

Deploying Google Cloud Functions in Python with Cloud Build

A tutorial by Benjamin Echelmeier

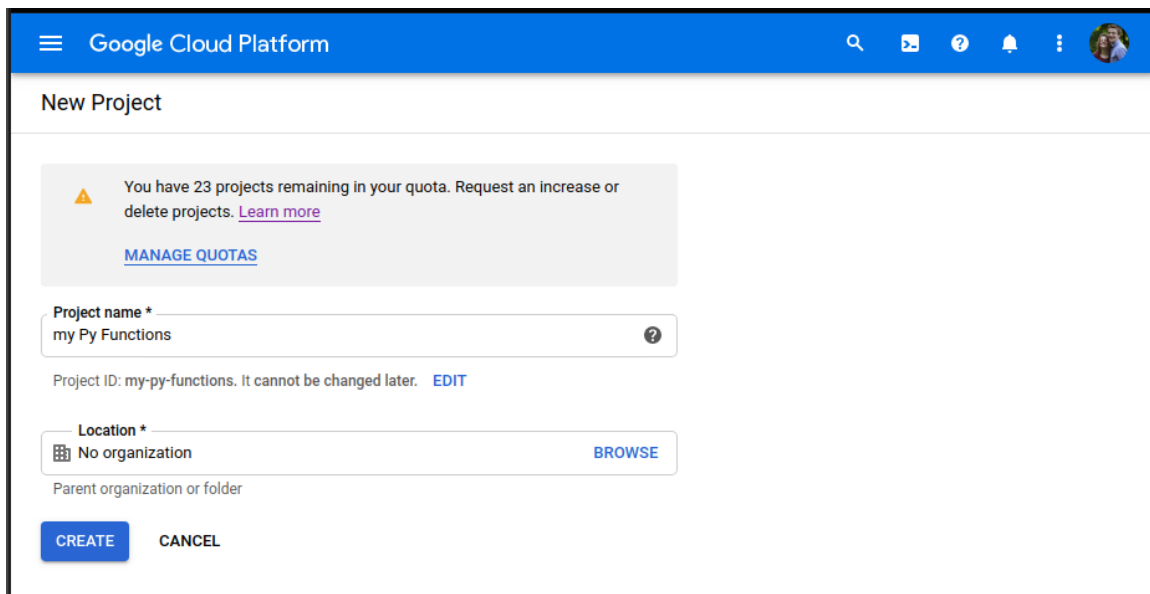
Introduction

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.

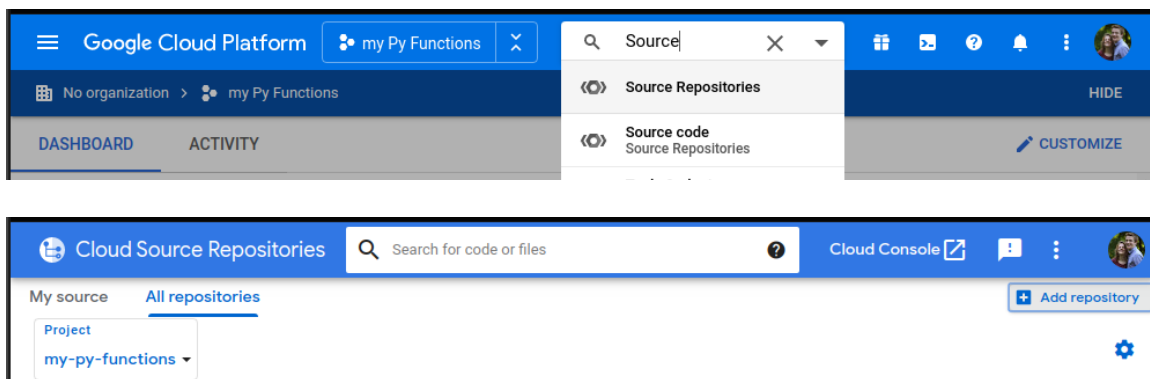
Setting up a new Google Cloud Project and Repository

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".

