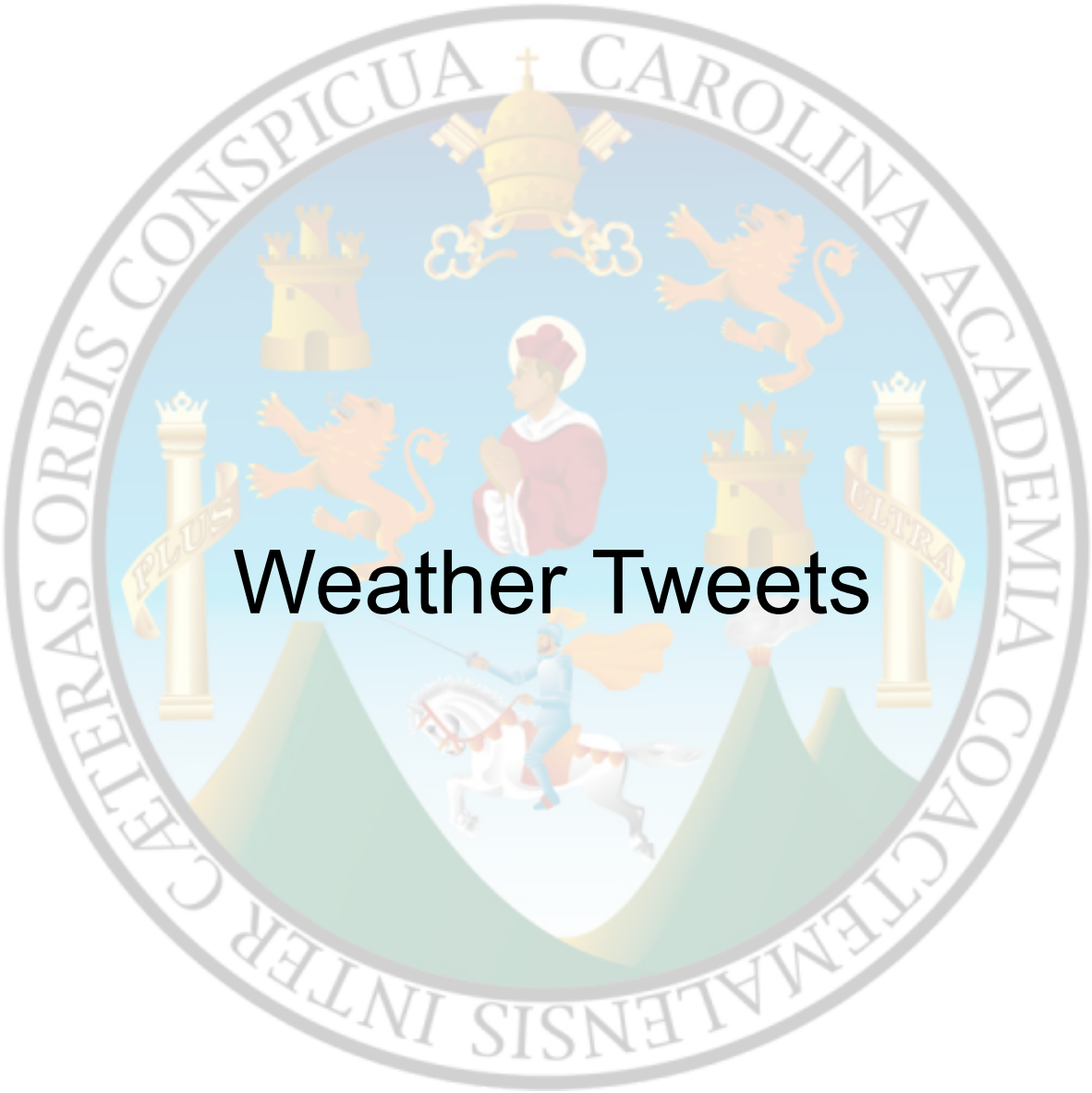
MANUAL TECNICO



Weather Tweets

Tabla de Contenido

Introducción

El proyecto “Weather Tweets” se enfoca en desarrollar una arquitectura distribuida para mostrar tweets relacionados con el clima a nivel mundial. Este sistema está diseñado para ser altamente escalable y promover la sostenibilidad ambiental mediante la monitorización del consumo de energía y emisiones de CO2. Utiliza tecnologías nativas de la nube y está implementado en un entorno de Kubernetes para facilitar la orquestación y gestión de contenedores.

