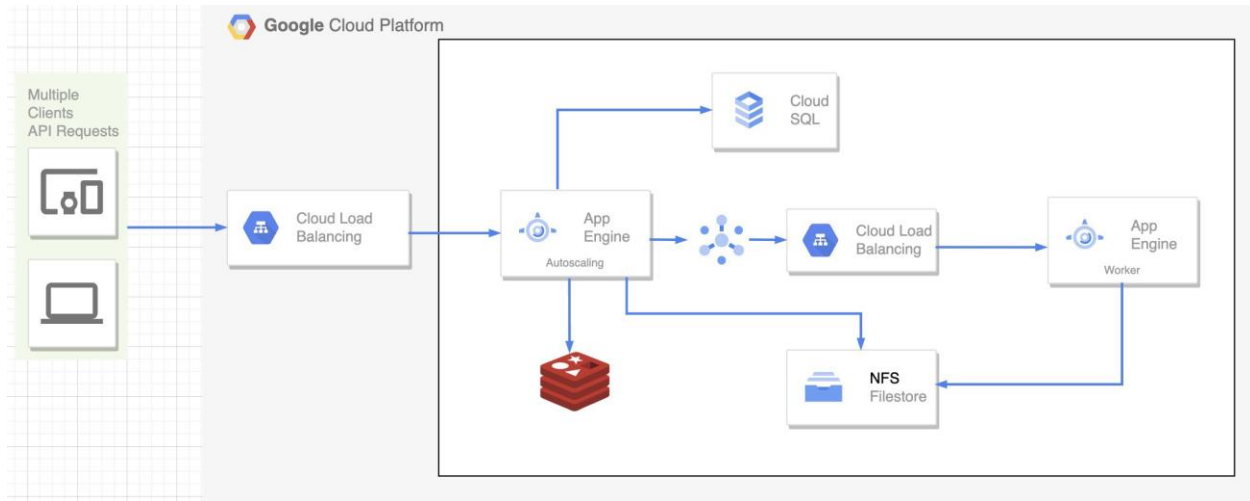


Entrega 5 – Despliegue en PAAS, Grupo 20

Migración De Una Aplicación Web A Una Plataforma Como Servicio En La Nube Pública

1 Arquitectura de la Aplicación

1.1 Diagrama de la Arquitectura



1.2 Infraestructura

Para la infraestructura se utilizó las siguientes soluciones tecnológicas:

- Google App Engine: Decidimos realizar la utilización de Google App Engine como plataforma de PaaS para ejecutar la aplicación. La aplicación se configuró como un solo recurso de aplicación que consta de varios servicios. Cada uno de estos servicios se configuró para utilizar entornos de ejecución y configuración de rendimiento específicos.
- Cloud SQL: Utilizamos Cloud SQL, configurando así una instancia corriendo PostgreSQL 14, esta instancia se configuró en modo desarrollo, con 1 núcleo, 1 GB de RAM y 10 GB de almacenamiento para reducir costos.
- Cloud Monitoring: Se configuró Google Cloud Monitoring para supervisar el estado y rendimiento de los servicios utilizados en la aplicación, como también la instancia de SQL.
- Autoscaling: Se utilizó un grupo de autoscaling de App Engine para la capa web de la aplicación. Esto nos permitió ajustar automáticamente la cantidad de instancias de App Engine ejecutadas según la demanda de usuarios.

- Cloud Pub/Sub: Se utilizó Cloud Pub/Sub para permitir que las aplicaciones intercambiaran mensajes de forma confiable, rápida y asíncrona. Facilitando así la comunicación entre los distintos componentes de la aplicación.
- Filestore: Se utilizó Filestore ya que este nos permitió implementar un servicio de archivos compartidos (NFS).
- 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, en las cuales se realizaron algunas modificaciones para cumplir los requerimientos para un despliegue en PAAS:

- 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: A diferencia de la entrega 4, utilizamos Redis como sistema de almacenamiento en memoria, caché y agente de mensajes. En la arquitectura propuesta, Redis se ejecutó utilizando un servidor de Redis autoescalable. En este caso, el servidor está configurado para escalar según la demanda y proporcionar un alto rendimiento y disponibilidad.

Paquetes de Python utilizados para configurar nuestra aplicación en App Engine:

```
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
pydub==0.25.1
pylzma==0.5.0
google-cloud-storage
```

2 Análisis de Capacidad

2.1 Entorno de Pruebas

Google App Engine: Se configuraron múltiples servicios dentro de App Engine para la ejecución de la aplicación. Cada servicio se ajustó para utilizar entornos de ejecución y configuraciones de rendimiento específicos según sus requisitos. Esto permitió una segmentación adecuada y un control más granular sobre el rendimiento y los recursos asignados a cada componente de la aplicación.

