Entrega 4 -ESCALABILIDAD DISEÑO E IMPLEMENTACIÓN DE UNA APLICACIÓN WEB ESCALABLE EN NUBE PÚBLICA

Grupo 3
C. Camilo Baquero Gómez, Franklin A. Pinto Carreño, Julian Yamid Torres Torres
Desarrollo de Aplicaciones Cloud
Universidad de los Andes, Bogotá, Colombia
c.baquero@uniandes.edu.co, f.pintoc@uniandes.edu.co, jy.torres@uniandes.edu.co
Fecha de presentación: Mayo 7 de 2023

LINK APLICACIÓN WEB: http://comprimemelo.com:5000/
Github: https://github.com/camilooob/comprimemelo.com

1. Arquitectura de Aplicación

La aplicación web de compresión de archivos se encuentra implementada bajo el modelo desacoplamiento utilizando una instancia web que se encarga de desplegar el front en un compute engine, y procesar los archivos de compresión en un Worker comunicado mediante una cola de Pubsub, el front y un worker que procesa la compresión de archivos.

Cuando el server web o worker supera un uso de cpu en un 60% se activa la regla de autoscaling y despliega un nodo adicional y lo agrega al Balanceador de Carga. El modelo implementa un conjunto de métodos para crear, modificar, eliminar, consultar, comprimir y descomprimir archivos y tiene acceso directo al motor de persistencia.

La vista está implementada en formato html para los formularios y páginas de presentación en capa web, y para las api rest, se utiliza el formato json, para capturar y responder las peticiones web. El controlador es el intermediario entre el modelo y la vista para interpretar las peticiones y entregar una respuesta a cada petición web realizada por un usuario.

Diagrama de arquitectura

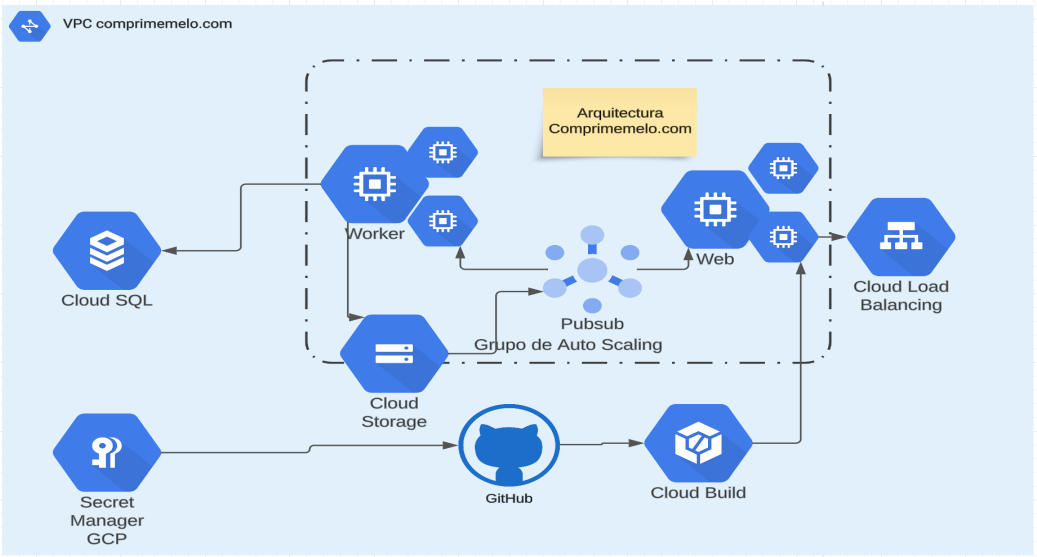


Figura 1. Diagrama de arquitectura aplicación de compresión de archivos

La arquitectura está diseñada para tener un despliegue de la aplicación para aplicaciones escalables sobre GCP, eso incluye el uso de un balanceador de carga hasta de 3 servidores web, políticas de autoscaling, además de una configuración en el servicio de balanceo de carga para poder desplegar en varios servidores web.

Nosotros definimos una estrategia para que los servidores web escalan de manera automática cuando estos alcancen el 60% de cpu , activamos el modo de autoscaling.