Objetivos

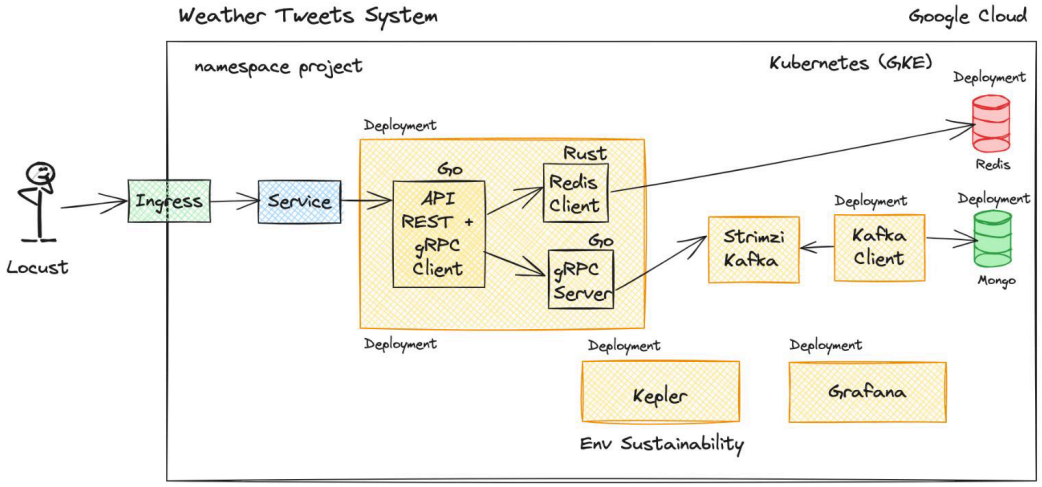
- Comprender la teoría de concurrencia y paralelismo para el desarrollo de sistemas distribuidos.
- Experimentar y evaluar tecnologías nativas de la nube que faciliten el desarrollo de sistemas distribuidos modernos.
- Diseñar estrategias para sistemas distribuidos que optimicen la respuesta ante alta concurrencia.
- Monitorear el consumo de energía y las emisiones de CO2 en despliegues Kubernetes, enfocándose en proyectos de sostenibilidad ambiental.
- Implementar contenedores y orquestadores en entornos de sistemas distribuidos.
- Utilizar comunicación contemporánea de microservicios mediante RPC.
- Desarrollar un sistema HPC para el procesamiento eficiente de mensajes.

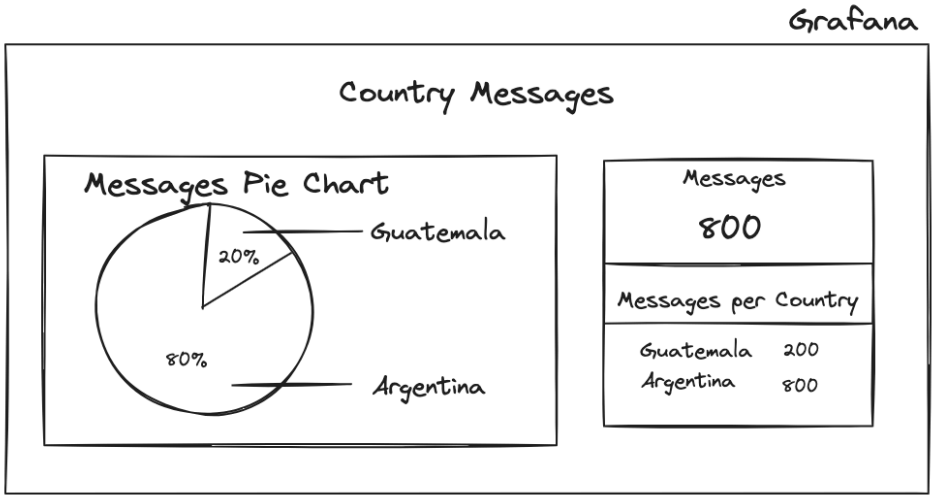
Resumen

Este proyecto tiene como objetivo crear una arquitectura distribuida que visualice tweets sobre el clima a nivel global, enfocándose en escalabilidad y sostenibilidad ambiental. Se busca demostrar la capacidad de gestionar alta concurrencia de tweets y medir el impacto en el consumo de energía y emisiones de CO2 generadas por las implementaciones del sistema.