Cloud SQL: Se creó una instancia de PostgreSQL 14 en modo de desarrollo dentro de Cloud SQL. Esta instancia se configuró con 1 núcleo, 1 GB de RAM y 10 GB de almacenamiento para reducir los costos en el entorno de pruebas. La instancia de SQL proporcionó la persistencia de datos necesaria para la aplicación y permitió realizar pruebas de integración y rendimiento utilizando una base de datos real.

Cloud Monitoring: Se configuró Google Cloud Monitoring para supervisar el estado y rendimiento de los servicios utilizados en el entorno de pruebas. Esto incluyó la monitorización de las instancias de App Engine y la instancia de SQL.

Autoscaling: El grupo de autoscaling de App Engine se configuró específicamente para el entorno de pruebas de la capa web de la aplicación. Esto permitió ajustar automáticamente la cantidad de instancias de App Engine en ejecución según la carga de trabajo generada por las pruebas. El autoscaling garantizó que el entorno de pruebas tuviera suficiente capacidad para manejar cargas de trabajo variables y permitió una optimización de costos al escalar hacia abajo cuando la demanda disminuía.

Cloud Pub/Sub: En el entorno de pruebas, se utilizó Cloud Pub/Sub para permitir la comunicación asíncrona entre los distintos componentes de la aplicación. Se configuraron los temas y las suscripciones correspondientes para el intercambio de mensajes entre las aplicaciones durante las pruebas.

Filestore: En el entorno de pruebas, Filestore se utilizó para implementar un servicio de archivos compartidos basado en NFS. Esto permitió a las aplicaciones acceder y compartir archivos de manera eficiente durante las pruebas, facilitando el desarrollo y la validación de funcionalidades que requerían acceso a datos en archivos.

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 del grupo de autoscaling creado.

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

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 o fallo del escenario, además se autoriza a través de jwt.

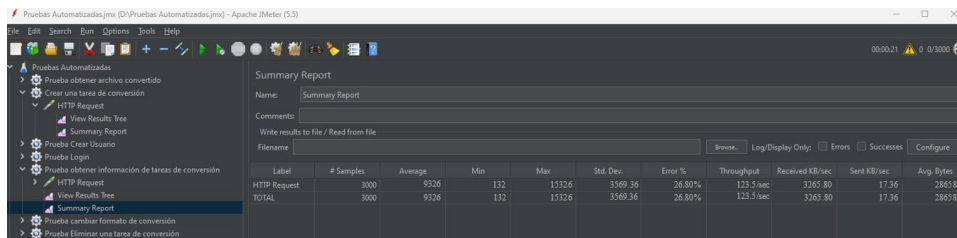
Los parámetros para el POST se configuran inicialmente en 500 threads y van decreciendo hasta el punto de éxito o fallo del escenario, además se autoriza a través de jwt.

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.



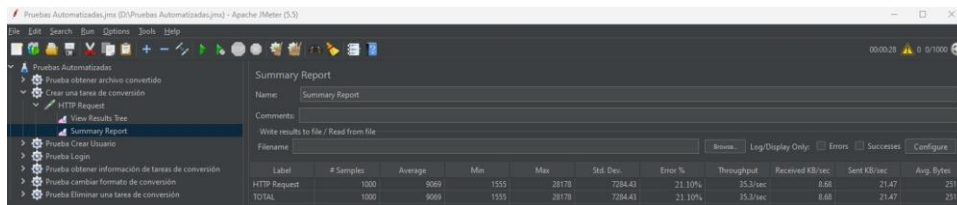
The screenshot shows the Apache JMeter Summary Report for a test named 'Summary Report'. The test consists of several steps, with the 'HTTP Request' step being the primary focus. The report displays various performance metrics for 3000 samples.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	3000	93.26	132	1532.6	3369.36	26.80%	123.5/sec	3265.80	17.36	28658.6
TOTAL	3000	93.26	132	1532.6	3369.36	26.80%	123.5/sec	3265.80	17.36	28658.6

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



The screenshot shows the Apache JMeter Summary Report for a test named 'Summary Report'. The test consists of several steps, with the 'HTTP Request' step being the primary focus. The report displays various performance metrics for 1000 samples.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1000	90.69	1535	2017.8	7284.43	21.10%	15.3/sec	8.68	21.47	231.4
TOTAL	1000	90.69	1535	2017.8	7284.43	21.10%	15.3/sec	8.68	21.47	231.4

3 Documentación Postman

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

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