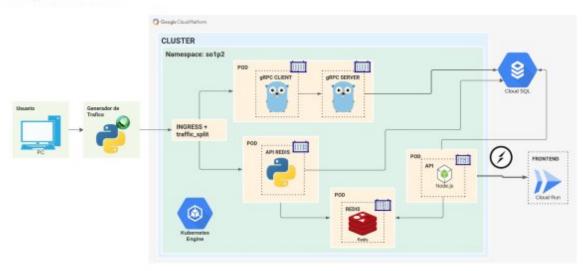
Manual Técnico

Arquitectura del proyecto:

Arquitectura



Herramientas utilizadas:

Para las diferentes API'S desarrolladas se utilizaron los lenguajes: NodeJS, y Python.

Para el cliente grpc y el servidor grpc se utilizó Golang.

Para almacenar la información se utilizó **MYSQL** para los datos estáticos y **REDIS** para los datos dinámicos.

Para el desarrollo de contenedores se utilizó la herramienta de Docker.

Se emplearon clústeres de Kubernetes de Google Cloud Platform (GCP) para la infraestructura en la nube.

Para alojar el frontend se utilizó un servicio de Cloud Run.

Se empleó **Locust** para simular el tráfico de datos.

Comandos utilizados

Para realizar el deployment de cada aplicación, se utilizó el comando *kubectl apply –f archivo.yaml*Estructura de un archivo .yaml utilizado en el proyecto:

```
api-mode > 6 api-mode yami

api-version: apps/v1

kind: Deployment

metadata:

ame: api-mode

name: api-mode

spec:

role: api-mode

spec:

role: api-mode

spec:

spec:

role: api-mode

spec:

spec:

role: api-mode

syrey:

spec: api-mode

name: api-mode

name: api-mode

particle below:

labels:

role: api-mode

particle api-mode

syrey: not linglydate

role: api-mode

particle api-mode

syrey: not linglydate

role: api-mode

particle api-mode

labels:

role: api-mode

particle api-mode

particle api-mode

labels:

role: api-mode

api-work api-mode

particle api-mode

particle api-mode

particle api-mode

api-work a
```

Para listar los servicios en ejecución dentro del clúster se utilizó el comando *kubectl get services –n nombreNamespace*. Ejemplo:

En este caso el namespace que generamos se llama so1p2.

```
mrrobot10@mrrobot10: $ kubectl get services -n solp2
NAME
                 TYPE
                                CLUSTER-IP
                                               EXTERNAL-IP
                                                                 PORT(S)
                                                                                  AGE
                 LoadBalancer
                                10.44.11.156
                                               35.224.108.103
                                                                 3004:30023/TCP
                                                                                  3h44m
apinode
                                                34.27.186.83
                                                                 5001:32556/TCP
apiredis
                 LoadBalancer
                                10.44.11.1
                                                                                  2d5h
                                10.44.4.93
                                                34.30.141.43
                                                                                  2d16h
clientgrpc
                 LoadBalancer
                                                                 3002:31320/TCP
ingresstraffic
                 LoadBalancer
                                10.44.7.195
                                                35.223.250.249
                                                                 4001:31841/TCP
                                                                                  4h46m
                                                                 3001:30669/TCP
                                                                                  2d17h
                 LoadBalancer
                                10.44.6.242
                                               34.121.200.86
servergrpc
```

Para listar los pods: *kubectl get pods –n so1p2*:

```
mrrobot10@mrrobot10:~$ kubectl get pods -n so1p2
NAME
                                   READY
                                           STATUS
                                                      RESTARTS
                                                                 AGE
apinode-879bcbd7d-5wmpm
                                   1/1
                                           Running
                                                      0
                                                                 3h51m
apinode-879bcbd7d-m49qv
                                   1/1
                                                                 3h51m
                                                      0
                                           Running
                                           Running
apiredis-84f5b8cb9b-pzx6c
                                   1/1
                                                      0
                                                                 9h
clientgrpc-7c88cf9669-j5vw4
                                   1/1
                                           Running
                                                      0
                                                                 2d16h
ingresstraffic-f94f444b6-b8pw6
                                   1/1
                                                      0
                                                                 4h54m
                                           Running
                                                      0
servergrpc-5f4c9968bb-4h648
                                   1/1
                                           Running
                                                                 2d17h
```

grpc

gRPC es un framework de código abierto desarrollado por Google que se utiliza para crear servicios y aplicaciones de comunicación de alta eficiencia en redes. A continuación, se destacan algunas de las ventajas y características clave de gRPC:

- Eficiencia en la comunicación: gRPC utiliza el Protocol Buffers (protobuf) como su lenguaje de descripción de interfaces, lo que permite una serialización binaria eficiente de datos. Esto reduce el tamaño de los datos transmitidos a través de la red y mejora la velocidad de transferencia.
- 2. Multiplataforma: gRPC es compatible con una variedad de lenguajes de programación, lo que facilita la creación de servicios y aplicaciones interoperables en diferentes entornos.
- 3. Protocolo HTTP/2: gRPC se basa en HTTP/2, lo que permite la multiplexación de múltiples solicitudes y respuestas en una sola conexión, reduciendo la latencia y mejorando el rendimiento.

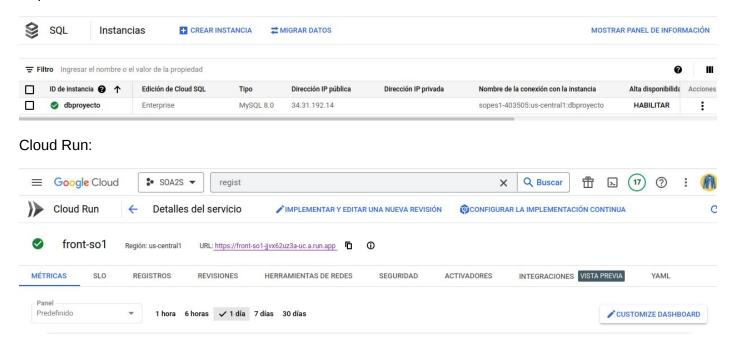
Archivo proto utilizado:

Comando utilizado para generar el archivo pb.go : *protoc proto/user.proto --go_out=plugins=grpc:.*

Script de la base de datos

```
CREATE TABLE nota(
idNota int auto_increment,
carnet varchar(25),
nombre varchar(100),
curso varchar(25),
nota int,
semestre varchar(25),
year varchar(25),
primary key(idNota)
);
```

Captura de la instancia:



Cargas de Trabajo:

