Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores
(Computer Engineering Academic Area)

Programa de Licenciatura en Ingeniería en Computadores
(Licentiate Degree Program in Computer Engineering)

SOA4ID Arquitectura Orientada a Servicios Aplicada a Sistemas Emergentes

Proyecto 1
Documentación
(Documentation)

Realizado por:
Made by:
Jason Salazar González

Profesora:
(Professor)
Alejandra Bolaños

Fecha de entrega:
(Date)
19 de abril 2022

Introducción

En este proyecto se construye un flujo de desarrollo, despliegue y puesta en producción de una aplicación, empleando los principios de arquitectura e infraestructura de software.

El sistema es únicamente un MVP, por lo cual no cuenta con interfaz gráfica. Y está inicialmente conformado exclusivamente por elementos creados vía IaC.

La intención es proveer a los futuros desarrolladores del proyecto, con una infraestructura estable, con el uso de la automatización en las etapas del desarrollo de aplicaciones.

La aplicación toma una imagen (fotografía de una persona) desde el cloud bucket, y consulta el GCP VISION API, mediante el cual la imagen será analizada, y la emoción que la fotografía refleja será analizada por el API. Además esta emoción se imprime como resultado de la llamada a la función.

En este documento se detalla el sistema implementado, así como la toma de decisiones de su diseño.

Set de tecnologías

A continuación se indican las diferentes tecnologías y herramientas utilizadas en el desarrollo de este proyecto, así como una descripción breve de cada una de estas.

- **Google Cloud Platform (GCP):** También conocido como Google Cloud, es un proveedor de recursos de computación en la nube que se utilizan para desarrollar, implementar y operar aplicaciones en la web. Este ofrece más de cien productos y servicios [1].

  Nota: Cada uno de los componentes de esta plataforma utilizados en el proyecto (**Cloud Storage buckets, Cloud Vision API, Cloud Build** y **Cloud**

Functions) se detallan en la sección de Componentes y conectores del sistema.

- **GitHub:** Es un repositorio online gratuito que permite gestionar proyectos y controlar versiones de código [2].
- **Python:** Es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código, se utiliza para desarrollar aplicaciones de todo tipo [3].
- **Terraform:** Es un programa para gestionar IaC (Infrastructure as Code). Define una estructura y sintaxis generales para el código y lo ejecuta para alcanzar el estado deseado. Terraform es, por tanto, declarativo: lo que se indica es el estado final que se quiere para la plataforma, no los pasos para llegar a ese estado [4].
- **Yaml:** Es un lenguaje de serialización de datos que suele utilizarse en el diseño de archivos de configuración [5].

Componentes y conectores del sistema

A continuación se detallan cada unos de los componentes y conectores del sistema, así como sus responsabilidades.

- **Cloud Storage buckets:** Este componente de GCP es un servicio de almacenamiento de archivos que permite almacenar y acceder a datos en GCP. En este trabajo se utilizan tres instancias de este componente por ambiente de trabajo. Uno de estos es utilizado para almacenar el estado de Terraform como un estado remoto. Otro es utilizado para almacenar las imágenes que van a ser procesadas por otros componentes. Y el otro es utilizado para almacenar el código fuente de la Cloud Function implementada.

- **Cloud Vision API:** Este componente permite integrar con facilidad las funciones de detección de visión en las aplicaciones, como el etiquetado de imágenes y la detección de rostros. En este trabajo este componente se utiliza para analizar imágenes y así detectar emociones en rostros humanos.
- **Cloud Build:** Este es un servicio que ejecuta compilaciones en la infraestructura de Google Cloud Platform. En este trabajo se utiliza Cloud Build para la creación de un pipeline de CI/CD. Para esto Cloud Build importa el código fuente del pipeline desde GitHub, y ejecuta la compilación como una serie de pasos de compilación, en los que cada uno se ejecuta en un contenedor de Docker.
- **Cloud Functions:** Es un entorno de ejecución sin servidores para compilar y conectar servicios en la nube. Permite programar funciones simples de un solo propósito vinculadas a eventos emitidos desde una infraestructura y servicios en la nube. En este proyecto se programa una función por ambiente de trabajo, la cual permite el procesamiento de imágenes en tiempo real. Esta función responde al evento generado a causa de cargar una imagen a un bucket de la infraestructura. Esta nueva imagen es procesada por la función, que por medio de Cloud Vision API detecta si hay algún rostro humano en la imagen para luego detectar si este presenta alguna emoción (ira, sorpresa o alegría).