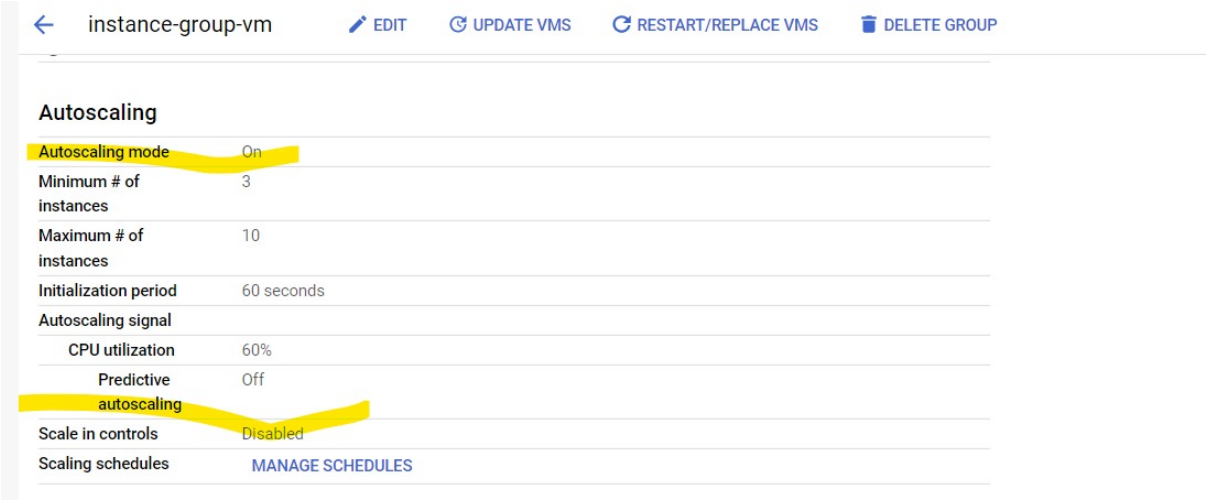
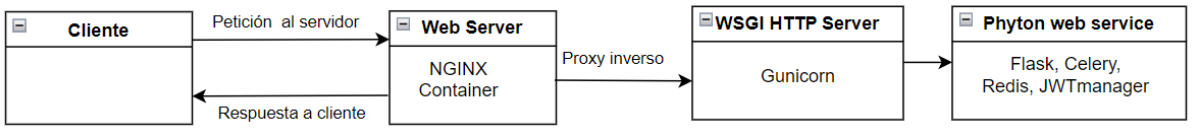
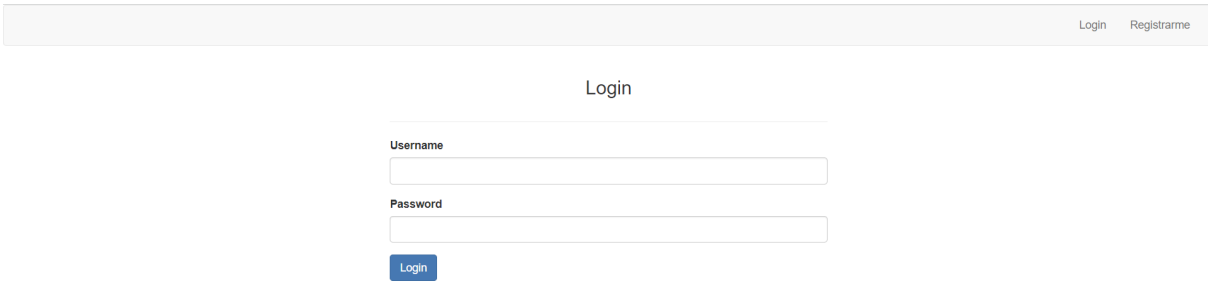


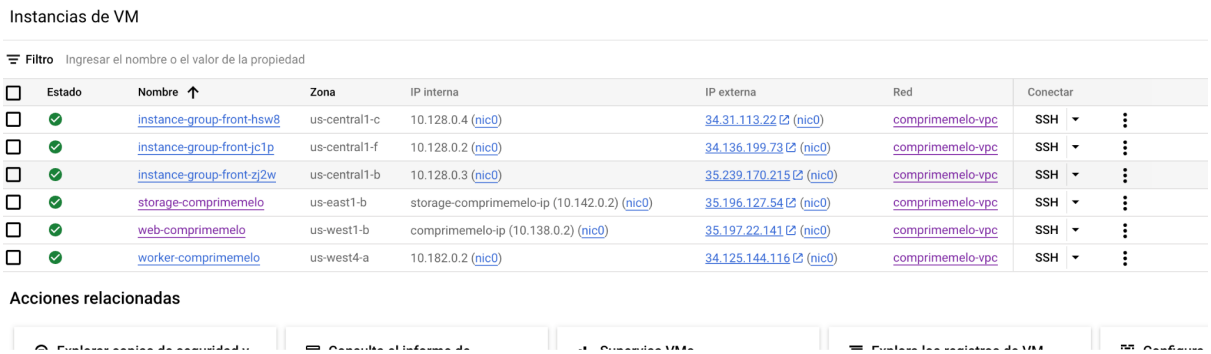
Diagrama de despliegue



La aplicación está corriendo en el dominio www.comprimemelo.com:5000 , desarrollada en Flask la cual está dividida en 3 VM de Compute Engine de la siguiente manera: Worker, Web Server y Cloud Storage. La aplicación Flask (VM Compute Engine Web Server) interactúa con la base de datos Mysql Cloud SQL, el Worker procesa las colas de pubsub de los archivos y Cloud Storage guarda los archivos.

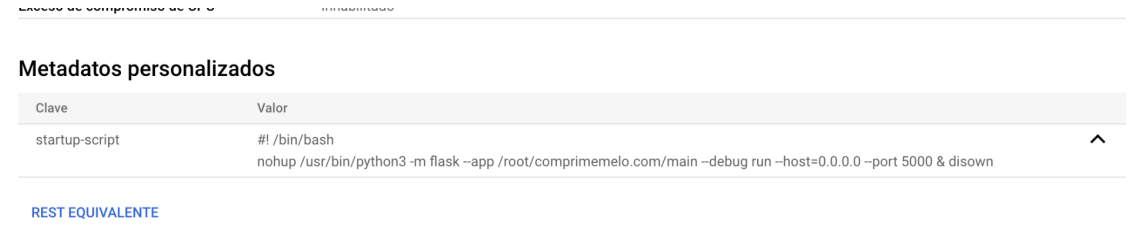


- **Instancias generadas con el autoscaling:**
- En los buckets se crearon los siguientes ,con una clase de almacenamiento estándar, de esa manera es posible almacenar todos los archivos subidos por los usuarios, tanto los originales como los procesados.



