Entrega 4 - Arquitectura, conclusiones y consideraciones

Javier Alejandro Gómez, Alejandra Guerrero, Diego Alejandro Peña, Juan Ignacio Arbeláez

{ja.gomez1003, a.guerrero10, da.pena20, ji.arbelaez}@uniandes.edu.co

Desarrollo de soluciones cloud - MINE

Universidad de los Andes, Bogotá, Colombia

Fecha de presentación: mayo 7 de 2023

Tabla de contenido

1. Introducción

En el presente documento se expone la arquitectura implementada para generar una aplicación web en Flask, la cual permita la compresión de archivos por medio de peticiones HTTP y utilizando un enfoque de procesamiento asíncrono. También se presentan las consideraciones que se tuvieron en cuenta en el desarrollo de este proyecto para cumplir con el objetivo, el cual era aplicar los conceptos de escalabilidad, una de las características principales de los ambientes en nube.

El código y la documentación de este proyecto se pueden consultar en los siguientes enlaces:

- Repositorio GitHub: https://github.com/aguerrero10/Desarrollo-cloud-grupal/tree/vm_gcp
- Documentación de Postman: https://documenter.getpostman.com/view/13843294/2s93CPrY2y
- Video de sustentación: Link del video

2. Arquitectura planteada

En este trabajo, se ha desarrollado una arquitectura robusta y escalable para la gestión de datos masivos que utiliza los siguientes componentes:

- Base de datos PostgreSQL usando el servicio Cloud SQL de GCP
- Un balanceador de carga asociado a un grupo de instancias de máquinas virtuales que son creadas a partir de una plantilla de máquinas virtuales para el servicio web
- Para el almacenamiento de archivos se usó Cloud Storage
- Para el despliegue de los workers se está usando Cloud Functions
- Se usa Pub/Sub para la comunicación entre el servicio web y Cloud Functions
- Por último, Cloud Monitoring para el monitoreo del proyecto y sus componentes

En el siguiente diagrama se puede evidenciar la arquitectura de la aplicación:

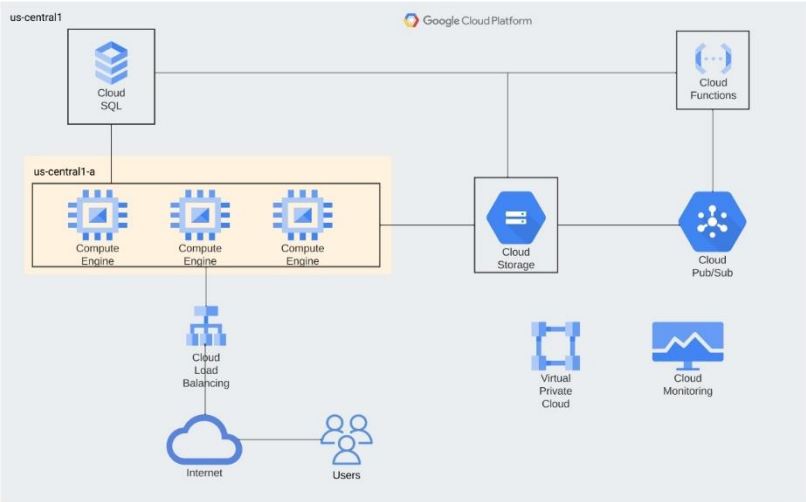


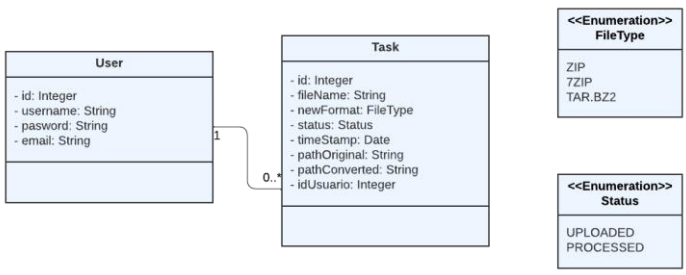
Figura 1. Diagrama de despliegue

Como se evidencia en el diagrama de despliegue, todos los componentes pertenecen a una misma región, aunque en distintas zonas, y se interconectan entre sí por medio de una red privada. La capa web server de la aplicación está compuesta por un balanceador de carga que se encarga de escalar y responder las solicitudes HTTP para la creación de usuarios y la gestión de compresión de archivos. La capa del web server inicialmente cuenta con una instancia de Compute Engine y a medida que se cumplen las reglas de escalabilidad (uso del 80% de la CPUD) se van a adicionar instancias de Compute Engine hasta alcanzar un máximo de 3 instancias. Las instancias de Compute Engine van a conectarse con Cloud SQL para poder almacenar de forma independiente y transversal a la capa de procesamiento los metadatos de los usuarios y de sus solicitudes de compresión, así como también habrá una conexión con Cloud Storage para la persistencia de los archivos pendientes de comprimir y comprimidos.

Por otra parte, la capa worker se compone de una función en Cloud Functions, la cual se activa cuando el tema de Pub/Sub al que se encuentra suscrito recibe una notificación de carga de archivo en Cloud Storage. Es entonces cuando la función se encarga de comprimir el archivo, actualizar el estado de esa tarea en la base de datos y notificar por correo electrónico al usuario para que proceda a descargar su archivo en versión comprimida.

Por otro lado, la lógica de negocio (backend) y la base de datos siguen el modelo de datos que se muestra en el siguiente diagrama:

Diagrama de clases



Es decir, el modelo de datos se compone de la entidad Usuario la cual se compone de sus datos básicos y puede o no tener múltiples tareas de compresión de archivos. Por otra parte, la entidad tarea tiene los datos del archivo a comprimir, su estado y tipo de archivo de archivo de compresión, modeladas con una enumeración.

3. Consideraciones de la aplicación

En el caso de la arquitectura planteada se eligieron las siguientes características de las instancias de servicios en la nube. La plantilla de instancias de Compute Engine se creó con un tipo de maquina F1-micro, con un SO Debian 11, tráfico HTTP habilitado y con el siguiente script para la configuración del servidor web:

```
#!/bin/bash
sudo apt-get update
cd ..
sudo mkdir proyecto
cd proyecto
yes | sudo apt-get install git
sudo git clone -b vm_gcp https://github.com/aguerrero10/Desarrollo-cloud-grupal
gsutil cp gs://proyecto-dsc/resources/dsc-proyecto-b6565f206c96.json
/proyecto/Desarrollo-cloud-grupal/backend/flaskr/dsc-proyecto-b6565f206c96.json
cd Desarrollo-cloud-grupal
yes | sudo apt install virtualenv python3-virtualenv
sudo virtualenv -p python3 environment
source environment/bin/activate
yes | sudo python3 -m pip install -r requirements.txt
cd backend/flaskr
sudo flask run --host=0.0.0.0
```

El grupo de instancias se creó a partir de la plantilla anterior y se configuró para tener entre 1 a 3 instancias de VM, cuya regla para crear nuevas instancias era si el uso del CPU de la máquina virtual era mayor a 80%. Estos dos elementos fueron usados para crear el balanceador de cargas, que tenía un servicio de backend y otro de frontend. El servicio de frontend fue creado con una IPv4 que tenía disponible el puerto 80, por otra parte, el backend fue creado teniendo en cuenta el grupo de instancias anterior, apuntando hacia el puerto 5000 de las instancias de Compute Engine y el cual permite 50 solicitudes por segundo. Ahora bien, la instancia de Cloud SQL tiene una base de datos en Postgres de 10GB de almacenamiento y se configuró con ip privada para la conexión con las VM y Cloud Functions y con una ip pública para permitir su monitoreo por medio de pgadmin. El servicio de Cloud Storage se creó con almacenamiento estándar y se utilizó el comando "`gsutil notification create -f json -t tareas -p files/uploads/ -e OBJECT_FINALIZE gs://proyecto-dsc`" para publicar en el tema "tareas" de PubSub la adición de nuevos archivos en el bucket.

Finalmente, se configuró una función worker en Cloud Functions para que ejecutara en máximo 3 instancias el siguiente código a cargo de la compresión de los archivos: compresión de archivos

4. Escenarios de pruebas

Escenario 1 – Pruebas de Escalabilidad Parte A, es decir, pruebas de carga solo con los ajustes de escalabilidad a la capa web.

Descripción

Con el fin de realizar una carga web a la aplicación, anteriormente se definió un escenario específico para el registro de usuarios mediante el uso de un archivo CSV que contenía la información requerida para el registro de los usuarios en la aplicación. Esto genera una carga a la parte web específicamente a la funcionalidad de registro de usuarios.