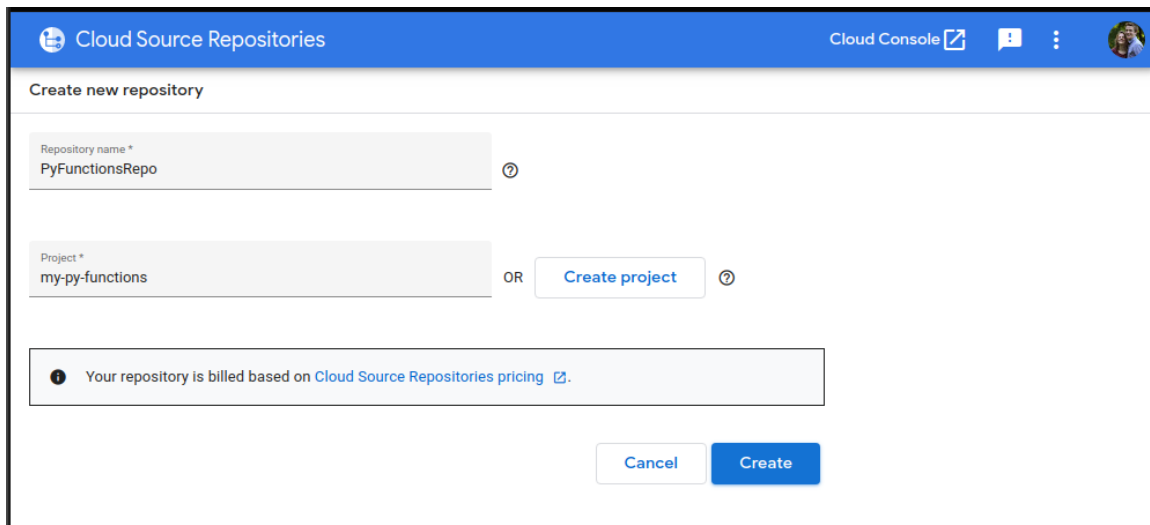


The screenshot shows the 'New Project' form in the Google Cloud Platform console. At the top, there's a blue header with the Google Cloud Platform logo and navigation icons. Below the header, the title 'New Project' is displayed. A warning message states: 'You have 23 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)'. Below this, there's a 'Project name' field with the value 'my Py Functions' and a help icon. Underneath, it shows 'Project ID: my-py-functions. It cannot be changed later. [EDIT](#)'. The 'Location' section has a dropdown menu set to 'No organization' with a 'BROWSE' button. At the bottom, there are 'CREATE' and 'CANCEL' buttons.

