

# IaC: Dataproc and Terraform

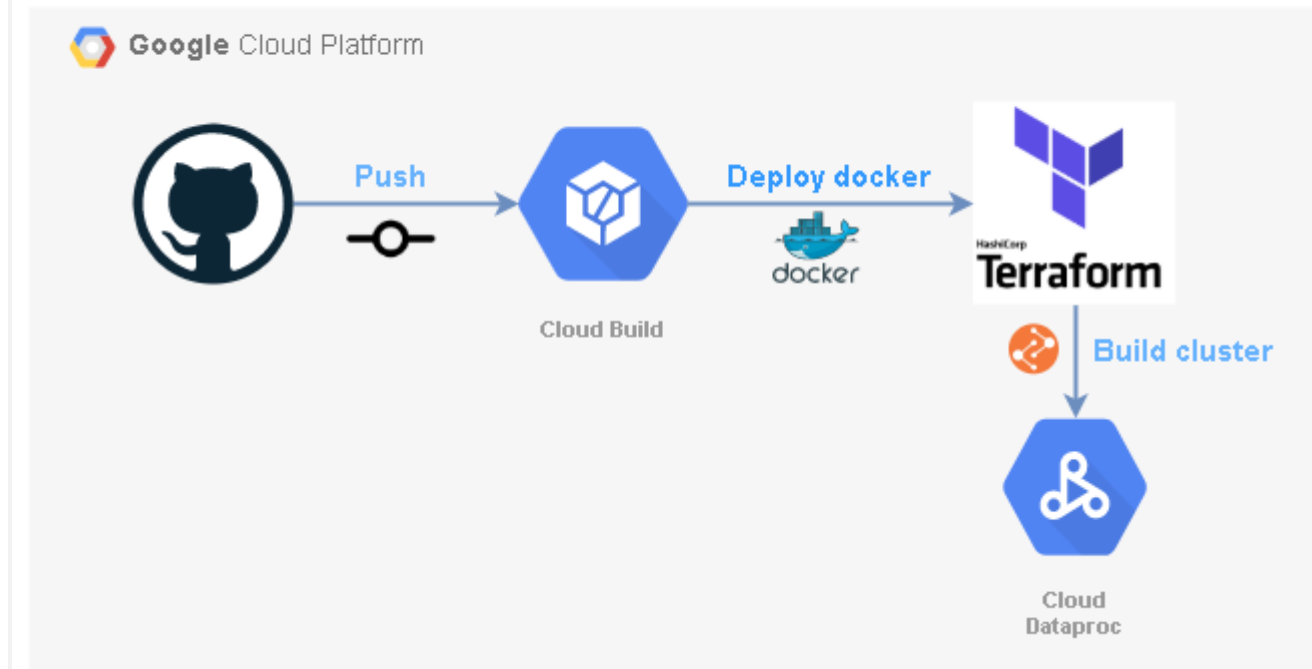
Imagine you want to start building some data pipelines in Spark or implement a model with Spark ML, the first step before anything is to deploy a Spark Cluster to make it easy you could set up in minutes a [Dataproc](#) cluster, It's a fully-managed cloud service that includes Spark, Hadoop, and Hive. Now imagine doing it many times, reproducing it in other [projects](#) or your organization want to make your [Dataproc](#) configurations a standard.

**Infrastructure as Code**, IaC is the process of managing and provisioning IaaS/PaaS/SaaS through human-readable definition files, rather than physical hardware configuration or interactive configuration tools.

**Cloud Build, Terraform** and its last integration with **Github** enable deployment of GCP components on-demand.

## Architecture and Scenario

## IaC: Continuous Dataproc Cluster Deployment



Components used

*Our objective is to make a simple repeatable pipeline to deploy a Dataproc Cluster.*

The workflow will start when we push our code to a remote repository already linked with our Google Cloud Build, this will execute automatically all the steps in the ***cloudbuild.yaml*** file that mainly deploys a Dataproc Cluster in our GCP Project. So our job for this scenario will be:

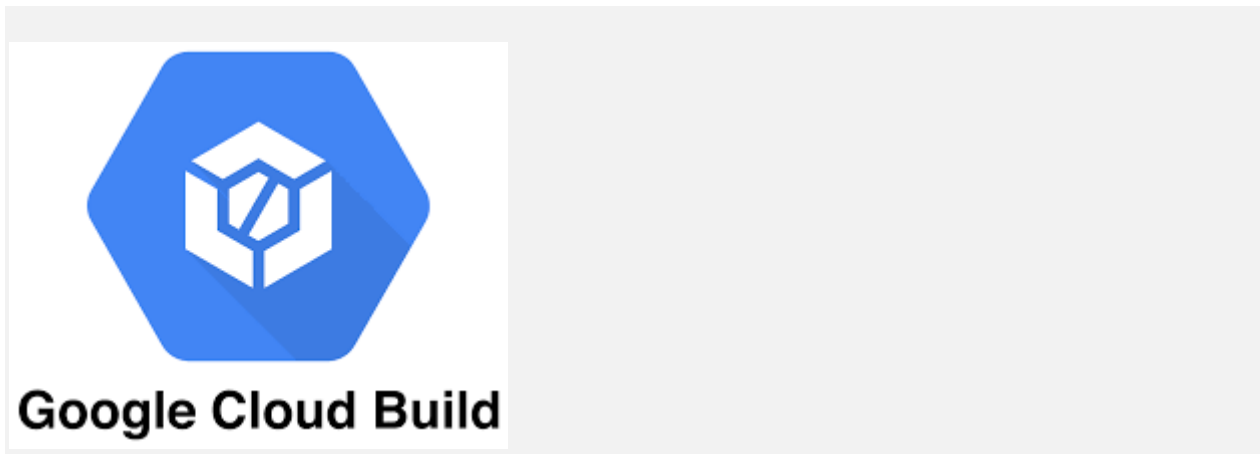
1. Fork the boilerplate repository
2. Connect Cloud Build with your GitHub repo

3. Writing the code on Cloud Build needed to deploy a Terraform container (*cloudbuild.yaml*)
4. Writing the Terraform code needed to build a Dataproc Cluster (*main.tf*)
5. Execute!

It means that with simple configurations and only two files we could have a pipeline to continuously deploy and destroy Dataproc clusters. Before start let's first check some definitions about the components we'll be using.

## Important Information

### Cloud Build



*Cloud Build executes your build as a series of build steps, where each build step is run in a Docker container* [[GCP Documentation](#)]. For this scenario, we'll use Cloud Build to build a docker with Terraform.

## Terraform



*Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions [[Terraform Documentation](#)]. In this opportunity, we are going to use Terraform for building a custom Dataproc cluster on the Google Cloud Platform.*

## Dataproc



*Dataproc is a fast, easy-to-use, fully managed cloud service for running [Apache Spark](#) and [Apache Hadoop](#) clusters in a simpler, more cost-efficient way [[Dataproc page](#)]. Since our*

purpose is only going up to deployment, we are just testing basic functionalities in order to check the correct working.

## GitHub



We don't need to explain it although GitHub is not only a Git repository hosting service, we'll also be using an [app](#) for automatically build containers on commits to our GitHub repository.



## Development


### **Before and during execution**


Please read carefully to avoid the most common error during execution time.

- Check if the Cloud Build API is enabled visiting the link

```
https://console.developers.google.com/apis/library/cloudbuild.googleapis.com?project=[YOUR-PROJECT-ID]&pli=1
```

 Google APIs  IaC-Demo-Project ▼


 API Library



## Cloud Build API

Google

Continuously build, test, and deploy.




[ENABLE](#) [TRY THIS API](#) 


Type	Overview
<a href="#">APIs &amp; services</a>	Cloud Build, Google Cloud's continuous integration (CI) and continuous delivery (CD)


Enable Cloud Build API


- Check if the Dataproc API is enable


```
https://console.cloud.google.com/dataproc/clusters?project=[YOUR-PROJECT-ID]&folder
```

