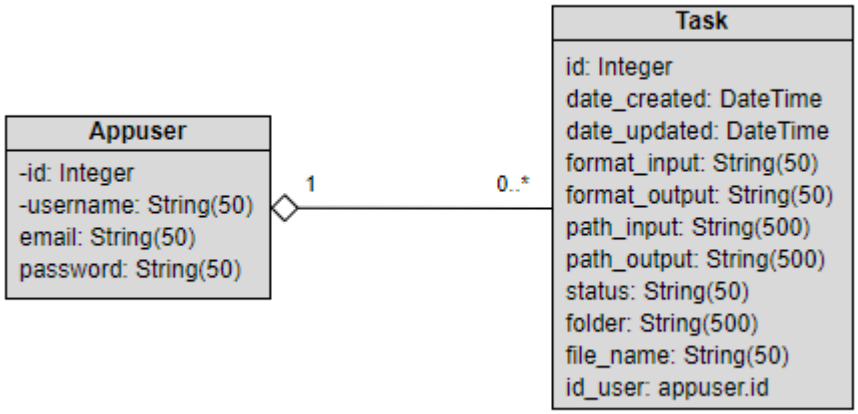
Entrega 5 - Arquitectura, conclusiones y consideraciones

A. Arquitectura

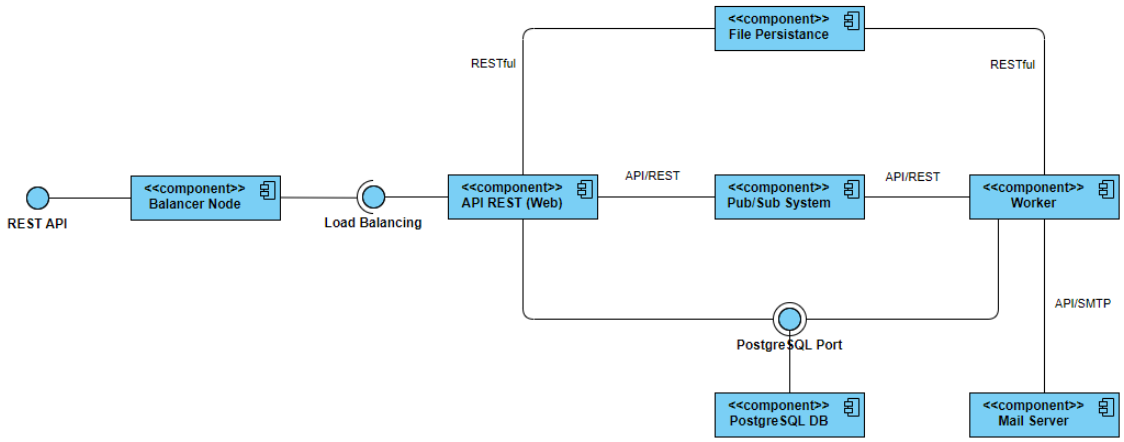
Se diagrama la arquitectura desde Vista de Diseño (Diagramas de entidades, y de componentes), para explicar la estructura de tablas SQL definidas para gestionar seguimiento a las tareas a encolar en el worker, y una Vista de Despliegue (Diagrama de despliegue) para que se identifiquen los diversos componentes tipo SaaS, incluidos los servicios de App Engine y los proveedores de almacenamiento, base de datos, y sistema de broker de mensajes.

a. Diagrama de entidades (PostgreSQL)



Esto se refleja en las tablas Appuser y Task creadas en el entorno Cloud SQL, sabor PostgreSQL.

