Project1
Nombre del proyecto: 2023 Weather Tweets
Ubicación: Guatemala
Profesor: Sergio Méndez
Autores: Sergio Méndez

Contenido

DESCRIPCIÓN

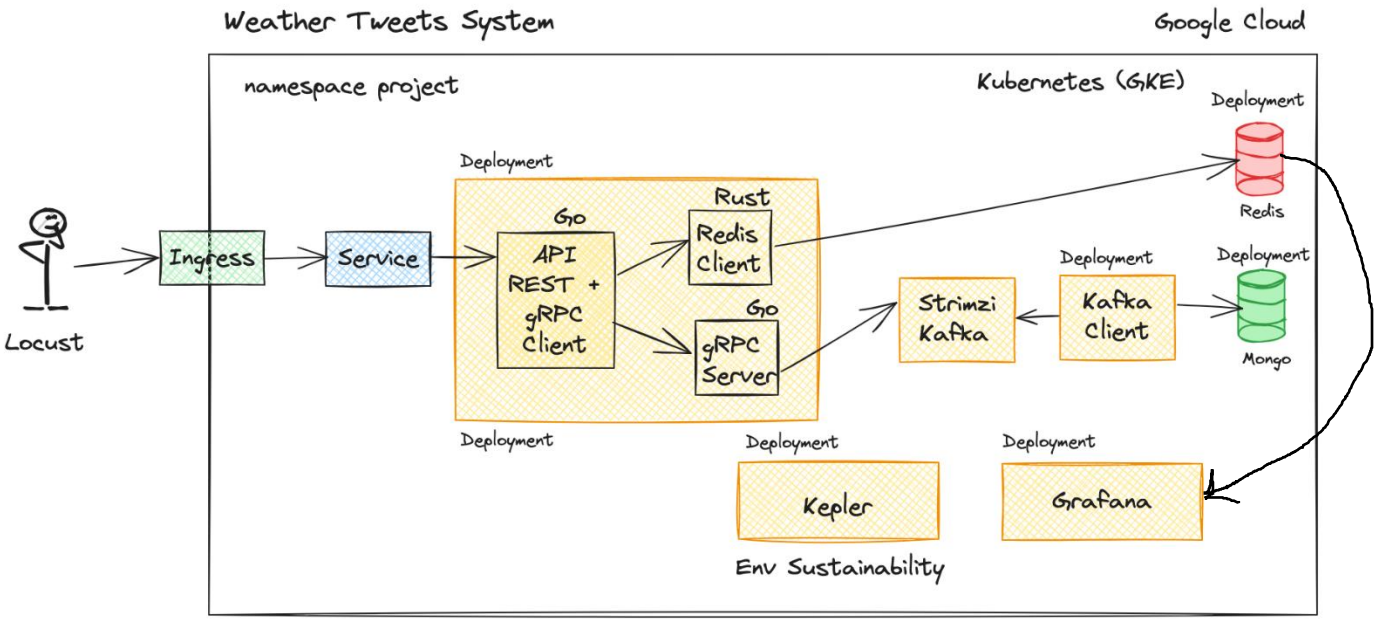
Cree una arquitectura de sistema distribuida genérica que muestre los tweets sobre el clima en todo el mundo. Esto se procesa mediante una arquitectura conceptual que puede ser escalable. Este proyecto pretende mostrar la concurrencia de tweets en el sistema. Además, mida la cantidad de energía de un CO2 utilizado por las implementaciones del proyecto.

OBJETIVOS

- Comprender la teoría de concurrencia y paralelismo para desarrollar sistemas distribuidos.
- Experimente y pruebe con tecnologías nativas de la nube que ayudan a desarrollar sistemas distribuidos modernos
- Diseñar estrategias de sistemas distribuidos para mejorar la respuesta de alta concurrencia.
- Monitorear la energía y el CO2 utilizados por los despliegues en Kubernetes para monitorear los proyectos de sostenibilidad ambiental que los estudiantes pueden experimentar.
- Implementar contenedores y orquestadores en sistemas distribuidos.
- Utilizar la comunicación contemporánea de microservicios RPC.
- Crear un sistema HPC para procesar mensajes.