 Google Cloud Platform  IaC-Demo-Project ▼ 

 Dataproc

 Clusters

 Jobs

 Workflows

### Cloud Dataproc Clusters

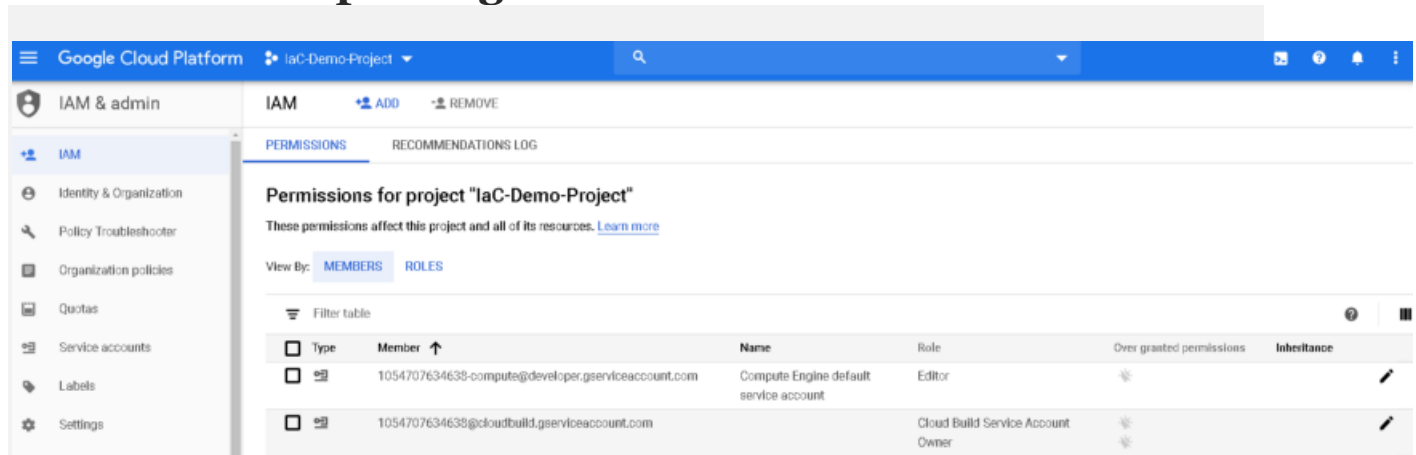
Google Cloud Dataproc lets you provision Apache Hadoop clusters and connect to underlying analytic data stores.

Create your first cluster to get started.

[Enabling](#) ...

Enable the Dataproc API

- Give the permissions necessary for the Cloud Build service account, for this article we are giving Owner Role **but remember for a real case to give the correct privileges.**



Owner is not a recommended role

- Enable the Cloud Resource Manager API

Google APIs IaC-Demo-Project

API Library

## Cloud Resource Manager API

Google

Creates, reads, and updates metadata for Google Cloud Platform resource containers.

[ENABLE](#) [TRY THIS API](#)

Type	Overview
APIs & services	Creates, reads, and updates metadata for Google Cloud Platform resource containers.
Last updated	

Cloud Build needs this API enables

- Validate if you have enough resources available in your region, so you'll avoid quotas problems, to update your machine type edit the file *terraform.tfvars*

```
Error: Error creating Dataproc cluster: googleapi: Error 400: Multiple validation errors:
- Google Cloud Storage bucket does not exist 'dataproc-bucket-6997dca22ab306ca'.
- Insufficient 'CPUS' quota. Requested 12.0, available 8.0.
- This request exceeds CPU quota. Some things to try: request fewer workers (a minimum of 2 is required), use smaller master and/or worker machine types (such as n1-standard-2)., badRequest
```

Problems with the CPU quota



```
# replace with n1-standard-1
machine_types = {
  "master" = "n1-standard-2"
  "worker" = "n1-standard-2"
}
```

CPU size requested

- Have a Staging Bucket(GCS) already created and defined in *terraform.tfvars*.

```
#Staging bucket, used used to stage files,
staging_bucket = "ctic-classes" You, a
```

Update this code with your own GCS

## Fork the boilerplate repository

In order to make it simpler, I published a repository with the code needed for the next steps, please fork it. I'll be explaining all the files in the following paragraphs.

### antoniocachuan/IaC-boilerplate

This repository is part of the article published on Medium This repository aims to deploy a Dataproc Cluster using...

