

Entrega 3 – Sistema de Conversión Cloud

Escalabilidad en la Capa Web

Grupo 20

Juan P Hernández

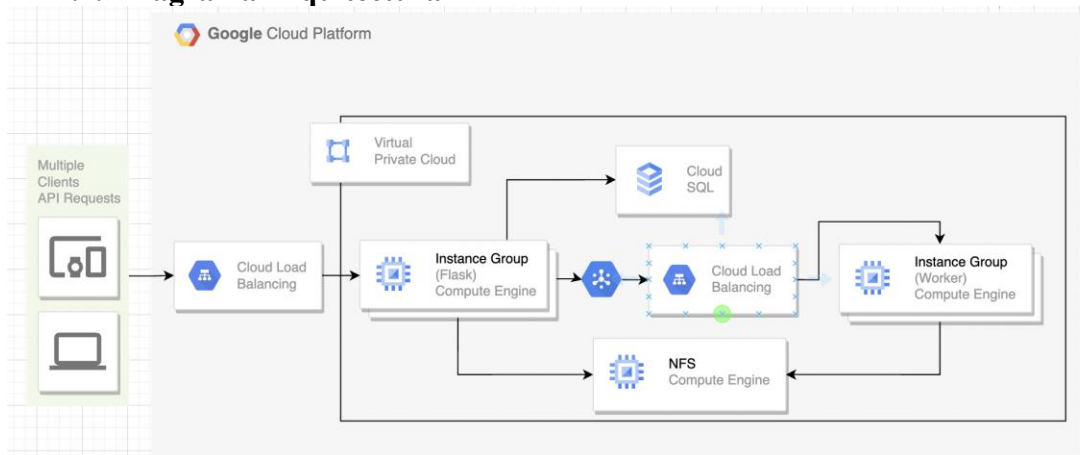
Juan D Diaz

Juan F Castaño

Kevin H Castrillón

1. Arquitectura de la aplicación.

1.1. Diagrama Arquitectura



1.2. Infraestructura

Para la infraestructura se utilizó las siguientes soluciones tecnológicas:

Máquina virtual: se configuraron instancias de tipo N1 con los requerimientos de cómputo y almacenamiento definidos, es decir, 614 MiB de RAM, 1vCPU y 10GB de RAM. El sistema operativo instalado en las mismas es Ubuntu 20.04. Tienen unas etiquetas de red para que se pudiese proceder a comunicarse entre ellas, y además que aceptaran conexiones a través de internet.

Instancia SQL: se configuró una instancia de SQL en la cual corrimos PostgreSQL 14, esta instancia se configuró en modo de desarrollo, con 1 núcleo y 1GB de ram, y 10 gb de almacenamiento interno.

Red VPC: se aprovechó la red VPC configurada por defecto en un proyecto de Google Cloud. Se configuraron reglas de firewall respectivas para poder acceder a la máquina virtual por TCP en el puerto 5000. Se aprovechó el puerto 3389 que ya está configurado por defecto para acceder por TCP.

Cloud Storage: En este caso para almacenar todos los archivos originales como también procesados.

Cloud Monitoring: Se configuró Google Cloud Monitoring para supervisar el estado y rendimiento de las instancias de la máquina virtual y la instancia de SQL.

Autoscalling: Se implementó el escalado automático utilizando Google Cloud Autoscaling. Se estableció una regla basada en la carga de CPU de la instancia de la máquina virtual para aumentar o disminuir automáticamente la cantidad de instancias en ejecución.

Load Balancer: Se implementaron dos Google Cloud Load Balancer para distribuir el tráfico entre las diferentes instancias de la máquina virtual. Esto permitió una mayor disponibilidad y escalabilidad de la aplicación. Se implementó un balanceador de carga para procesar las peticiones

Para el ambiente de software se usaron las siguientes tecnologías:

A continuación, se listan las tres principales herramientas tecnológicas usadas en la máquina virtual para el desarrollo de la entrega 3:

Flask: framework web minimalista para Python. Es una herramienta popular utilizada para construir aplicaciones web pequeñas y medianas que son rápidas y eficientes.

Celery: herramienta de Python que permite la programación de tareas en segundo plano y la ejecución de trabajos de forma asíncrona y distribuida. Se utiliza ampliamente en aplicaciones web para procesar tareas que pueden ser realizadas fuera del ciclo de solicitud-respuesta.

Redis: sistema de almacenamiento en memoria de código abierto, rápido y eficiente que se utiliza como base de datos, caché y agente de mensajes. Fue diseñado para manejar grandes cantidades de datos y proporcionar un alto rendimiento y disponibilidad.