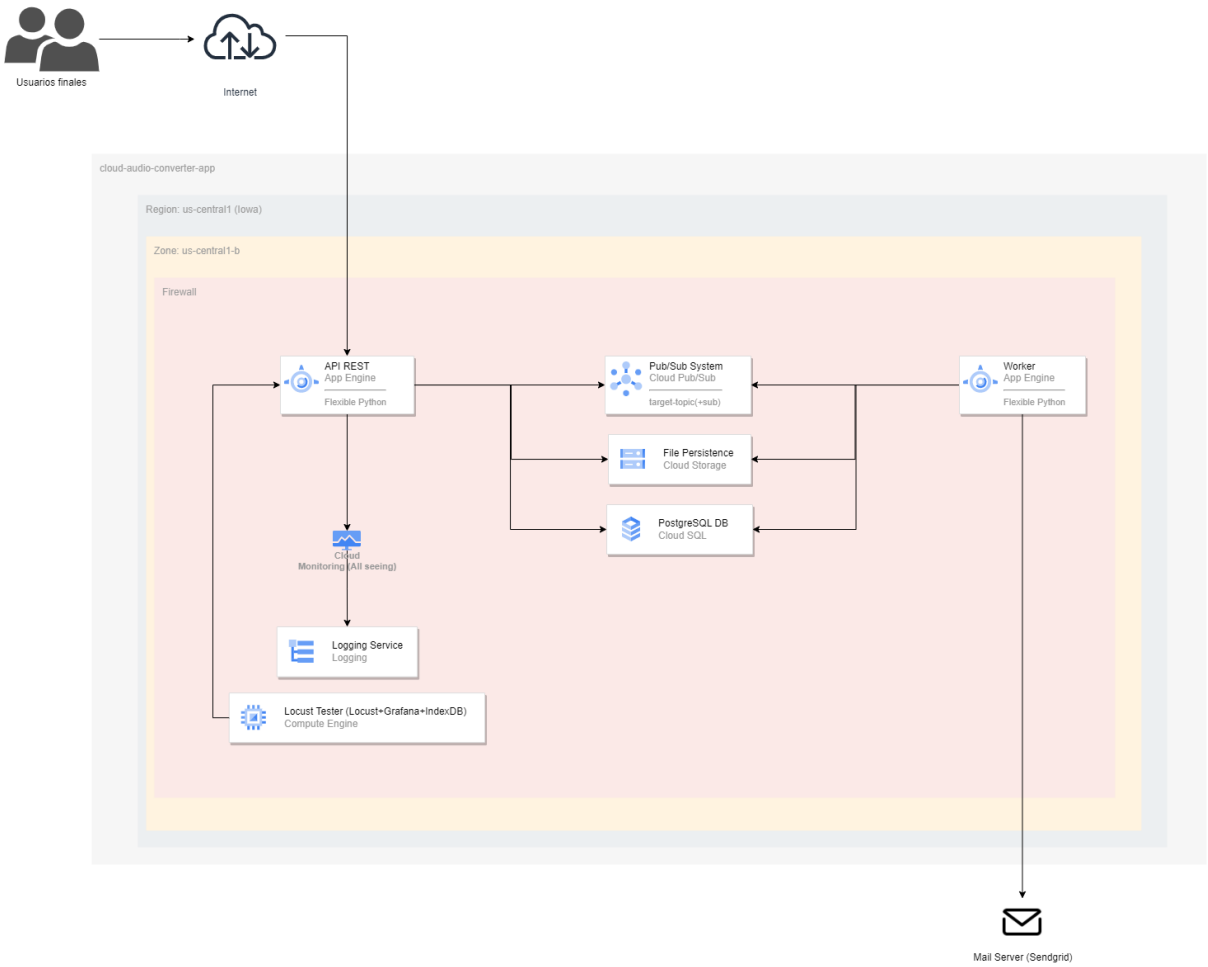
b. Diagrama de componentes



A grandes rasgos, el diagrama de componentes expone brevemente la solución actualmente empleada por el equipo de desarrollo. Se tiene el Balancer node que es una abstracción del actor responsable de balanceo de carga (gestionado por App Engine), quien a su vez se conecta al API REST asociado a la capa Web de la solución (pues está será balanceada y autoescalada de ser necesario, aunque en esta edición no se habilita autoscaling), el componente Worker que procesa los trabajos de conversión, quién se suscribe a un tópico de una instancia de Google Cloud

Pub/Sub (modo PULL). Como componentes compartidos se tiene la base de datos en PostgreSQL DB, el servidor de envío de correos como Mail Server, y un Cloud Storage que se encarga de guardar los archivos recibidos por Web a convertir, y los archivos convertidos por el Converter Worker (conforme consume mensajes de petición de conversión en el tópico Pub/Sub).

c. Diagrama de Despliegue



Enlace: [Diagrama de despliegue Cloud - App Engine.png]

El diagrama de despliegue, mediante las cajas de color identifica las distintas fronteras:

- Rojo: Componentes protegidos por firewall.
- Amarillo: Zona donde se crean los componentes.
- Azul: Región que alberga zona para despliegue.
- Gris: Frontera del proyecto contenedor de los componentes de la solución.

Las flechas indican las dependencias o uso de recursos explícitos desde cada componente de Google Cloud Platform. Hay que resaltar que Cloud Storage y PostgreSQL reflejan componentes tipo SaaS que emplea la solución actual, de tal forma que están completamente desacopladas del dominio de los servicios desplegados en la aplicación de App Engine para la solución, pero que de todas formas sólo existen dentro del contexto del proyecto cloud *cloud-audio-converter-app*. En particular, se reusan los nombres del diagrama de componentes en este diagrama, para homologar los componentes identificados inicialmente en el diseño abstraído de la solución, es decir, sin indicar proveedores de nube puntuales:

- API REST (Web) se despliega como un service en Google App Engine, sin autoescalamiento (1 instancia), y en caso de necesitar autoescalamiento se configura la cantidad de instancias máximas (no requiere Instance Groups).
- Worker se despliega como un service en Google App Engine, sin autoescalamiento (1 instancia), y en caso de necesitar autoescalamiento se configura la cantidad de instancias máximas (no requiere Instance Grous).
- El bucket de Cloud Storage usa modo Standard debido a que se accede con frecuencia a los archivos de audio a procesar, y procesados. Adicionalmente App Engine despliega Buckets que el mismo autogestiona, 2 para toda la aplicación App Engine, no se diagraman siguiendo las recomendaciones de diagramado de Google.
- PostgreSQL DB se aprovisiona con una instancia SaaS de Cloud SQL “sabor” PostgreSQL.
- El Mail Server lo provee Sendgrid y existe afuera de la solución. No se aprovisiona desde GCP directamente.

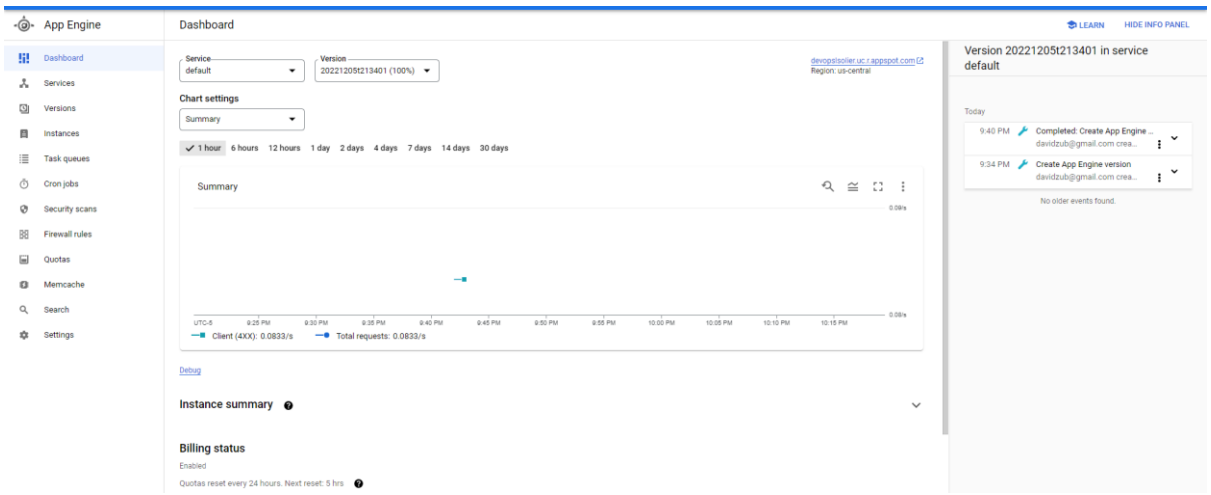
Se mantiene encolado de trabajo mediante broker de mensajería ofrecido por una instancia de Google Cloud Pub/Sub. Permite mantener el desacoplamiento a nivel de microservicios, es decir entre API REST y Worker. Además de este modo, de ser necesario se puede habilitar autoescalamiento nuevamente, de manera transparente sin necesidad de modificar ninguno de los microservicios.

Adicionalmente se levanta un Compute Engine con la aplicación Locust empleada en las pruebas de estrés, denominado Locust Tester.

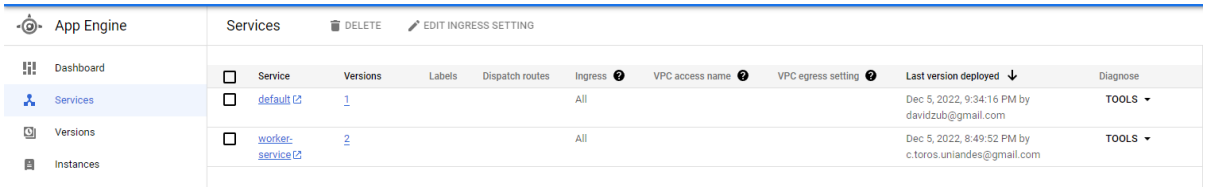
Los servicios de Monitoring y Logging vienen por defecto en el proyecto de Cloud, no hay necesidad de configuraciones adicionales, ya que App Engine se encarga de monitorear toda instancia de todo servicio que contenga (aunque ofrece opción de librería Python para escribir logs más ricos de ser necesario).

Tanto el servicio de API REST (default en App Engine) como el de worker (worker-service en App Engine) usan la configuración de hardware por defecto para App Engine Flexible, es decir, 1 vCPU, con 0.6 GB de RAM y 13 GB de disco de almacenamiento.

