

Build a Serverless Github Bot in GCP

Franklin E. Diaz

frank378@gmail.com

November 19, 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.

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.

1.1 Outline

add a diagram here that shows the overall workflow

A high level overview of the learning path is as follows:

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

1.2 Some Interesting Examples

2 Prerequisites

There are some requirements that must be met to successfully complete this workshop. This workshop was developed in a Linux environment, with some testing on Macintosh. You should use a similar environment, or you can use the containerized development environment if you wish. This may be a good option for folks who don't have access to a reliable Linux environment, or may be having issues getting the proper packages installed.

- Install the latest VS Code.
- Clone the project repository.
- Open the containerized development environment.

2.1 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 1 is from [1]. It illustrates the relationship between components mentioned in Table 1.

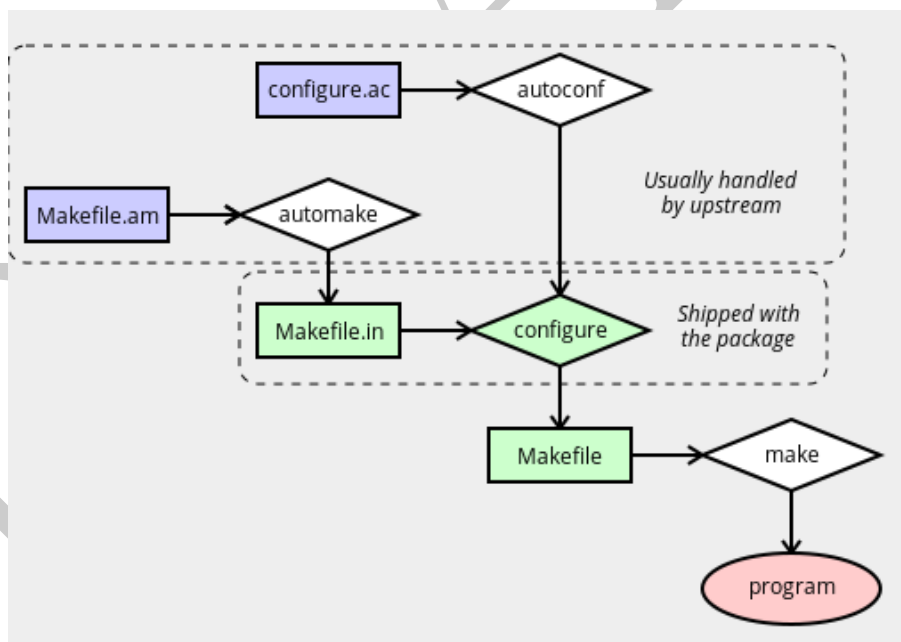


Figure 1: A basic overview of how the main Autotools components fit together.

3 Set up A Development Environment

The development environment can be configured using the “bootstrap.sh” script and GNU autotools. This process should work well in Linux or Mac environments. There is also a development container that you can build if you are having issues configuring these, wish to run in a different Operating System, etc.

3.1 Option 1: Set up on Linux and Mac

1. Clone the Github repository from <https://github.com/devsecfranklin/workshop-codemash-2023>.
2. In the newly created “workshop-codemash-2023” directory, execute the “bootstrap.sh” script. Note that on Macintosh this will update your existing brew installation.
3. Next, run the “./configure” command.
4. Finally, run the “make python” command to prepare the necessary Python modules.

3.2 Option 2: Set up the dev Container

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 repository.
3. Add the Github action to the repository.

4 Github Setup

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. Add the Github action to the repository.
4. Configure a webhook to connect Github to Cloud Function.

You can add a cool icon to the bot account Github profile.

