UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

SISTEMAS OPERATIVOS 1 SECCIÓN N

ING. JESUS GUZMAN POLANCO

AUX. ALVARO NORBERTO GARCÍA

AUX. SERGIO ALFONSO FERRER GARCÍA

SEGUNDO SEMESTRE 2024

PROYECTO 2

Olimpiadas USAC

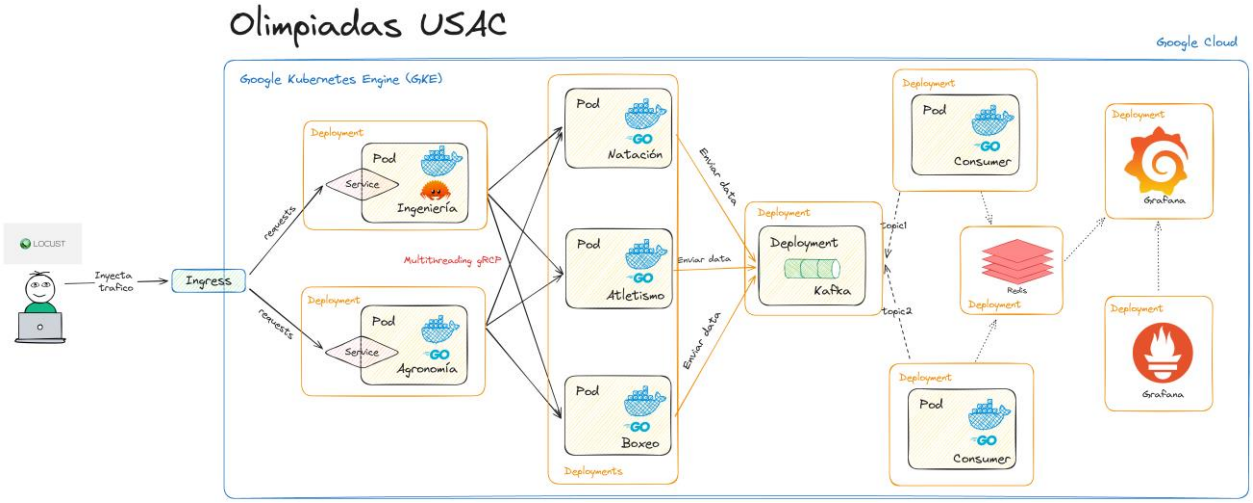
Objetivos

- Administrar una arquitectura en la nube utilizando Kubernetes en Google Cloud Platform (GCP).
- Utilizar el lenguaje de programación Golang, maximizando su concurrencia y aprovechando sus librerías.
- Crear y desplegar contenedores en un Container Registry.
- Entender el funcionamiento de un message broker utilizando Kafka.

Introducción

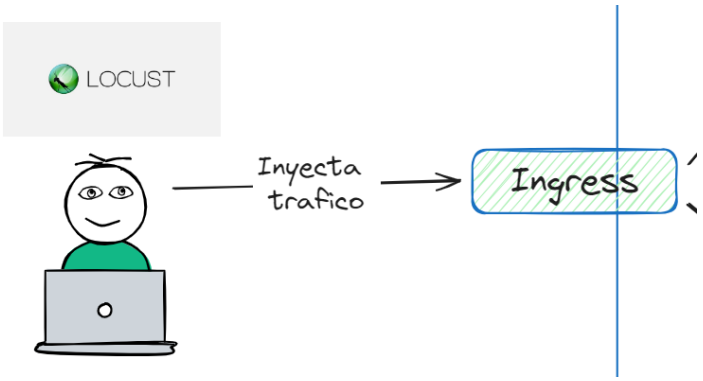
Este proyecto tiene como propósito aplicar los conocimientos adquiridos en las unidades 1 y 2 mediante la implementación de una arquitectura en Google Cloud Platform (GCP) utilizando Google Kubernetes Engine (GKE). El sistema monitorizará las Olimpiadas de la Universidad de San Carlos de Guatemala. A través de Kubernetes y contenedores, se deberá desplegar una arquitectura en la nube capaz de soportar grandes volúmenes de tráfico generado por los participantes de las facultades de Ingeniería y Agronomía, que competirán en Natación, Boxeo y Atletismo. El sistema mostrará en tiempo real las medallas obtenidas por cada facultad utilizando Grafana para el análisis de datos.