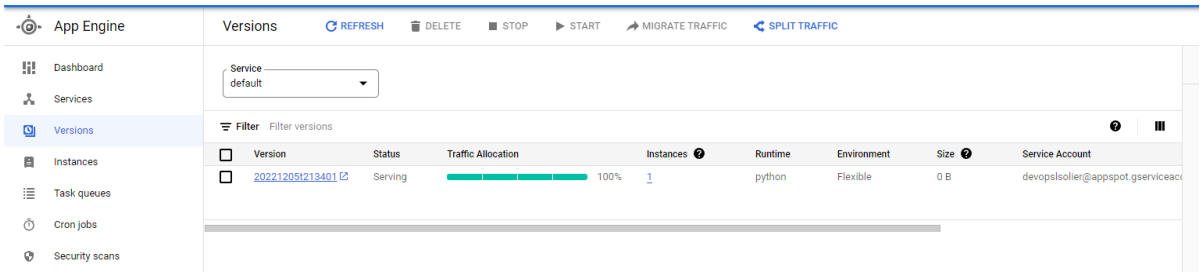
Acá, el dashboard de App Engine



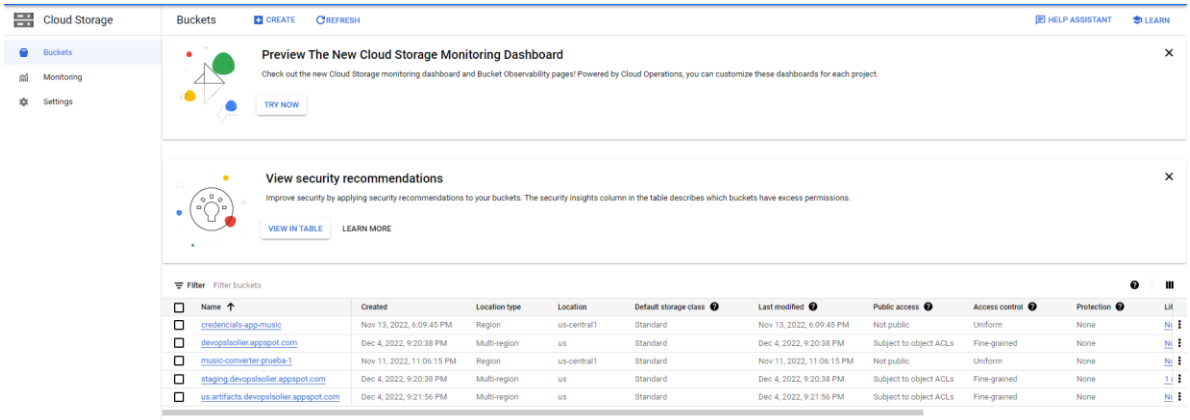
Servicios activos



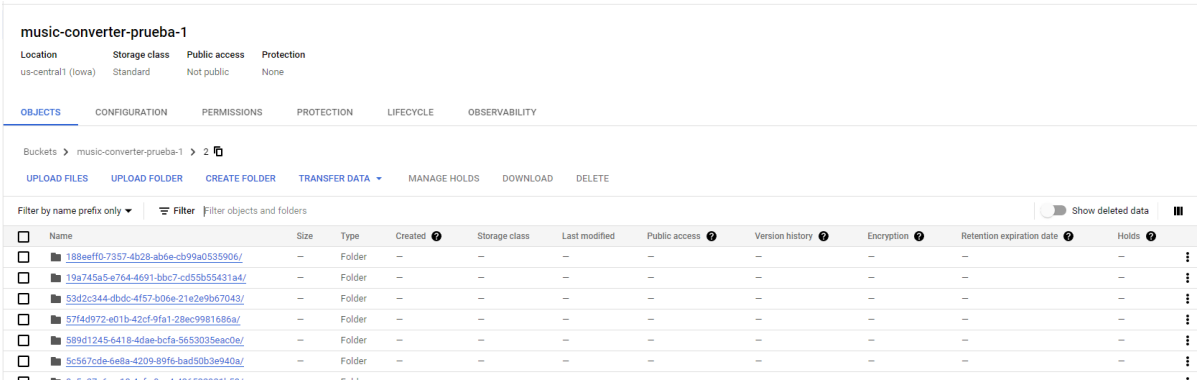
Versiones (sólo aplica para servicios web)



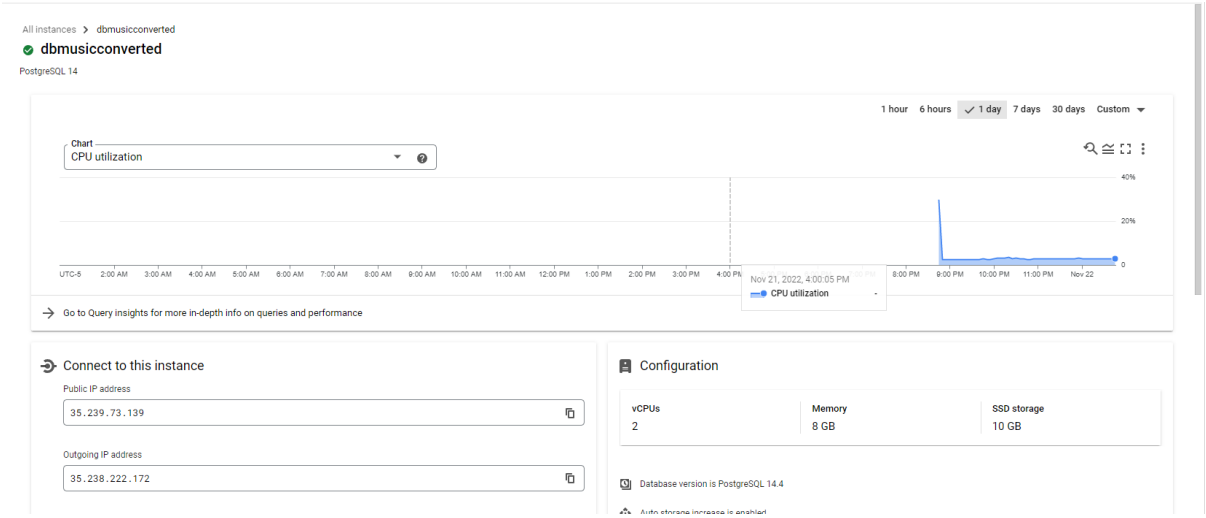
Buckets configurados (credenciales-app-music ya no se usa), :



