Deploying Google Cloud Functions in Python with Cloud Build

A tutorial by Benjamin Echelmeier

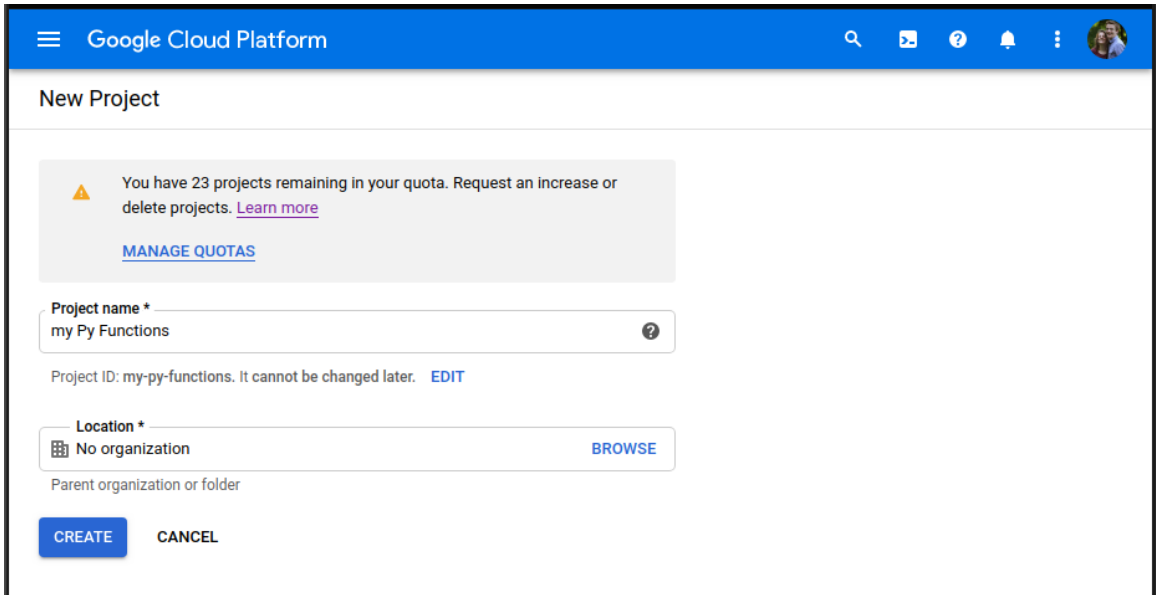
Introduction

In this tutorial, we'll be utilizing the Google Cloud Platform to deploy Cloud Functions in Python using Cloud Build for Continuous Integration or Continuous Deployment. All the technologies used here should be covered in depth, but there is an assumption of basic Python and a degree of understanding cloud technology.

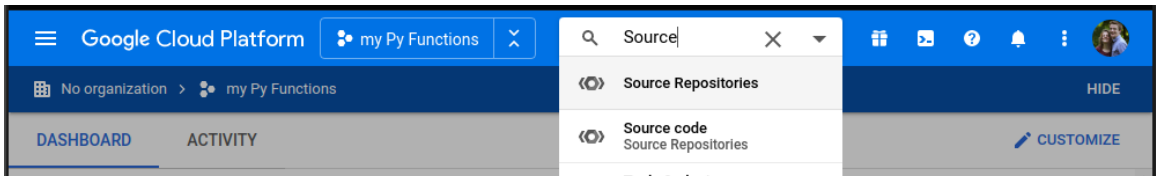
If you are unfamiliar with Cloud Functions, check out this quickstart from the Google Cloud documentation; it will guide you through a basic function that we will use to explore Cloud Build.

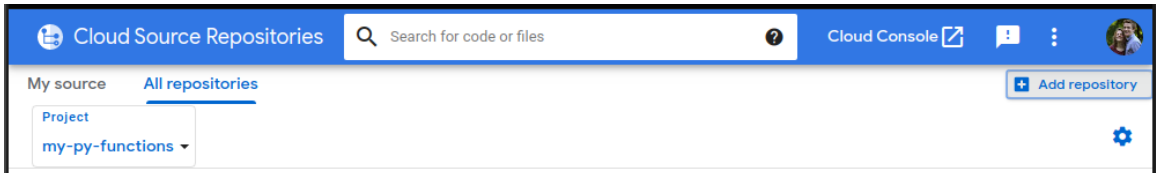
Setting up a new Google Cloud Project and Repository

To begin, of course, we'll need to sign in to (or create) our Google Cloud account and create a new project. I'll be naming mine "my Py Functions".

