

COMPONENTES Y SUS RESPONSABILIDADES

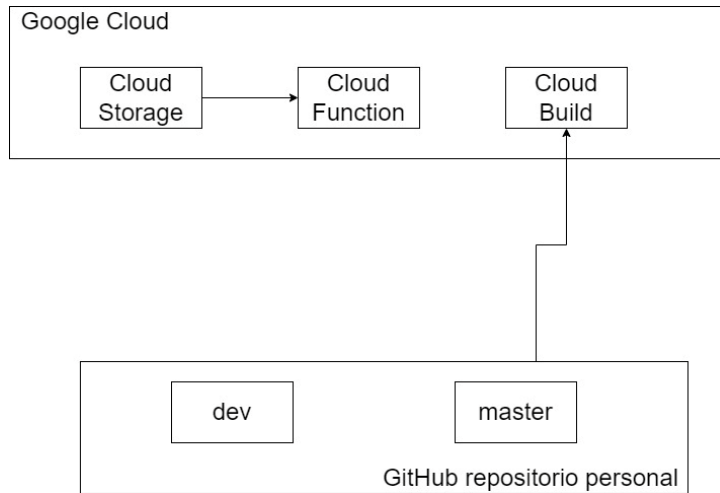


Ilustración 1 Componentes

Para este proyecto se utilizaron dos plataformas principales GitHub y Google Cloud. A continuación, y usando el diagrama presentado al inicio se explicará cada componente y que función cumple dentro del proyecto.

1. Cloud Storage

En este bucket de Google cloud es donde se estará subiendo las imágenes que se espera que el API de cloud vision vaya a revisar. Una vez que una imagen se sube a este bucket entonces se activará el cloud function que ejecutará el Python creado inicialmente y que ya se le hizo terraform apply.

2. Cloud Function

Es donde está la función que se ejecuta, que se activa una vez que se sube una imagen al cloud storage. Esta función es la encargada de llamar al API de vision para verificar cual es la emoción principal que se ve en la persona. Y va a devolver esta emoción y que tan probable es que sea la que muestra la persona de la imagen.

3. Cloud Build

En el cloud build es donde se corre el pipeline para asegurar una integración continua del proyecto. En el pipeline se deben ejecutar 5 pasos esenciales para garantizar un buen trabajo sin que ninguno falle.

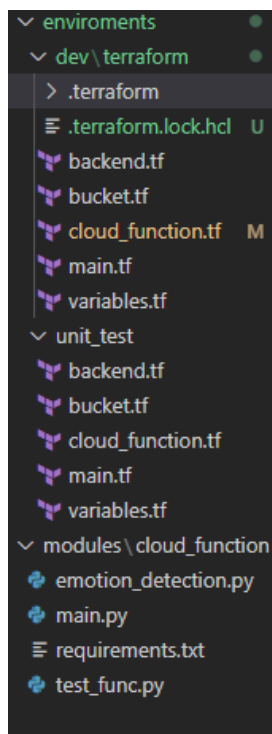
- Verificación del código por medio de lint.
- Ejecución del código para pruebas unitarias, esto con el fin de asegurar que no se está subiendo nada que quiebre el código.
- Terraform init para inicializar el proceso de terraform.
- Terraform plan para generar un reporte y saber que es lo que se le hará deploy y los cambios que van a suceder.

Para poder hacer el deploy al cloud function esta debe de hacerse manualmente, esto con el fin de evitar crear un proceso que luego quede ejecutándose continuamente y se llegue a cobrar. Para realizar el deploy se debe ejecutar el "Terraform apply"

4. GitHub

En GitHub es donde se llevará el control de versiones del código.

En el repositorio no solo debe estar el archivo de Python donde se obtienen las emociones, sino que también se debe realizar la estructura para poder ejecutar el terraform y el pipeline.



En la ilustración se muestra como se debe configurar el repositorio. En enviroments se debe hacer una carpeta para cada Branch en donde se quiera correr el pipeline y terraform. Ya que el cloudbuild.yaml que esta en el root del proyecto se ejecuta un bucle para pasar por cada una de estas carpetas siempre y cuando exista ese Branch en el repositorio.

Por otro lado, en la carpeta de modules es donde está el código del proyecto como tal. En el main.py es el que se va a ejecutar cuando el cloud function se ponga a funcionar. El test_func.py es donde está el unit testing que se ejecuta en el pipeline. El emotion_detection.py es el encargado de conectar con el api de vision para detectar la emoción y devolverla. Y por último requirements.txt es donde se deben poner dependencias para ser instaladas durante el pipeline en la máquina virtual donde se corre el mismo.

Ilustración 2 Módulos y componentes del código

Conectores entre cada componente

Los componentes mencionados anteriormente se conectan entre ellos por medio de Google Cloud y sus diferentes aplicaciones.