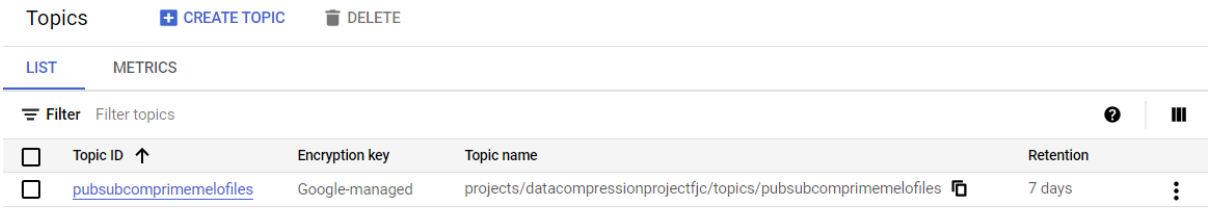
En esta parte nosotros colocamos el siguiente script para iniciar la app , cuando se crea un nuevo nodo con el autoscaling . Es un archivo que contiene comandos que se ejecutan cuando se inicia una instancia de máquina virtual, permitiendo expandir la red, discos, y

administración de los metadatos.

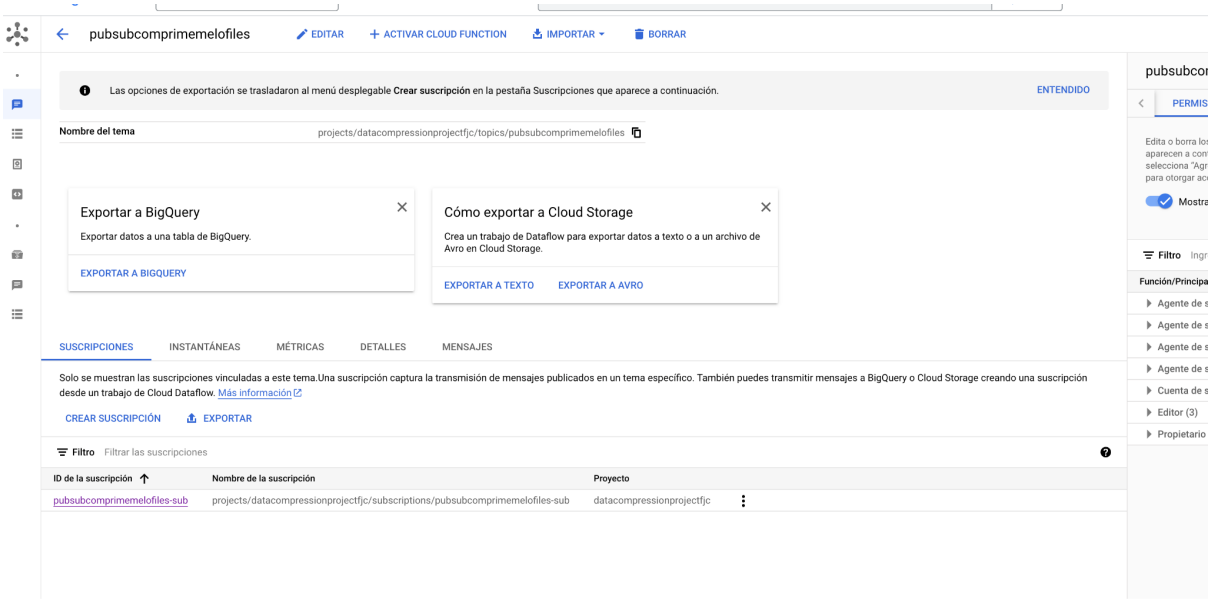


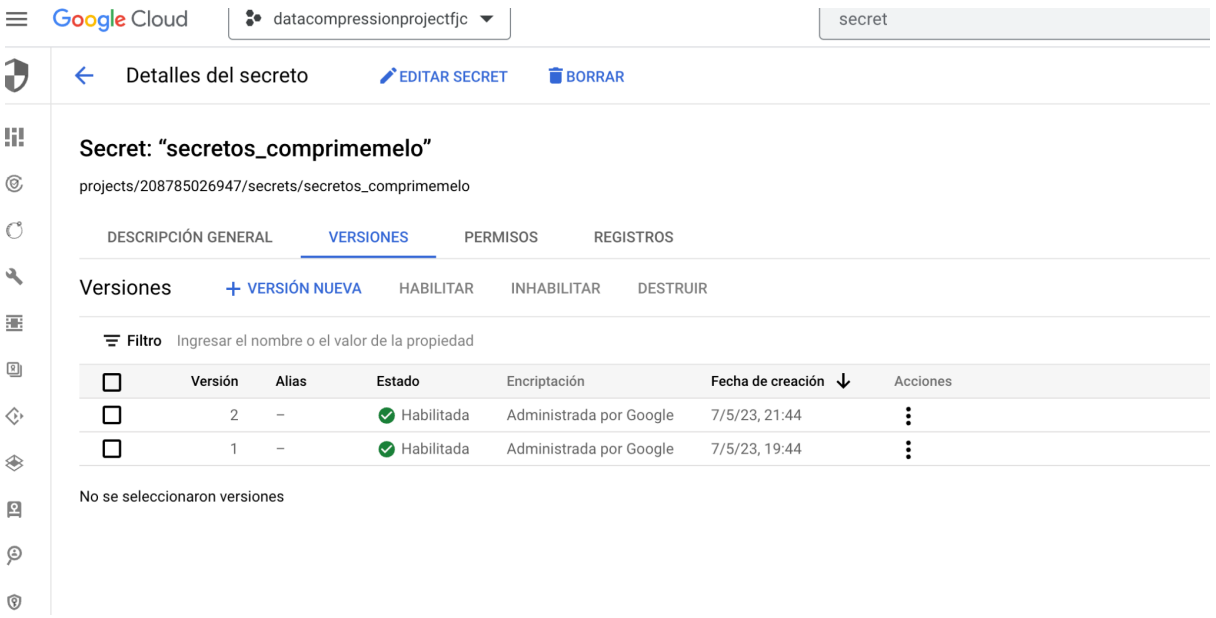
- **Pubsub y Secret manager configurado**