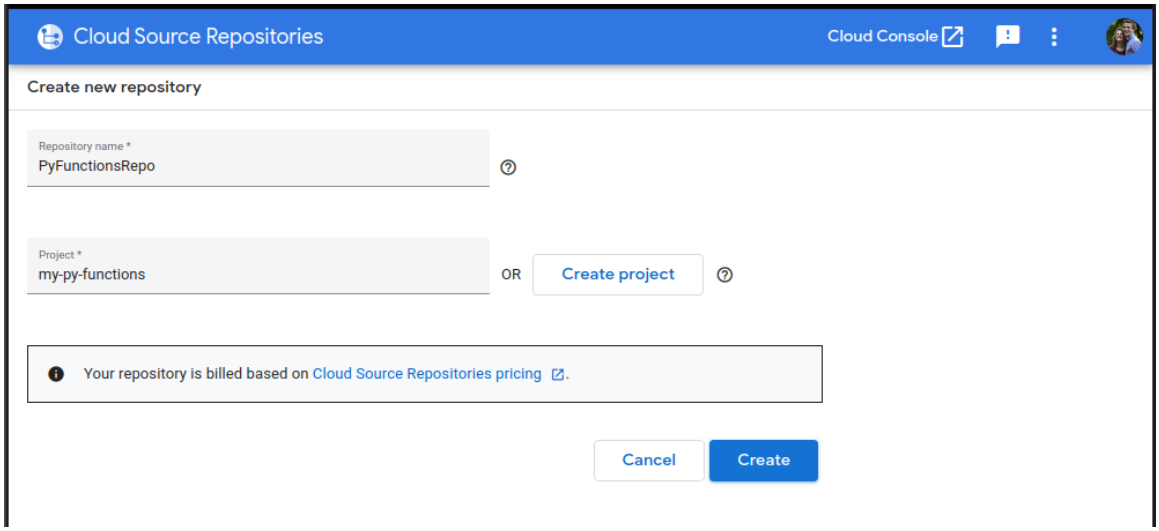


After the project has been created, head over to your Source Repositories and add a repository.

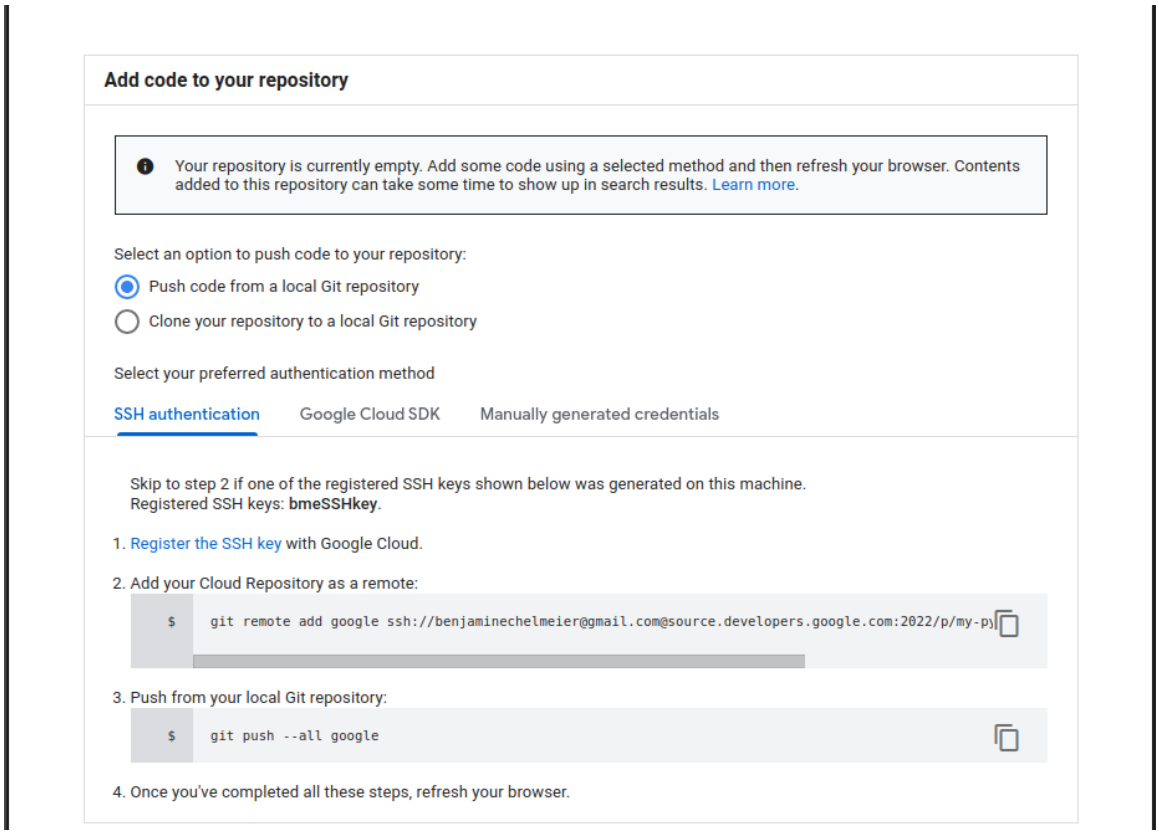




Go ahead and *Create new Repository* with your chosen name inside your project.

