UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

SISTEMAS OPERATIVOS 1 SECCIÓN N

ING. JESUS GUZMAN POLANCO

AUX. JOSÉ DANIEL VELÁSQUEZ OROZCO

AUX. JHONATHAN DANIEL TOCAY

PRIMER SEMESTRE 2024

# PROYECTO 2

# Sistema Distribuido de Votaciones

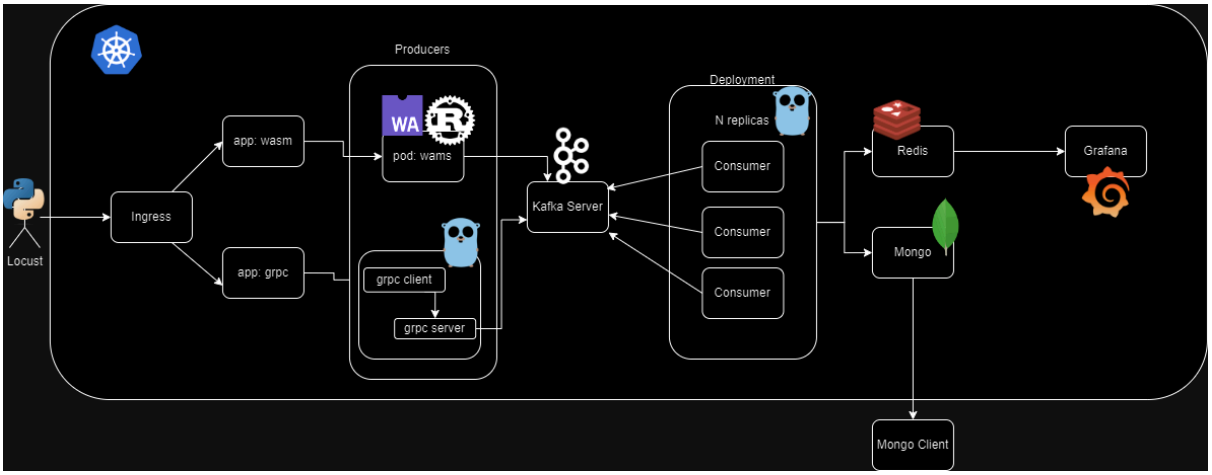
## Objetivos

- Implementar un sistema distribuido con microservicios en kubernetes.
- Encolar distintos servicios con sistemas de mensajerías.
- Utilizar Grafana como interfaz gráfica de dashboards.

## Introducción

En este proyecto, se tiene como objetivo principal implementar un sistema de votaciones para un concurso de bandas de música guatemalteca; el propósito de este es enviar tráfico por medio de archivos con votaciones creadas hacia distintos servicios (grpc y wasm) que van a encolar cada uno de los datos enviados, así mismo se tendrán ciertos consumidores a la escucha del sistema de colas para enviar datos a una base de datos en Redis; estos datos se verán en dashboards en tiempo real. También se tiene una base de datos de Mongodb para guardar los logs, los cuales serán consultados por medio de una aplicación web.

# Arquitectura




A continuación se describe cada uno de los módulos y servicios de la arquitectura:

## Locust

Es un generador de tráfico y su trabajo es enviar datos a los distintos servidores desplegados en Kubernetes. Esta tecnología debe ser desarrollada con python y será ejecutada de forma local. Se pide utilizar 2 bandas con 2 álbumes cada una. El archivo de entrada puede ser creado utilizando Mackaroo.

## Estructura de Datos

Dado que se trata de una competencia de bandas musicales guatemaltecas, se debe implementar la siguiente estructura para los datos que serán enviados por cada uno de los servicios:

```
{
 "name": "Name of the band",
 "album": "Name of the album",
 "year": "Year of release",
 "rank": "Ranking for this album"
}
```

## Kubernetes

Se debe utilizar un clúster de kubernetes en la plataforma de Google Cloud, el cual contenga productores y consumidores; también en ese entorno existirá el servidor de Kafka. Habrán dos productores; el propósito de ambos es hacer comparaciones de tiempos y sacar una conclusión entre los servicios que se piden, se pueden clasificar de la siguiente forma:

- ### Productores

  Los dos productores deben ser capaces de enviar la información a una cola de Kafka, en este caso tenemos:

  - gRPC

El servidor y cliente productor de gRPC debe ser programado en lenguaje Golang y existen en el mismo pod.

- Wasm

  El servidor productor de Web Assembly debe ser programador en lenguaje Rust. Se recomienda utilizar wasmedge.

- Servidor de Kafka

  Se tendrá un servidor de Kafka dispuesto a recibir peticiones de los dos productores y encolarlas para luego ser puestas a disposición del consumidor. Se recomienda que el servidor Kafka sea instalado con Strimzi.

- Consumidor

  Se tendrá un daemon consumidor escrito en lenguaje Golang. Este estará deployado en un pod de 2 réplicas con un auto scaling hasta de 5 réplicas. Esta daemon a su vez estará alojando datos en ambas bases de datos por medio de rutinas de go.