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

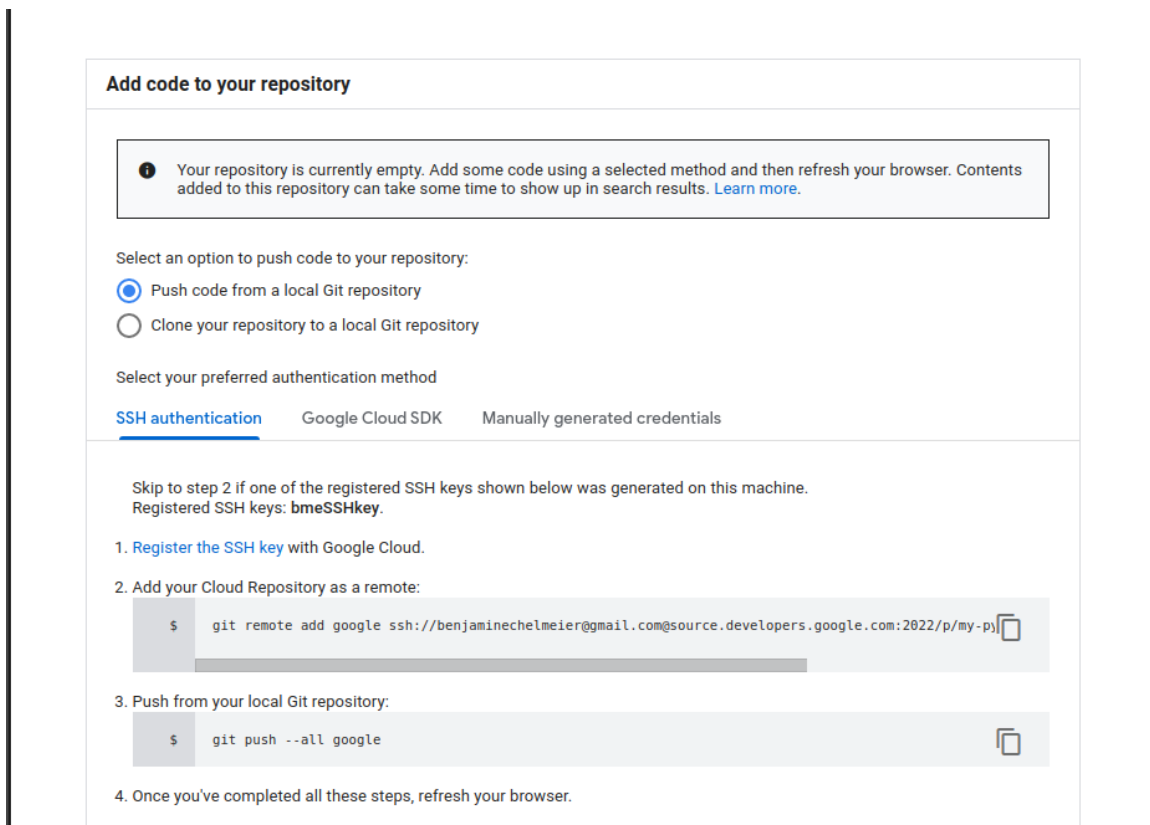


Go ahead and *Create new Repository* with your chosen name inside your project.



The screenshot shows the 'Create new repository' interface in the Google Cloud Console. At the top, there's a blue header with the 'Cloud Source Repositories' logo and 'Cloud Console' link. Below the header, the title 'Create new repository' is displayed. The main form has two input fields: 'Repository name *' with the value 'PyFunctionsRepo' and a help icon, and 'Project *' with the value 'my-py-functions' and a 'Create project' button. Below these fields is a message: 'Your repository is billed based on Cloud Source Repositories pricing'. At the bottom, there are 'Cancel' and 'Create' buttons.

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)



The screenshot shows the 'Add code to your repository' page. It starts with a message: 'Your repository is currently empty. Add some code using a selected method and then refresh your browser. Contents added to this repository can take some time to show up in search results. Learn more.' Below this, there are two sections. The first section, 'Select an option to push code to your repository:', has two radio buttons: 'Push code from a local Git repository' (selected) and 'Clone your repository to a local Git repository'. The second section, 'Select your preferred authentication method:', has three tabs: 'SSH authentication' (selected), 'Google Cloud SDK', and 'Manually generated credentials'. Under the 'SSH authentication' tab, there's a message: 'Skip to step 2 if one of the registered SSH keys shown below was generated on this machine. Registered SSH keys: bmeSSHkey.' Below this, there are four steps: 1. 'Register the SSH key with Google Cloud.' 2. 'Add your Cloud Repository as a remote:' with a terminal command: `git remote add google ssh://benjaminechelmeier@gmail.com@source.developers.google.com:2022/p/my-py`. 3. 'Push from your local Git repository:' with a terminal command: `git push --all google`. 4. 'Once you've completed all these steps, refresh your browser.'

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

Cloud Source Repositories

Search for code or files

Cloud Console

PyFunctionsRepo > master

Start DebuggingCloneEdit code

FilesOutline

Repository root

- README.md
- cloudbuild.yaml
- main.py
- requirements.py

Repository Root

Links

GCP Functions for Cloud Build

This is a simple template for basic CI/CD using Google Cloud Platform's Source Repositories, Cloud Build, and Cloud Functions.