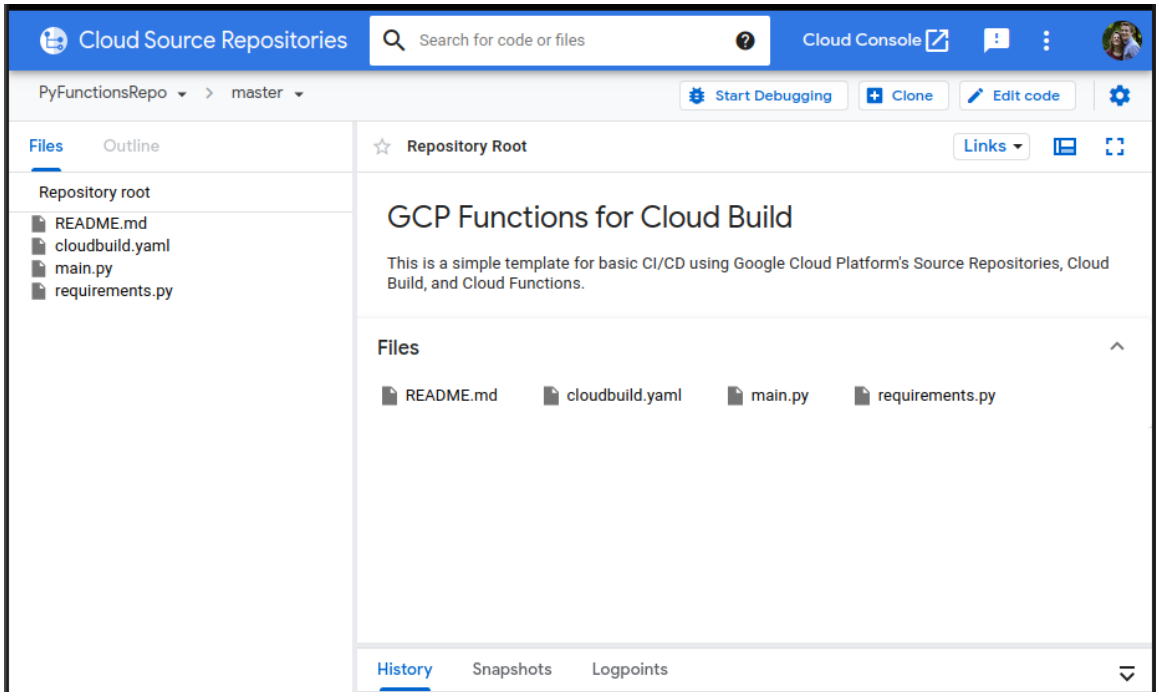


I'll be pushing a clone of [this repository](), which contains a slightly modified version of the function mentioned in the introduction, and following the instructions to *Push code from a local Git repository*. (Feel free to fill the repository in whatever way is familiar to you)



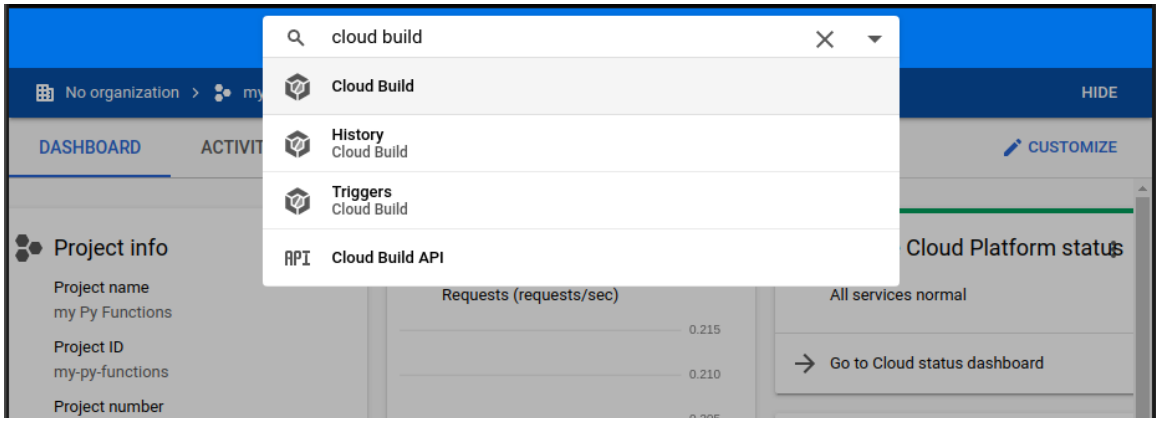
After refreshing our browser, we should have a repository similar to the one pictured below. We have our README and our .py files for our function, which should be familiar, and

our cloudbuild.yaml, which is shown and explained later in the “Understanding and Editing Cloudbuild Files” section.