Paquetes de Python

```
flask==2.2.3
flask-SQLAlchemy==3.0.3
flask-RESTful==0.3.9
flask-marshmallow==0.15.0
marshmallow_sqlalchemy==0.29.0
marshmallow==3.19.0
flask-jwt-extended==4.4.4
flask-cors==3.0.10
werkzeug==2.2.3
celery==5.2.7
redis==4.5.4
gunicorn==20.1.0
psycpg2-binary==2.9.2
pydump==0.25.1
pylzma==0.5.0
```

2. Análisis de capacidad:

2.1 Entorno de pruebas.

Máquina virtual: para esta entrega se configuraron tres instancias de tipo N1 con los requerimientos de cómputo y almacenamiento definidos, es decir, 1 GB de RAM, 1vCPU y 20GB de Almacenamiento interno. El sistema operativo instalado en las mismas es Ubuntu

20.04. Se le agregó una etiqueta de red para que las reglas necesarias de firewall para acceder desde internet fueran posibles.

Instancia SQL: se configuró una instancia de SQL en la cual corrimos PostgreSQL 14, esta instancia se configuró en modo de desarrollo, con 2 núcleos y 1GB de ram, y 10 gb de almacenamiento interno.

Red VPC: se aprovechó la red VPC configurada por defecto en un proyecto de Google Cloud. Se configuraron reglas de firewall respectivas para poder acceder a la máquina virtual por TCP en el puerto 5000. Se aprovechó el puerto 3389 que ya está configurado por defecto para acceder por TCP.

Load Balancer: Se implementó un Round Robin como Load Balancer para distribuir el tráfico entre las diferentes instancias de la máquina virtual.

Autoscaling: Implementamos el uso de autoscaling de manera que cuando exista un 70% de uso en una máquina, se realice la creación automática de otra instancia con las mismas características. También se estableció un límite máximo de 5 instancias. Se configuró una política de eliminación de instancias para que las instancias no utilizadas se eliminen automáticamente después de un tiempo determinado.

2.2 Criterios de aceptación

Los criterios de aceptación para las pruebas de estrés incluyen los siguientes objetivos y limitaciones:

Tiempo de respuesta: se espera que la aplicación responda a las solicitudes de los usuarios en un tiempo razonable, sin generar tiempos de espera excesivos. El objetivo es que el tiempo de respuesta promedio sea inferior a 3 segundos, con un límite máximo de 5 segundos.

Rendimiento: se espera que la aplicación pueda manejar una carga de trabajo adecuada, sin experimentar una disminución significativa en el rendimiento. El objetivo es que la aplicación pueda manejar al menos 500 usuarios concurrentes sin afectar negativamente el rendimiento gracias al uso de los servicios de auto escalabilidad de GCP configurados por nosotros.

Utilización de recursos: se espera que la aplicación utilice los recursos del sistema de manera eficiente, sin generar cuellos de botella en el procesamiento o en la utilización de memoria. El objetivo es que la utilización de recursos se mantenga por debajo del 80% en todo momento gracias a la configuración de los load balancers.

Además de estos objetivos y limitaciones, los criterios de éxito del proyecto también incluyen la identificación de los cuellos de botella en la aplicación y su infraestructura, y la determinación de las configuraciones óptimas para maximizar el rendimiento.

2.3 Escenarios de prueba

Escenarios de prueba:

1. Conversión de (1, 10, 100) archivos pequeños a ZIP.
2. Conversión de (1, 10, 100) archivos grandes a ZIP.
3. Conversión de (1, 10, 100) archivos pequeños a 7Z.
4. Conversión de (1, 10, 100) archivos grandes a 7Z.
5. Conversión de (1, 10, 100) archivos pequeños a TAR.GZ.
6. Conversión de (1, 10, 100) archivos grandes a TAR.GZ.
7. Conversión de (1, 10, 100) archivos pequeños a TAR.BZ2.
8. Conversión de (1, 10, 100) archivos grandes a TAR.BZ2.
9. Obtención de (1, 10, 100) archivos grandes.
10. Obtención de (1, 10, 100) archivos pequeños.
11. Autenticación de (1, 10, 100) usuarios concurrentes.
12. Crear (1, 10, 100) usuarios.
13. Obtener (1, 10, 100) tareas de conversión de forma simultánea.
14. Eliminar una tarea.
15. Cambiar el formato de una tarea (1, 10, 100) de forma simultanea

Escenarios clave:

- Comprobar que la aplicación puede manejar diferentes tipos de archivos y tamaños.
- Verificar que la conversión de los archivos se realiza correctamente a los diferentes formatos.
- Validar que la aplicación no pierda información o datos durante el proceso de compresión.
- Asegurar que la aplicación mantiene la estructura de carpetas durante la compresión.
- Asegurar que un usuario pueda crear una cuenta.
- Asegurar que un usuario pueda hacer log-in de forma correcta.
- Permitir que se pueda comprobar que las tareas se crean y se suben en el sistema.

Variabilidad entre servicios representativos:

Se puede simular la variabilidad en los servicios representativos de la aplicación mediante la generación de diferentes tipos de archivos de diferentes tamaños y formatos.

