Instituto Tecnológico de Costa Rica

Arquitectura Orientada A Servicios Aplicada A Sistemas Emergentes
Primer Semestre Año 2022

Proyecto No 1

Grupo:

No 1

Profesor:
Alejandra Bolaños Murillo

Estudiantes:

Adrián González Jiménez – 2019173563

Introducción

En el siguiente proyecto se va a desarrollar un programa de software que incorpora el api de Vision de Google y una arquitectura de desarrollo basada en la Continuous Integration y Continuous Delivery usando Google Build.

A continuación, se procederá a explicar los componentes encontrados en este proyecto así como las responsabilidades de cada uno.

Componentes y sus responsabilidades:

1. **Cloud-Bucket:** Este componente va a estar encargado de servir para almacenamiento y de accionar un trigger en caso de que se suba una imagen, dicho trigger llama a la cloud function que evoca el proceso de análisis.
2. **Componente Serverless de Cloud function:** Este componente corresponde al script de Python que invoca a la REST de Vision y opera su resultado, trabaja realizando la consulta al bucket.
3. **RestApi de Visión:** Este api será la encargada de recibir una imagen y retornar el análisis solicitado, en este caso, corresponde al análisis de detección de rostros, que contiene información sobre los sentimientos de la imagen.
4. **Repositorio de GitHub:** Este componente se encargará de respaldar versiones del código, pero también de ejecutar acciones que permitan facilitar el CI Y CD, tales acciones son: la ejecución de un lint, ejecución de pruebas unitarias, ejecución de plan de terraform y aprobación manual (estos dos últimos en caso de un push request)
5. **Terraform:** Nos sirve para gestionar arquitecturas como servicio y genera un estado final que representa el proyecto.

Diagrama del sistema

A continuación se presentarán los diagramas contemplados para la arquitectura de este proyecto, primeramente, se verá el diagrama de primer nivel que relaciona de manera sencilla los componentes y sus conectores.

Figura 1

Diagrama de primer nivel de arquitectura del sistema

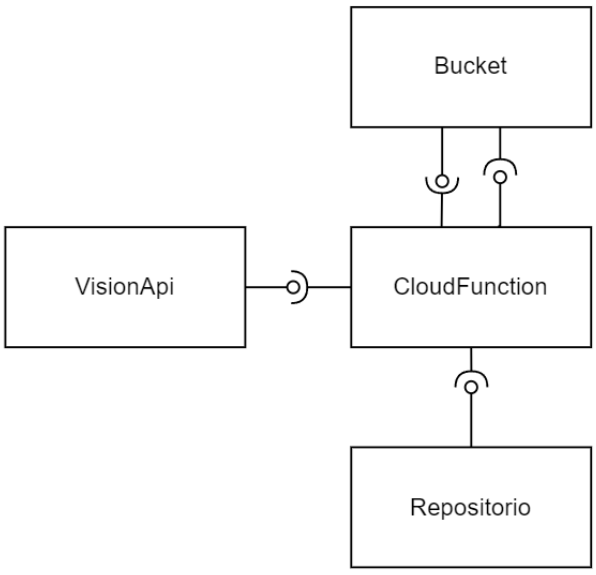
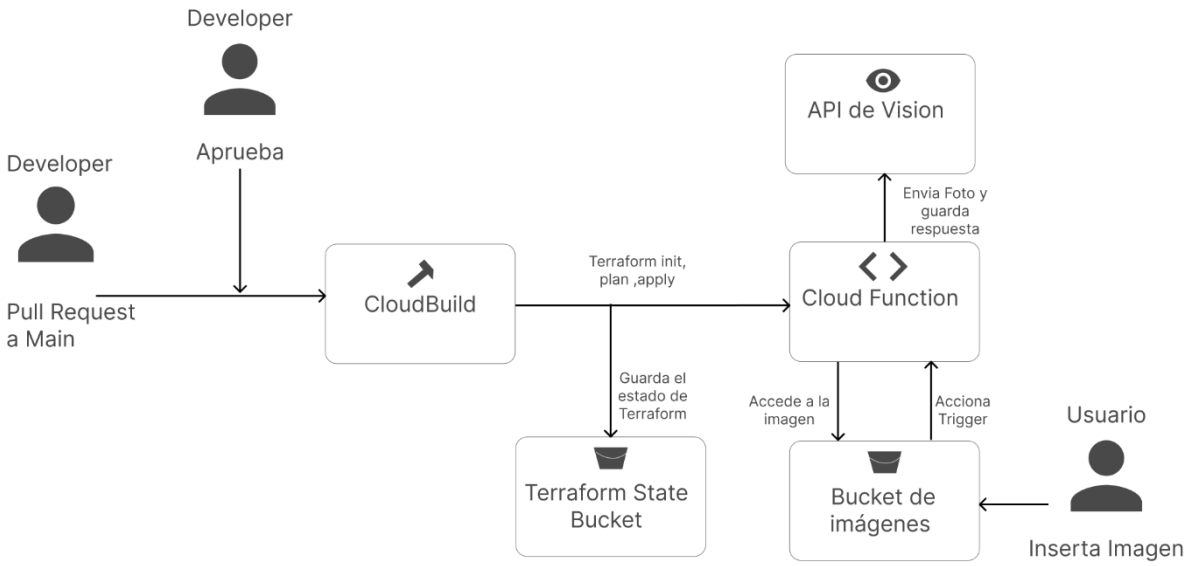


