

## Entrega 4 – Escalabilidad

# DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB ESCALABLE EN NUBE PÚBLICA

Paul A. Calvache Tapia, Fabian O. Ramírez, David A. Vásquez Pachón, Mateo Zapata López

Universidad de los Andes, Bogotá, Colombia

{pa.calvache, fo.ramirez50, da.vasquez11, m.zapatal2}@uniandes.edu.co

Fecha de presentación: mayo 07 de 2023

### Contents

Arquitectura .....	2
1. Balanceo de cargas.....	2
2. Auto escalamiento.....	2
3. Almacenar archivos en el bucket de Cloud storage .....	3
4. Script inicio Automático .....	4
5. Capa Worker tareas asíncronas .....	4
Escalamiento de los workers mediante el uso de Cloud Functions .....	6
1. Pub sub.....	7
2. Alta disponibilidad.....	8
Escenario 1 .....	8
Escalabilidad de la parte A .....	¡Error! Marcador no definido.
Escenario 2:.....	14
Escalabilidad de la parte B.....	8

# PARTE A

## MODELO DE DESPLIEGUE - ESCALABILIDAD EN LA CAPA WEB

### Arquitectura

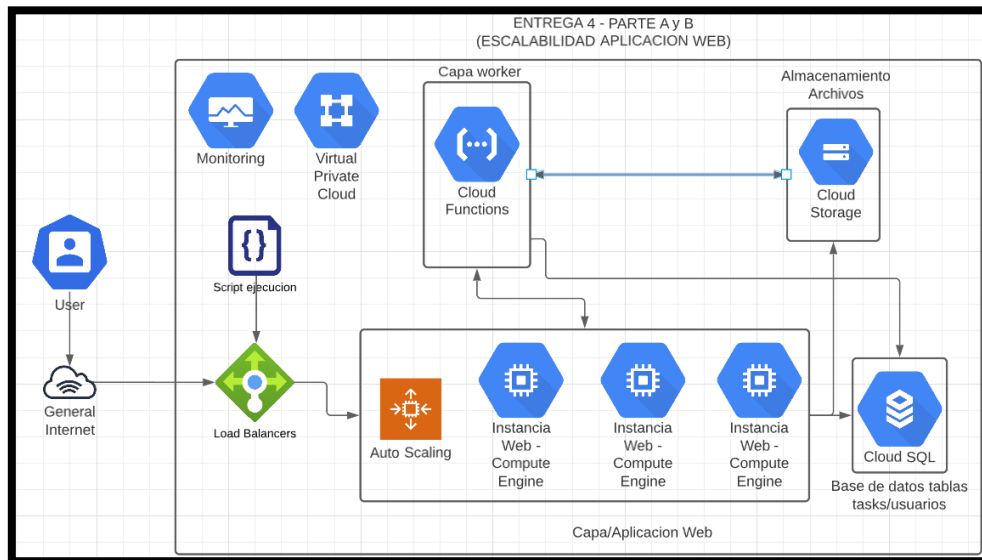


Ilustración 1 Arquitectura de la parte A

1. **Balanceo de cargas:** el balanceador de cargas es una característica de la infraestructura en la nube que permite dividir el trabajo de las aplicaciones en diferentes instancias con lo cual evita que se colapsen las aplicaciones que se despliegan en una sola máquina virtual, se configura con el grupo de instancias, el objetivo de escalación es 60% y el modo de balanceo es de 80% de las RPS máximas: 10 (por instancia). El balanceador de cargas lo nombramos “web-serverlb”.

La configuración del backend es la siguiente:

Backend					
Servicios de backend					
1. backendsrv					
Protocolo de extremo	Puerto con nombre	Tiempo de espera	Verificación de estado	Cloud CDN	Registros
HTTP	http	30 segundos segundos	<a href="#">healthcheck1</a>	Inhabilitada	Inhabilitada
Afinidad de sesión	Tiempo de espera para el vaciado de conexiones	Política de seguridad del backend	Política de seguridad de Edge	<a href="#">Identity-Aware Proxy</a>	
Ninguno	300 segundos	Ninguna	Ninguna	Inhabilitada	

Ilustración 2, configuración balanceo de cargas

También se configuró, una verificación de estado “healthcheck1” sencilla con un intervalo de tiempo de espera de 5 seg, y umbrales de buen y mal estado de 2 seg.

2. **Auto escalamiento:** Condiciones de auto escalamiento y periodo de enfriamiento:

La política de auto escalamiento se definió en la capa web para crear máquinas virtuales iguales a la desplegada por el script, para las cuales se consideró que se crearan cuando la VM tuviera más de un 60% de uso de la CPU para empezar a desplegar la siguiente VM, se tiene que se debe auto escalar hasta un máximo de 3 VM, y se puso una configuración de escalamiento hacia abajo cuando después de 400 seg las CPU de las maquinas estén por debajo de 60%. Se debe aclarar que las VM están asociadas a un grupo de instancias con las cuales se pueden desplegar en una misma zona establecida y que a su vez la instancia de grupo se asoció a una plantilla para tener un script para desplegar la aplicación automáticamente.