Datos de prueba:

- Archivos pequeños y grandes de diferentes formatos.
- Varios archivos para comprimir.
- Si el archivo comprimido se puede abrir correctamente.
- Si la estructura de carpetas se mantiene durante la compresión.
- Credenciales de acceso.
- Datos en texto plano correspondiente a formularios.
- Información de una tarea de conversión.

2.4 Parámetros de configuración

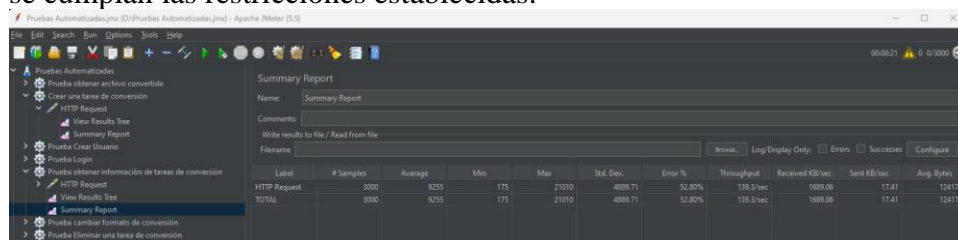
Los parámetros para las peticiones GET, PUT y DELETE se configuran inicialmente en 100 threads y van creciendo o decreciendo hasta el punto de éxito de éxito o fallo del escenario, además se autoriza a través de jwt.

Los parámetros para el POST se configuran inicialmente en 500 threads y van decreciendo hasta el punto de éxito de éxito y fallo del escenario, además se autoriza a través de jwt.

2.5 Escenario 1

Obtener un archivo convertido (GET).

Se prueba obtener un archivo procesado, la restricción es la siguiente (En las pruebas de estrés el tiempo de respuesta promedio de la aplicación debe ser de máximo 1.500 ms, si este tiempo no se cumple, se concluye que el sistema NO soporta la cantidad de requests de la prueba. En caso de que durante una prueba se generen más de un 1% de errores en los requests de la prueba, se concluye que la aplicación NO soporta la cantidad de requests de la prueba.), se desea saber cuántas conexiones máximas se pueden tener en simultaneo, tal que se cumplan las restricciones establecidas.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	3000	9255	175	21010	4889.71	52.80%	139.3/sec	1699.06	17.41	12417.4
TOTAL	3000	9255	175	21010	4889.71	52.80%	139.3/sec	1699.06	17.41	12417.4

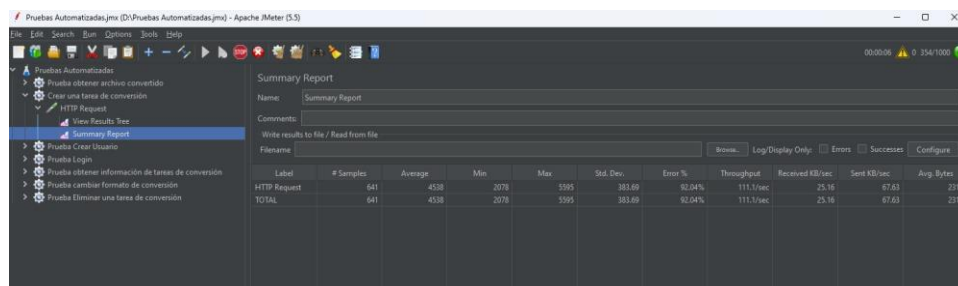
Ensayo con 3000 hilos: Se evidencia el anterior resultado el cual tiene un % de error del 52.8

Análisis: este escenario realiza muchas peticiones y al alcanzarse el límite de la métrica definida se ajusta el tamaño del grupo de instancias por medio de autoscaling. Mientras se ejecuta este proceso de escalamiento se pierden peticiones, tras lo cual se vuelven a retomar. También es importante tener en cuenta que la base de datos actualmente está desplegada con las configuraciones más básicas lo cual representa un cuello de botella también.

2.6 Escenario 2

Crear una tarea (POST).

Se prueba enviar y procesar un archivo, se busca saber cuántos archivos máximo se pueden procesar en un tiempo menos a 600 segundos y con 0% de error que nos indica que se procese correctamente



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	641	4538	2078	5595	383.69	92.04%	111.1/sec	25.16	67.63	231.9
TOTAL	641	4538	2078	5595	383.69	92.04%	111.1/sec	25.16	67.63	231.9

Ensayo con 641 hilos: Se evidencia que en un primer momento este escenario genera un porcentaje de 92% de error, pero al final de la ejecución se evidencia un porcentaje de error menor.

3. Documentación Postman

En el siguiente link se encuentra la documentación de las pruebas realizadas en Postman.

<https://documenter.getpostman.com/view/19568698/2s93Y5PKYM>