Todo el proceso se inicia cuando se hace un push o un pull request en GitHub hacia cualquier Branch. En el Cloud Build se configuró un trigger que detecta esa acción y busca por el archivo llamado “cloudbuild.yaml” en el cual se especifican los pasos del pipeline como se muestra en la ilustración 3.

Estos pasos se ejecutan en una virtual machine que Google Cloud crea solo para eso y se hace el build como en la ilustración 4.

```
steps:
# lint
- id: 'lint'
  name: 'python:3.8-slim'
  entrypoint: '/bin/sh'
  args:
  - '-c'
  - | ...

# unittest
- id: 'unittest'
  name: 'python:3.8-slim'
  entrypoint: '/bin/sh'
  args:
  - '-c'
  - | ...

# terraform init
- id: 'tf init'
  name: 'hashicorp/terraform:1.0.0'
  entrypoint: 'sh'
  args:
  - '-c'
  - | ...

# terraform plan
- id: 'tf plan'
  name: 'hashicorp/terraform:1.0.0'
  entrypoint: 'sh'
  args:
  - '-c'
  - | ...
```

Ilustración 3 cloudbuild.yaml







 Successful: a69dedef	
Started on Apr 19, 2022, 8:11:06 AM	
Steps	Duration
 Build Summary 4 Steps	00:00:54
 0: lint /bin/sh -c cd Proyecto\ 1/Viviana\ Villalobos/code/modules...	00:00:23
 1: unittest /bin/sh -c cd Proyecto\ 1/Viviana\ Villalobos/code/modules...	00:00:17
 2: tf init sh -c cd Proyecto\ 1/Viviana\ Villalobos/code/enviroments/...	00:00:06
 3: tf plan sh -c cd Proyecto\ 1/Viviana\ Villalobos/code/enviroments/...	00:00:03

Ilustración 4 Build en Cloud Build

Seguidamente esta información del build aparece en el GitHub específicamente si es un pull request dado que si este falla no se podrá completar el mismo tal como se muestra en la ilustración 5.

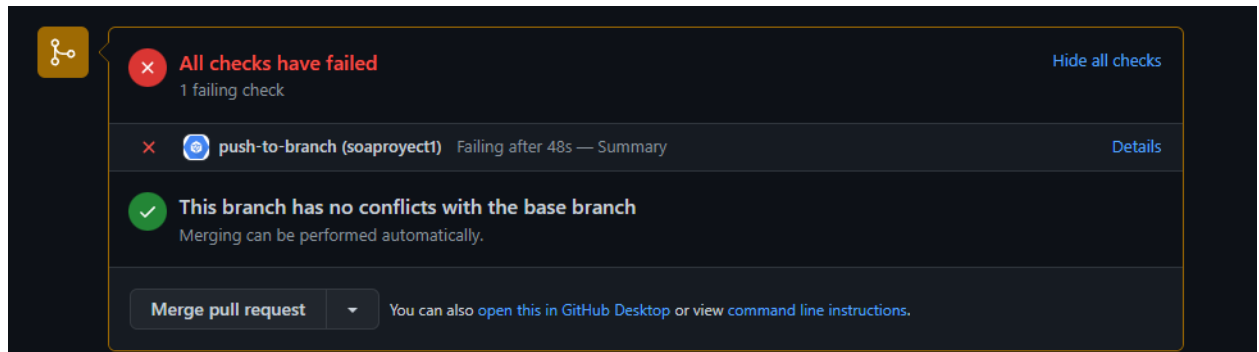


Ilustración 5 Fallo del build evita el pull request

Una vez finalizado todo el proceso a mano debe de hacerse el terraform apply para hacerle deploy al proyecto en el cloud function. Este a su vez queda esperando al Cloud Storage a que se suba una imagen así haciendo trigger a la función la cual se conecta con el API de Google Vision para obtener la emoción mejor representada entre las que sabe determinar.

Diagrama de la arquitectura de la infraestructura

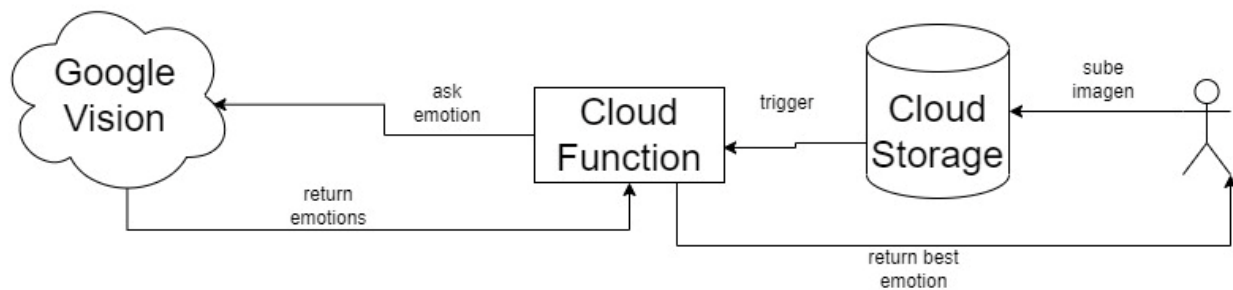


Ilustración 6 Diagrama de arquitectura

En la ilustración 6 se muestra el diagrama de arquitectura del proyecto y como interactúan los distintos componentes una vez que ya la función está en la nube y escuchando.