Arquitectura



Enlace para acceder al recurso:
https://excalidraw.com/#json=xHeUPmwmGjOhWg5Cg9i9L,ZZTGNkzahYFcfwwpV6BrEQ

Locust (Generador de tráfico)



Locust es una herramienta para la generación de tráfico hecha con Python, la cual puede instalarse de manera local y será el encargado de generar el tráfico el cual será enviado al *ingress* (puerta de acceso) de la arquitectura desplegada en Kubernetes.

Notas:

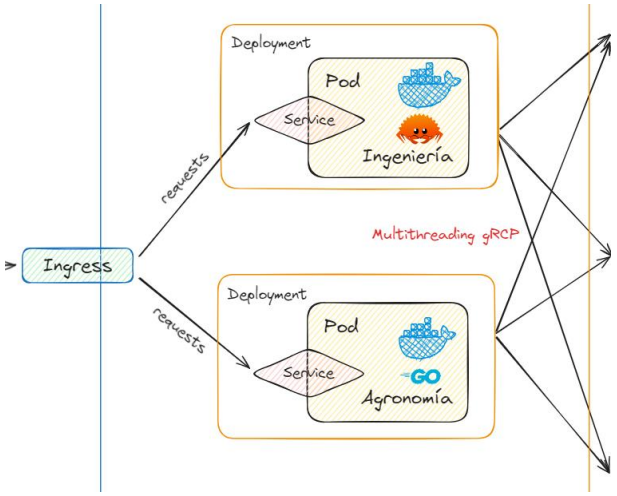
- La estructura JSON que se debe de manejar para enviar como *body* en las peticiones *HTTPS* es la siguiente:
    - *Faculty* se divide en dos tipos “Ingeniería” y “Agronomía”
    - *Discipline* corresponde de la siguiente manera
        - 1 = Natación
        - 2 = Atletismo
        - 3 = Boxeo

```
{
  "student": "Alvaro García",
  "age": 20,
  "faculty": "Ingeniería",
  "discipline": 1
}
```

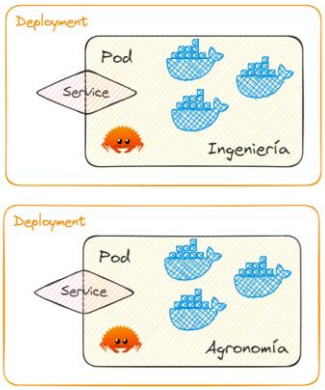
Sugerencias:

- Como mínimo manejar arreglos de 10,000 objetos para tener una mejor visualización.

Deployments de los servidores de las facultades



Con ayuda del *ingress* el tráfico que ingrese mediante *Locust* será redirigido dependiendo de la facultad a la que pertenece cada alumno (objeto). Cada servicio debe recibir la petición luego utilizando *threads* y gRPC para enviar al alumno a la disciplina que desea competir. Además, estos deployments podrán escalar horizontalmente según la demanda, la imagen de abajo demuestra cómo se verían estos deployments con auto escalado horizontal.



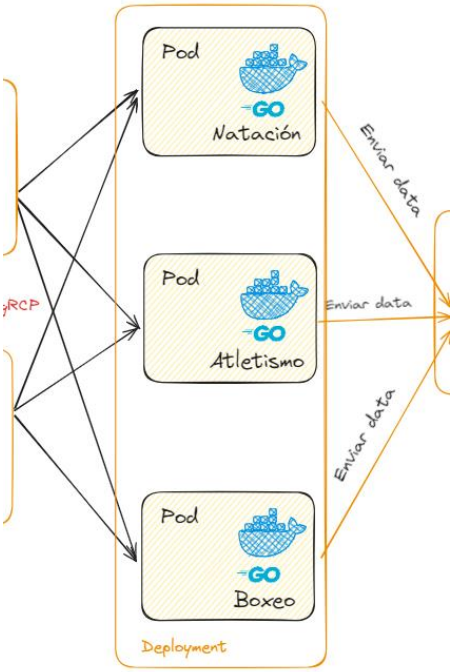
Notas:

1. Un deployment contendrá un pod con un contenedor hecho con el lenguaje de programación golang.
2. Un deployment contendrá un pod con un contenedor hecho con el lenguaje de programación Rust.
3. En el caso de golang utilizar *channels* o *go routines* para enviar mediante gRPC los alumnos a los servidores de las disciplinas correspondientes.
4. En el caso de rust utilizar *threads* para enviar mediante *gRPC* los alumnos a los servidores de las disciplinas correspondientes.
5. La estructura de los archivos. proto queda a discreción del estudiante, pero es importante mantener los datos del alumno como: la facultad que pertenece y la disciplina que aplica.
6. Configurar un HPA (*Horizontal Pod Autoscaler)* para aumentar el número de pods cuando se sobrepase los recursos.
7. Cada deployment deberá de tener su respectivo servicio para poder dirigir el tráfico correspondiente.