4.1 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
```

4.2 Github Webhook Configuration

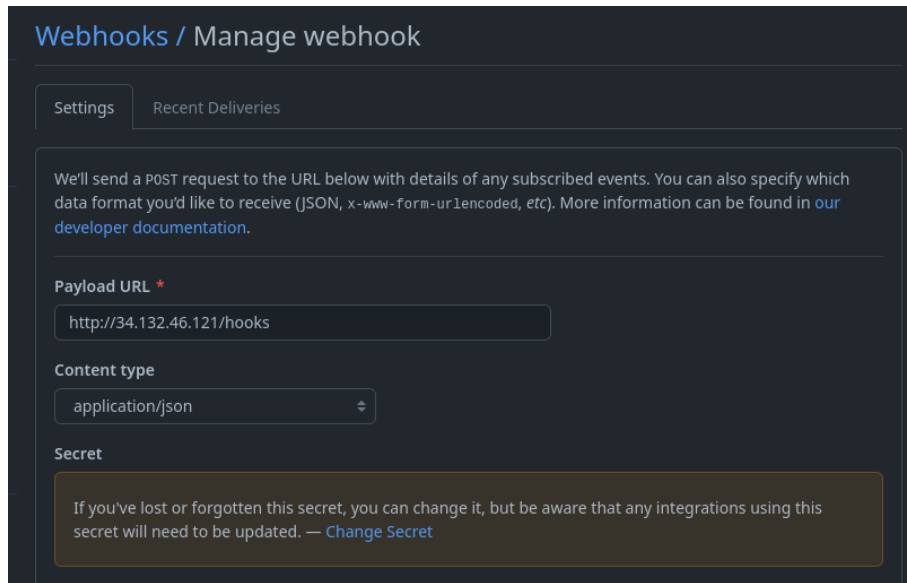
There are three parts to configuring the webhook in Github.

- 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 the next section.

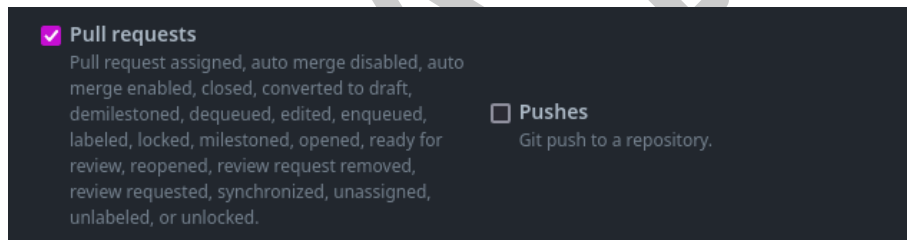
4.3 Github Secrets Configuration

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.



The screenshot shows the 'Webhooks / Manage webhook' interface. It has two tabs: 'Settings' and 'Recent Deliveries'. The 'Settings' tab is active. Below the tabs, there is a text block explaining that a POST request will be sent to the 'Payload URL' with details of subscribed events. The 'Payload URL' is set to 'http://34.132.46.121/hooks'. The 'Content type' is set to 'application/json'. Below this, there is a 'Secret' section with a warning: 'If you've lost or forgotten this secret, you can change it, but be aware that any integrations using this secret will need to be updated. — [Change Secret](#)'.

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



The screenshot shows the 'Manage webhooks' page with a list of trigger events. The 'Pull requests' event is selected with a checkmark. The list of events for 'Pull requests' includes: 'Pull request assigned, auto merge disabled, auto merge enabled, closed, converted to draft, demilestoned, dequeued, edited, enqueued, labeled, locked, milestone, opened, ready for review, reopened, review request removed, review requested, synchronized, unassigned, unlabeled, or unlocked.' The 'Pushes' event is not selected, indicated by an unchecked checkbox. The list of events for 'Pushes' includes: 'Git push to a repository.'

Figure 3: Webhook trigger events.

5 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 because 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
```

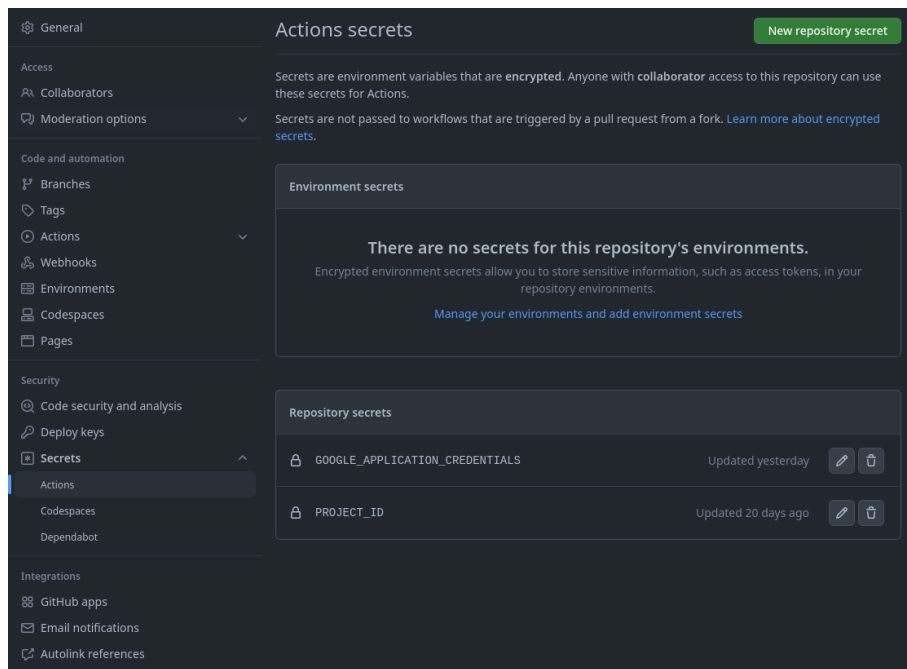


Figure 4: Setting repository secrets.

```
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.

6 Terraform

7 Testing

8 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.

9 Cleanup

```
docker system prune
```

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.

DRAFT