Location	
Zone	us-central1-a
Autoscaling	
Autoscaling mode	On
Minimum # of instances	1
Maximum # of instances	3
Initialization period	400 seconds
Autoscaling signal	
CPU utilization	60%
Predictive autoscaling	Off
Scale in controls	Disabled
Scaling schedules	<a href="#">MANAGE SCHEDULES</a>

Ilustración 3, configuración autoscaling de la capa web

3. Almacenar archivos en el bucket de Cloud storage: Este es un servicio de almacenamiento de objetos en la nube, con él se puede crear y configurar un bucket el cual se encarga de hospedar los objetos, se creó un bucket “cloudentrega4” para almacenar los archivos originales subidos por cada usuario y los archivos comprimidos.

















cloudentrega4							
Ubicación	Clase de almacenamiento	Acceso público	Protección				
us-central1 (Iowa)	Standard	No público	Ninguno				
<a href="#">OBJETOS</a>	<a href="#">CONFIGURACIÓN</a>	<a href="#">PERMISOS</a>	<a href="#">PROTECCIÓN</a>	<a href="#">CICLO DE VIDA</a>	<a href="#">OBSERVABILIDAD</a>	<a href="#">NUEVO</a>	<a href="#">INFORMES DE INVENTARIO</a>
Depósitos > cloudentrega4							
<a href="#">SUBIR ARCHIVOS</a>	<a href="#">SUBIR CARPETA</a>	<a href="#">CREAR CARPETA</a>	<a href="#">TRANSFERIR LOS DATOS</a>	<a href="#">ADMINISTRAR CONSERVACIONES</a>	<a href="#">DESCARGAR</a>	<a href="#">BORRAR</a>	
Filtrar solo por prefijo de nombre <input type="text"/> Filtro Filtrar objetos y carpetas <input type="text"/>							
<input type="checkbox"/>	Nombre	Tamaño	Tipo	Fecha de creación	Clase de almacenamiento	Última modificación	Acc
<input type="checkbox"/>	 4x4 cruz amarilla.png	275.6 KB	image/png	7 may 2023 13:19:17	Standard	7 may 2023 13:19:17	No  
<input type="checkbox"/>	 OtunPanas.jpeg	200.1 KB	image/jpeg	7 may 2023 14:49:55	Standard	7 may 2023 14:49:55	No  
<input type="checkbox"/>	 OtunPanas.jpeg.zip	200.2 KB	application/zip	7 may 2023 16:25:36	Standard	7 may 2023 16:25:36	No  
<input type="checkbox"/>	 aaaa.png	94.9 KB	image/png	7 may 2023 12:17:15	Standard	7 may 2023 12:17:15	No  
<input type="checkbox"/>	 archivos/	—	Carpeta	—	—	—	— 
<input type="checkbox"/>	 archivosComprimidos/	—	Carpeta	—	—	—	— 

Ilustración 4, bucket cloud storage

Imagen 1. Bucket en el Cloud Storage.

4. **Script inicio Automático** El Script para desplegar la aplicación de manera automática es una configuración importante a realizar en la plantilla del grupo de instancias, ya que permite correr los comandos que se establezcan apenas se crea una VM se realizó con una plantilla que corre todo el script de inicio que necesitamos que se copie en todas las instancias a las que se va a escalar, las instalaciones de librerías, la clonación del repositorio de Github y también la puesta en marcha de Flask. También se decidieron las máquinas como N1 con el tamaño standard pues la instalación de Github es bastante exigente en CPU, por esto se utilizó una instancia con mayor capacidad.

Metadatos personalizados	
Clave	Valor
startup-script	<pre> sudo apt update &amp;&amp; sudo apt -y install git sudo apt install python3-pip -y git clone https://github.com/mzapatal2/Cloud_Entrega1.git sudo apt-get install python3-venv -y python3 -m venv venv source venv/bin/activate cd Cloud_Entrega1 sudo pip install -r requirements.txt export FLASK_APP=apirest flask run --host=0.0.0.0 </pre>

*Imagen 2. Script para desplegar la capa web de manera automática.*

Como se puede apreciar en la imagen 5 los comandos que realiza el script es la instalación de git para poder clonar el repositorio, la instalación de Python 3 para poder crear el ambiente y activarlo, la clonación del repositorio, la instalación de los requerimientos para la aplicación y correr la aplicación con su variable de ambiente.

#### 5. **Capa Worker tareas asíncronas:**

La capa worker, se realizó con un servicio llamado Cloud Functions el cual permite implementar secciones de código específicas a realizar cuando existan peticiones HTTP, o que existan un trigger que accione la tarea, entre otras. En nuestro caso fue configurado para comprimir los archivos que se almacenan en el Cloud Storage (en el bucket del proyecto) y realice una actualización de los registros que tengan estado “uploaded” a “processed”.

La función de nube tiene un escuchador que está pendiente de los eventos registrados en le storage, en el momento que se genera un evento de archivar o crear un elemento, empieza a ejecutarse el cloud, que se encarga de revisar que ese archivo nuevo no sea de exención comprimida. Para luego comprimirlo y enviarlos a la carpeta correspondiente. Esta función se ejecuta de manera automática junto con los cambios de estado en la base de datos.



*Imagen 3. Función para comprimir archivos en Cloud Function.*

```

1  import os
2  import zipfile
3  import psycopg2
4  from google.cloud import storage
5
6
7  def process_new_file(event, context):
8      # Configuración de la conexión de PostgreSQL
9      db_name = "libros"
10     db_user = "postgres"
11     db_password = "libros"
12     db_host = "34.29.121.22"
13     db_port = "5432"
14     idUsuario = "uploaded"
15
16     # Conexión a la base de datos PostgreSQL y ejecución de consulta
17     connection = psycopg2.connect(
18         dbname=db_name, user=db_user, password=db_password, host=db_host, port=db_port
19     )
20     with connection.cursor() as cursor:
21         cursor.execute("""UPDATE public.task SET
22                             status =%s
23                             WHERE status = %s """, ("process", "uploaded",))
24         affected_rows=cursor.rowcount
25         connection.commit()
26     connection.close()
27

```

*Imagen 4. Función para comprimir archivos y actualizar los registros en la base de datos.*

La función es de primera generación y se creó en el lenguaje de programación Python 3.7, esta debe estar asociada a un bucket para poder consultar los archivos almacenados allí y comprimirlos y dejarlos en el mismo bucket.