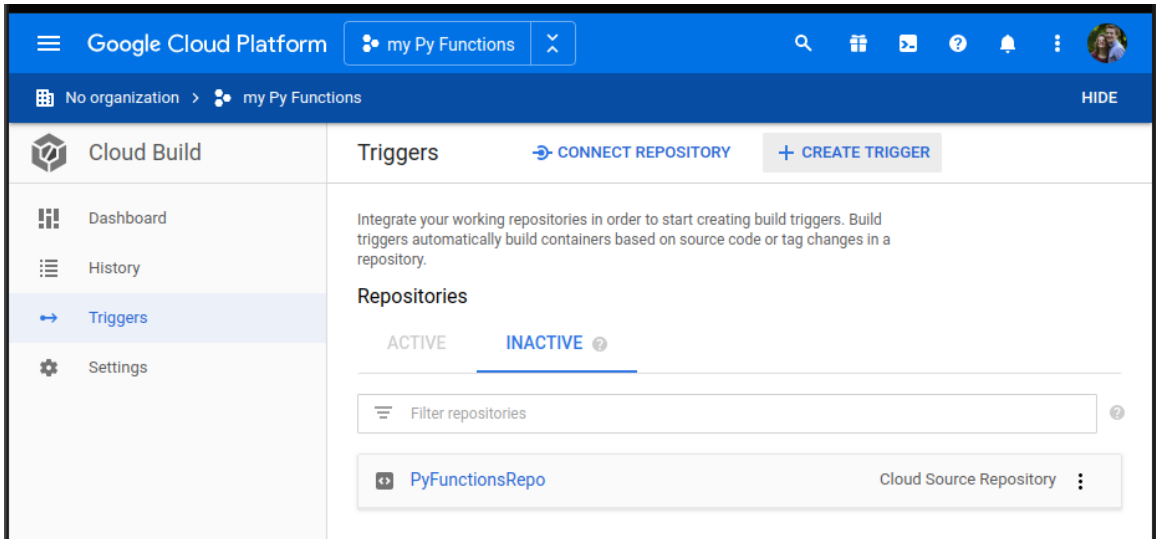


Setting up our Cloud Build

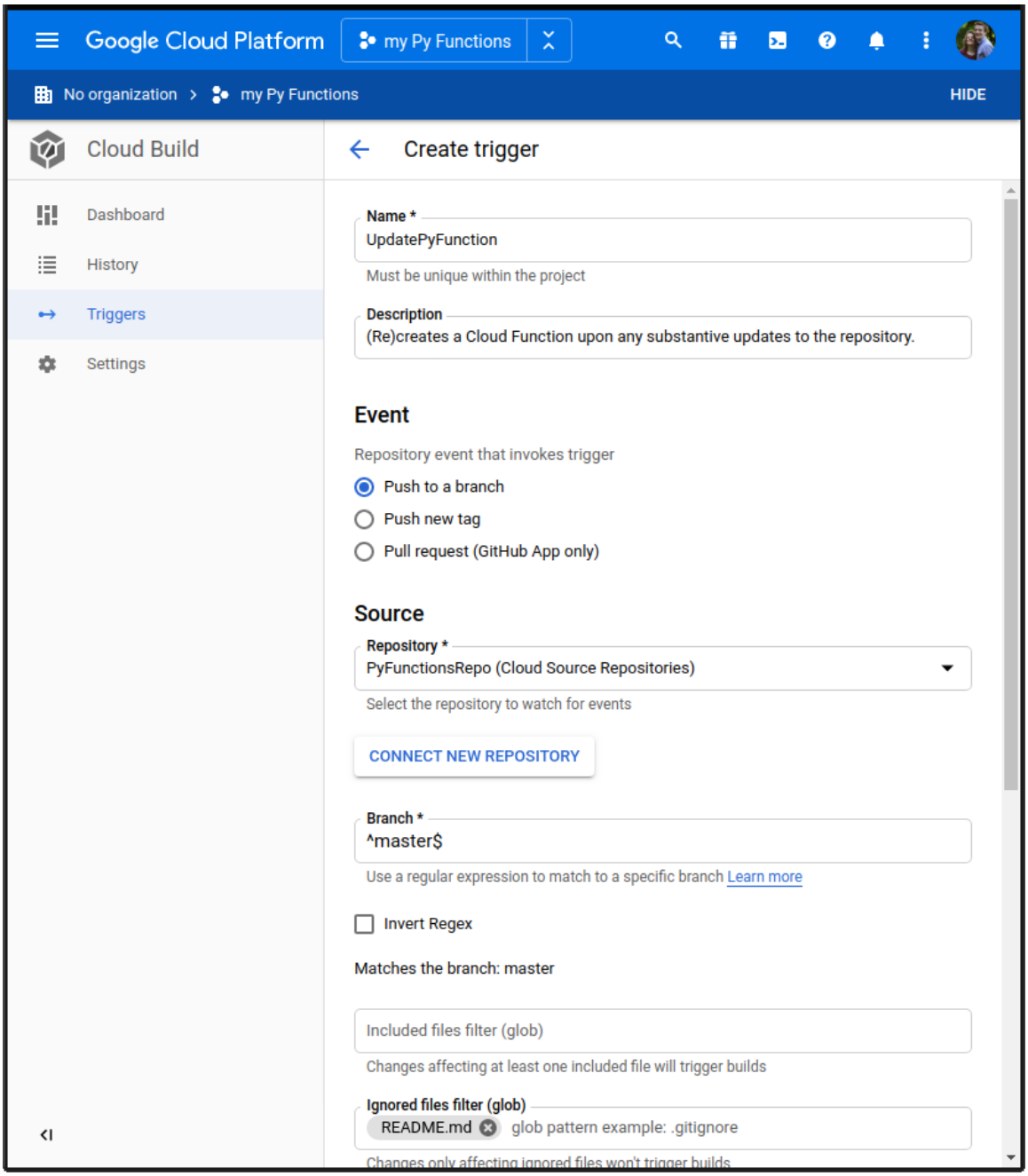
Now that we have our dummy code (or other code, if you’ve already customized things for your project) set up in our repository, lets hop back to the *Cloud Console* and navigate to our *Cloud Build* API.