Arquitectura

A continuación, se muestran imágenes de la arquitectura del proyecto.





Descripción

Implementamos una arquitectura moderna utilizando Kubernetes como plataforma base. Usamos Locust para simular tráfico concurrente hacia un dominio expuesto por NGINX. Una API escrita en Go procesa solicitudes HTTP y las envía a Kafka a través de gRPC. Kafka, gestionado con Strimzi en Kubernetes, recibe datos y los almacena en un tema denominado messages. Además, un contenedor Rust implementa WebSockets para insertar datos en Redis, que se utiliza para visualización en Grafana. Utilizamos Kepler, instalado con Helm, para monitorear consumo de energía y emisiones de carbono por Pod en Kubernetes.

Tecnologías

Locust

Es una herramienta de prueba de carga de código abierto que permite simular usuarios virtuales en un sistema para evaluar su rendimiento y escalabilidad bajo diferentes condiciones de carga.

gRPC

Es un framework de comunicación de código abierto desarrollado por Google que permite la comunicación eficiente y rápida entre servicios distribuidos utilizando el protocolo HTTP/2 y la serialización de datos en formato Protocol Buffers.

Rust

Es un lenguaje de programación de sistemas que enfatiza la seguridad, especialmente en concurrencia y gestión de memoria. Es especialmente útil para desarrollar software de alto rendimiento y de bajo nivel, como sistemas operativos, motores de juegos y aplicaciones embebidas.

Redis

Es una base de datos en memoria de código abierto que se utiliza comúnmente como almacenamiento en caché, almacén de claves-valor y cola de mensajes. Proporciona alta velocidad y rendimiento para operaciones de lectura y escritura de datos.

Kubernetes

Es una plataforma de código abierto diseñada para automatizar la implementación, escalado y administración de aplicaciones en contenedores. Proporciona un entorno de ejecución de aplicaciones altamente disponible y distribuido en clústeres de servidores.

Kafka

Es una plataforma de streaming de código abierto que proporciona una infraestructura escalable y tolerante a fallos para la transmisión y procesamiento de datos en tiempo real.

Mongo

MongoDB es una base de datos NoSQL de código abierto, orientada a documentos y altamente escalable. Almacena datos en documentos flexibles similares a JSON, lo que permite una fácil integración y manipulación de datos.

Grafana

Es una plataforma de análisis y visualización de código abierto que permite crear paneles dinámicos y gráficos interactivos para monitorear y analizar datos en tiempo real provenientes de diversas fuentes.

Kepler

Se refiere al uso del paquete Helm para desplegar Kepler en Kubernetes. Kepler (Kubernetes-based Efficient Power Level Exporter) es una herramienta de monitoreo que mide el consumo de energía y las emisiones de CO2 generadas por los despliegues en Kubernetes, proporcionando métricas para evaluar la sostenibilidad ambiental de las aplicaciones.

Deployments

En Google Cloud los archivos YAML son fundamentales para desplegar y gestionar aplicaciones en contenedores. A continuación se muestran cada uno de los archivos creados.

consumer.yml

Este archivo YAML define un Deployment que creará y gestionará un pod con un único contenedor. El contenedor ejecutará una aplicación basada en una imagen subida a docker hub, y se le asignan límites específicos de CPU y memoria. Este Deployment asegura que siempre haya una instancia de la aplicación ejecutándose y puede ser escalado fácilmente modificando el número de réplicas.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: consumer
  template:
    metadata:
      labels:
        app: consumer
    spec:
      containers:
      - name: consumer
        image: racs0/consumer-kafka
        resources:
          limits:
            cpu: "0.4"
            memory: "700Mi"

```

grafana.yml

Este archivo YAML define un Deployment que creará y gestionará un pod con un único contenedor que ejecuta Grafana. El contenedor se basa en una imagen, y se le asignan recursos específicos de CPU y memoria. El contenedor expondrá el puerto 3100 y se configurará mediante una variable de entorno para utilizar ese puerto HTTP.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grafana
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grafana
  template:
    metadata:
      labels:
        app: grafana
    spec:
      containers:
      - name: grafana
        image: grafana/grafana:8.4.4
        ports:
        - name: grafana
          containerPort: 3100
        env:
        - name: GF_SERVER_HTTP_PORT
          value: "3100"
        resources:
          limits:
            memory: "1Gi"
            cpu: "1000m"
          requests:
            memory: 500M
            cpu: "500m"
---
```