Adicional a esta para poder llevar a cabo las ejecuciones de funciones de Nube, se crearon servicios accounts que les permitieran a las funciones tener los permisos de escribir sobre el storage y las instancias SQL que contienen los estados de las compresiones de los archivos.

## PARTE B

### MODELO DE DESPLIEGUE - ESCALABILIDAD EN EL BACKEND

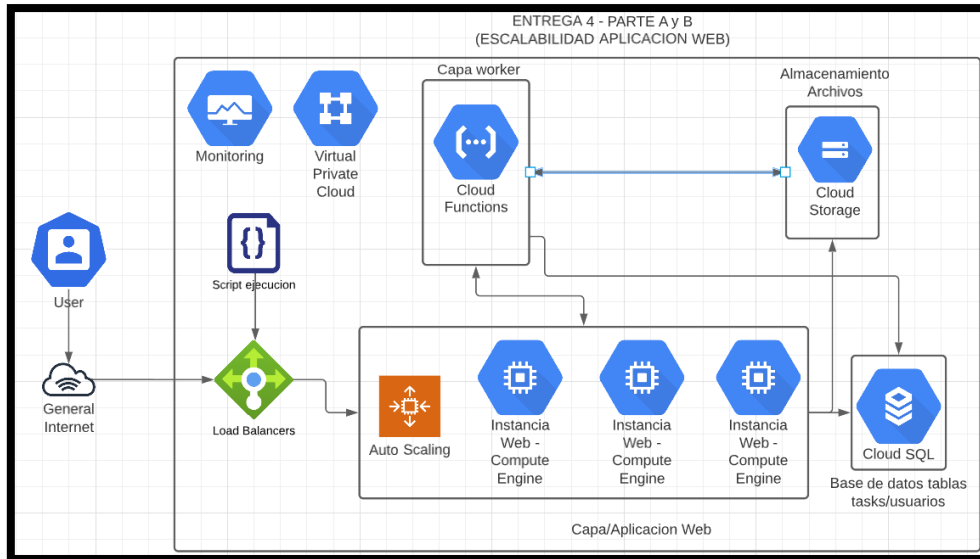


Ilustración 5, Arquitectura para la aplicación B

#### Escalamiento de los workers mediante el uso de Cloud Functions

El escalamiento de la capa worker se realiza mediante la configuración de la función “storage” creada para realizar la compresión y actualización de los archivos ubicados en el Bucket “cloudentrega4” de Cloud Storage y la actualización del campo “status” de los registros de la base de datos.

General Information	
Last deployed	May 7, 2023 at 4:21:32 PM GMT-5
Region	us-central1
Memory allocated	256 MB
Timeout	60 seconds
Minimum instances	0
Maximum instances	3000
Service account	<a href="mailto:storagetriguer@entrega4a.iam.gserviceaccount.com">storagetriguer@entrega4a.iam.gserviceaccount.com</a>
Build Worker Pools	—
Container build log	<a href="#">36c4bd73-d35c-49df-b020-4bd69f099095</a>

Ilustración 6, configuración del autoscaling

Como se visualiza en la imagen 7 el máximo número de instancias a desplegar cuando hay un alto número de peticiones es de 3000 y el parámetro timeout es de 60 seg que indica en qué momento empieza a realizar un escalado hacia abajo.

1. Pub sub: como se ve en la gráfica, en nuestra arquitectura no se utilizó la herramienta pub-sub, esto tiene su justificación pues la herramienta no se necesita por 2 razones concretar:

- a. La alta disponibilidad de la herramienta Cloud Functions (serverless) se escala de manera automática, con esta hacemos todo el proceso de compresión, y el trigger de este proceso es la subida de algún archivo al Cloud Storage, esto nos ahorra el proceso de colas y nos da la alta disponibilidad, necesaria para la aplicación.
- b. El proceso de comunicación del Cloud Function, se realiza mediante una función que cambia el estado de la tabla del SQL de “loaded” a “proccessed”, por esto, con esta arquitectura, se ve innecesario utilizar esta herramienta en cuanto a el proceso de comunicación.

```
storage_client = storage.Client()
bucket_name = "cloudentrega4"

def push_to_gcs(file,bucket):
    """ Writes to Google Cloud Storage. """
    file_name = file.split('/')[-1]
    print(f"Pushing {file_name} to GCS...")
    blob = bucket.blob(file_name)
    blob.upload_from_filename(file)
    print(f"File pushed to {blob.id} succesfully.")

root = os.path.dirname(os.path.abspath(__file__))
def zip_file(event, context):
    """Cloud Function triggered by Cloud Storage changes.
    Args:
        event (dict): The Cloud Storage event payload.
        context (google.cloud.functions.Context): Metadata of triggering event.
    """
    print(event)
    print(context)
    file_name = event['name']
    file_extension = os.path.splitext(file_name)[1]

    # Skip files that are already a ZIP archive
    if file_extension == '.zip':
        print(f'File {file_name} is already a ZIP archive. Skipping compression.')
        return

    bucket = storage_client.bucket(bucket_name)
    blob = bucket.get_blob(file_name)
    blobstr = blob.download_as_string()
    print(blob.path)

    zipped_files_path = f'/tmp/{file_name}.zip'
    with ZipFile(os.path.join(root, zipped_files_path), 'w') as myzip:
        myzip.writestr(file_name,blobstr)
    push_to_gcs(os.path.join(root, zipped_files_path),bucket)

    # Create a temporary file to hold the compressed data
```

*Ilustración 7, Cloud función para ejecutar la compresión de los archivos*

2. Alta disponibilidad, despliegue de la aplicación de manera Multi-Region.  
Para esta parte es necesario configurar el Load Balancer como multi-region y también realizar la configuración de la VPC como multi-region.