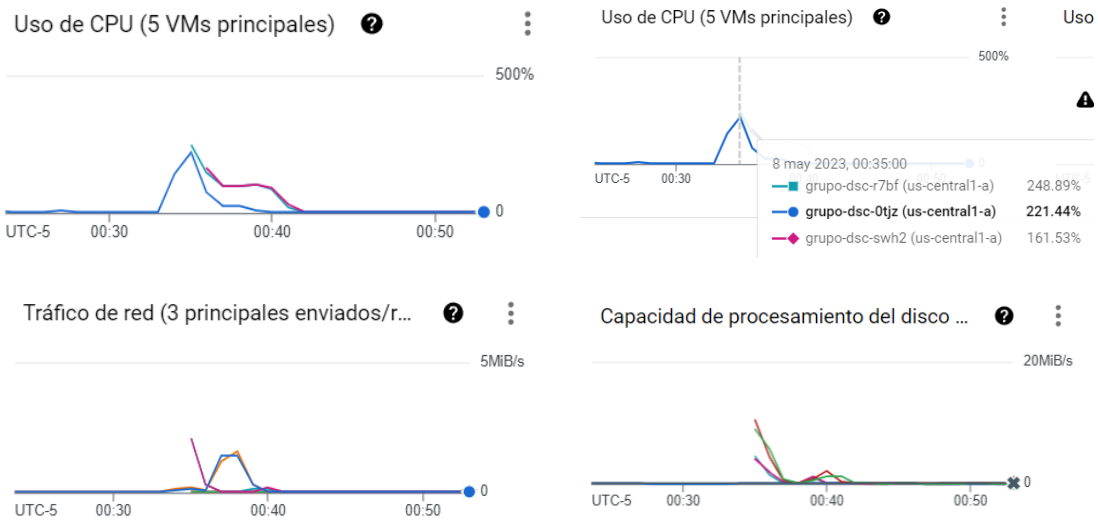
Configuración usada

Tras realizar varias pruebas con diferentes configuraciones de usuarios concurrentes y número de registros por realizar, se observó que la configuración deseada para ver la escalabilidad es la siguiente:

- Número de usuarios concurrentes: 15
- Número de solicitudes totales enviadas: 1000

Resultados

Al enviar la carga con la configuración mencionada, se observó que el sistema desplegó dos instancias adicionales para lograr procesar todas las solicitudes. El principal componente afectado fue las instancias de "Compute Engine" que contenían la funcionalidad web de la aplicación. A continuación, se puede observar el pico de procesamiento que tuvo el componente:



Las métricas obtenidas tras el procesamiento para el número de solicitudes descrito son:

Error %	Throughput	Received KB/sec	Sent KB/sec
0.04%	89.6/sec	48.22	34.05
0.00%	89.7/sec	61.93	0.00
0.02%	179.2/sec	110.08	34.05

Escenario 2 – Pruebas de Escalabilidad Parte B, es decir, pruebas de carga a la aplicación considerando la escalabilidad de la capa web y la capa de procesamiento asíncrono.

Descripción

Con el fin de realizar una carga tanto en la aplicación web como en el procesamiento asíncrono de los datos, anteriormente se definió un escenario específico para el inicio de sesión y envío de archivos por parte de los usuarios. Mediante el uso de un archivo CSV se especificó la información requerida para el inicio de sesión de los usuarios en la aplicación, así como el formato de compresión deseado y el archivo específico.

Configuración usada

Tras realizar varias pruebas con diferentes configuraciones de usuarios concurrentes y número de registros por realizar, se observó que la configuración deseada para ver la escalabilidad es la siguiente:

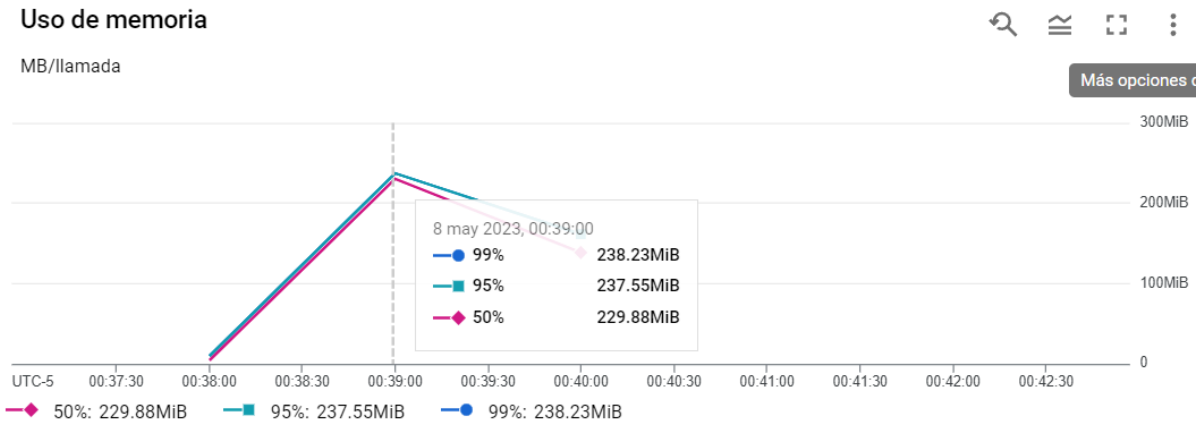
- Número de usuarios concurrentes: 10
- Número de solicitudes totales enviadas: 1000