Diagrama de la arquitectura de la infraestructura

En la figura 1 se muestra el diagrama de la arquitectura de la infraestructura. Este diagrama permite ver cómo están conectados los componentes de la infraestructura mencionados anteriormente.

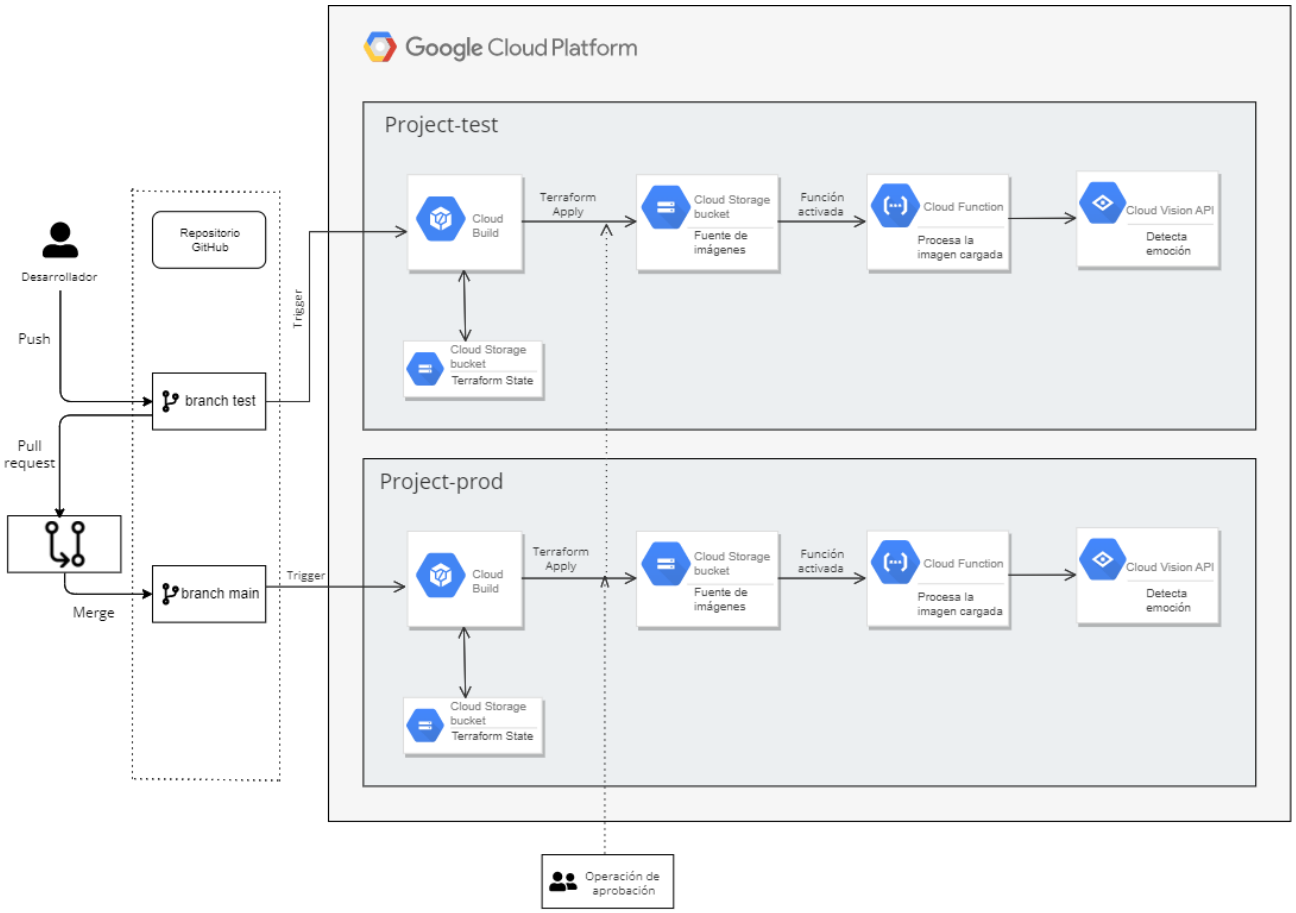

Figura 1. Diagrama de la arquitectura de la infraestructura.

Arquitectura del flujo de trabajo del pipeline de CI/CD

Para tener un mejor entendimiento del sistema implementado, en la figura 2 se muestra la arquitectura del flujo de trabajo del pipeline de CI/CD.

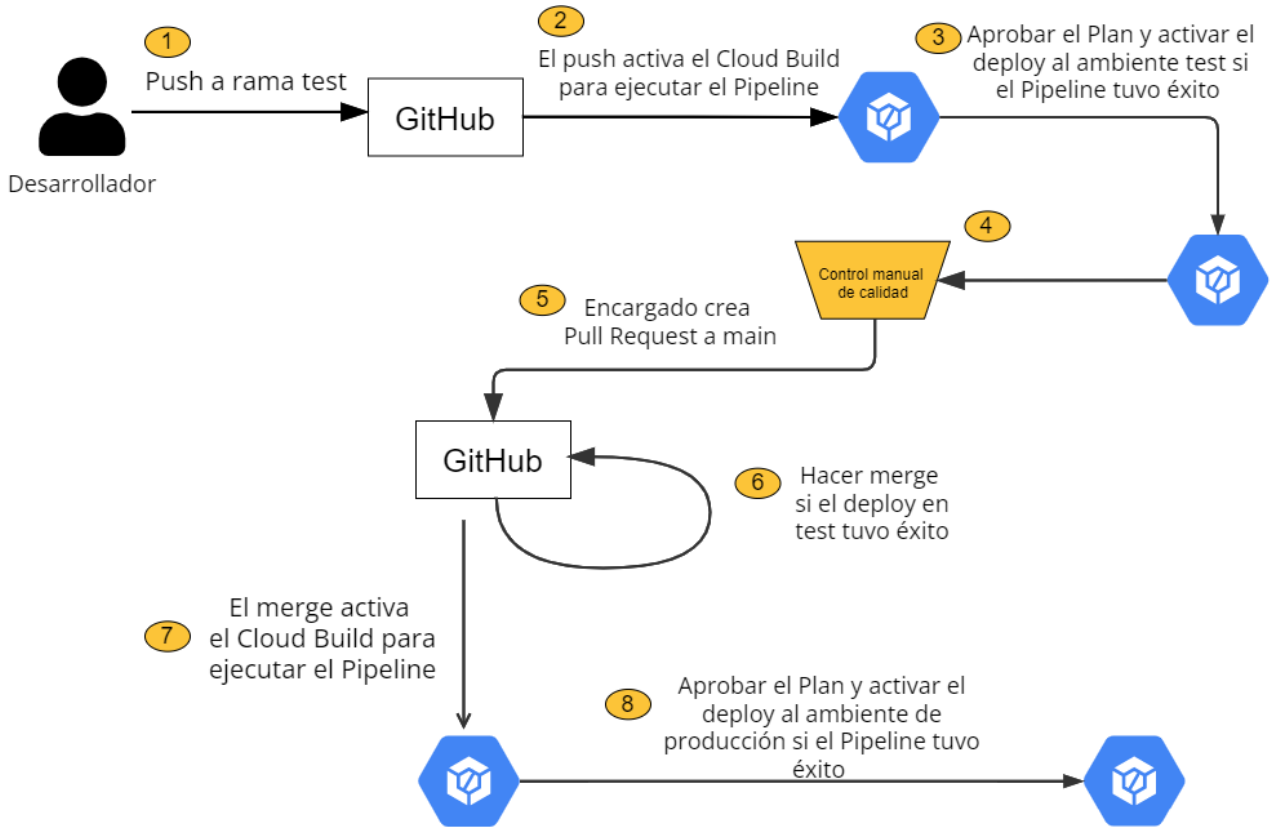

Figura 2. Arquitectura del flujo de trabajo del pipeline de CI/CD.

Documentación de decisiones

- **Branching strategy:** Para escoger una adecuada branching strategy para administrar el proyecto y optimizar el flujo de trabajo se analizaron dos opciones, estas son: Git Flow y GitHub Flow.

  En el caso de GitHub Flow es un flujo de trabajo relativamente simple que permite que los equipos más pequeños, agilicen su trabajo. En GitHub Flow, la rama principal contiene su código listo para producción. Las otras ramas, feature branches, deben contener trabajo de nuevas características y correcciones de errores y se fusionarán nuevamente en la rama principal cuando el trabajo esté terminado y revisado adecuadamente [7].