En el Cloud Pub/Sub utilizamos uno, con el Topic ID pubsubcomprimemelofiles, con el topic name projects**/datacompressionprojectfjc/topics/pubsubcomprimemelofile**s, de esa forma nosotros diseñamos el sistema de comunicación entre los servidores web y los procesos workers permitiendo que se comunicaran entre ellos. Esto es muy importante porque permite que se añadan o se creen las diferentes solicitudes para procesar nuevos archivos y los workers pueden procesar dicha cola.

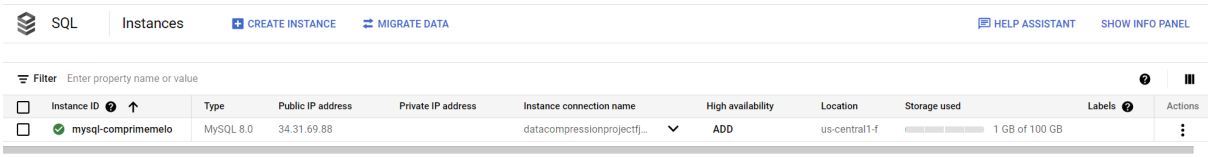


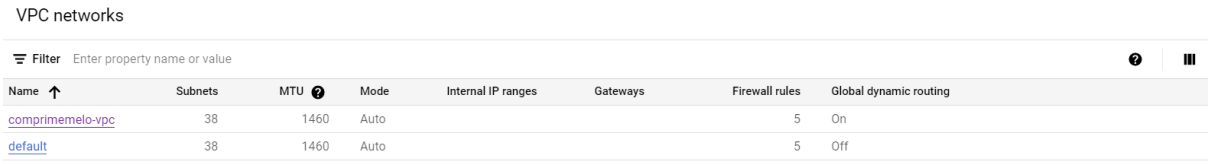
A continuación podemos ver como se encuentra configurado el servicio de pub/sub





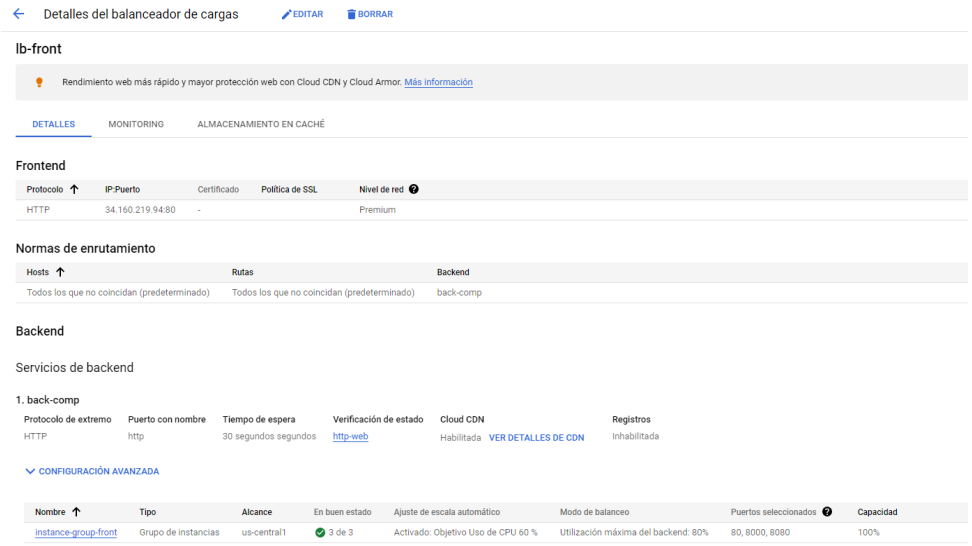
- **La base de datos Mysql conectada a la VPC interna y recibiendo los registros de la VM.**