Figura 2

Diagrama de nivel superior de la arquitectura del sistema



Luego, podemos ver la figura 2 que explica un poco más la arquitectura y el flujo de funcionamiento del sistema.

Existen dos flujos de interacción, primero, el desarrollador cuando realiza un pull request a main y otro desarrollador lo autoriza , cloud build acciona un trigger asociado a dicho evento y ejecuta el proceso de compilación, que se especifica en un documento.yaml con los pasos a seguir, en este proyecto se utilizará terraform como manejador de arquitectura, por lo que para guardar su estado este debe almacenarse en un bucket, una vez se compila el proceso, este se despliega a una cloud function. Segundo, el usuario cuando inserta una imagen se acciona un trigger que llama a la cloud function, esta cloud function obtiene la imagen que el usuario depositó en el bucket, posteriormente, envía esta información al api de visión, que retorna información al respecto.

Conectores entre cada componente

A continuación, se describirán los conectores que componen el proyecto

1. Bucket-CloudFunction: Cada vez que subamos una imagen ejecutaremos un trigger que se encargará de llamar a la CloudFuntion.
2. CloudFunction-Vision: Cada vez que se llame al proceso la cloud function le enviará la fotografía a dicha api, que nos retorna un array de información acerca de las caras en la foto.
3. Github-CloudFunction: Cada vez que se ejecute un push request, el pipeline en GitHub se va a encargar de realizar una serie de pasos, entre ellos el terraform init, plane y apply a Cloud.
4. CloudBuild-TerraformBucket: Como este tipo de funciones en la nube son “stateless” debemos asegurarnos de guardar el estado de la aplicación de terraform en un bucket.

Toma de decisiones

A continuación, se explicarán las decisiones de tecnología y demás para este proyecto, hay que remarcar que nuestro objetivo es desarrollar una arquitectura escalable que nos permita responder ágilmente al cambio no solo de features, si no también de personal, tecnologías y demás.

Las tecnologías mencionadas a continuación buscan desarrollar la solución del proyecto dando énfasis, en popularidad, curva aprendizaje inicial y funcionalidad, es importante destacar que la popularidad es crítica, dado que entre más popular sea una tecnología, mas respuestas existen ante los posibles errores, también una curva de aprendizaje pequeña nos ayuda a poder integrar nuevas personas rápidamente, reconocer errores y plantear soluciones.

- **Estrategia de Branching:**

Para este proyecto vamos a utilizar la estrategia de Git Flow, primeramente, porque es bastante sencillo de entender y segundo que debido a su estructuración nos permite trabajar de manera descentralizada entre equipos, pero manteniendo un grado esencial de centralización a la hora de trabajar en el proyecto, dicha estructuración nos ofrece múltiples ventajas, como un ambiente intuitivo, predecible y testeable.

Las razones de peso, por la que se escoge esta branching strategy y no otra, es por la simplicidad y popularidad, debido a que la curva de aprendizaje no es tan acentuada y existe bastante documentación al respecto. Entonces se espera preparar el ambiente de CI/CD y posteriormente crear los branches y protecciones.