EXPLICACIÓN ARQUITECTURA

En esta sección se explica cómo implementar la siguiente arquitectura:



Con esto vamos a explicar cómo se implementa.

paso 1(Locust + Go)

- Esta parte consiste en el uso de Locust para enviar tráfico concurrente. Este tráfico será recibido por equilibradores de carga públicos (entradas de k8s) en este caso:
  http://IP.nip.io/input , este dominio se expone mediante un controlador de ingreso, en este caso NGINX

- Locust debe utilizar parámetros de concurrencia para simular al menos 10 clientes concurrentes y 10000 solicitudes en total.

- Locust shoul leer un archivo JSON con datos aleatorios, en el siguiente formato:

```
{
"texto": "Su lluvia",
"país": "Argentina",
},
{
"texto": "Llueve de nuevo",
"país": "Argentina",
},
{
"texto": "El tiempo es tan agradable",
"país": "Guatemala",
}
```

API (Implementación)

En esta implementación y API creada con Go, recibe las solicitudes e inserta esta información en un tema de Kafka usando gRPC. En paralelo hay una vaina que consta de tres contenedores:

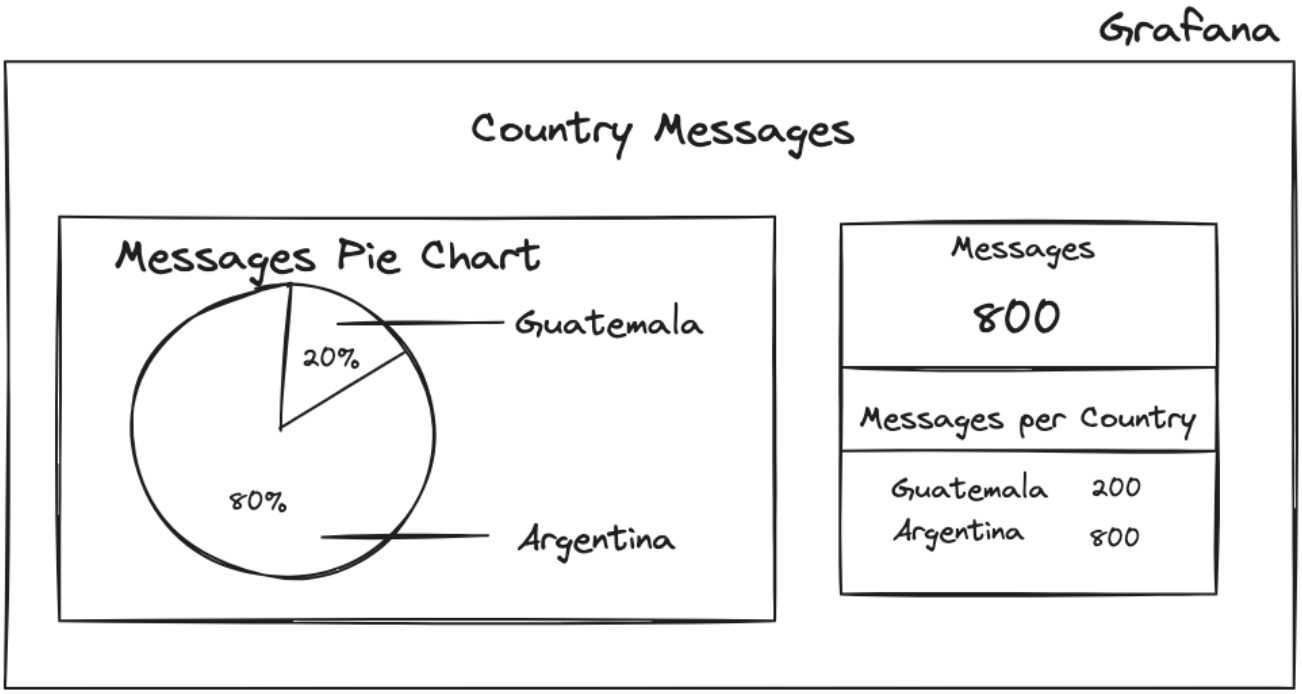
1. api container: Lee la información REST y llama al servidor gRPC usando un cliente gRPC y envía el texto y la información del país para ser almacenados en Mongo. Además, debe llamar a un contenedor Rust para insertar datos en Redis.