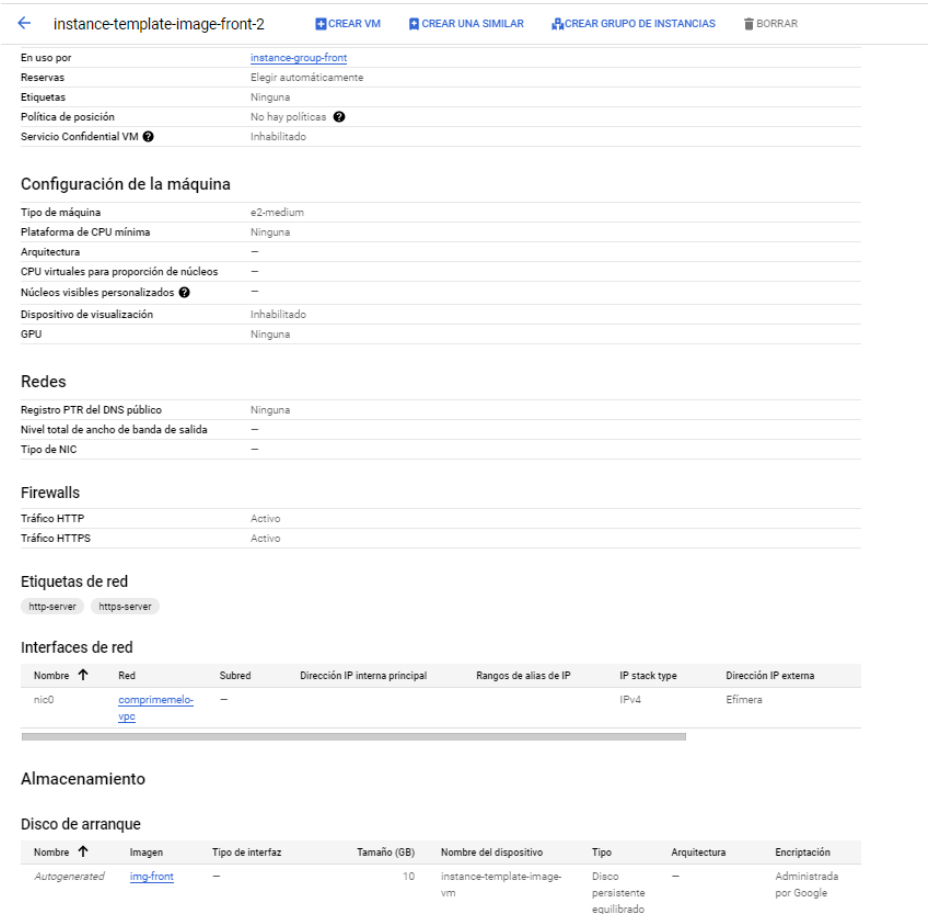


Configuración del balanceador de carga de instancia web

El balanceador de carga de red nos permite a nosotros colocar un puerto de origen y de destino, una dirección IP , como también asignar diferentes protocolos para el reenvío de paquetes, de esa forma podemos mejorar el resentimiento de la web



Plantilla de replicación de instancia web

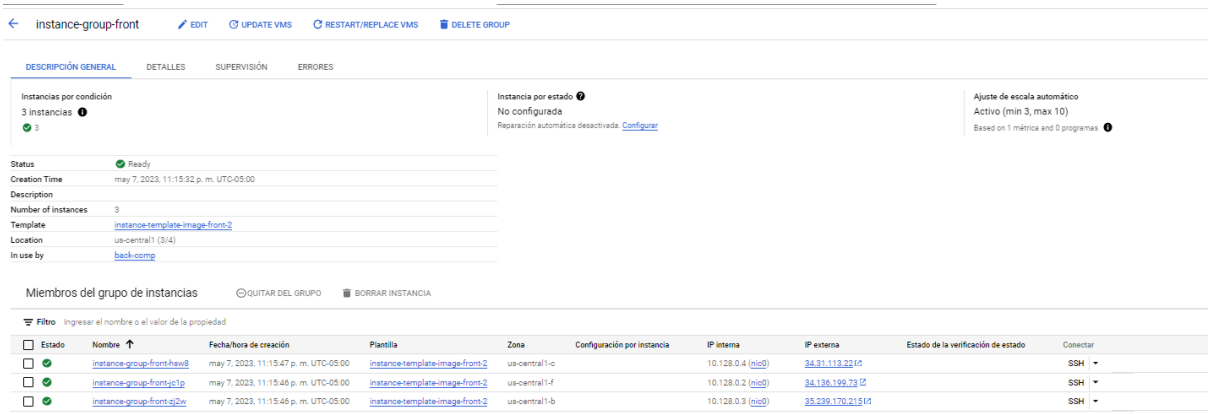


La plantilla nos permite obtener una copia de la imagen real de la instancia donde se encuentra configurada la aplicación web, la cual es usada en el grupo de instancias para la replicación en el autoescalamiento.

Adicionalmente con el uso de la plantilla podemos definir el tipo de máquina, imagen del disco de arranque o del contenedor, como también establecer una secuencia de comandos

de inicio. El uso de plantillas de instancias no está vinculado a una zona o región, sin embargo es recomendable especificar cuáles recursos serán utilizados en algunas zonas en específico.

Configuración de grupo de instancias

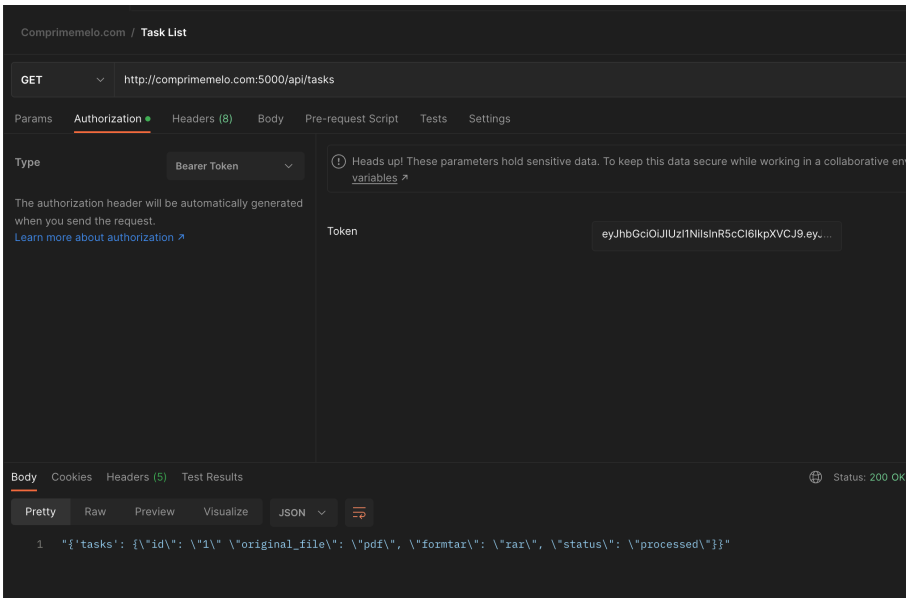


El grupo de instancias se basa en la plantilla que previamente se creó y se definen la cantidad mínima de instancias, ciclo de vida de instancias, configuración autoscaling y ubicación de las nuevas máquinas virtuales.