## Análisis de capacidad

Escalabilidad de la aplicación:

Escenario 1:

Para el escenario de prueba 1 se realizaron dos etapas la primera con la creación de 100 tareas en 10 segundos lo que no causó estrés a la aplicación por lo cual el performance fue el adecuado.

Como se muestra a continuación en las imágenes donde visualizamos que la totalidad de peticiones se completan satisfactoriamente.

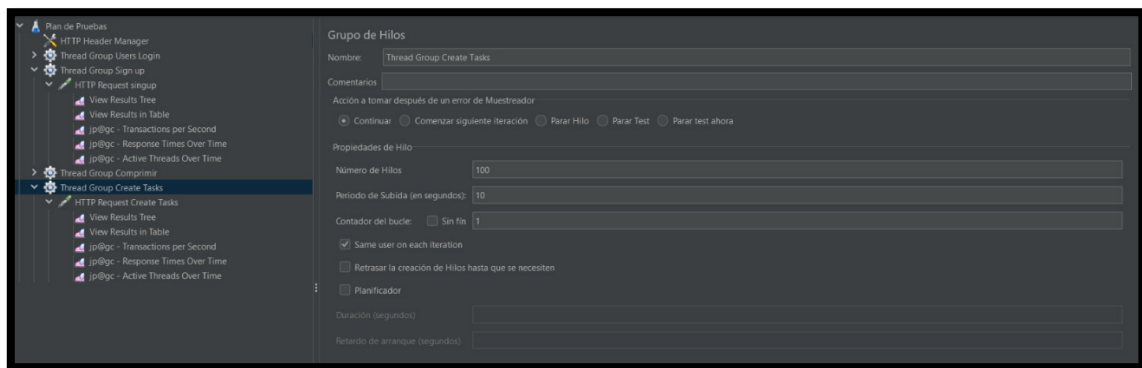


Ilustración 8. Escenario de prueba creación de 100 tareas en 10 segundos configuración.

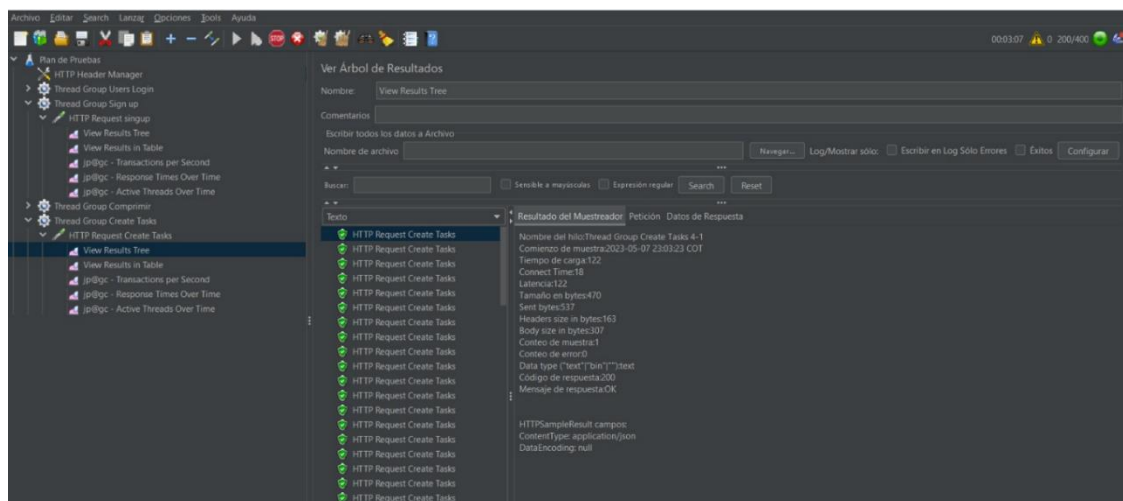


Ilustración 9. Completitud de las 100 peticiones en estado ok.



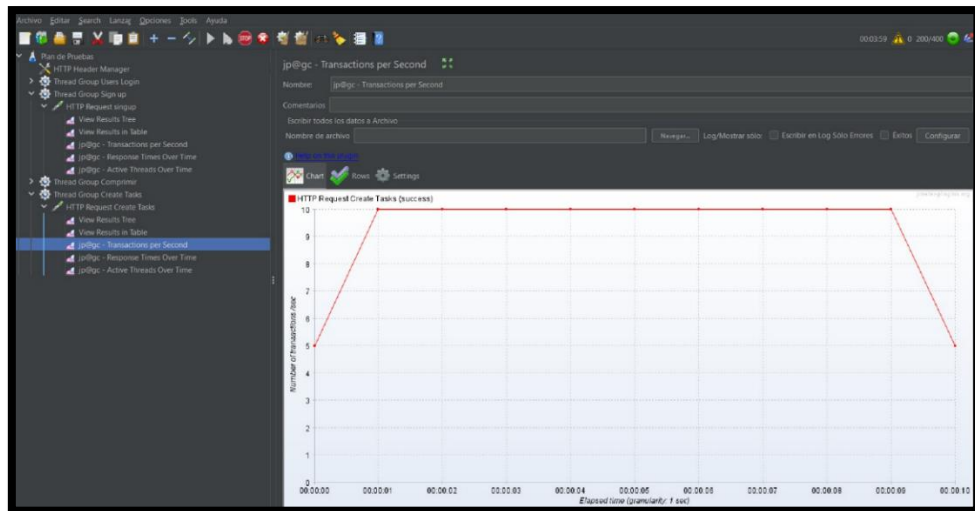


Ilustración 10. Grafico que muestra las transacciones por segundo.

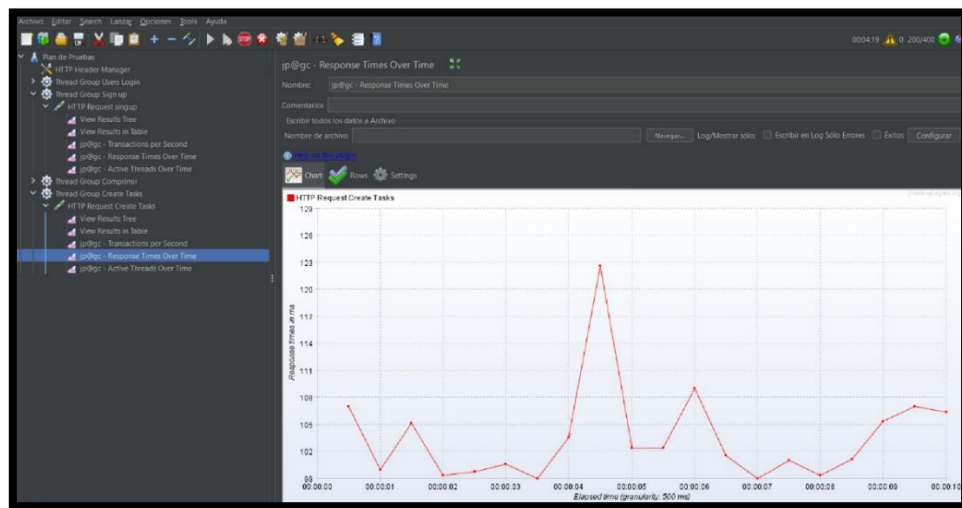


Ilustración 11. Tiempo de respuesta en el tiempo para escenario 1 etapa 1.

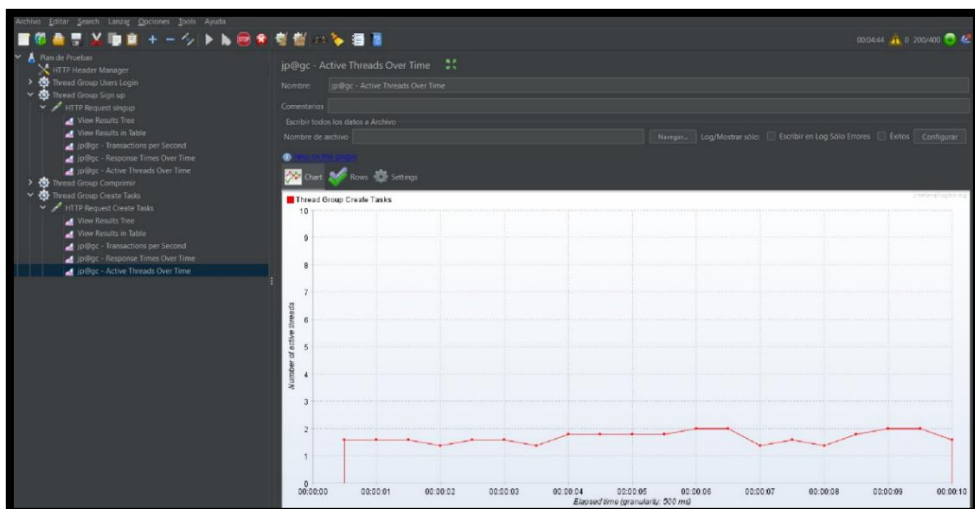


Ilustración 12. Hilos realizados en el tiempo. (peticiones).

La segunda etapa fue crear 1 millón de peticiones en 10 segundos lo cual estreso a la maquina hasta llegar a utilizar el 100 por ciento del CPU de la primera VM.



Ilustración 13. Etapa 2 escenario 1, 1 millón de peticiones.

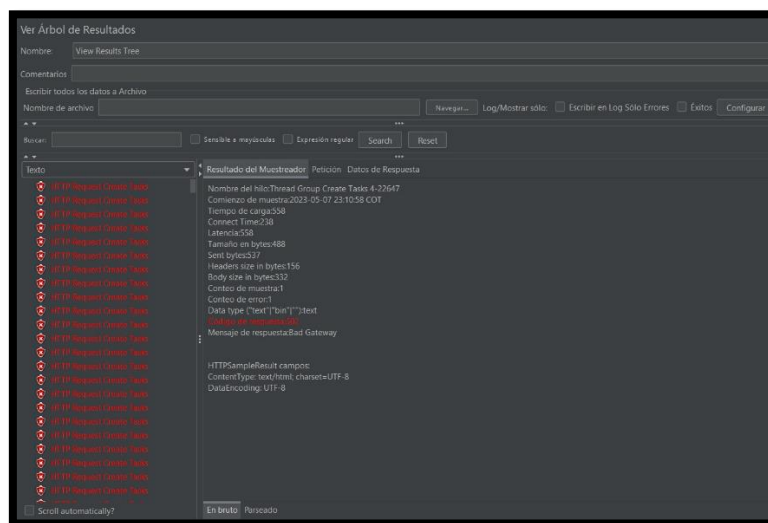


Ilustración 14. Fallos en las peticiones.

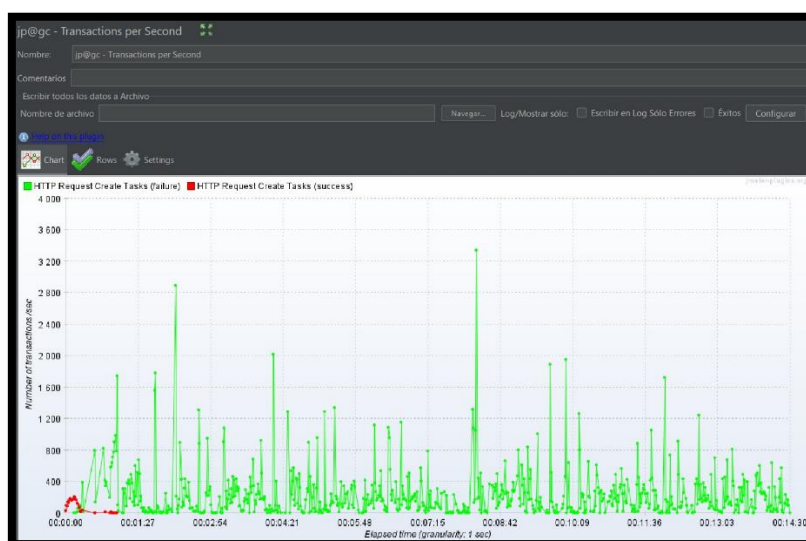


Ilustración 15. Colapso de las transacciones por segundo de la capa web.