**Nota:** Debe utilizar namespaces que usted considere para cada pod o servicio.

## Bases de Datos

- Redis

  Se tendrá una base de datos Redis construída con Docker la cual estará obteniendo los contadores que cada uno de los consumidores está enviando en tiempo real.

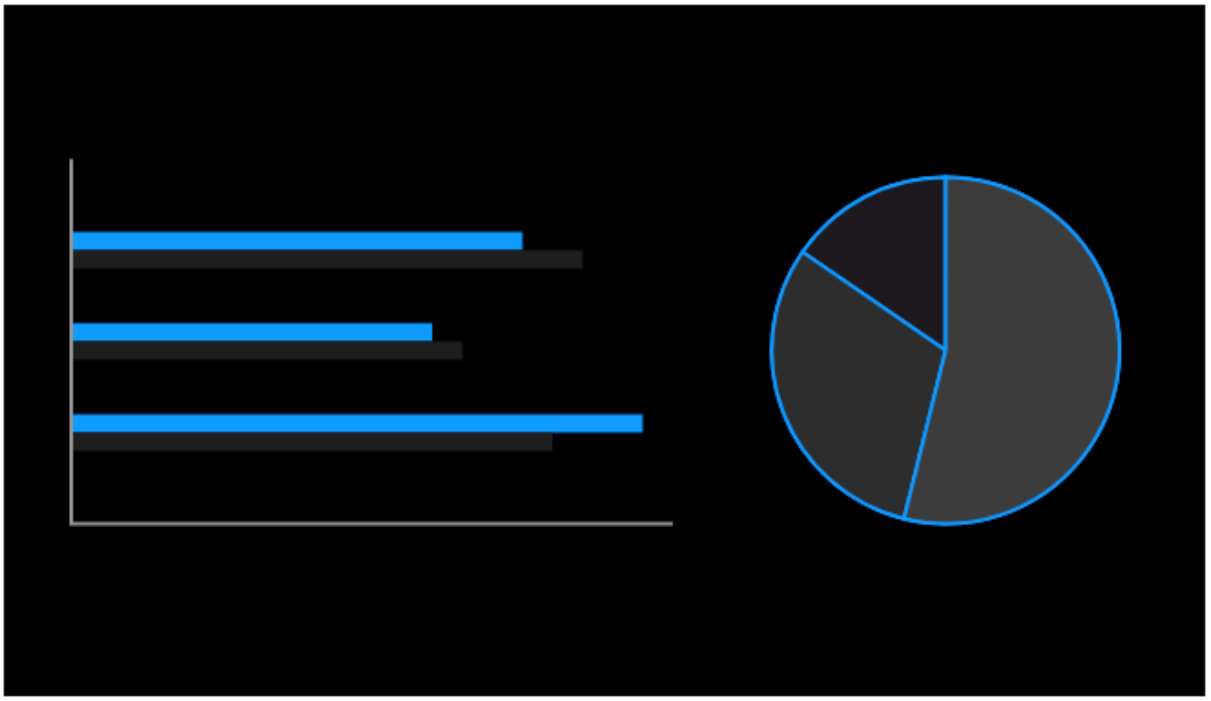
- MongoDB

  Se tendrá un clúster de MongoDB construído con Docker el cual estará recibiendo cada uno de los logs que más adelante serán consultados mediante una webapp.

**Nota:** Las bases de datos deben subsistir en el mismo cluster de kubernetes.

## Grafana

Este será el sistema de dashboards para los contadores en tiempo real guardados en la base de datos, por lo que ésta debe estar conectada a Redis. El dashboard de Grafana debe mostrar por medio de dos gráficas diferentes (queda a discreción del estudiante los tipos de gráfica) como van fluyendo las votaciones en tiempo real. Por ejemplo:



## Cloud Run

Se tendrá una api en Node y una webapp con Vue JS en la cual se podrán observar los registros de los logs de MongoDB. Debe estar desplegada en el puerto 80 utilizando proxies. Ambas deben estar en servicios de Cloud Run. La aplicación web debe mostrar los últimos 20 logs de la base de datos, este puede tener un aspecto parecido al de una terminal como se muestra a continuación:


## Preguntas a Resolver

- ¿Qué servicio se tardó menos? ¿Por qué?
- ¿En qué casos utilizarías grpc y en qué casos utilizarías wasm?

## Requisitos Mínimos

- Todos los servicios deben estar en Google Kubernetes Engine a excepción de los que no se solicitan dentro.
- Documentación con:
    - Introducción
    - Objetivos
    - Descripción de cada tecnología utilizada
    - Descripción de cada deployment y service de K8S.
    - Ejemplo de funcionamiento con capturas de pantalla.
    - Conclusiones

## Restricciones

- El proyecto se realizará de forma individual.
- Las imágenes de los contenedores deben de ser publicadas de Docker Hub.

## 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: **DannyLaine158** y **JhonathanTocay2020**

## 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 26 de abril de 2024.
- La forma de entrega es mediante UEDI subiendo el enlace del repositorio.
- Manual técnico con explicación de todos los servicios y tecnologías utilizadas en el proyecto, así como su forma de uso clara y entendible.

## Referencias

- Repositorio de Ejemplo Wasm y Rust
- Repositorio de Ejemplo Strimzi y Kafka