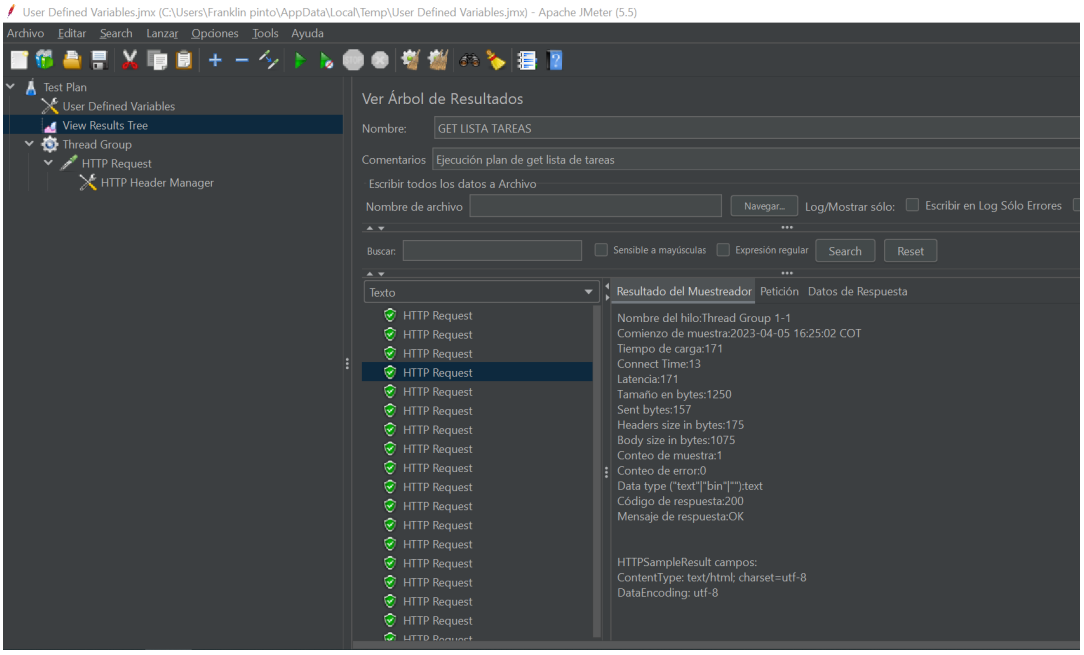
Pruebas de Escalabilidad

Prueba escenario 1. *Listar todas las tareas de conversión de un usuario*

El servicio entrega el identificador de la tarea, el nombre y la extensión del archivo original, a qué extensión desea convertir y si está disponible o no. El usuario debe proveer el token de autenticación para realizar dicha operación.



Se ejecuta el plan de pruebas desde JMETER



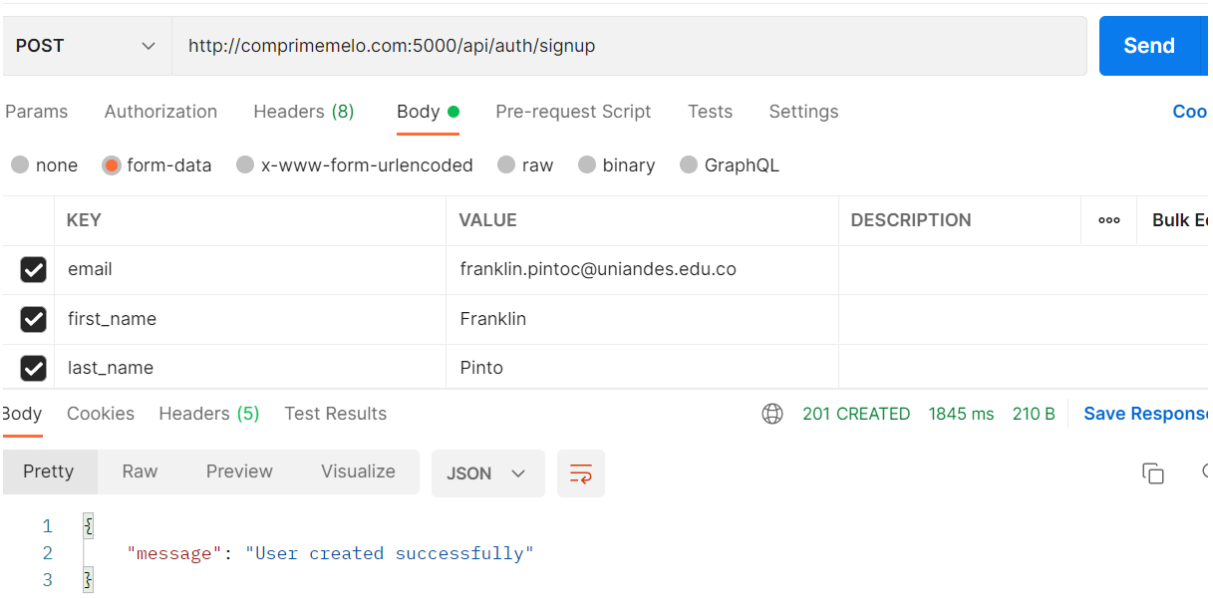
Con 100,500 y 1000 peticiones en concurrencia no falló ninguna petición mientras que al usar 2000 comenzó a presentar fallos por timeout. Sin embargo, a nivel funcional se afirma que está dentro de los límites, dado que el requerimiento menciona que debe soportar 1000 peticiones concurrentes para este servicio.

