.
- Review the Python source for the bot.
- Configure Terraform and deploy the bot.
- Test it out.
- Explore possibilities for extending the functionality.

Part II
Setup

2 Getting Set Up

A Dockerfile that has everything you need to participate in this workshop is provided. The fastest way to get set up is to clone the project repository from Github, then open that repository in Visual Studio Code.

2.1 VS Code

Visual Studio Code is a popular integrated development environment (IDE) that works with Linux, Mac, and other less popular operating systems. Click here to download the latest VSCode for your system. The other thing that make VSCode a great platform for our workshop is the idea that we can develop our project in a containerized environment.

2.2 Github Repository Setup

There are two significant branches in the Github repository, named "workshop" and "develop". The live session will be based on the workshop branch of the repository. The goal of having a separate workshop branch is an effort to declutter the environment, thus making it easier to follow along.

It is possible, and perfectly acceptable, to clone the workshop files from the develop branch of the workshop repository. This advanced path is only recommended for folks who already have a decent grasp on most of the files, packages, and paradigms we will use during the live session. You will be more or less on your own for installation and configuration, since most folks will probably opt for the golden path. That said, I encourage you to try!

Once you clone the Github repository, you should open that folder in VS Code. In the bottom right of the IDE window you should see a message similar to Figure 1. Click the "Reopen in Container" button to proceed.

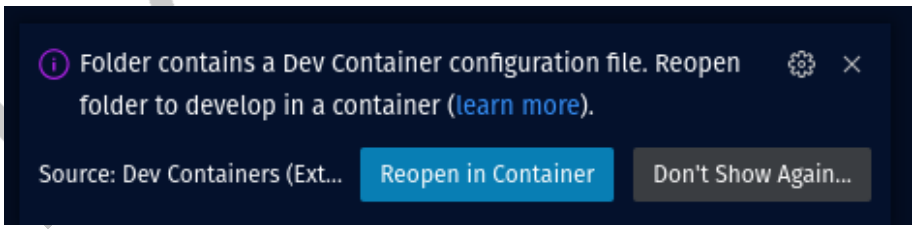

Figure 1: Opening the development container environment.

2.3 Set Up a Second Github Account

The second Github account will be your "bot account". Note that this requires a separate e-mail address. Once you have registered, preform the following steps.

1. Create a new Github account using the second e-mail address.
2. As your "main user", invite the bot user "bot-account" as a collaborator on the repo.
3. Log out of Github, log in on your second account and accept the invitation to join the repo as a contributor.

Note that you can add a cool icon to the bot accountś Github profile.

2.4 GH Action YAML File

```
name: 'codemash-cloudbot'
on:
- pull_request
env:
PR_NUMBER: ${{ github.event.pull_request.number }}
jobs:
phone-home:
    name: 'codemash-cloudbot'
    runs-on: ubuntu-latest
    steps:
    - name: Set up Cloud SDK
        uses: 'google-github-actions/setup-gcloud@v0'
        with:
        project_id: ${{ secrets.PROJECT_ID }}
        service_account_key: ${{ secrets.GOOGLE_APPLICATION_CREDENTIALS }}
        export_default_credentials: true
    - name: Use gcloud CLI
        run: |
        curl --request POST \
            --header "Content-Type:application/json" \
            --header "Authorization: Bearer $(gcloud auth print-identity-token)" \
            --data "{\"message\": \"Pull request number $PR_NUMBER by \
            $GITHUB_ACTOR on repository $GITHUB_REPOSITORY\", \"pr_number\": \
            \"$PR_NUMBER\", \"user\": \"$GITHUB_ACTOR\", \"repo\": \
            \"$GITHUB_REPOSITORY\", \"ref\": \"$GITHUB_REF\", \"commit_sha\": \
            \"$GITHUB_SHA\"}" \
            \ https://us-central1-gcp-gcs-pso.cloudfunctions.net/codemash-cloudbot
```

2.5 Github Webhook Configuration

There are three parts to configuring the webhook in Github.

- Set the payload URL.
- Select the correct content type from the drop down.
- Add the webhook secret.

Note that the webhook secret is different from the two Github secrets we will set in section 2.6.