[github.com](https://github.com)

The project looks like the image, and each file or folder has a different purpose.

**cloudbuild.yaml**=file used by Cloud Build to run Terraform dockers.  
**clouddestroy.yaml**=file used by Cloud Build to run a Terraform

docker that could destroy the Dataproc cluster (use carefully, and read about [State Locking](#)).

**environments/master**=principal branch that we'll be using

**environments/master/backend.tf**=file that indicate where will be saved the metadata related to Terraform.

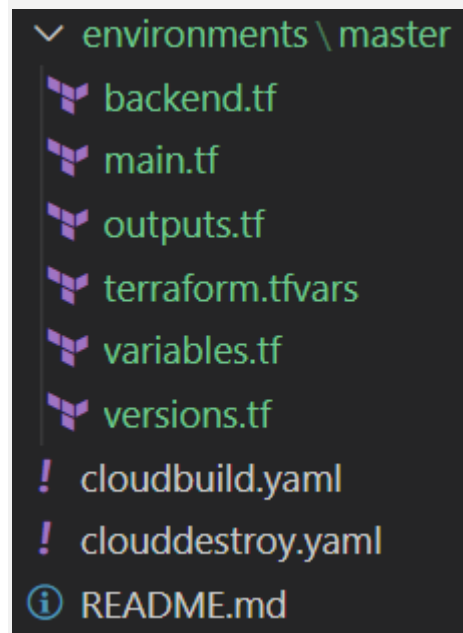
**environments/master/main.tf**=principal file that will implement all the components of the Dataproc cluster

**environments/master/outputs.tf**=reference some output variables generated during the Dataproc creation time.

**environments/master/terraform.tfvars**=file for defining manual input variables values like project name, region, machine types, etc that will be use in the main.tf. Normaly It's more frequently update than variables.tf

**environments/master/variables.tf**=space for defining all variables and its default values.

**environments/master/versions.tf**=version required for Terraform



Project structure

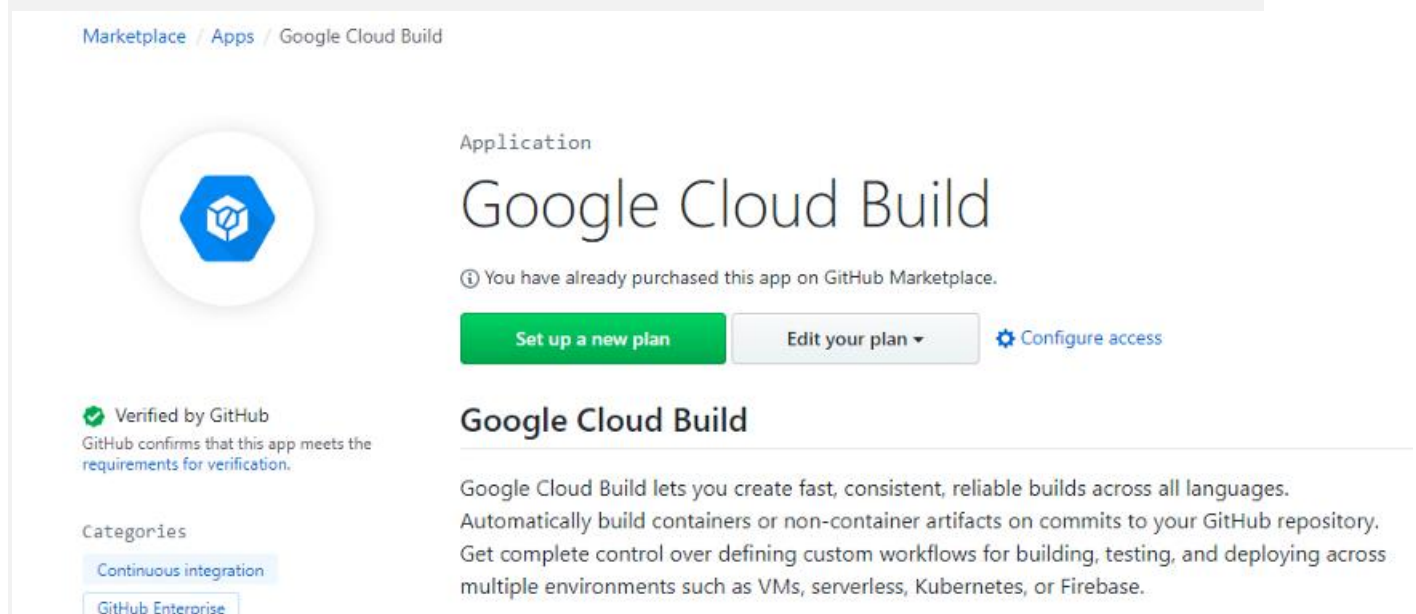
## Connect Cloud Build with a GitHub repo