Prueba escenario 2. *Crear una cuenta de usuario en la aplicación*

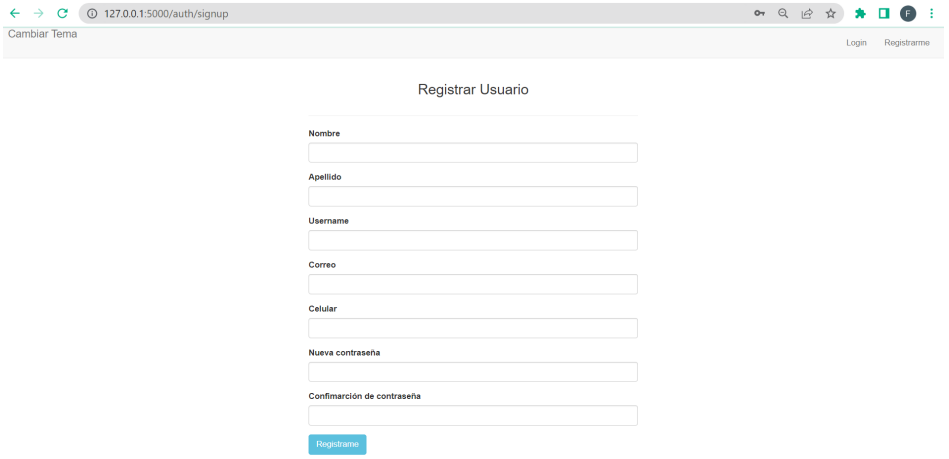
Para crear una cuenta se deben especificar los campos: usuario, correo electrónico y contraseña. El correo electrónico debe ser único en la plataforma dado que este se usa para la autenticación de los usuarios en la aplicación.

Para crear una cuenta de usuario es posible hacerlo a través de un formulario web ó consumiendo la api rest a través de un cliente http, ejemplo postman..

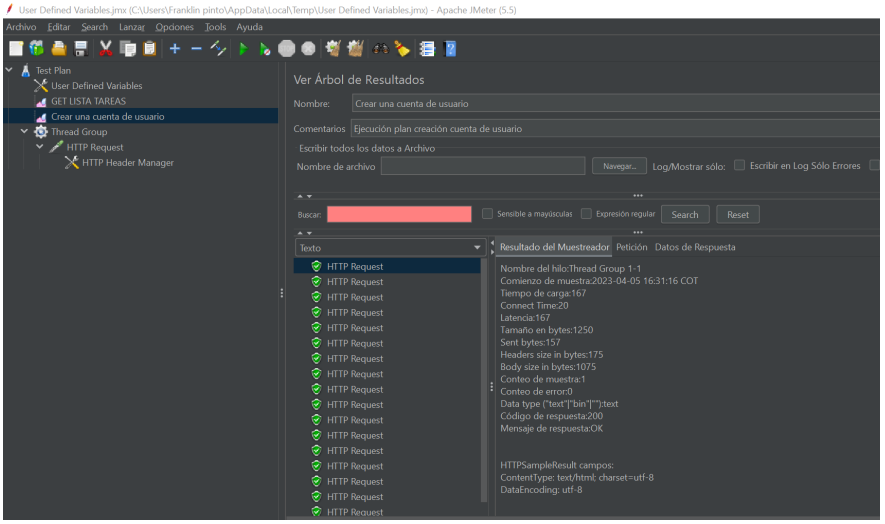
El siguiente endpoint con método post, permite crear una cuenta de usuario: http://comprimemelo.com:5000/api/auth/signup



o ingresando a la aplicación web en la siguiente url: http://comprimemelo.com:5000/api/auth/signup
presentará el siguiente formulario para el respectivo diligenciamiento de datos