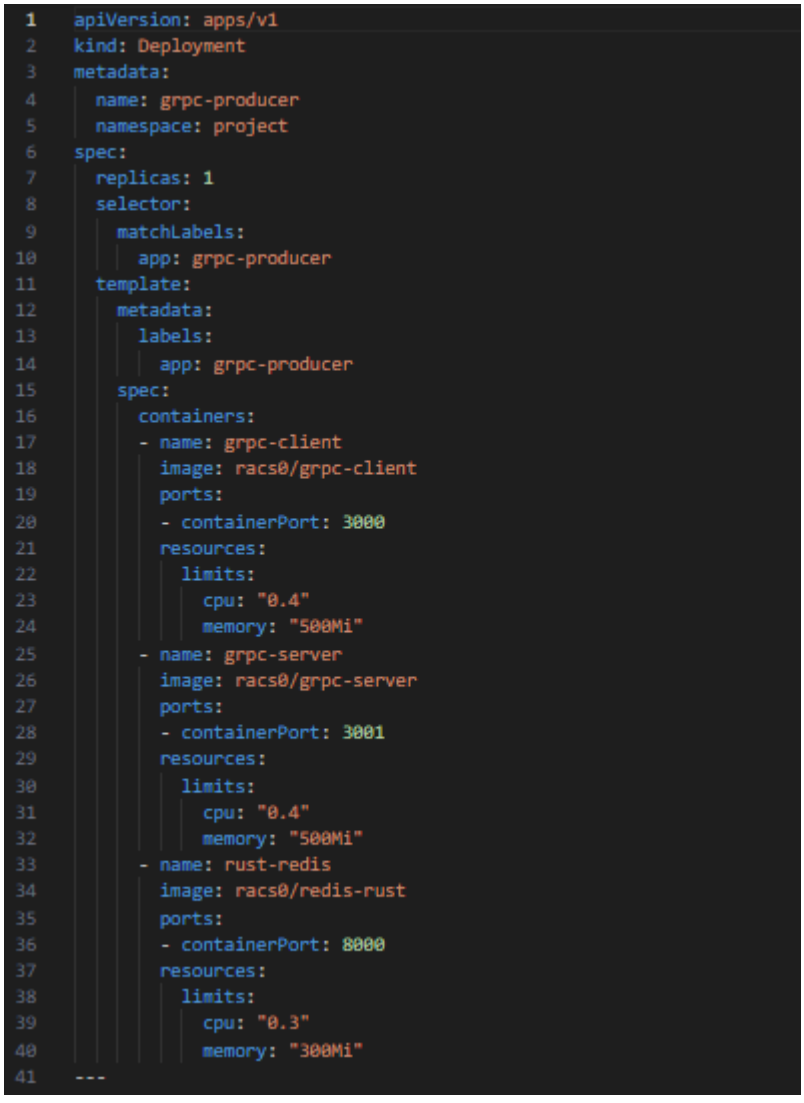
grpc.yml

Este archivo YAML define un Deployment que creará y gestionará un pod con tres contenedores:

grpc-client: Un cliente gRPC que escucha en el puerto 3000.

grpc-server: Un servidor gRPC que escucha en el puerto 3001.

rust-redis: Un servidor Redis escrito en Rust que escucha en el puerto 8000.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: grpc-producer
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      app: grpc-producer
  template:
    metadata:
      labels:
        app: grpc-producer
    spec:
      containers:
      - name: grpc-client
        image: racs0/grpc-client
        ports:
        - containerPort: 3000
        resources:
          limits:
            cpu: "0.4"
            memory: "500Mi"
      - name: grpc-server
        image: racs0/grpc-server
        ports:
        - containerPort: 3001
        resources:
          limits:
            cpu: "0.4"
            memory: "500Mi"
      - name: rust-redis
        image: racs0/redis-rust
        ports:
        - containerPort: 8000
        resources:
          limits:
            cpu: "0.3"
            memory: "300Mi"
---
```

ingress.yml

Este archivo YAML define un Ingress que expone dos servicios al mundo exterior, grpc-client-service y grafana. El Ingress utiliza reglas de enrutamiento y el tipo de ruta Prefix para enrutar el tráfico entrante a los servicios correspondientes.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: so1-proyecto2-ingress
  namespace: project
spec:
  ingressClassName: nginx
  rules:
    - host: 34.66.144.235.nip.io
      http:
        paths:
          - path: /input
            pathType: Prefix
            backend:
              service:
                name: grpc-client-service
                port:
                  number: 3000
          - path: /
            pathType: Prefix
            backend:
              service:
                name: grafana
                port:
                  number: 3100