First, let's create a simple GitHub repository, in the following image since it is not the first time doing this connection we could grant access to the Marketplace app to the repository directly, although we'll be doing like it was the first time.

Creating a simple repository

Make the first push containing all the code from the boilerplate.

```
git init
git add .
git commit -m "[ADD] Committing boilerplate"
git remote add origin
https://github.com/YOUR\_GITHUB\_ACCOUNT/YOUR\_GITHUB\_REPO.git
git push -u origin master
```




The screenshot shows the GitHub Marketplace page for the Google Cloud Build application. At the top, there's a breadcrumb trail: Marketplace / Apps / Google Cloud Build. The main header area features the Google Cloud Build logo (a blue hexagon with a white 'G' and a cloud icon) and the text 'Application Google Cloud Build'. Below this, a message states: 'You have already purchased this app on GitHub Marketplace.' There are three buttons: 'Set up a new plan' (green), 'Edit your plan' (grey with a dropdown arrow), and 'Configure access' (blue with a gear icon). On the left side, there's a 'Verified by GitHub' badge with a green checkmark and the text 'GitHub confirms that this app meets the requirements for verification.' Below this, under the 'Categories' section, there are two tags: 'Continuous integration' and 'GitHub Enterprise'. The main content area has a heading 'Google Cloud Build' followed by a description: 'Google Cloud Build lets you create fast, consistent, reliable builds across all languages. Automatically build containers or non-container artifacts on commits to your GitHub repository. Get complete control over defining custom workflows for building, testing, and deploying across multiple environments such as VMs, serverless, Kubernetes, or Firebase.'

Google Cloud Build App Page

Now let's make the connection accessing the [Cloud Build App Page](#), consider that this app offers a free tier and charge per build minute above it.

Google Cloud Build

Pay-as-you-go



Cloud Build charges per build minute above a free tier.

- ✓ 120 free build minutes a day. \$0.003/minute after that.
- ✓ Up to 10 concurrent builds included.
- ✓ Automatically push built images to Google Container Registry (storage costs may apply).
- ✓ See the Cloud Build pricing page for full details.  
<https://cloud.google.com/cloud-build/pricing>

Account: [InteligenciaClientes](#) ▼

Setup with Google Cloud Build

Next: Confirm your installation location.

Pay-as-you-go model

You need to allow all the permissions for the app to your GitHub account and select the repository you will be using, then you will have to select and allow a GCP Project that will be connected to the repository and finally creating a push trigger that will run a build every time a push is made to any branch.

1. Granting the app access to the Github repository



## Google Cloud Build

Connect your selected GitHub repositories to your Google Cloud Platform projects **BETA**

With this GitHub application, you can connect your repositories to your Google Cloud projects and start using Cloud Build.



Project settings

2

Repository selection

3

Trigger settings

Select repositories to connect with your project



Filter repositories



Select all repositories



BigDataCTIC/laC\_Dataproce\_Template

Connect repository

Cancel

### 2. Selecting the repository

# Google Cloud Build

Connect your selected GitHub repositories to your Google Cloud Platform projects **BETA**

With this GitHub application, you can connect your repositories to your Google Cloud projects and start using Cloud Build.