Se ejecuta el plan de pruebas desde JMETER

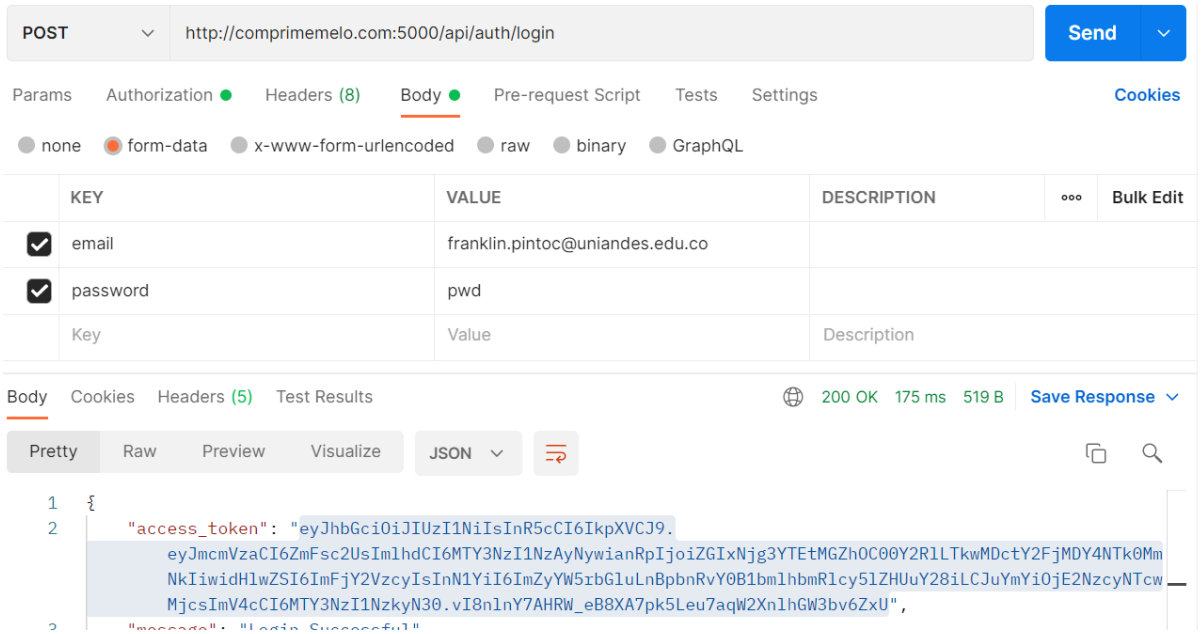


Para este escenario se identifica que al ejecutar 300 peticiones concurrentes el sistema comienza a responder con timeout, esto posiblemente se debe es por las validaciones de existencia del usuario, creación del mismo y envío de correo simultáneo. *Según los requerimientos del usuario se especifica que con 200 peticiones el sistema responde adecuadamente.*

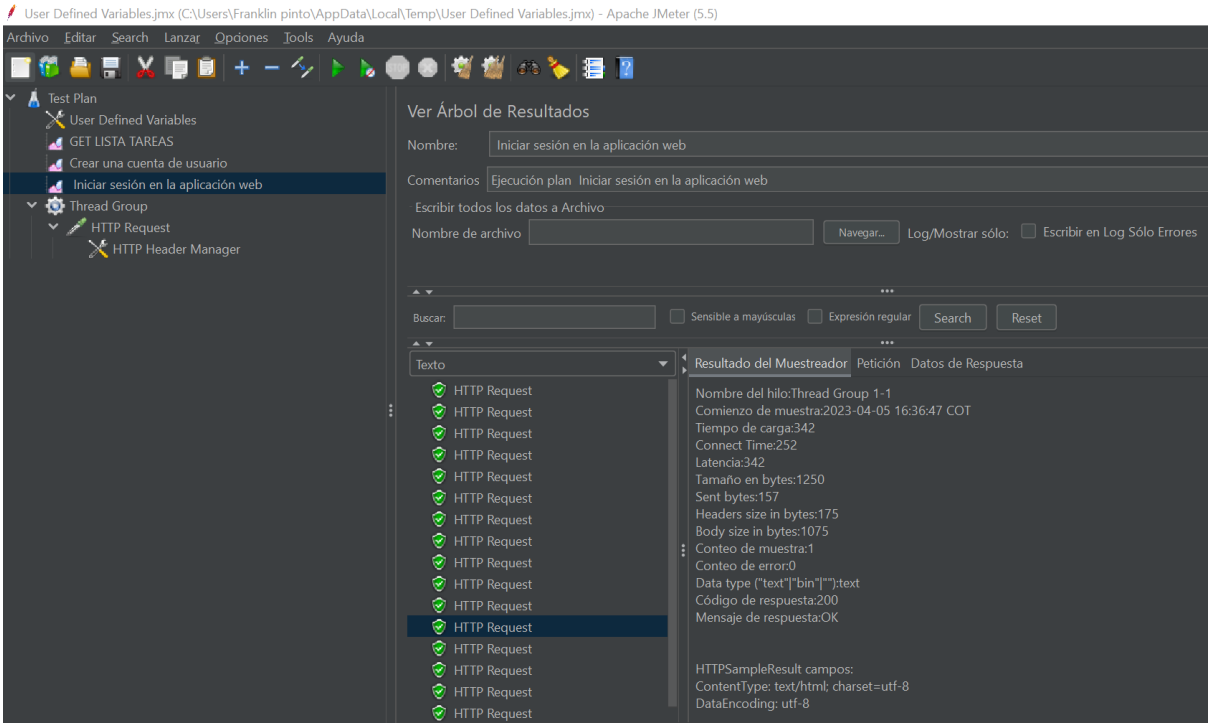
Prueba escenario 3. *Iniciar sesión en la aplicación web*

El usuario provee el correo electrónico/usuario y la contraseña con la que creó la cuenta de usuario en la aplicación. La aplicación retorna un token de sesión si el usuario se autenticó de forma correcta, de lo contrario indica un error de autenticación y no permite utilizar los recursos de la aplicación.

Para iniciar sesión en la aplicación se debe autenticarse ingresando las credenciales de usuario a través de la siguiente url http://comprimemelo.com:5000/api/auth/login
Para obtener el token de autenticación de usuario a través de la api rest debe usar el siguiente endpoint con metodo POST: http://comprimemelo.com:5000/api/auth/login



Se ejecuta el plan de pruebas desde JMETER



En esta prueba de concurrencia con 500 hilos comienza a presentarse timeout, hasta 350 peticiones concurrentes soporta sin presentar errores internos de servidor, sin embargo con las 350 peticiones se puede afirmar que cumple con los requerimientos del usuario.

5. Bibliografía

https://docs.celeryq.dev/en/stable/
https://flask.palletsprojects.com/en/2.2.x/
https://flask-jwt-extended.readthedocs.io/en/stable/