*Ilustración 16. Demora de las respuestas a las peticiones.*



*Ilustración 17. Hilos activos sobre el tiempo, demora en la respuesta.*

Se activo la segunda Maquina, pero como no es una máquina de grandes características y el script toma un tiempo de aproximadamente 5 minutos en que instale todas las dependencias. Se observa una caída en el uso del CPU en ambas maquinas.

Instancias de VM

CREAR INSTANCIA

IMPORTAR VM

ACTUALIZAR

APRENDIZAJE

INSTANCIAS

OBSERVABILIDAD

PROGRAMAS DE LAS INSTANCIAS

Instancias de VM

Filtro

Ingresar el nombre o el valor de la propiedad

?

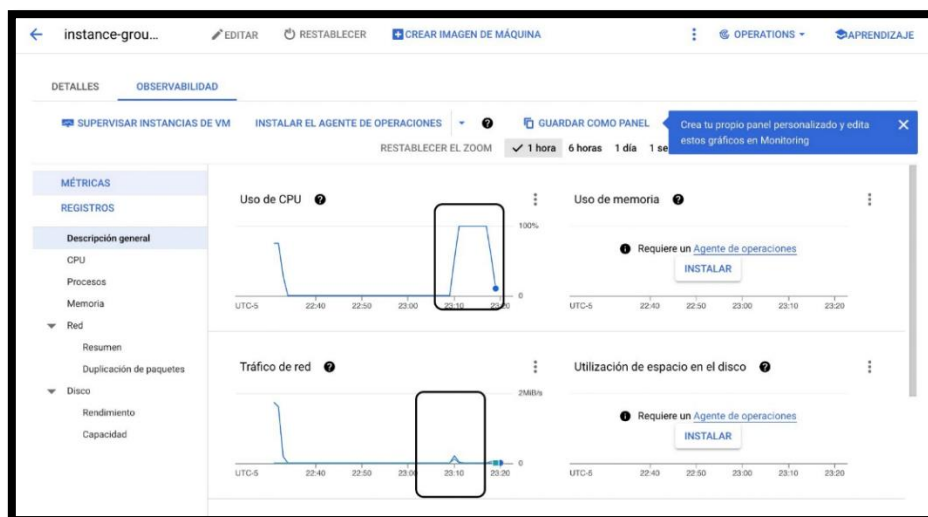
III

<input type="checkbox"/>	Estado	Nombre ↑	Zona	Conectar
<input type="checkbox"/>	✓	<a href="#">instance-group-2-xcgv</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-3-r76g</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-4-32ws</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-5-6fmx</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-6-drl8</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-7-v7jj</a>	us-central1-a	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-8-06w6</a>	us-central1-c	SSH ▾ ⋮
<input type="checkbox"/>	✓	<a href="#">instance-group-8-zvbb</a>	us-central1-f	SSH ▾ ⋮

Acciones relacionadas

HIDE

Ilustración 18. Auto scaling, creación de las instancias.



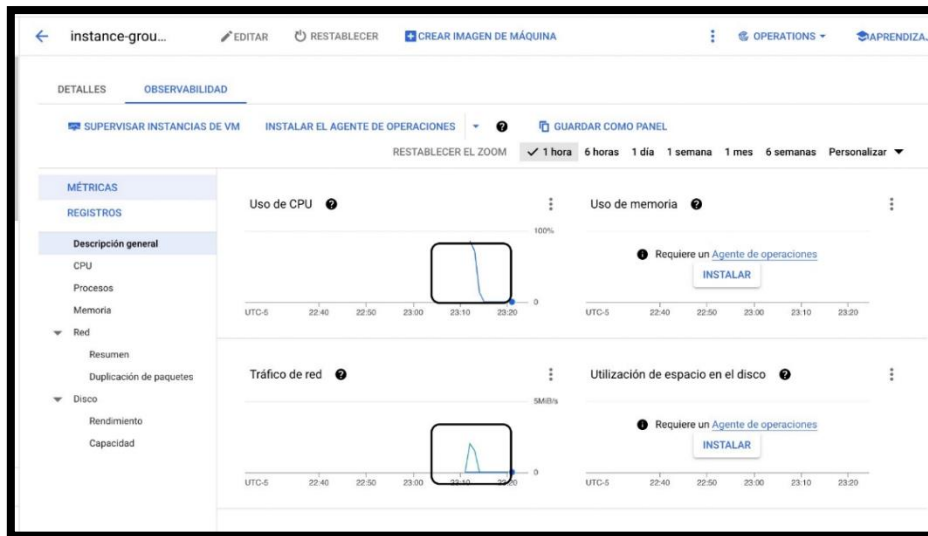


Ilustración 20. Creación segunda instancia capa web, desplome uso de CPU.