- ✓ Project settings
- ✓ Repository selection
- 3 Trigger settings

## Create a push trigger (optional)

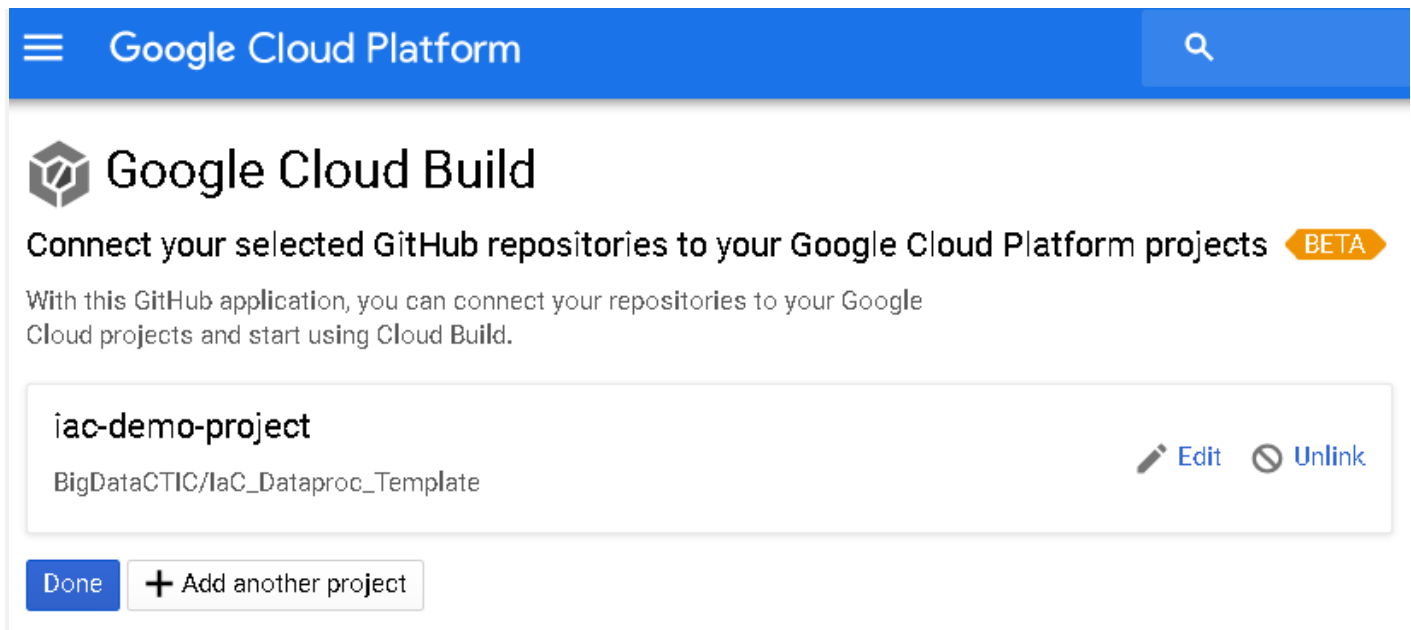
To get started, create a push trigger for each repository. This trigger will run a build every time a push is made to any branch. The trigger will look like the one below for each of your repositories. You can edit the settings for this trigger later.

Description	Type	Filter	Build configuration
Push to any branch	Push to branch	.*	Autodetected ?

Create a push trigger for these repositories:

- ✓ Select all repositories
- ✓ BigDataCTIC/laC\_Dataproce\_Template

## 3. Creating a push trigger



#### 4. Set up complete

Completing this, you have Cloud Build ready to build containers each time you push changes to any branch in your GitHub Repository.

### Writing the code on Cloud Build needed to deploy a Terraform container (*cloudbuild.yaml*)

Now the project is linked with GitHub and each push will deploy all the steps described in the file `cloudbuild.yaml`. This file is used by Cloud Build and in order to make it work on each step, you need to indicate the docker name and by default, it will look in [Docker Hub Repository](#) or [other builders provided by Google](#) and [the community](#) and deploy the docker that will be available just during the execution of the step. Additionally you can submit some arguments and this will allow us to submit Terraform commands.

If you observe the file It contains 4 steps, in all the cases when the code run it will assume that it is located in the root of the repository that is the reason you will see `cd environments/$BRANCH_NAME` in step 2 to 4. In this case, Cloud Build only runs Terraform commands to have more ordered code I divided into 4 Steps

**Step 1:** use the [alpine docker](#) to just print the branch we are using.

**Step 2:** deploys a [terraform docker](#) to launch the `init` command It is used to initialize a working directory containing Terraform configuration files [[Terraform docs](#)].