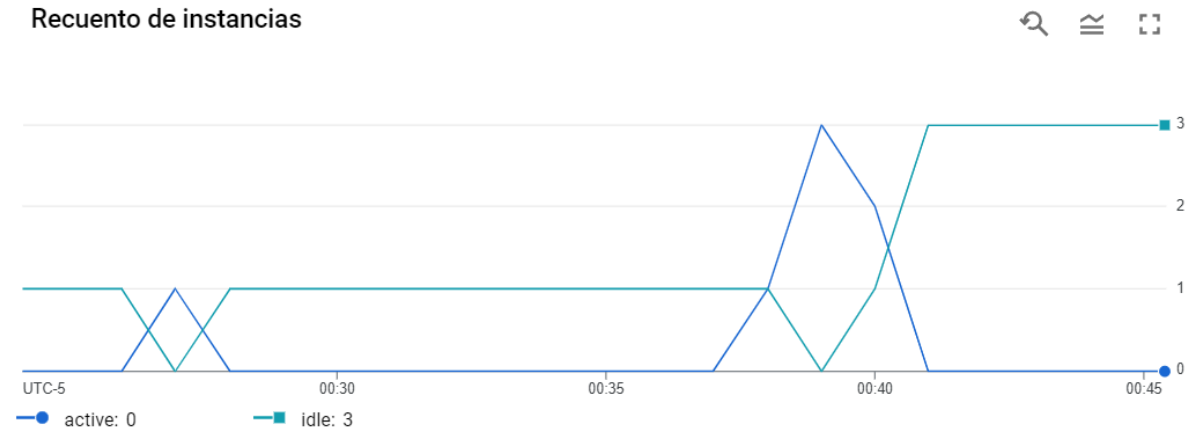
- Tamaño del archivo: 5MB

Resultados

El envío de solicitudes masivas para la subida del archivo implicó el uso del máximo número de instancias disponibles de acuerdo con la configuración para atender todas las solicitudes requeridas.

En las siguientes gráficas se puede observar el desempeño del componente de “Cloud Function” para las solicitudes enviadas durante la prueba:





El número de solicitudes atendidas por segundo fue mucho menor al obtenido anteriormente en las pruebas de la capa web. Esto tiene sentido teniendo en cuenta que la tarea de compresión de archivos es más demandante para la aplicación. Adicionalmente, es necesario resaltar que se contó con la configuración de 3 instancias únicamente para el proceso de compresión asíncrono, lo que limito el número de transacciones procesadas.

Min	Max	Std. Dev.	Error %	Throughput
93	384	66.12	0.00%	2.0/sec
87	362	60.62	0.00%	2.0/sec
928	7446	1713.01	27.78%	1.9/sec
0	1	0.19	0.00%	1.6/sec
0	7446	1443.68	7.46%	6.9/sec

5. Conclusiones

A partir del desarrollo de este proyecto se comprueba como el rendimiento de la aplicación mejora al separar la capa de procesamiento y almacenamiento, lo cual permite el uso de herramientas que se encargan de escalar las tareas de procesamiento y administración de la aplicación, y el uso de componentes transversales para la persistencia de los datos.

En la primera y segunda versión de la aplicación se pudo comprobar el funcionamiento del API, restringido a limitantes de capacidad fija de recursos. Con esta nueva versión de la aplicación, fue posible definir un rango de instancias (1 a 3) para responder a crecimientos repentinos en el uso de los servicios ofrecidos sin caer en problemas de sobreaprovisionamiento. No obstante, consideramos que las máquinas elegidas (N1-F1) aún son pequeñas para responder a múltiples solicitudes concurrentes de compresión de archivos sin impactar el rendimiento.

Para finalizar, consideramos que la implementación de soluciones en la nube plantea retos como elegir la herramienta que mejor se ajusta a las necesidades de un proyecto, en este caso consideramos diversas opciones como Compute Engine, Cloud Run, Cloud Functions para el despliegue del worker y optamos por la última herramienta buscando facilidad de desarrollo y disminución de administración de la infraestructura. También plantea retos como saber implementar la manera más eficiente para conectar los componentes de la arquitectura sin disminuir los niveles de seguridad de la aplicación.