Decisiones relevantes

a. Pipeline:

Se tomó la decisión de realizar el pipeline en la aplicación de Google Cloud llamada Cloud Build. Esto porque tiene una mejor integración con GitHub. Por otro lado, siempre es importante elegir opciones conocidas, cuando se puede, para dar el mejor resultado posible en este caso al haber trabajado en clase con terraform y Google Cloud entonces se convierte en la mejor opción.

Cloud Build da la opción de poder añadir un Docker file o un YAML file para identificar los pasos del pipeline a ejecutar. En este caso se decidió por utilizar YAML dado que permite escribir paso a paso lo que se debe de ir haciendo.

b. Bucket:

Para el Bucket este se realizó en Google Cloud al igual que el pipeline. Esto para que todo se encuentre en un mismo lugar. Al estar todo bajo la misma plataforma no hay problemas de autorizaciones y las funciones se pueden desarrollar de la mejor manera.

c. Terraform apply:

Se tomó la decisión de no añadir el terraform apply en el pipeline dado que al generar un servicio que se cobra podría suceder un error humano y no destruirse el terraform terminando en un cobro.

d. Branching strategy:

Para el uso de GitHub y los branches se decidió utilizar únicamente uno. Esto porque al ser solo una persona trabajando es más fácil realizar los cambios. No hay necesidad de hacer pull request porque no hay aprobación de terceros. Así mismo, el pipeline ayuda a saber que el código no se ha quebrado y que funciona según lo esperado.

Referencias consultadas

<https://www.youtube.com/watch?v=pEbL TT9cHg>

https://www.youtube.com/watch?v=9FMVZkxfrfA&t=182s&ab_channel=ritvikmath

https://developers.google.com/resources/api-libraries/documentation/vision/v1/csharp/latest/classGoogle_1_1Apis_1_1Vision_1_1v1_1_1Data_1_1ImageSource.html#details

<https://www.codegrepper.com/code-examples/python/python+input+parameter>

https://www.youtube.com/watch?v=Z3S2gMBUkBo&ab_channel=SimplyExplained

<https://www.blazemeter.com/blog/how-to-integrate-your-github-repository-to-your-jenkins-project>

<https://stackoverflow.com/questions/26677637/how-to-uninstall-jenkins-on-windows>

https://www.youtube.com/watch?v=Zd014DjongE&ab_channel=Fireship

https://sonarcloud.io/?gads_campaign=SouthAmerica-Generic&gads_ad_group=ALMs&gads_keyword=github%20cloud%20integration&gclid=Cj0KCQjw0umSBhDrARIsAH7FCodHtFEaVrJxc-tIWZlvqFVdAunCHy3QDS5Ew1by7kUUY1KeATqn8xoaAvQIEALw_wcB

<https://stackoverflow.com/questions/56605865/cloud-build-not-able-to-find-the-dockerfile>

<https://cloud.google.com/storage/docs/creating-buckets?hl=es-419#storage-create-bucket-console>

<https://cloud.google.com/storage/docs/getting-bucket-information?hl=es-419#prereq-rest>

<https://cloud.google.com/storage/docs/access-control/making-data-public?hl=es-419#prereq-console>

<https://stackoverflow.com/questions/59017827/google-cloud-build-always-connects-to-wrong-github-account>

<https://stackoverflow.com/questions/59017827/google-cloud-build-always-connects-to-wrong-github-account>

<https://cloud.google.com/build/docs/automating-builds/build-repos-from-github>

<https://cloud.google.com/architecture/managing-infrastructure-as-code#architecture>

<https://github.com/GoogleCloudPlatform/solutions-terraform-cloudbuild-gitops>

<https://cloud.google.com/build/docs/build-push-docker-image>

<https://stackoverflow.com/questions/36183486/importerror-no-module-named-google>

<https://stackoverflow.com/questions/7877522/how-do-i-disable-missing-docstring-warnings-at-a-file-level-in-pylint>

<https://towardsdatascience.com/deploy-cloud-functions-on-gcp-with-terraform-111a1c4a9a88>