Sugerencias:

- Se recomienda crear por cada *deployment* su *service* correspondiente.
- Versionar las imágenes de los contenedores que se van a subir al *registry*.
- En el caso de estos dos *deployments* actuarán como clientes de las peticiones *HTTPS* y como clientes de *gRPC.*
- Deberá asignarle pocos recursos a los pods para provocar un escalado horizontal.

Deployments de los servidores de las diferentes disciplinas



Estos deployments contendrán contenedores en Go, los cuales representarán las disciplinas. Los alumnos serán enviados a estos deployments mediante gRPC. Cada servidor ejecutará un algoritmo de probabilidad sencillo (como lanzamiento de moneda) para decidir si el alumno es ganador. Los ganadores y perdedores serán enviados a una cola de Kafka con su respectivo tópico.

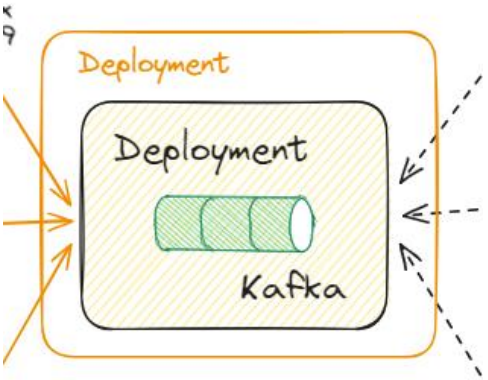
Notas

1. Al momento que una petición llegue al servidor, una función será la encargada de decidir si este alumno fue o no el ganador, la manera en que la función decidirá que alumnos son ganadores o perdedores, es utilizando un algoritmo sencillo de probabilidad como por ejemplo *lanzamiento de moneda* donde pueden existir dos estados, ganador o perdedor.
2. En el caso de que el alumno sea el ganador, este será enviado a las colas de Kafka utilizando el tópico “winners” para las tres disciplinas (servidores).
3. En el caso de que el alumno sea el perdedor, este será enviado a las colas de kafka utilizando el tópico “losers” para las tres disciplinas (servidores).
4. Configurar un HPA (*Horizontal Pod Autoscaler)* para aumentar el número de pods cuando se sobrepase los recursos.
5. Cada deployment deberá de tener su respectivo servicio para poder recibir el trafico de los servidores de la facultad, estos servicios deben funcionar con gRPC.

Sugerencias:

- En el caso de estos dos *deployments* actuarán como servidores de las peticiones *gRPC* y como *publishers* para las colas de Kafka.
- Se recomienda configurar los deployments de tal manera que “provoquen” un escalado automático horizontal.

Deployment de Kafka



Apache Kafka es una plataforma de streaming distribuida de código abierta diseñada para gestionar flujos de datos en tiempo real de forma rápida. Kafka cuenta con muchas características y la que será esencial para esta arquitectura será *Publisher* y *Subscriber.* En este caso los servidores de las disciplinas (*Publishers)* serán los encargados de generar y publicar los mensajes a las colas de Kafka, separadas mediante tópicos, del otro lado estarán los consumidores escuchando los nuevos mensajes entrantes para ser procesados.

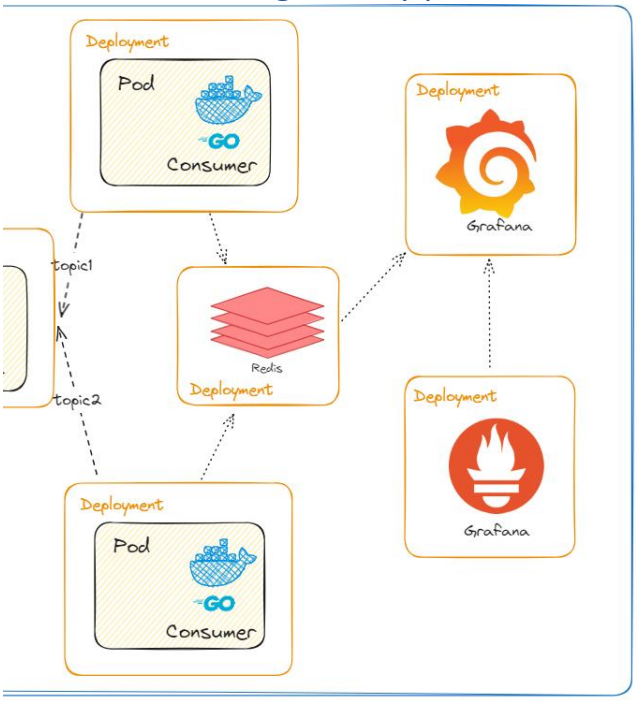
Notas:

1. Manejar un mismo tópico tanto para los publicadores como consumidores para que los mensajes convivan en la misma cola.
2. Enviar a Kafka información del alumno que resultó ganador o perdedor de dicha disciplina.

