Profesor:

Jesse Padilla Agudelo

Integrantes:

Brenda Catalina Barahona Pinilla (201812721)
Juan Diego González Gomez (201911031)
Kevin Steven Gamez Abril (201912514)
Sergio Julian Zona Moreno (201914936)

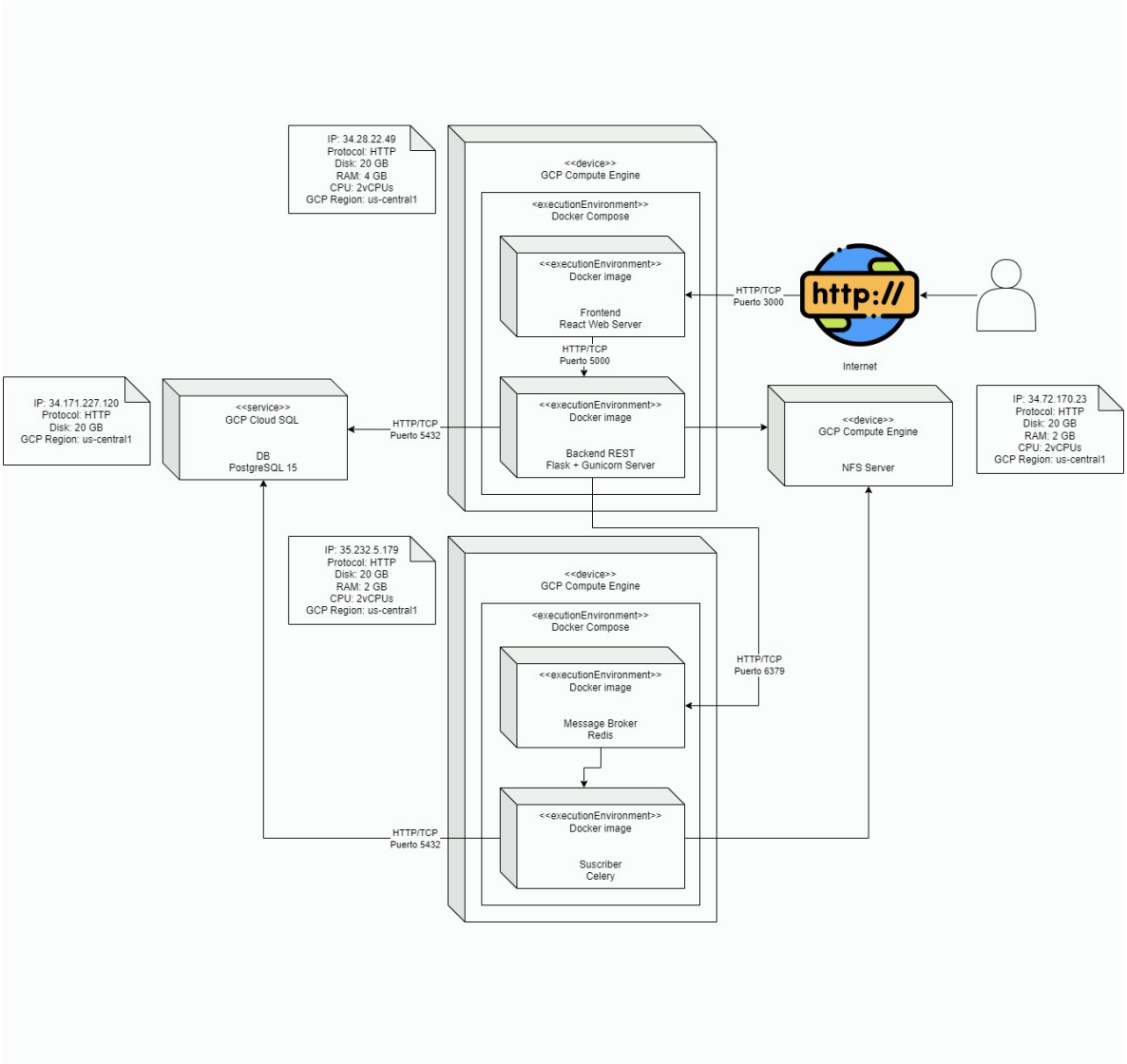
Tabla de contenido

1 Introducción

En este documento se documenta la solución a la Entrega 2 del proyecto del curso *Desarrollo de soluciones cloud*. El propósito central es desplegar una aplicación Web completa (Frontend + Backend + Base de datos relacional + Servicios asincrónicos) utilizando contenedores de Docker, y *Docker Compose* y desplegar los servicios de manera modularizada en diferentes instancias virtuales de *Compute Engine* en GCP.

2 Arquitectura de Software

2.1 Diagrama de despliegue y descripción



La arquitectura de Software que se adoptó fue la misma que recomendó el profesor Jesse en clase. Para simplificar la arquitectura la dividiremos en los diferentes servicios:

- Servicio de Backend y Frontend: a este servicio accede el usuario por medio de internet a través del Protocolo HTTP. Se conecta a una instancia de *Compute Engine* de GCP por medio del puerto 5000 al servicio de Frontend, luego el servicio de Frontend accede a todos los endpoints necesarios del back por medio del puerto 3000 de la aplicación. Este servicio se conecta de manera externa a: el servicio de procesamiento asincrónico de archivos, el servidor NFS donde se almacena de manera unificada los archivos procesados, y a la base de datos de GCP Cloud SQL donde extrae información de manera eficiente.

- Servicio de procesamiento de archivos (asincrónico): con el fin de mantener la tarea de procesamiento de archivos de manera separada del servidor principal, se delegó esta tarea a una máquina virtual específicamente. Redis y Celery son quienes permiten el funcionamiento asincrónico del backend, siendo Redis el Message Broker al cuál se suscribe Celery, el cual posteriormente inicia los Workers necesarios para procesar los archivos y actualizar la base de datos de PostgreSQL que se encuentra en un servicio de GCP Cloud SQL. Este servicio se conecta de manera externa a: al servicio de backend del cual recibe todas las peticiones, la base de datos GCP Cloud SQL y el servidor NFS donde se almacena de manera unificada los archivos procesados.
- GCP Cloud SQL: instancia de la base de datos de PostgreSQL en su versión 15, almacena de manera separada toda la información. Se conecta directamente al servicio de procesamiento asincrónico donde recibe constantemente actualizaciones. De manera paralela, el backend tiene una conexión directa con la base de datos para poder extraer la información de manera rápida y enviarla al cliente en el frontend.
- Servidor NFS: funciona como sistema de almacenamiento unificado al cual se conecta el backend de la aplicación para extraer rápidamente los archivos, y el servicio de procesamiento de archivos el cual almacena los PDFs en la unidad una vez fueron procesados.

Con estos servicios se da cumplimiento a los componentes que se solicitaban en el taller: instancia *Web server*, *worker* de procesamiento de archivos, *file server* de almacenamiento y una *base de datos relacional* en una instancia separada.

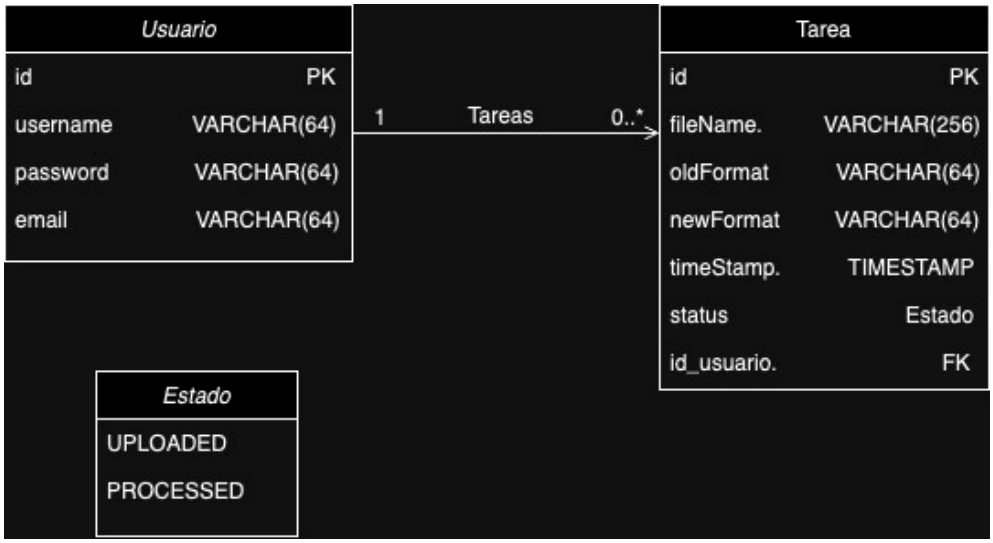
2.2 Características de las máquinas y las instancias

Todas las máquinas tienen las siguientes características:

- SO: Ubuntu 22.04 LTS
- Protocolo de conexión: HTTP
- Disk: 20 GB
- RAM: 2 GB
- CPU: 2vCPUs
- GCP Region: us-central1

Exceptuando la máquina que contiene el Servicio de Backend y Frontend a la cuál se le otorgó una capacidad de 4 GB de RAM.

2.3 Diagrama entidad relación



El modelo presentado en la imagen describe un sistema de gestión de usuarios y tareas, estructurado en dos tablas principales: Usuario y Tarea, con una relación de uno a muchos. Esto significa que un usuario puede tener asignadas múltiples tareas, pero cada tarea está específicamente vinculada a un solo usuario. La tabla Usuario incluye campos para el identificador único del usuario, nombre de usuario, contraseña y correo electrónico, siendo este último único para cada usuario. Por su parte, la tabla Tarea consta de un identificador único, el nombre del archivo asociado, los formatos original y nuevo del archivo, una marca de tiempo que registra la creación o modificación de la tarea, el estado de la tarea que puede ser 'UPLOADED' (subido) o 'PROCESSED' (procesado), y una clave foránea que enlaza cada tarea a su usuario correspondiente.