```

mongo.yml

Este archivo YAML define un contenedor llamado mongo, que ejecuta la imagen de MongoDB de Docker Hub. Al contenedor se le asigna 0,5 unidades de CPU y 500Mi de memoria.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongo-db
  namespace: project
  labels:
    app: mongo-db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongo-db
  template:
    metadata:
      labels:
        app: mongo-db
    spec:
      containers:
      - name: mongo
        image: mongo
        ports:
        - containerPort: 27017
        resources:
          limits:
            cpu: "0.5"
            memory: "500Mi"
          requests:
            cpu: "0.3"
            memory: "256Mi"
---
```

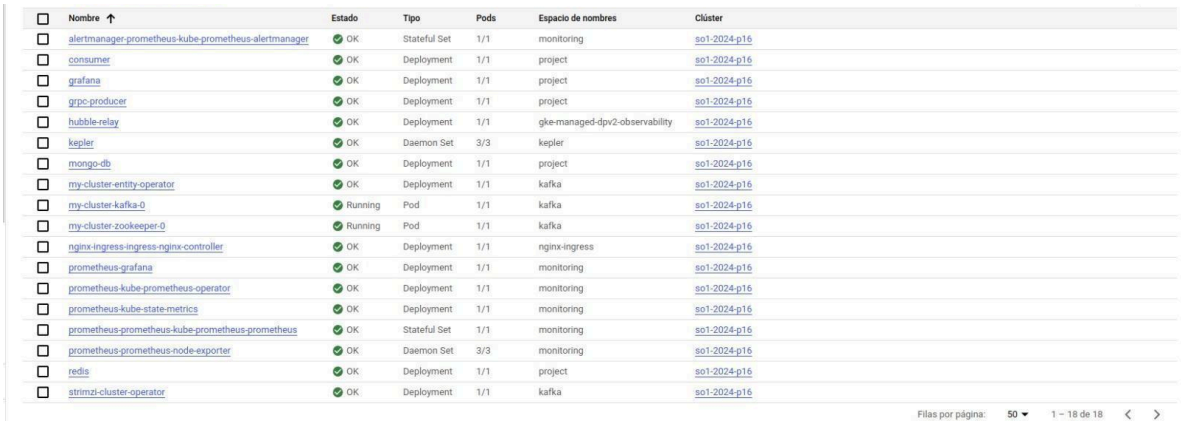
redis.yml

Este archivo YAML define un contenedor llamado redis, que ejecuta la imagen de redis de Docker Hub. Al contenedor se le asigna 0,2 unidades de CPU y 128Mi de memoria.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: redis
  name: redis
  namespace: project
spec:
  replicas: 1
  selector:
    matchLabels:
      run: redis
  template:
    metadata:
      labels:
        run: redis
    spec:
      containers:
      - name: redis
        image: redis:6.2
        ports:
        - containerPort: 6379
        resources:
          limits:
            cpu: "0.2"
            memory: "128Mi"
---
apiVersion: v1
kind: Service
metadata:
  labels:
    run: redis
  name: redis
  namespace: project
spec:
  ports:
  - port: 6379
    protocol: TCP
    targetPort: 6379
  selector:
    run: redis
  type: ClusterIP

```

A continuación, se muestra una imagen de los despliegues creados.