  Por otra parte, la idea principal detrás de Git Flow es aislar su trabajo en diferentes tipos de ramificaciones. Hay cinco tipos diferentes de ramas en total. Las dos ramas principales son main y develop. Hay tres tipos de ramas de soporte con diferentes propósitos: feature, release, and hotfix [7].

  Algunas beneficios de Git Flow son [7]:

  - Los diversos tipos de ramas hacen que sea fácil e intuitivo organizar su trabajo.
  - El proceso de desarrollo sistemático permite pruebas eficientes.
  - El uso de release branches le permite soportar de manera fácil y continua múltiples versiones de código de producción.

Aprovechando los beneficios de Git Flow mencionados anteriormente se decide utilizar esta estrategia para el desarrollo de este trabajo. Debido a las características de este trabajo, aquí se emplean únicamente las dos ramas principales de este flujo. Develop (rama llamada test en este caso) contendrá código listo para pruebas,

mientras main contendrá código listo para producción. Entonces, una vez que el código de la rama test sea probado, podrá ser combinado en la rama main.

En la imagen 3 se muestra el diagrama del flujo del branching strategy utilizado en este trabajo.

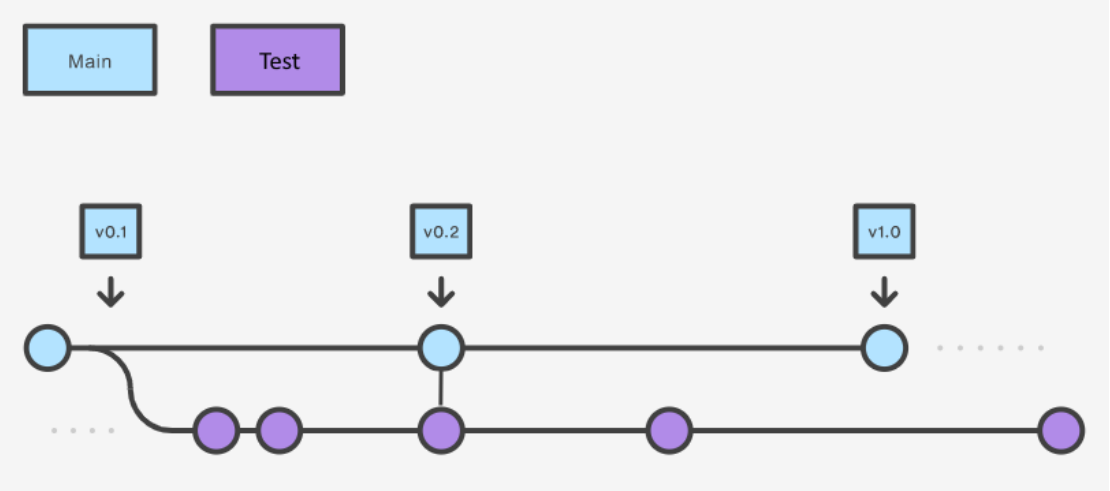

Figura 3. Diagrama del branching strategy, GitFlow.

- **Pipeline de CI/CD:** Para el caso de la implementación del pipeline se comparan dos opciones de servidores de integración continua, esto son: Jenkins y Google Cloud Build. A continuación se realiza una tabla donde se comparan algunas de las características más importantes de ambos [6]:

	Jenkins	**Cloud Build**
Ofrece plan gratis	Sí. Gratis y de código abierto.	Se ofrecen 120 minutos de compilación gratis por día.
Soporte/SLA	No (Parcial) No hay soporte oficial. Pero debido a su popularidad se encuentra soporte en muchos foros.	Sí. Incluso brinda soporte de telefónico 24/7.

Paralelismo	Parcial.	Sí.
Soporte de Containers/ Ambientes de desarrollo	No(Parcial) Ejecuta todas las compilaciones en el mismo entorno que el propio servidor de compilación.	Sí. Compatibilidad nativa con Docker y Packer
Management support (Qué tan fácil es administrar usuarios/proyectos/ asignar roles y permisos, etc.)	Faltan las funciones de colaboración integradas en otros productos similares, al igual que las funciones de gobernanza.	Sí.
Auditing	Las instancias de Jenkins son realmente administradas por un único usuario con privilegios administrativos.	Sí.