Sugerencias:

- Se recomienda utilizar Strimnzi para el deploy de los pods y servicios de Kafka.

Deployments de consumidores, redis, grafana y prometheus



Estos *deployments* representan la parte final de la arquitectura y son fundamentales para la correcta visualización de los datos en tiempo real y monitoreo.

Después que los datos hayan sido despachados a Kafka y sus colas, estarán dos deployments consumiendo todos los mensajes de las respectivas colas de los tópicos asignados en paralelo sin repetir los datos. Posteriormente estos datos serán guardados en redis y consumidos por Grafana donde se podrá visualizar en tiempo real la gráfica de los alumnos por facultad y disciplinas.

Por último, se tendrá un monitoreo del Cluster. Prometheus es un sistema de monitorización de código abierto diseñado para recopilar métricas de forma automática, dentro de un Cluster de Kubernetes este proporcionará una visión detallada del estado de los sistemas. Estas métricas deben visualizarle en Grafana mediante la creación de dashboards.

Notas:

1. Deberá utilizar 5 deploysments para los consumidores, redis, grafana y prometheus.
2. Los consumidores deberán conectarse a Redis para guardar la información entrante.
3. En grafana se deberá mostrar 3 dashboards sobre los alumnos de la siguiente forma:
    a. Conteo de alumnos por facultad.
    b. Conteo de las disciplinas de donde provienen los ganadores.
    c. Conteo de alumnos por facultad que perdieron.
4. En grafana se deberá mostrar 3 dashboards sobre promethues de la siguiente forma:
    a. Resumen del Clúster que contenga los nodos, pods, conectores y servicios.
    b. Rendimiento de las aplicaciones para ver las peticiones http y latencia.
    c. Recursos del sistema como la CPU, Memoria y Disco.

Sugerencias:

- Utilizar hash para poder organizar de mejor manera los datos y que sean fáciles de representar en gráfana.
- Para los dashboards de prometheus se recomienda utilizar los templates utilizados por la comunidad para evitar hacer PromQL.
- Se recomienda el uso de Helm para el despliegue de Grafana, Redis y Prometheus.

Requisitos Mínimos

- Documentación de los deployments y una breve explicación con ejemplos.
- Cluster de Kubernetes en GCP.
- Servidores de facultades, disciplinas.
- Sistema de streaming con sus consumers.

Restricciones

- El proyecto se realizará de forma individual.
- La información que ingrese será generada por Locust.
- Utilizar GKE.
- Utilizar gRCP para los servidores de las facultades y disciplinas.

Github

- El código fuente debe de ser gestionado por un repositorio privado de Github
- Crea una Carpeta en el repositorio llamado: Proyecto2
- Agregar a los auxiliares al repositorio de GitHub: **AlvaroG13191704** y **SergioFerrer9**

Calificación

- Al momento de la calificación se verificará la última versión publicada en el repositorio de GitHub
- Cualquier copia parcial o total tendrán nota de 0 puntos y serán reportadas al catedrático y a la Escuela de Ciencias y Sistemas.
- Si el proyecto se envía después de la fecha límite, se aplicará una penalización del 25% en la puntuación asignada cada 12 horas después de la fecha límite. Esto significa que, por cada período de 12 horas de retraso, se reducirá un 25% de los puntos totales posibles. La penalización continuará acumulándose hasta que el trabajo alcance un retraso de 48 horas (2 días). Después de este punto, si el trabajo se envía, la puntuación asignada será de 0 puntos.

Entregables

- La entrega se debe realizar antes de las 23:59 del 01 de noviembre de 2024.
- La forma de entrega es mediante UEDI subiendo el enlace del repositorio.
- Manual técnico en formato Mark Down o PDF.

Referencias

- https://github.com/AlvaroG13191704/SO1_S2_2024
- operating-systems-usac-course/lang/en/projects/projects.md at master · sergioarmgpl/operating-systems-usac-course · GitHub
- https://devopscube.com/setup-prometheus-monitoring-on-kubernetes/