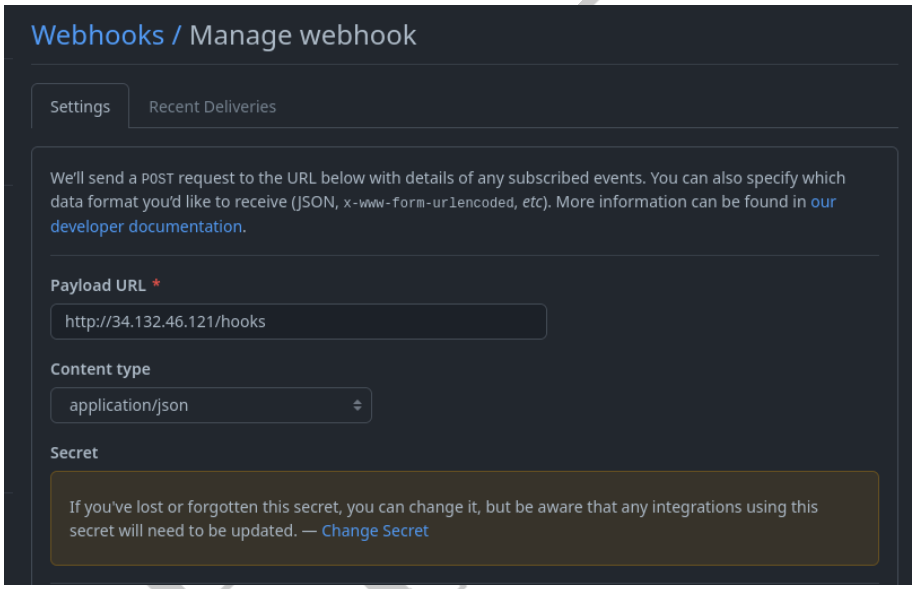

Figure 2: Payload URL, Content type, and Webhook Secret.

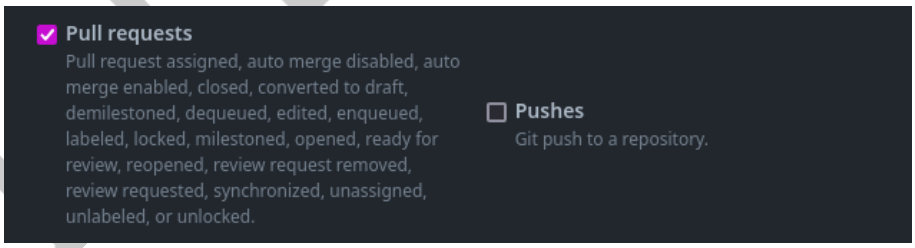

Figure 3: Webhook trigger events.

2.6 Github Secrets Configuration

The image in Figure 4 shows where to set the secrets in your Github repository. These should be named to match the values given in the GH action YAML file.

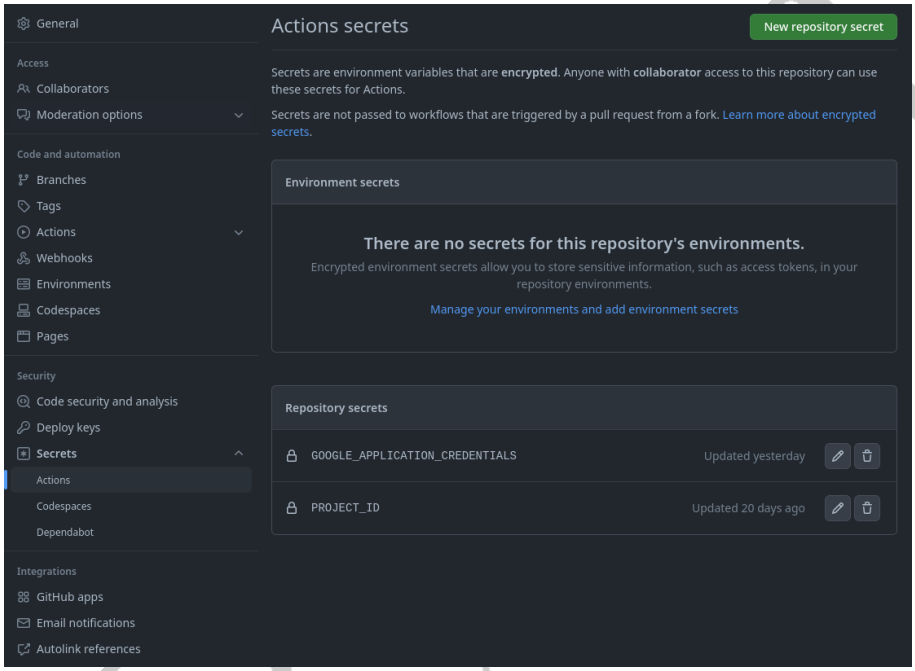

Figure 4: Setting repository secrets.