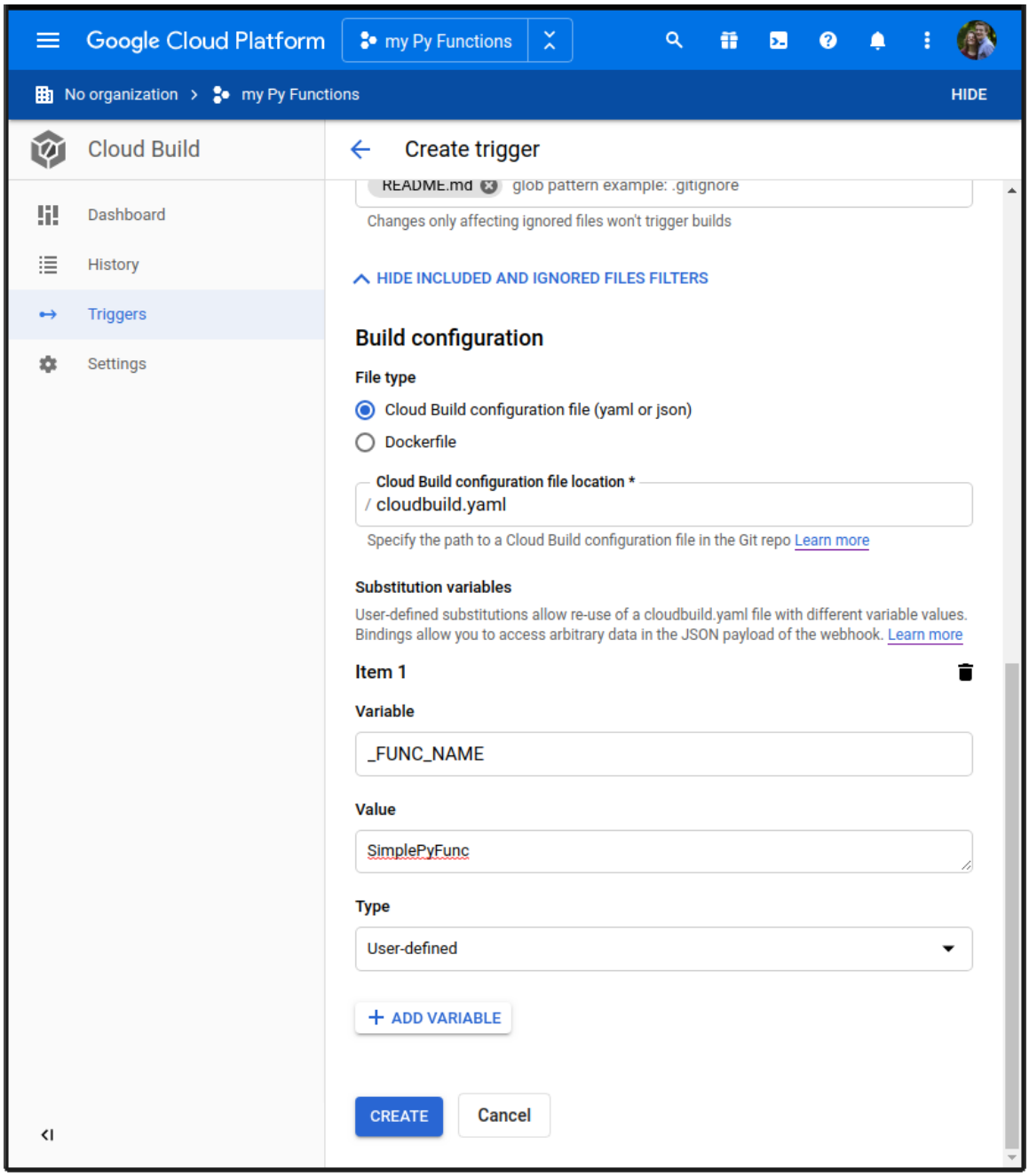
Go ahead and *enable* the API, and navigate to the *Triggers* menu on the sidebar and *Create a Trigger*.



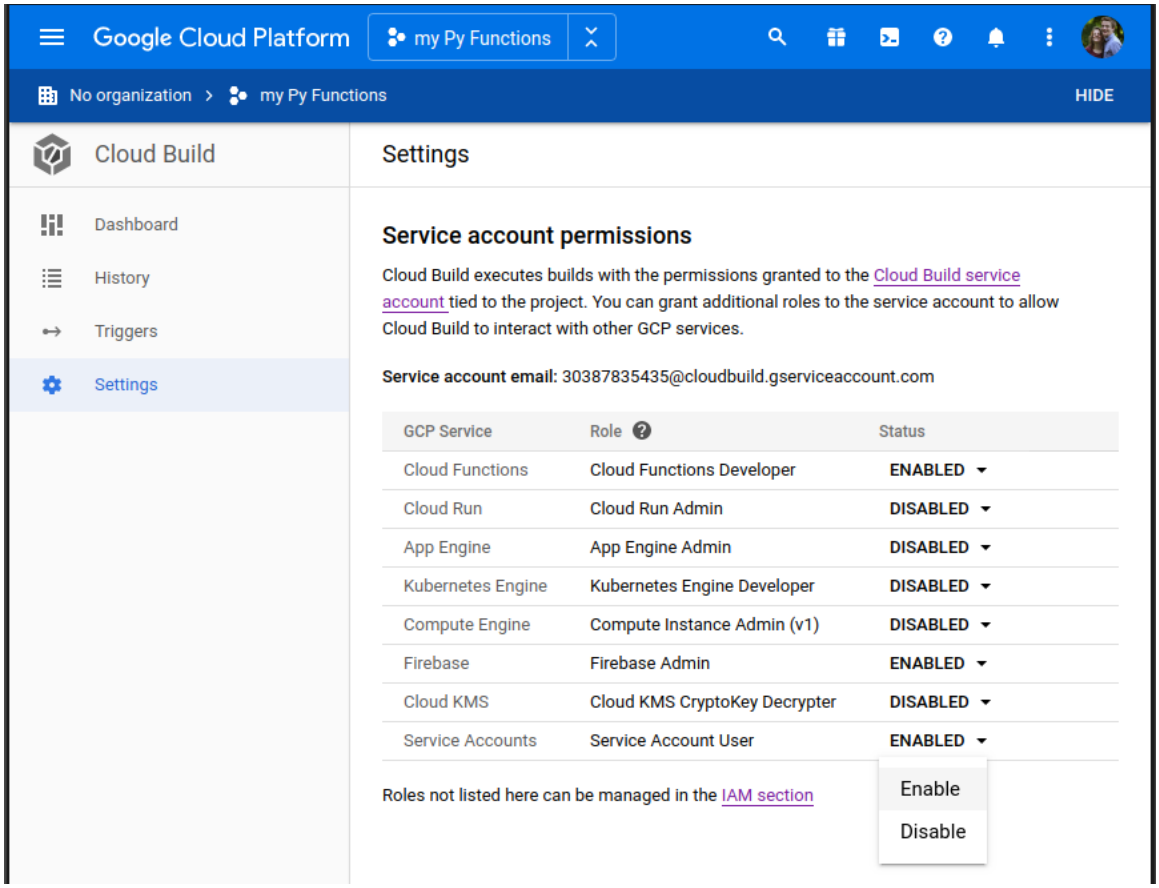
Fill in the *name* and *description*, and select the *Push to a branch* Event to invoke the trigger. Select your repository and your branch (I picked master), and, if you'd like, you can filter out changes to certain files, like my README, so as to avoid triggering unnecessary builds.