A nivel de Base de datos no se observa ninguna carga importante excepto en transacciones por segundo

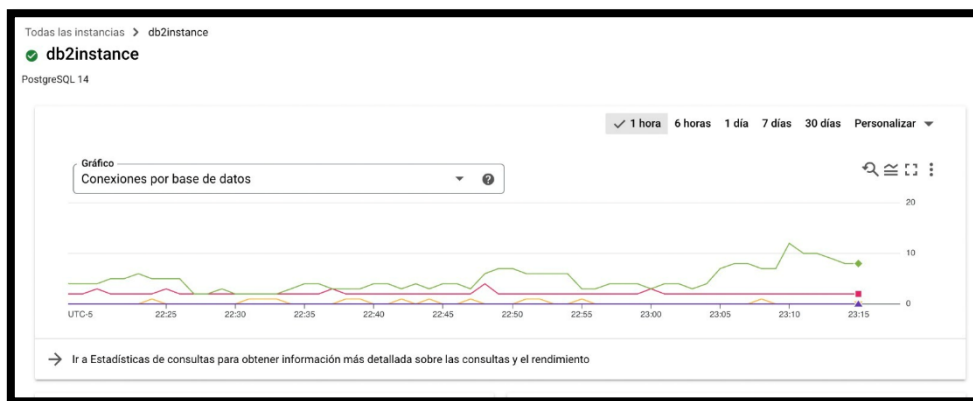


Ilustración 21. Conexiones realizadas a la base de datos.



Ilustración 22. Uso de CPU de la base de datos, alrededor del 5%.

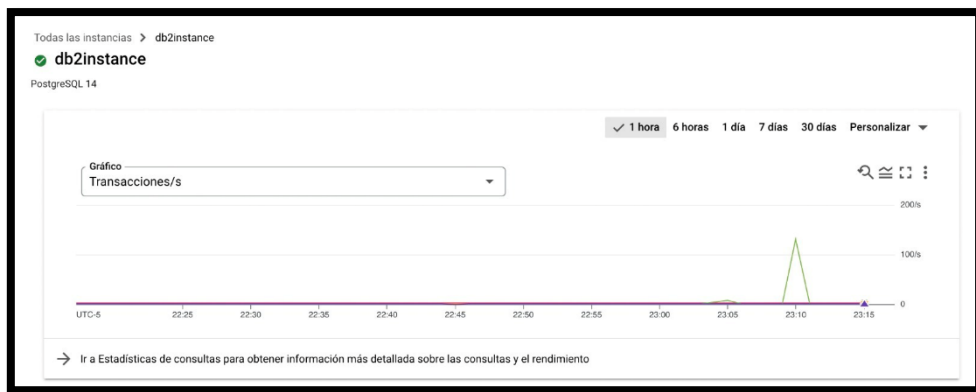


Ilustración 23. Transacciones realizadas a la base de datos.

Con esto se concluye que para el tema de base de datos escala de mejor manera que con Compute Engine, por lo cual para el caso propuesto en clase es mejor usar un App Engine que un Compute Engine o un Serverless.

## Escenario 2:

En este escenario se estresó el endpoint de Compresión, y el endpoint de ejecución a la base de datos.

El primer escenario nos muestra como la cloud function encargada de la compresión recibe una carga de 100 archivos y automaticamente auto escala las instancias que tiene disponibles de 2 a 6 instancias:

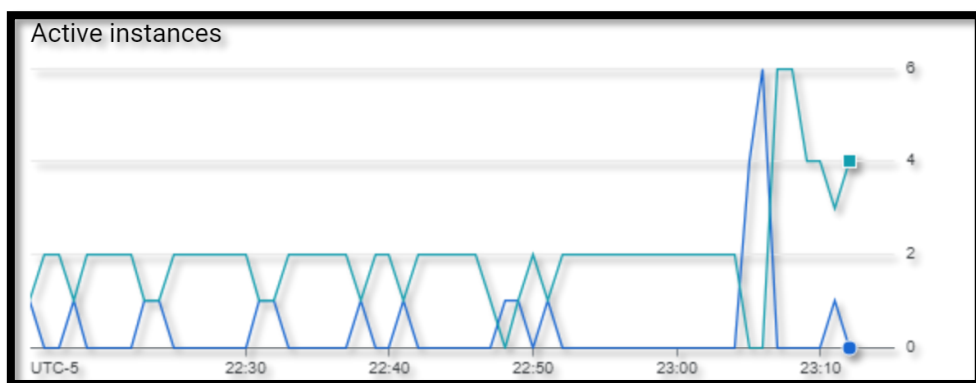


Ilustración 24, instancias activas para ejecutar la cloud Function

El pico que se logra evidenciar se refleja en las demás graficas de desempeño, tanto como escalamiento de aprovisionamiento como el escalamiento en disminución de su capacidad, luego de terminar la compresión de los archivos. Se enviaron 100 archivos de compresión por segundo. Acá podemos ver una de las grandes ventajas del auto escalamiento del Paas o de las soluciones serverless que me permiten ocuparme en el negocio, mientras que el proveedor autogestiona la demanda de las necesidades técnicas.