Basado en las ventajas mencionadas de Cloud Build en el cuadro anterior, la facilidad de utilizar este en GCP y su fácil acoplamiento con GitHub se decide utilizar esta herramienta.

- **Elemento serverless:** Para la ejecución de la funcionalidad del sistema se analiza entre dos herramientas, estas son: Google Cloud Functions y GCP App Engine.

  Por una parte se tiene que App Engine es para crear una aplicación serverless con una web o un backend de API y admite varios lenguajes de desarrollo sin necesidad de preocuparse por el soporte de infraestructura. Cuando se tiene una aplicación que necesita comunicarse con varios servicios, como una aplicación web o API, Google Cloud App Engine es una solución adecuada [8].

Mientras que las soluciones basadas en eventos que se extienden a los servicios de Google y de terceros son una buena opción para las Cloud Functions, así como aquellas que necesitan escalar rápidamente [8].

Si bien App Engine admite muchos servicios diferentes dentro de una sola aplicación, Cloud Functions admite servicios individualizados. Si los requisitos no incluyen múltiples servicios, Cloud Functions es una excelente opción [8].

Por lo anterior se decide utilizar Google Cloud Functions para implementar la funcionalidad del sistema.

- **Lenguaje de programación:** Como parte de los requerimientos para la elaboración de este trabajo, se solicita que se utilice Python como lenguaje de programación para implementar la funcionalidad del sistema.
    - **Check code:** Para revisar la calidad del código se utiliza **Pylint**. El cual es una herramienta que busca errores en el código de Python, e intenta aplicar un estándar de codificación. También puede buscar ciertos errores de tipo, y puede recomendar sugerencias sobre cómo se pueden refactorizar bloques particulares [9].
    - **Pruebas Unitarias:** Para la elaboración de pruebas unitarias se utiliza el framework **pytest**. Ya que este facilita la escritura de pruebas pequeñas y legibles, y puede escalar para admitir pruebas funcionales complejas para aplicaciones y bibliotecas [10].

Referencia bibliográficas

[1] V. Trafaniuc. "Descubre qué es Google Cloud Platform y sus ventajas". Maplink. https://maplink.global/blog/es/que-es-google-cloud/ (accedido el 17 de abril de 2022).
[2] "¿Qué es GITHub y para qué sirve?" Webempresa. https://www.webempresa.com/hosting/que-es-github.html (accedido el 17 de abril de 2022).
[3] "¿Qué es Python? - Datademia". Datademia. https://datademia.es/blog/que-es-python (accedido el 17 de abril de 2022).
[4] "¿Qué es Terraform?". Deloitte España. https://www2.deloitte.com/es/es/blog/todo-tecnologia/2021/que-es-terraform.html (accedido el 17 de abril de 2022).
[5] "¿Qué es YAML?" Red Hat - We make open source technologies for the enterprise. https://www.redhat.com/es/topics/automation/what-is-yaml (accedido el 17 de abril de 2022).
[6] "Jenkins vs Google Cloud Build comparison of Continuous Integration servers". Speed up your tests with optimal test suite parallelisation. https://knapsackpro.com/ci_comparisons/jenkins/vs/google-cloud-build (accedido el 18 de abril de 2022).
[7] "What is the best Git branch strategy? | Git Best Practices". GitKraken. https://www.gitkraken.com/learn/git/best-practices/git-branch-strategy (accedido el 18 de abril de 2022).
[8] "The Differences Between GCP App Engine, Cloud Run & Cloud Function". Sphere Partners. https://www.sphereinc.com/blogs/when-to-choose-app-engine-vs-cloud-functions-or-cloud-run-in-gcp/#:~:text=While%20App%20Engine%20supports%20many,Functions%20is%20a%20great%20choice. (accedido el 18 de abril de 2022).
[9] "Pylint 2.14.0-dev0 documentation". Pylint 2.14.0-dev0 documentation. https://pylint.pycqa.org/en/latest/ (accedido el 18 de abril de 2022).
[10] "pytest: helps you write better programs — pytest documentation". pytest: helps you write better programs — pytest documentation. https://docs.pytest.org/en/7.1.x/ (accedido el 18 de abril de 2022).