2. gRPC server container: Insertar datos en un tema de Kafka llamado **messages** cuando es llamado por su servicio. Esto utiliza el lenguaje Go.

3. Contenedor Rust: Esto implementa un WebSockets que escucha los datos para insertarlos en Redis. Esto utiliza el lenguaje Rust.

Grafana (Implementación)

Esta implementación funciona para visualizar los mensajes en el sistema de la siguiente manera:



Esta implementación utiliza Redis para visualizar estos datos, por lo que los datos insertados en Redis deben contener toda la información necesaria.

Un consejo rápido es usar tablas Hash en Redis para almacenar un contador de país y un contador total de mensajes.

Cliente Kafka (Implementación)

Debe crear un código de cliente Kafka, usando Go, usando las bibliotecas Kafka Confluent para crear un cliente que consuma mensajes de un tema llamado Message. Esto usa el lenguaje Go.

Kepler (Helm instalado)

Es necesario instalar Kepler para visualizar el consumo de energía y las emisiones de carbono por Pod. Esto debe instalarse utilizando Helm.

Strimzi (Kafka)

Debe configurar el clúster de Kafka utilizando Strimzi, un operador de Kubernetes que simplifica la instalación. Esto debe instalarse con Helm.

CONSIDERACIONES

- **Cloud and system**
  Tienes que crear tus propias imágenes de contenedor usando Docker y un clúster de Kubernetes en Google Cloud.

- **Bases de datos**
  Este proyecto utiliza Redis y Mongo DB por lo que debe instalarse utilizando implementaciones con su propio servicio para permitir el acceso de estas bases de datos dentro de la red interna de Kubernetes. De esta manera, las implementaciones pueden insertar datos en estas bases de datos.

- **GIT**
  Git será la forma de almacenar y versionar código en github. Git/Github será utilizado como una herramienta para crear un entorno colaborativo de desarrollo estudiantil.

- **NAMESPACE**
  Utilice namespace para organizar sus objetos Kubernetes, en este caso use **project***.

- **LOAD BALANCERS**
  - Utilice el controlador de ingreso NGINX para exponer sus API.
  - Esta parte es la manera de exponer la aplicación al mundo exterior.

- **RPC y Brokers**
  La idea principal en esta parte es crear una forma de alto rendimiento para escribir datos en bases de datos NoSQL, utilizando comunicación RPC (gRPC) y Brokers (Kafka).

  - gRPC:** Es un framework RPC de alto rendimiento que puede ejecutarse en cualquier entorno. Se usa primario para conectar servicios de backend.
  - Kafka:** Es un modo de sistema de cola HA para transmitir datos para aplicaciones en tiempo real.
  - Tenga en cuenta las siguientes preguntas:
  - ¿Cómo funciona Kafka?
  - ¿Qué es el comportamiento Kafka cuando se procesan datos? ¿Es lento?

- **Data**
  - Los registros tienen que ser almacenados en MongoDB.
  - Los datos en tiempo real deben almacenarse en Redis.
  - Replicar datos reales basados en Twitter(X) en los últimos días.

PENALIZACIONES

- El proyecto debe desarrollarse en dúos
- Deben implementarse con las entradas y los idiomas seleccionados
- Redactar un manual técnico y de usuario
- Si se encuentran copias, el dúo recibirá una puntuación de 0 puntos y se informará.
- No se aceptarán proyectos tardíos

ENTREGABLES

- Código fuente en Github
- Manuales en formato PDF

REFERENCIAS

- https://kubernetes.io/
- https://grpc.io/
- https://www.mongodb.com/
- https://redis.io/
- https://strimzi.io/quickstarts/
- https://docs.locust.io/en/stable/
- Covid Realtime Project: https://www.youtube.com/watch?v=IOMQ2ijkKQs
- gRPC Intro: https://www.youtube.com/watch?v=ftefB0t61w
- Bibliotecas Kafka confluentes: https://github.com/confluentinc/confluent-kafka-go
- Kepler: https://sustainable-computing.io/
- Código que puede usar: [view](yaml)