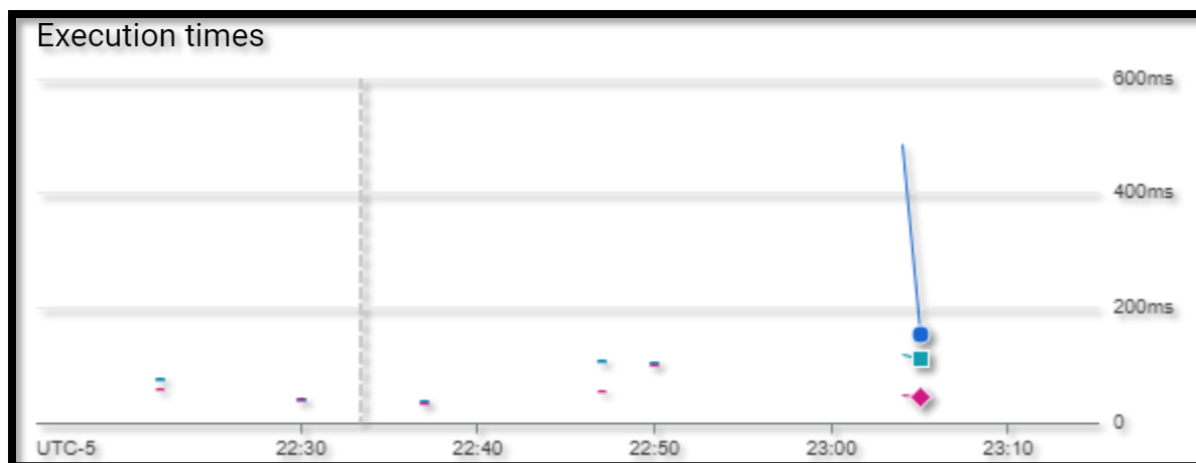


Ilustración 25, Gráficas de ejecución de cloud function encargada del worker.

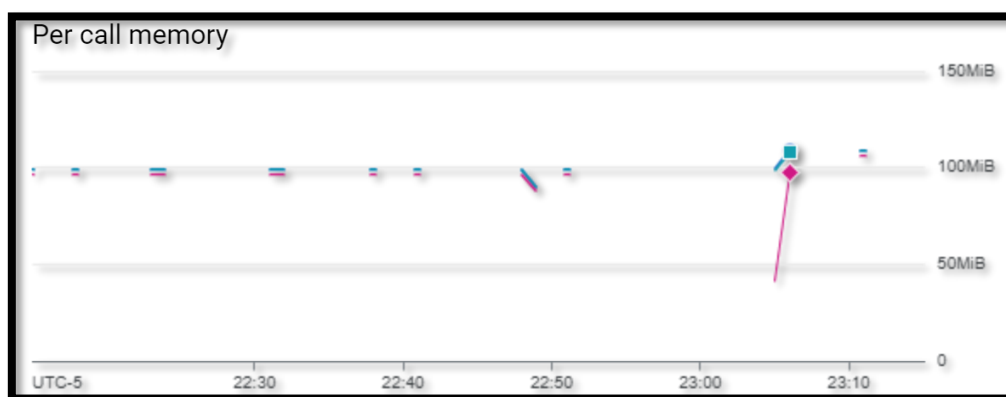
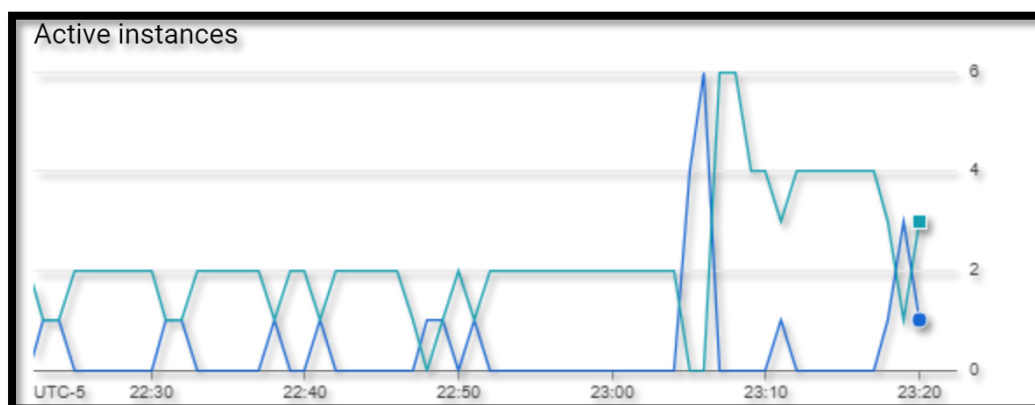


Ilustración 26, Grafica de llamado de memoria de cloud función en estrés de 100 compresiones por segundo.

A continuación, se presentan los resultados de desempeño de la escritura hacia la base de datos de SQL por 100 tareas por segundo:



En la escritura también se evidencia el auto escalamiento automático que realiza la función de la nube para mantener continuo el proceso de escritura, aumentando las instancias de acuerdo a la demanda de carga que está recibiendo la función.

## **Conclusiones**

- La implementación de cloud Functions, luego de compararla con la arquitectura de workers tradicional, ahorra muchos flujos de trabajo en cuanto implementación de código, se genera en unas pocas líneas, tiene auto escalamiento y se autogestiona al momento de estresarse.
- De igual manera la implementación de cloud Functions limita al desarrollador a tener control sobre el control de errores sobre el entorno de GCP, pues lanzar las pruebas tiene una demora de más o menos 3 minutos por intento y el manejo de errores se debe adecuar a los errores de la plataforma.
- Se debió aumentar el tamaño de maquina planteado para la infraestructura debido a que la instalación de los requerimientos estresaba la máquina para el escalamiento, con lo cual los planes de carga se deben modificar para estándares superiores.
- Cuando la iniciación de la maquina es muy pesada, como lo fue el caso de la instalación de los requerimientos de nuestra aplicación, el periodo de enfriamiento se ve afectado, para ello limitamos el uso máximo de la cpu al 60%, y un periodo de 100 seg, con lo cual el auto escalador logra inicializar y manejar las peticiones adicionales de la aplicación.
- Los desarrollos con elementos de Paas o serverless, le permiten al usuario tener Sprint más rápidos de desarrollo sin tener preocupaciones de infraestructura, enfocándose en el desempeño del negocio y estética del mismo. Gracias a la autogestión de estos productos.

## **Bibliografía**

1. <https://cloud.google.com/functions/docs/securing/function-identity>
2. <https://cloud.google.com/functions/docs/how-to>
3. <https://cloud.google.com/sql/docs/mysql/iam-roles?hl=es-419>
4. <https://cloud.google.com/load-balancing/docs/https/setting-up-https>