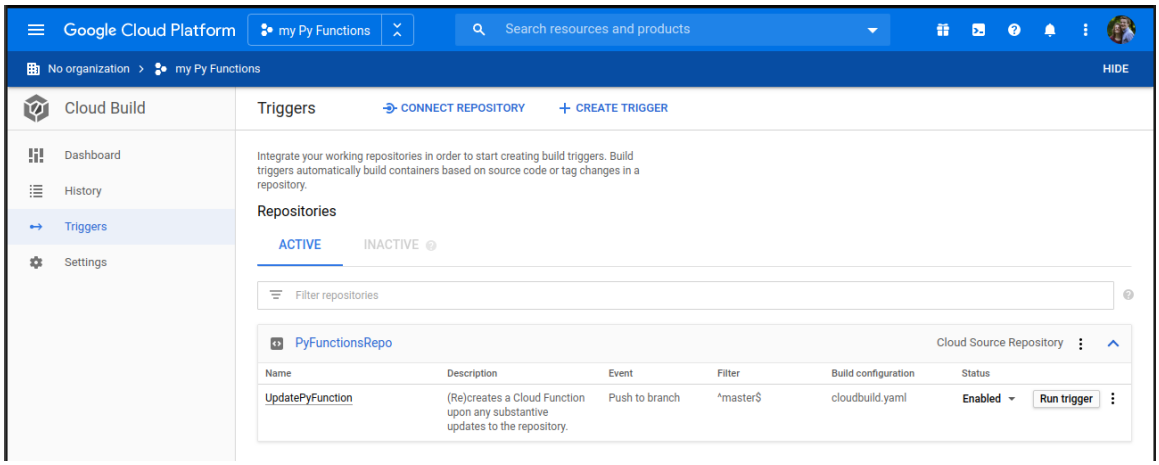
The defaults should lead directly to your cloudbuild.yaml file. Then we are going to select *add variable*. Those who studied our cloudbuild.yaml file, might recognize the _FUNC_NAME variable. Hopefully it is fairly obvious that the value here will be the name of your Cloud Function. Once this is all to your liking, click *create*.



Now, we have our trigger, but before the build can run properly we will need to grant permission to build Cloud Functions. Go to *Settings* on the side bar and *enable* Cloud Functions, Firebase, and Service Accounts.