Nombre	Estado	Tipo	Pods	Espacio de nombres	Clúster
alertmanager-prometheus-kube-prometheus-alertmanager	OK	Stateful Set	1/1	monitoring	so1-2024-p16
consumer	OK	Deployment	1/1	project	so1-2024-p16
grafana	OK	Deployment	1/1	project	so1-2024-p16
grpc-producer	OK	Deployment	1/1	project	so1-2024-p16
hubble-relay	OK	Deployment	1/1	gke-managed-dpv2-observability	so1-2024-p16
kepler	OK	Daemon Set	3/3	kepler	so1-2024-p16
mongo-db	OK	Deployment	1/1	project	so1-2024-p16
my-cluster-entity-operator	OK	Deployment	1/1	kafka	so1-2024-p16
my-cluster-kafka-0	Running	Pod	1/1	kafka	so1-2024-p16
my-cluster-zookeeper-0	Running	Pod	1/1	kafka	so1-2024-p16
nginx-ingress-ingress-nginx-controller	OK	Deployment	1/1	nginx-ingress	so1-2024-p16
prometheus-grafana	OK	Deployment	1/1	monitoring	so1-2024-p16
prometheus-kube-prometheus-operator	OK	Deployment	1/1	monitoring	so1-2024-p16
prometheus-kube-state-metrics	OK	Deployment	1/1	monitoring	so1-2024-p16
prometheus-prometheus-kube-prometheus-prometheus	OK	Stateful Set	1/1	monitoring	so1-2024-p16
prometheus-prometheus-node-exporter	OK	Daemon Set	3/3	monitoring	so1-2024-p16
redis	OK	Deployment	1/1	project	so1-2024-p16
strimzi-cluster-operator	OK	Deployment	1/1	kafka	so1-2024-p16

Filas por página: 50 1 – 18 de 18

grpc-producer

Al igual que con la aplicación de rust, aquí se tienen 2 contenedores en el pod, y estos cuentan con el client y server de Go.

redis

Aquí se tiene un pod que obtiene la imagen de Redis. Este tambien se coloco loadbalancer para utilizarla de manera más fácil.

mongo-db

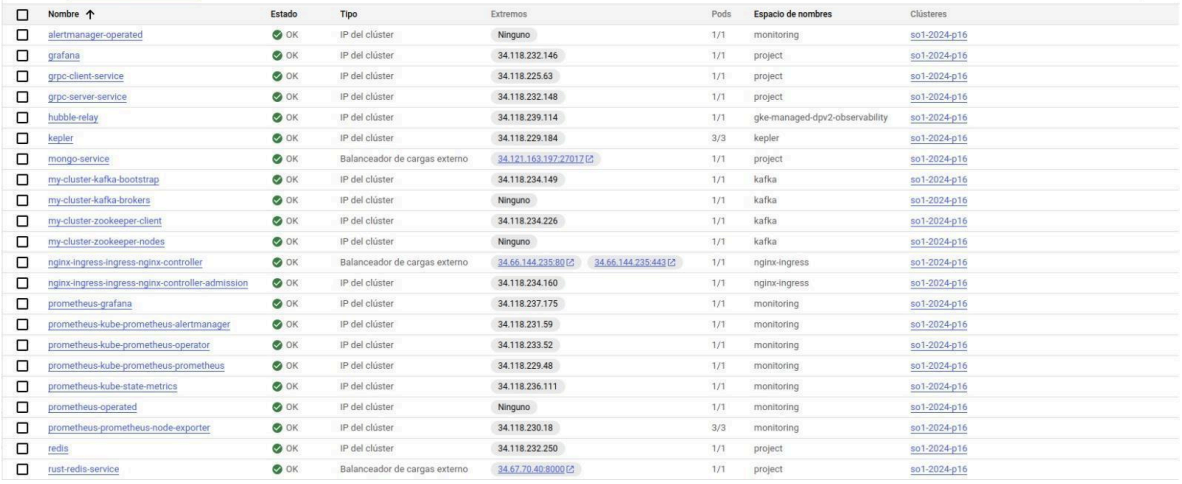
Aquí se tiene un pod que obtiene la imagen de mongoDB. Este se pone de tipo loadBalancer para poder entrar a la base de datos con la IP.

grafana

Este deployment cuenta con la imagen de grafana, la cual se utiliza para ver todos los contadores de redis.

Services

A continuación, se muestran los servicios para poder comunicar entre el cluster.