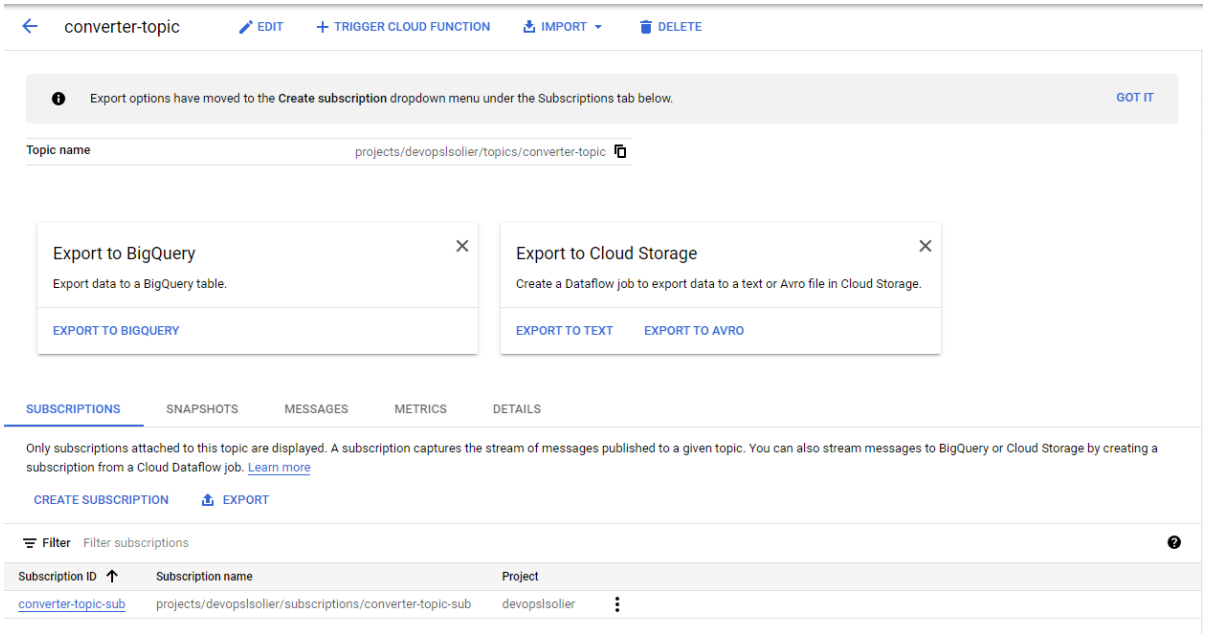
Dominios de appspot referencian buckets de uso exclusivo de App Engine