2.4 Release

https://github.com/JuanDiegoGonzalez/Proyecto_Cloud_Grupo2/releases/tag/2.0

2.5 Video de sustentación

https://youtu.be/-Z7gpzTLB6Q

3 Pruebas de estrés, análisis y documentación

Para efectos prácticos seleccionamos JMeter como herramienta de pruebas que nos permitirá evaluar los dos escenarios planteados en la entrega previa.

Escenario	Descripción	Subescenario	Tipo	Header	Número de peticiones	Latencia máxima	Error máximo	Servicio(s) que afecta
Escenario 1	Peticiones POST para subir un archivo	Escenario 1.1	POST	JSON	1000	300 ms	1%	VM 1, VM 2, Cloud SQL y Linux NFS
		Escenario 1.2	POST	JSON	2000	500 ms	2%	VM 1, VM 2, Cloud SQL y Linux NFS
Escenario 2	Peticiones GET para descargar un archivo	Escenario 2.1	GET	JSON	1000	300 ms	1%	VM 1, Cloud SQL y Linux NFS
		Escenario 2.2	GET	JSON	2000	500 ms	2%	VM 1,Cloud SQL y Linux NFS

3.1 Escenario 2

Los resultados que se obtuvieron del escenario 2 para un *ramp-up* de 1 seg fueron:

- Para 1000 peticiones:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	1000	2540	181	5054	1280.54	0.00%	159.6/sec	27.90	78.25	179.0
TOTAL	1000	2540	181	5054	1280.54	0.00%	159.6/sec	27.90	78.25	179.0

- Para 2000 peticiones:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	2000	4735	173	11121	2585.49	0.00%	167.0/sec	29.20	81.89	179.0
TOTAL	2000	4735	173	11121	2585.49	0.00%	167.0/sec	29.20	81.89	179.0

Conclusiones del escenario 2:

Para este escenario en particular pudimos observar que el % error esperado se mantuvo en 0% hasta las 4000-4500 peticiones (siendo este el punto de quiebre). Lo cuál indica que contamos con una capacidad considerable para atender usuarios simultáneamente. Sin embargo, el tiempo de latencia no fue el esperado, tenemos aproximadamente 10 veces más tiempo de latencia producto de nuestro proveedor de red, la capacidad de procesamiento de las máquinas y la ubicación de la Zona de disponibilidad.

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
HTTP Request	4546	12425	207	30585	7146.65	3.83%	139.2/sec	37.95	65.64	279.2
TOTAL	4546	12425	207	30585	7146.65	3.83%	139.2/sec	37.95	65.64	279.2

Se planteo de SLO de menos del 5% de error en alta demanda. Nuestro SLI para 4546 peticiones fue de 3.83% por lo que cumplimos de manera satisfactoria el SLO.

3.2 Escenario 1

Conclusiones del escenario 1:

Como era de esperarse, el escenario 1 obtuvo métricas inferiores en términos de desempeño de la aplicación. Esto se debe a que afectaba más servicios, por lo que la latencia aumentó en más de un 40%. Asimismo el % error no logra satisfacer la alta demanda con más de 4500 usuarios, teniendo un porcentaje de fallas considerable de más del 10%.

4 Limitaciones y consideraciones adicionales

4.1 Limitaciones

Dentro de las limitaciones podemos encontrar que:

- Solo existe una instancia que recibe peticiones, no hay replicación y se presenta ausencia de balanceadores de carga.
- Las pruebas de carga se ejecutan con el software de JMeter por lo que nos encontramos limitados a las funcionalidades que este disponga.
- Según la ubicación de los servidores y la disponibilidad que tengamos, puede que la latencia aumente debido a la distancia que se tiene con la zona de disponibilidad.
- La capacidad de las máquinas es limitada, debido a el número de créditos disponibles. De igual forma se tiene en cuenta que esta aplicación es de tamaño pequeño y para fines académicos. Asimismo, tampoco podemos aprovisionar múltiples máquinas por

lo que no es posible realizar pruebas de escalabilidad (y autoescalabilidad) y disponibilidad dentro del alcance del curso.

4.2 Consideraciones adicionales para escalar la aplicación

- Implementar un balanceador de carga que redirija a múltiples instancias (y no solo una) de un mismo servicio. Esto puede aumentar la capacidad de peticiones que se atienden simultáneamente.
- Ajustar (de acuerdo con los rubros disponibles para el curso) la capacidad de las máquinas especialmente en temas de procesamiento y memoria RAM, es decir, realizar un escalamiento vertical de las instancias.
- De manera paralela al balanceador, implementar una política de *autoscaling* que genere escalamiento tanto horizontal como vertical cuando detecte picos de peticiones superiores a un umbral del 85% de capacidad. Esta política de *autoscaling* se puede complementar con modelos de inteligencia artificial que permitan detectar de manera eficiente los picos de peticiones.