Nombre	Estado	Tipo	Extremos	Pods	Espacio de nombres	Clústeres
alertmanager-operated	OK	IP del clúster	Ninguno	1/1	monitoring	so1-2024-p16
grafana	OK	IP del clúster	34.118.232.146	1/1	project	so1-2024-p16
grpc-client-service	OK	IP del clúster	34.118.225.63	1/1	project	so1-2024-p16
grpc-server-service	OK	IP del clúster	34.118.232.148	1/1	project	so1-2024-p16
hubble-relay	OK	IP del clúster	34.118.239.114	1/1	gke-managed-dpv2-observability	so1-2024-p16
kepler	OK	IP del clúster	34.118.229.184	3/3	kepler	so1-2024-p16
mongo-service	OK	Balanceador de cargas externo	34.121.163.197:27017	1/1	project	so1-2024-p16
my-cluster-kafka-bootstrap	OK	IP del clúster	34.118.234.149	1/1	kafka	so1-2024-p16
my-cluster-kafka-brokers	OK	IP del clúster	Ninguno	1/1	kafka	so1-2024-p16
my-cluster-zookeeper-client	OK	IP del clúster	34.118.234.226	1/1	kafka	so1-2024-p16
my-cluster-zookeeper-nodes	OK	IP del clúster	Ninguno	1/1	kafka	so1-2024-p16
nginx-ingress-ingress-nginx-controller	OK	Balanceador de cargas externo	34.66.144.235:80 34.66.144.235:443	1/1	nginx-ingress	so1-2024-p16
nginx-ingress-ingress-nginx-controller-admission	OK	IP del clúster	34.118.234.160	1/1	nginx-ingress	so1-2024-p16
prometheus-grafana	OK	IP del clúster	34.118.237.175	1/1	monitoring	so1-2024-p16
prometheus-kube-prometheus-alertmanager	OK	IP del clúster	34.118.231.59	1/1	monitoring	so1-2024-p16
prometheus-kube-prometheus-operator	OK	IP del clúster	34.118.233.52	1/1	monitoring	so1-2024-p16
prometheus-kube-prometheus-prometheus	OK	IP del clúster	34.118.229.48	1/1	monitoring	so1-2024-p16
prometheus-kube-state-metrics	OK	IP del clúster	34.118.236.111	1/1	monitoring	so1-2024-p16
prometheus-operated	OK	IP del clúster	Ninguno	1/1	monitoring	so1-2024-p16
prometheus-prometheus-node-exporter	OK	IP del clúster	34.118.230.18	3/3	monitoring	so1-2024-p16
redis	OK	IP del clúster	34.118.232.250	1/1	project	so1-2024-p16
rust-redis-service	OK	Balanceador de cargas externo	34.67.70.40:8000	1/1	project	so1-2024-p16

grpc-client-service

Servicio tipo clusterIP para comunicar el deployment de grpc.

grpc-server-service

Servicio tipo clusterIP para comunicar el deployment de grpc.

rust-redis-service

Servicio tipo loadbalancer para comunicar la base de datos de redis.

Kafka

Son 4 servicios de kafka para poder hacer uso de la herramienta.

mongo-service

Servicio tipo loadbalancer para comunicar la base de datos de mongo.

grafana

Servicio para comunicar a través de un portFordwarding a los servicios de grafana, es de tipo ClusterIP

Ingress

A continuación, se muestra una imagen con los ingresos creados.

Conclusiones

1. El proyecto demuestra la implementación efectiva de una arquitectura distribuida utilizando una variedad de tecnologías modernas. Al combinar herramientas como Locust para pruebas de carga, Go para el desarrollo de APIs, Kafka para la mensajería, y gRPC para la comunicación entre microservicios, se logra una solución robusta y escalable capaz de manejar alta concurrencia y grandes volúmenes de datos en tiempo real. La elección de Redis y MongoDB para el almacenamiento de datos, junto con la visualización en tiempo real a través de Grafana, garantiza una monitorización eficiente y un acceso rápido a la información.
2. El uso de Kepler para medir el consumo de energía y las emisiones de CO2 de los despliegues en Kubernetes destaca el compromiso del proyecto con la sostenibilidad ambiental. Al integrar esta herramienta, se permite a los desarrolladores no solo monitorear el rendimiento del sistema, sino también evaluar y minimizar el impacto ambiental de sus aplicaciones. Esto refleja una tendencia creciente hacia la implementación de prácticas sostenibles en el desarrollo de software, alineándose con los objetivos de sostenibilidad del proyecto.
3. El proyecto enfatiza el uso de metodologías y herramientas modernas en el desarrollo de sistemas distribuidos. La utilización de Docker para la creación de contenedores, Helm para la gestión de paquetes en Kubernetes, y Strimzi para la configuración de clústeres de Kafka, asegura una implementación ágil y eficiente. Además, el uso de Git para la gestión de código y colaboración, junto con el uso de namespaces en Kubernetes para la organización, refleja las mejores prácticas en el desarrollo y despliegue de aplicaciones nativas de la nube.