**Step 3:** deploys a [terraform docker](#) to launch the `plan` command this command is used to create an execution plan [[Terraform docs](#)].

**Step 4:** deploys a [terraform docker](#) to launch the `command` command It is used to apply the changes required to reach the desired state of the configuration or the pre-determined set of actions generated by a `terraform plan` execution plan.[[Terraform docs](#)].

## Writing the Terraform code needed to build a Dataproc Cluster (*main.tf*)

We have the repository connected to Cloud Build, the steps defined for deploying the Terraform Dockers. Now is time to write the code required for deploying a Dataproc cluster, the code needs



to follow the [Terraform syntax](#), and if you look the code below the majority is self-explanatory:

- The first part is for the provider, in this case, GCP and the variables of the project, region, and zone.
- The second part is to enable APIs needed to create the components.
- Networking module lets define a custom network for the cluster, this part could be omitted if you prefer to use the default network.
- Then comes the Dataproc definition, we are deploying a cluster with 1 master and 2 workers, mainly we use here the variables from terraform.tfvars or variables.tf to set the amount of RAM, CPU, Disk and other necessary components.
- Finally, the firewall rules, to allow the ingress connection for SSH and Hive JDBC port.

## **Execute!**

With all the steps completed, we just need to make a simple git push to start building our Dataproc Cluster to see the steps and check if there is any error we could monitor the execution in the Cloud Build interface.

The screenshot shows the Google Cloud Platform interface for the 'laC-Demo-Project'. The left sidebar shows 'Cloud Build' with options for History, Triggers, and Settings. The main area displays 'Build details' for a specific build. The build information includes:

- Build id:** 5d53e74d-867f-4c32-b5c9-5d6bb2851723
- Image:** —
- Trigger:** Push to master branch (Push to any branch)
- Source:** GitHub BigDataCTIC/laC\_Dataproj\_Template
- Git commit:** 6c6c790a7b21391caea1d056de8d6f5b86dbb245
- Started:** December 27, 2019 at 1:59:00 AM UTC-5
- Duration:** 10 min 2 sec

Below the build information, the 'Build steps' section shows four steps:

- branch name:** 2 sec. Command: `alpine -- -c "echo "*****" echo "master" echo "*****" "`
- tf init:** 8 sec. Command: `hashicorp/terraform:0.12.9 -- -c "cd environments/master terraform init "`
- tf plan:** 14 sec. Command: `hashicorp/terraform:0.12.9 -- -c "cd environments/master terraform plan "`
- tf apply:** 9 min 32 sec. Command: `hashicorp/terraform:0.12.9 -- -c "cd environments/master terraform apply -auto-approve "`

Cloud Build execution

And finally, after around 8 minutes we have a working Dataproj cluster, so now you can deploy in any project a custom cluster.

The screenshot shows the Google Cloud Platform interface for the 'laC-Demo-Project' under the 'Compute Engine' section. The left sidebar shows 'VM instances' selected. The main area displays 'VM instances' with buttons for 'CREATE INSTANCE', 'IMPORT VM', and 'REFRESH'. A table lists the deployed VM instances:

Name	Zone	Recommendation	In use by
<input type="checkbox"/> <input checked="" type="checkbox"/> dataproc-cluster-6997dca22ab306ca-m	us-central1-c		
<input type="checkbox"/> <input checked="" type="checkbox"/> dataproc-cluster-6997dca22ab306ca-w-0	us-central1-c		
<input type="checkbox"/> <input checked="" type="checkbox"/> dataproc-cluster-6997dca22ab306ca-w-1	us-central1-c		

Dataproj cluster deployed

## Conclusion and Future Work

This article's intention was to understand how simple it is to start building infrastructure with code in GCP and is not only for Dataproc, you could deploy easily almost all GCP components if you want to go further exists many resources from [Google Cloud](#) like [Cloud Foundation Toolkit](#). Please feel free to drop a comment or send me a message on [LinkedIn](#).