3 The Python Application

The Python code is meant to run as a "Cloud Function" in GCP. This is a cost-effective way to run code becuase you can have your application hosted without provisioning and maintaining any supporting infrastructure. We get to focus our code and let Google take care of the rest.

Our Python is reliant on a set of modules that are declared in the "src/requirements.txt" file.

```
google-cloud-logging
google-cloud-secret-manager
requests
flask
PyGithub
```

There are a few other Python requirements files in the project that serve other purposes including testing and security.

4 Terraform

5 Testing

6 Going a Bit Further

This optional section describes how you can connect your Cloud Function to your Kubernetes cluster to extend the functionality even more.

7 Cleanup

docker system prune

Part III
Appendix

A GNU Autotools

At the risk of introducing greater complexity, GNU Autotools have been included in the code base for the workshop. Autotools are a well-maintained set of Open Source tools with a gentle learning curve and are included in the distribution of many Open Source packages that we all rely on daily, at least indirectly, and often unknowingly. While it may be possible to complete this workshop without at least learning how to configure and execute the tools in your environment, you will derive significantly less enjoyment in doing so.

The list of tools for using this Autotools configuration paradigm is show in Table 1.

Tool	Description
autoconf	Generates a configure script from configure.ac
automake	Generates a system-specific Makefile based on Makefile.am template
make	X

Table 1: GNU tools used in this project

The image shown in Figure 5 is from [1]. It illustrates the relationship between components mentioned in Table 1.

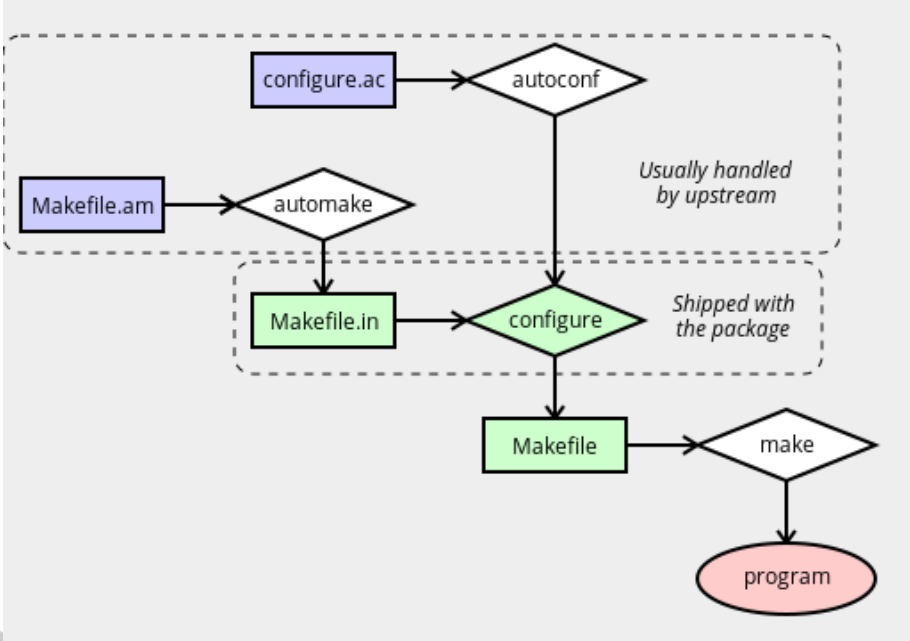

Figure 5: A basic overview of how the main Autotools components fit together.

Revision History

Revision	Date	Author(s)	Description
v0.1	October 2nd, 2022	Franklin Diaz	Initial Draft

References

[1] Gentoo Authors. The basics of autotools, 2022.