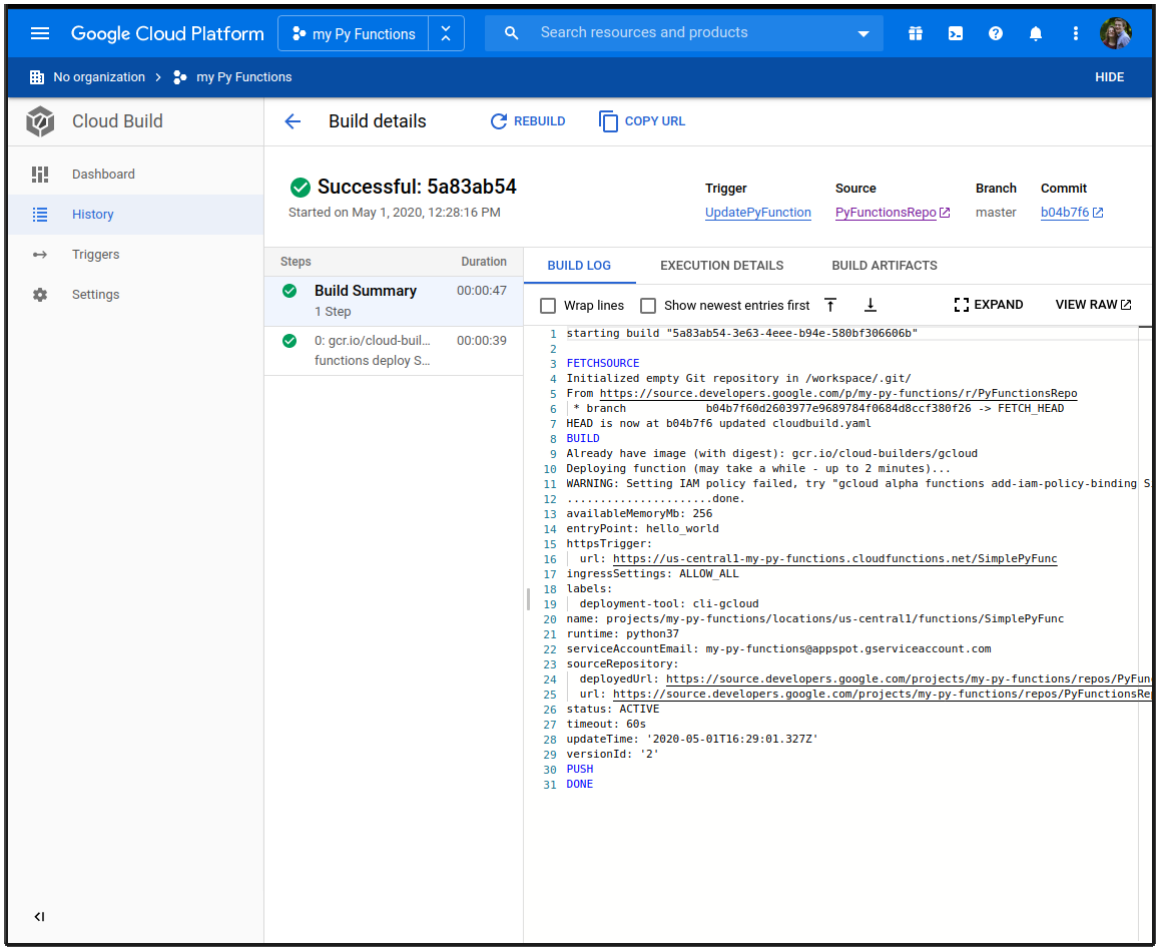


We can now go back to our trigger and click to *run trigger*. This can also be done by pushing a change to our repo.

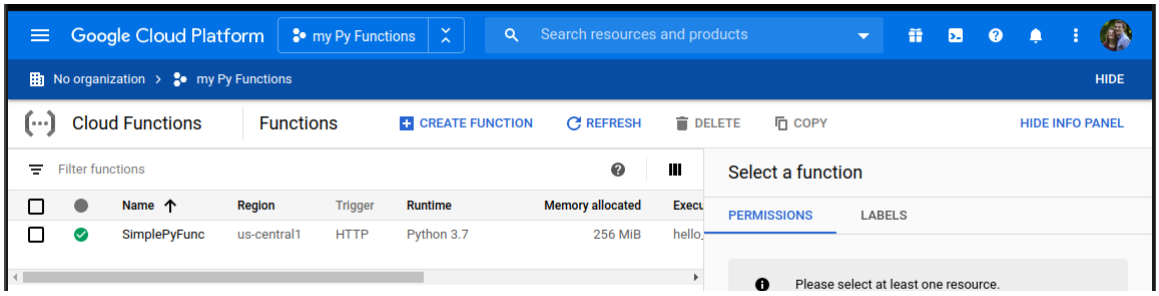


You should get a notification of your running build, or you can go to the *history* tab and select the running build. You can watch magic happen and, hopefully, after a few minutes you

have a successful build. If your build failed, try and troubleshoot based on the error message; they tend to be fairly helpful.



Once your build has succeeded, use the search bar to hop over to your Cloud Functions. There you should find your newly created Cloud Function.



Congratulations! You just made a CD build for your Cloud Function!

Customizing and Going Further

Understanding and Editing Cloudbuild Files

Google has good documentation on how to make a cloudbuild file here, and the documentation should be more than sufficient for making your own cloudbuild file to automate your needs. As you want to adapt and customize (or build from scratch), please look to these docs. They also have a guide to building a basic configuration file to get new users started, but I'll be providing a slightly more concrete breakdown of a simple file here.

First, let's take a look at the cloudbuild.yaml file used in this tutorial.

```
steps:
- name: 'gcr.io/cloud-builders/gcloud'
  args:
    [ 'functions',
      'deploy',
      '$_FUNC_NAME',
      '--runtime',
      'python37',
      '--entry-point',
      'hello_world',
      '--source',
      'https://source.developers.google.com/projects/$PROJECT_ID/repos/$REPO_NAME/
moveable-aliases/$BRANCH_NAME/paths//',
      '--trigger-http',
      '--allow-unauthenticated'
    ]
```

Note that if you want to copy this text to create your own file, you must remove the space in the url.

Practically any cloudbuild file will have a "steps," "name," and "args" field. "Steps" demarks that the contained will be the build steps, as opposed to, for instance, "options" or "tags." "Name" initiates a step by pointing to a cloud builder, and "args" are the meat of the file, containing the arguments passed to the builder.