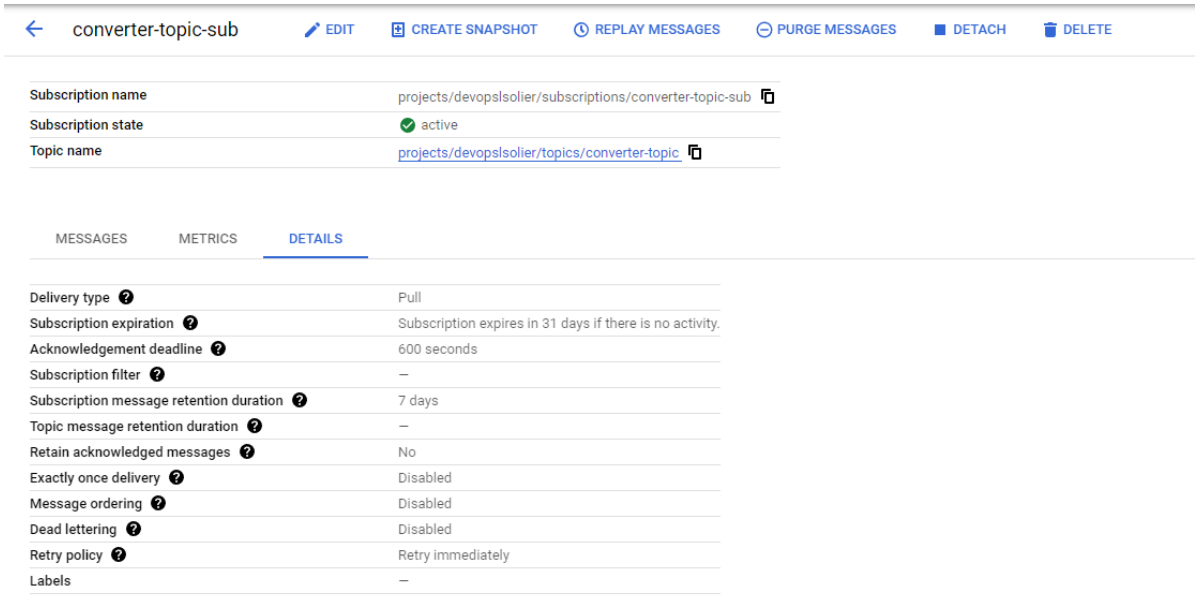
Cloud SQL configuración:



Tópico Pub/Sub:



Suscripción para consumir del Tópico Pub/Sub:



URL pública de solución migrada a nube: https://devopslsolier.uc.r.appspot.com/

Reemplazar en URL base de Postman Collections disponible en: https://github.com/edgar-melara-uniandes/proyecto_cloud/tree/entrega_5/documentation

B. Conclusiones de las pruebas de estrés

Desafortunadamente, la conclusión es que, si no se deja escalar o si al menos no se usa hardware virtualizado robusto, para las aplicaciones Python (en especial el converter), no se estará a la altura de las tareas de conversión solicitadas. Se observa hasta 12 minutos para terminar una tarea de conversión con unos pocos request. De hecho, al equipo de trabajo le parece extraño que no se solicite implementar autoscaling, después de realizar las entregas 4 y 5, pero igual se considera una instrucción válida, viendo los costos que maneja App Engine por consumo (son potencialmente más caros que aquellos en una solución equivalente pero basada en Compute Engine).

De todas formas, sí se observó que el mero levantamiento de una nueva versión de servicio (sea API REST o Worker) puede tomar varios minutos, entonces se asume que el autoscaling consiguiente tomaría varios minutos y no se podría adaptar de forma rápida a picos de demanda de la aplicación, al menos no sin modificar un poco el código de la aplicación. Pero, por otro lado, las instancias de servicios que levanta App Engine suelen ser “saludables”, es decir, están listas para operar y no requieren de calibrar un cooldown time precioa, como sí ocurre en el caso de MIGs para Compute Engine.