Files

README.md

cloudbuild.yaml

main.py

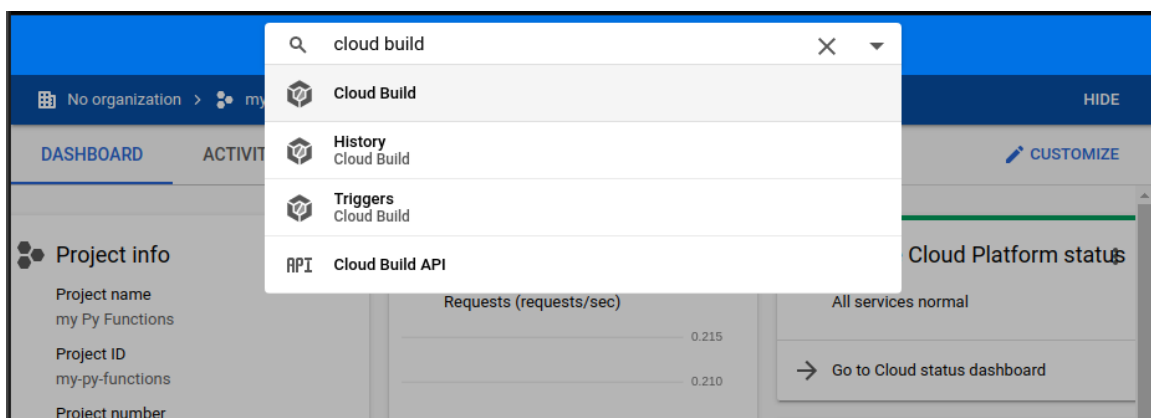
requirements.py

History

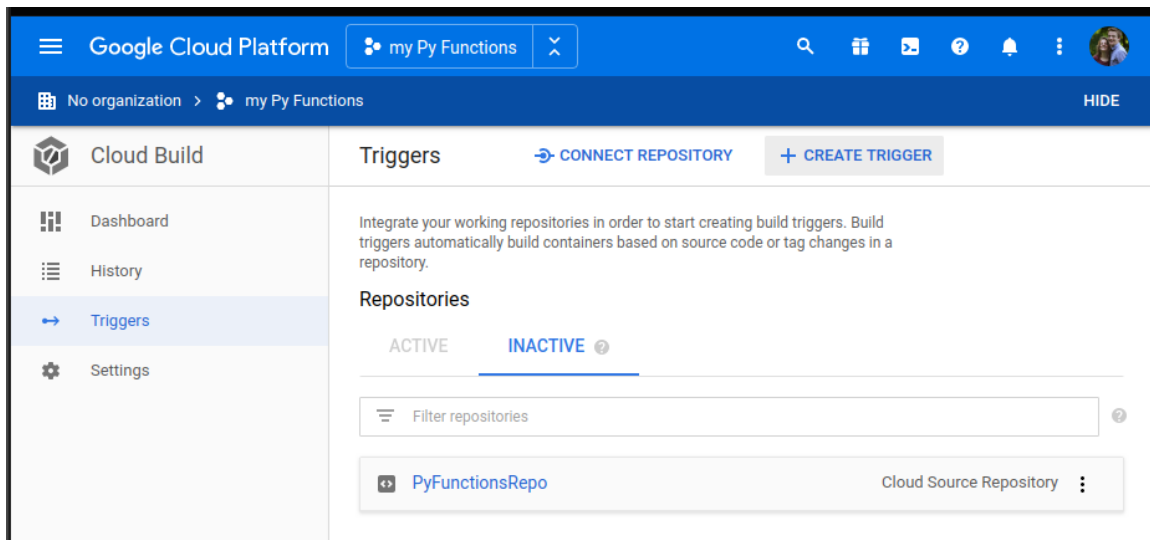
Snapshots

Logpoints

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 screenshot shows the 'Create trigger' page in the Google Cloud Platform console. The left sidebar contains navigation links for Cloud Build, Dashboard, History, Triggers, and Settings. The main content area is titled 'Create trigger' and includes the following sections:

- Name ***: A text input field containing 'UpdatePyFunction'. A note below states 'Must be unique within the project'.
- Description**: A text input field containing '(Re)creates a Cloud Function upon any substantive updates to the repository.'
- Event**: A section titled 'Repository event that invokes trigger' with three radio button options: 'Push to a branch' (selected), 'Push new tag', and 'Pull request (GitHub App only)'.
- Source**: A section titled 'Repository *' with a dropdown menu showing 'PyFunctionsRepo (Cloud Source Repositories)'. A note below says 'Select the repository to watch for events'. A button labeled 'CONNECT NEW REPOSITORY' is located below the dropdown.
- Branch ***: A text input field containing '^master\$'. A note below states 'Use a regular expression to match to a specific branch' with a link to 'Learn more'.
- Invert Regex**: An unchecked checkbox.
- Matches the branch**: A label indicating the current branch match is 'master'.
- Included files filter (glob)**: A text input field.
- Ignored files filter (glob)**: A text input field containing 'README.md' (highlighted in a grey pill) and a note 'glob pattern example: .gitignore'.

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*.

Google Cloud Platform my Py Functions

No organization > my Py Functions HIDE

Cloud Build

- Dashboard
- History
- Triggers
- Settings

Create trigger

README.md glob pattern example: .gitignore

Changes only affecting ignored files won't trigger builds

[HIDE INCLUDED AND IGNORED FILES FILTERS](#)

Build configuration

File type

☒ Cloud Build configuration file (yaml or json)

☐ Dockerfile

Cloud Build configuration file location *

/ cloudbuild.yaml

Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Substitution variables

User-defined substitutions allow re-use of a cloudbuild.yaml file with different variable values. Bindings allow you to access arbitrary data in the JSON payload of the webhook. [Learn more](#)

Item 1

Variable

_FUNC_NAME

Value

SimplePyFunc

Type

User-defined

[+ ADD VARIABLE](#)

[CREATE](#) [Cancel](#)

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.

The screenshot shows the Google Cloud Platform interface. The top navigation bar includes the Google Cloud Platform logo, a dropdown menu for 'my Py Functions', and various utility icons. The left sidebar shows the 'Cloud Build' section with options for Dashboard, History, Triggers, and Settings. The main content area is titled 'Settings' and 'Service account permissions'. It explains that Cloud Build uses the 'Cloud Build service account' and provides the service account email: 30387835435@cloudbuild.gserviceaccount.com. A table lists the permissions for various GCP services, with Cloud Functions, Firebase, and Service Accounts set to 'ENABLED'. A context menu is open over the 'ENABLED' status for Service Accounts, showing 'Enable' and 'Disable' options.

GCP Service	Role	Status
Cloud Functions	Cloud Functions Developer	ENABLED
Cloud Run	Cloud Run Admin	DISABLED
App Engine	App Engine Admin	DISABLED
Kubernetes Engine	Kubernetes Engine Developer	DISABLED
Compute Engine	Compute Instance Admin (v1)	DISABLED
Firebase	Firebase Admin	ENABLED
Cloud KMS	Cloud KMS CryptoKey Decrypter	DISABLED
Service Accounts	Service Account User	ENABLED

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.

The screenshot shows the Google Cloud Platform interface for the 'Triggers' section. The top navigation bar includes the Google Cloud Platform logo, a dropdown menu for 'my Py Functions', and a search bar. The left sidebar shows the 'Cloud Build' section with options for Dashboard, History, Triggers, and Settings. The main content area is titled 'Triggers' and includes a 'CONNECT REPOSITORY' button and a 'CREATE TRIGGER' button. It explains that triggers automatically build containers based on source code or tag changes in a repository. A table lists the repositories, with 'PyFunctionsRepo' selected. A table below shows the triggers for 'PyFunctionsRepo', with 'UpdatePyFunction' selected. The 'UpdatePyFunction' trigger is enabled and has a 'Run trigger' button.

Name	Description	Event	Filter	Build configuration	Status
UpdatePyFunction	(Re)creates a Cloud Function upon any substantive updates to the repository.	Push to branch	*master\$	cloudbuild.yaml	Enabled

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.