Esta estrategia como menciona Benet J.(2013) tiene como énfasis los siguientes tres principios:

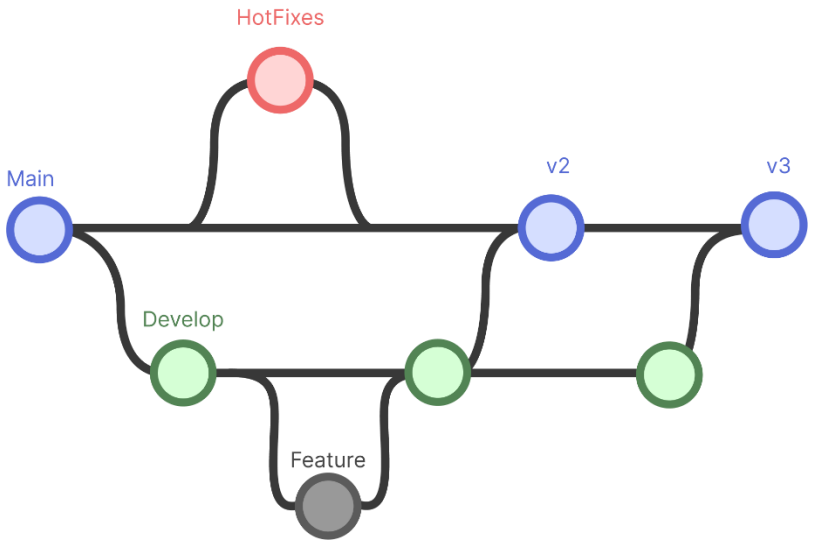
1. main siempre debe ser desplegable
2. Todos los cambios deben manejarse a través de los Features Branches a través de pull-request + merge
3. Rebasar para evitar o resolver comflictos y realizar merge a main

A continuación, se muestra el diagrama de branches del repositorio, donde la rama “main”, contiene una versión siempre funcional del código, una rama, llamada Develop parte de main y funcionará para crear ramas en las que se trabajan los features que se incorporarán, representadas como “Feature”, cuando se considere las nuevas funcionalidades estén listas, se unen a Develop, y si esta última también está lista, se une a Main.

Existe otra rama llamada hotfixes, que está pensada para solucionar bugs en producción un poco más rápido.

Figura 3

Diagrama de git Flow para este proyecto



Al observar el diagrama anterior, es posible ver, porqué nos permite trabajar de manera modular pero centralizada, dado que por cada feature existe una rama, no nos debería importar la existencia de otros cambios, pero al final debemos acoplarnos a develop por lo que seguimos la estructura del proyecto.

Esta estrategia nos permite solo preocuparnos por implementar las funcionalidades y evitar estar resolviendo conflictos cada vez que se hace un cambio nuevo.

- **Pipeline de CI/CD:**

Para este proyecto tenemos la opción de Jenkins y Google Cloud, según knapsackpro (2021), ambos servidores, está en términos de features, similares, con algunas variaciones entre características o de paga (Jenkins es código libre y abierto, mientras que Google cloud es un servicio de paga “Pay as you go”), como para este proyecto se piensa usar Google cloud functions, buckets y demás servicios pertenecientes de Google, se espera un acople más fácil y centralizado entre sus componentes, razón por la cual se va a utilizar Google cloud build.

- **Python:**

En este caso, es un requerimiento solicitado, pero también hay que destacar que para las acciones que vamos a realizar, tiene bastante soporte y una comunidad bastante extensa, lo que nos permite establecer que Python es bastante confiable para el objetivo de desarrollar una arquitectura escalable y mantenible.

- **Pruebas de código:**

Para la realización de pruebas unitarias se debe escoger un framework de Python, en este caso se escogió Pytest, ya que como menciona Lyubinsky S. (2020) tiene una comunidad gigantesca, open-source y es muy fácil de implementar, además, debido a su sencilla sintaxis, el

desarrollo de pruebas unitarias se hace muy práctico y limpio, lo cual es precisamente lo que buscamos para diseñar una arquitectura va a evolucionar con el tiempo.

Bibliografía:

Deloitte España (2021). *¿Qué es Terraform?* https://www2.deloitte.com/es/es/blog/todo-tecnologia/2021/que-es-terraform.html

Benet J.(2013). *A simple git branching model (written in 2013)* https://gist.github.com/jbenet/ee6c9ac48068889b0912/revisions

Lyubinsky S. (2020). *Top 8 Python Testing Frameworks in 2021*.https://blog.testproject.io/2020/10/27/top-python-testing-frameworks/

knapsackpro (2021). *Jenkins vs Google Cloud Build*. https://knapsackpro.com/ci_comparisons/jenkins/vs/google-cloud-build