Como se asume que el hardware subyacente a App Engine debe ser de naturaleza similar al provisto en Compute Engine, las posibilidades de escalamiento, se asume, no se deberían desviar de aquellas vistas con la solución generada en Entrega 4, dejando de lado observaciones previas. Para resumir, el autoescalamiento para este tipo de soluciones es esencial, en especial si se va a realizar este tipo de pruebas de estrés. A su vez, se debe responder de manera recíproca, es decir que la solución si o si esté configurada pensando en atendar dicha demanda artificial, sino cualquier prueba de estrés no entregará resultados satisfactorios.

Más detalles de las métricas de desempeño las pueden encontrar en el documento “Entrega 5 - Escenario y Pruebas de Estrés API REST y Batch”.

C. Consideraciones para el paso a entorno en nube, masivo

Sea por empleo de Compute Engine o App Engine, según se quiera extender funcionalidades a futuro de la app, se deberá favorecer una opción sobre otra, ya que ninguna representa una bala de plata (ver sección D, Comparación con solución basada en Compute Engine). Ambas permiten múltiples cantidades de clientes atendidos simultáneamente, si se tiene el presupuesto adecuado; de momento se sabe que hay un límite duro de servicios, versiones e instancias del lado de App Engine (20 instancias por defecto en el caso flexible), así que teóricamente gana Compute Engine, de nuevo si se tiene el presupuesto adecuado. Si no, gana App Engine al ofrecer configuraciones por defecto "robustas".

Por temas de presupuesto, no se puede explorar a profundidad configuraciones más elaboradas, sea modo Standard o modo Flexible, para jugar con el comportamiento de hardware subyacente. De todas formas, después de los múltiples ejercicios usar máquinas tipo N1 no es recomendable.

D. Comparación con solución basada en Compute Engine

- **Compute Engine**

Ventajas: El empleo de máquinas virtuales de Compute Engine ofrece mayor flexibilidad y poder para escalar la solución final según una demanda esperada del lado de los desarrolladores. Además, al tener acceso directo a la máquina virtual, se puede instalar software personalizado, hacer ajustes customizados al OS o cambiar el mismo, además de gestionar granularmente detalles como tipo de disco duro, configuraciones más finas de red y de permisos de firewall, y una diversidad de opciones.

Desventajas: Sin embargo, el empleo de una máquina virtual gestionada puede incurrir en costos mayor, tanto en tiempo como dinero; por principio de que se deben configurar y gestionar mayor diversidad de componentes para hacer que la solución de software escale de forma automática (Load Balancer, Managed Instance Groups, Instance Templates, etc.), además de peculiaridades propias como escoger el hardware objetivo, imágenes de OS o de máquina a emplear, y muchos otros elementos a tener en cuenta. Se requiere mayor expertise y tiempo dedicado de administración de infraestructura en nube.

- **App Engine**

Ventajas: App Engine se centra en gestionar todos los detalles técnicos y logísticos que no son relevantes funcionalmente a una aplicación de software, esto abarca prácticamente todas las capacidades de gestión que justamente se entregan con Compute Engine. Al hacer esto, un desarrollador de software (que trabaje con los lenguajes de software soportados por GAE) puede ignorar por (casi) completo la gestión de aprovisionamiento requerido para desplegar su aplicación

Desventajas: App Engine quita preocupaciones "logísticas" de despliegue de aplicaciones, pero a cambio el desarrollador cede mucho control y ello acarrea ciertas consecuencias, como no tener un consumo controlado de las aplicaciones desplegadas y por ende potencialmente incurrir en costos mayores. Aunque ofrece personalizaciones en términos de hardware desplegado (tipo de CPU y settings del servicio en si para el caso de Standard, y detalles de monitoreo, red e instancias para el caso flexible) no está pensado para un manejo granular de ello, como si ocurre en el caso de Compute Engine. Adicional en modo standard, no hay buena trazabilidad y los despliegues son engañosos, ejemplo puede indicar que se terminó el despliegue, pero observando logs el servicio puede estar aún arrancando y no estar listo para atender peticiones. El modo flexible no se puede comparar a un Compute Engine.