The screenshot shows the Google Cloud Platform interface for Cloud Build. The top navigation bar includes the Google Cloud Platform logo, a dropdown menu for 'my Py Functions', a search bar, and various utility icons. The main content area is titled 'Cloud Build' and shows 'Build details' for a successful build with ID '5a83ab54'. The build started on May 1, 2020, at 12:28:16 PM. The trigger is 'UpdatePyFunction', the source is 'PyFunctionsRepo', the branch is 'master', and the commit is 'b04b7f6'. The build log shows the following steps:

Steps	Duration
Build Summary 1 Step	00:00:47
0: gcr.io/cloud-buil... functions deploy S...	00:00:39

The build log details include:

```
1 starting build "5a83ab54-3e63-4eee-b94e-580bf306606b"
2
3 FETCHSOURCE
4 Initialized empty Git repository in /workspace/.git/
5 From https://source.developers.google.com/p/my-py-functions/r/PyFunctionsRepo
6 * branch      b04b7f60d2603977e9689784f0684d8ccf380f26 -> FETCH_HEAD
7 HEAD is now at b04b7f6 updated cloudbuild.yaml
8 BUILD
9 Already have image (with digest): gcr.io/cloud-builders/gcloud
10 Deploying function (may take a while - up to 2 minutes)...
11 WARNING: Setting IAM policy failed, try "gcloud alpha functions add-iam-policy-binding S
12 .....done.
13 availableMemoryMb: 256
14 entryPoint: hello_world
15 httpsTrigger:
16 | url: https://us-central1-my-py-functions.cloudfunctions.net/SimplePyFunc
17 ingressSettings: ALLOW_ALL
18 labels:
19 | deployment-tool: cli-gcloud
20 name: projects/my-py-functions/locations/us-central1/functions/SimplePyFunc
21 runtime: python37
22 serviceAccountEmail: my-py-functions@appspot.gserviceaccount.com
23 sourceRepository:
24 | deployedUrl: https://source.developers.google.com/projects/my-py-functions/repos/PyFun
25 | url: https://source.developers.google.com/projects/my-py-functions/repos/PyFunctionsRe
26 status: ACTIVE
27 timeout: 60s
28 updateTime: '2020-05-01T16:29:01.327Z'
29 versionId: '2'
30 PUSH
31 DONE
```

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.

The screenshot shows the Google Cloud Platform interface for Cloud Functions. The top navigation bar includes the Google Cloud Platform logo, a dropdown menu for 'my Py Functions', a search bar, and various utility icons. The main content area is titled 'Cloud Functions' and shows 'Functions' for 'my Py Functions'. The 'SimplePyFunc' function is listed with the following details:

Name	Region	Trigger	Runtime	Memory allocated	Execu
SimplePyFunc	us-central1	HTTP	Python 3.7	256 MIB	hello...

The 'SimplePyFunc' function is selected, and the 'PERMISSIONS' tab is active. A message at the bottom states: 'Please select at least one resource.'

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.

Google Cloud Platform my Py Functions

No organization > my Py Functions

Cloud Build
Create trigger

Name *
UpdatePyFunction-dev
Must be unique within the project

Description
(Re)creates a Cloud Function upon any updates to the dev branch.

Event
Repository event that invokes trigger
☒ Push to a branch
☐ Push new tag
☐ Pull request (GitHub App only)

Source
Repository *
PyFunctionsRepo (Cloud Source Repositories)
Select the repository to watch for events
CONNECT NEW REPOSITORY

Branch *
^dev\$
Use a regular expression to match to a specific branch [Learn more](#)
☐ Invert Regex

Matches the branch: dev

Included files filter (glob)
Changes affecting at least one included file will trigger builds

Ignored files filter (glob)
Changes only affecting ignored files won't trigger builds

[HIDE INCLUDED AND IGNORED FILES FILTERS](#)

Build configuration

File type
☒ Cloud Build configuration file (yaml or json)
☐ Dockerfile

Cloud Build configuration file location *
/cloudbuild.yaml
Specify the path to a Cloud Build configuration file in the Git repo [Learn more](#)

Substitution variables
User-defined substitutions allow re-use of a cloudbuild.yaml file with different variable values. Bindings allow you to access arbitrary data in the JSON payload of the webhook. [Learn more](#)

Variable	Value	Type
_FUNC_NAME	SimplePyFunc-dev	User-defined

[+ ADD VARIABLE](#)

CREATE Cancel

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.