Our builder is, of course, gcloud, the primary CLI for GCP, and our arguments are a single-line functions deploy argument. Manually entered into the gcloud CLI it would look like:

```
gcloud functions deploy $_FUNC_NAME --runtime python37 --entry-point hello_world --source
https://source.developers.google.com/projects/$PROJECT_ID/repos/$REPO_NAME/moveable-alias
es/$BRANCH_NAME/paths// --trigger-http --allow-unauthenticated
```

Obviously, the $VARIABLES look a bit out of place in gcloud; those are a terrific tool that allows for far more flexibility in our config files. Our _FUNC_NAME is, of course, the variable we defined, while the others (notably lacking the preceding underscore) are defaults. Questions about any arguments here will likely be answered by referencing the appropriate flag here.

Building for Multiple Branches

One of the best uses of Cloud Build is the ability to integrate CD into various branches, allowing for quicker feedback and collaboration. I'll take a brief moment here to create a "dev" branch that might be used to test our potentially unstable changes before we merge with the master branch and, thereafter, our primary function.

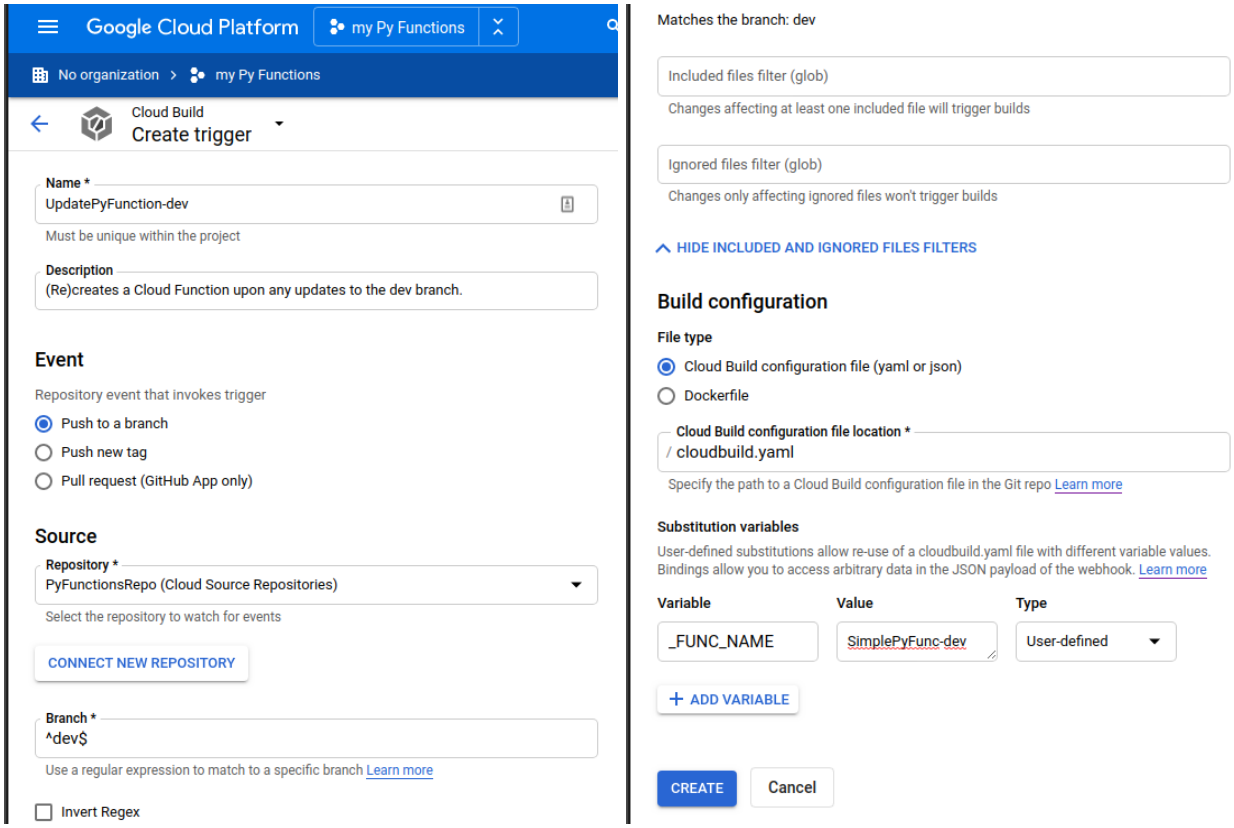
Let us return to our local git repo and make our dev branch in the terminal.

```
git checkout -b dev
```

Go ahead and add some text to the return statement before adding, committing, and pushing to your cloud repository.

```
git add .
git commit -m 'updated main.py with potentially unstable text'
git push --set-upstream google dev
```

Now that we have our branch, we can go ahead and create a new trigger for it. This should be quite similar to our previous trigger, but with a new name, branch, and _FUNC_NAME variable.



That's all it takes! *Create* your function and *run trigger* or push a change to your dev branch, and then go confirm that your new function has been created, complete with the new changes. This can and should be duplicated into a greater number of branches, however many suits your needs.

Conclusion

Congratulations! You've successfully created a functioning CI/CD environment with GCP cloud functions. This is, of course, just an entry point, and I encourage you to continue exploring the Console and Documentation provided by Google, both for any questions and for further application.

I'll finish by giving credit to Clemens Siebler and his similar post, [Deploying Azure Functions] in Python with Azure DevOps, which introduced me to the world of CI/CD and serves as inspiration for this guide. If you're looking to do something very similar in Azure, go check out his post.