Universidad San Carlos de Guatemala
Facultad de ingeniería.
Ingeniería en ciencias y sistemas

Arquitectura Distribuida en la Nube con Kubernetes

Competencia que desarrollaremos

Al finalizar este proyecto, el estudiante será competente en:

- Diseñar e implementar arquitecturas de microservicios en un entorno de nube (GCP).
- Orquestar contenedores utilizando Kubernetes (GKE), incluyendo deployments, services, ingress, y HPA.
- Desarrollar servicios concurrentes en Go y Rust, aprovechando sus librerías y paradigmas.
- Utilizar y configurar message brokers (Kafka) para la comunicación asíncrona entre servicios.
- Integrar y gestionar bases de datos en memoria (Valkey) para el almacenamiento de datos de alta velocidad.
- Implementar y utilizar un Container Registry (Zot) para la gestión de imágenes Docker.
- Analizar y comparar el rendimiento de diferentes componentes tecnológicos en un sistema distribuido.
- Generar y manejar carga de pruebas con herramientas como Locust.
- Visualizar métricas y datos del sistema utilizando herramientas como Grafana.

Objetivos del Aprendizaje

Objetivo General

Construir una arquitectura de sistema distribuido genérico en Google Kubernetes Engine (GKE) para simular el procesamiento de información sobre ventas de black friday, aplicando conceptos de concurrencia, mensajería, almacenamiento en memoria y visualización, y comparando el rendimiento de diferentes tecnologías clave.

Objetivos Específicos

Al finalizar el proyecto, los estudiantes deberán ser capaces de:

1. **Administrar una arquitectura en la nube utilizando Kubernetes en GCP:** Configurar y desplegar múltiples componentes interconectados en GKE, gestionando recursos como pods, services, ingress, y HPA
2. **Utilizar Go y Rust para desarrollo de servicios concurrentes:** Implementar los componentes de API (Rust) y los servicios de procesamiento y publicación (Go), maximizando su concurrencia y aprovechando sus librerías.
3. **Crear, desplegar y utilizar contenedores y un Container Registry:** Empaquetar todas las aplicaciones en imágenes Docker, publicarlas en un Container Registry privado (Zot) y desplegarlas en GKE.
4. **Entender y operar message brokers (Kafka):** Configurar, desplegar y utilizar Kafka para la comunicación asíncrona de mensajes, y comparar su rendimiento bajo carga.
5. **Implementar un sistema de alta concurrencia para manejo de mensajes:** Diseñar el flujo de datos para soportar un alto volumen de mensajes generados por Locust, desde la recepción en la API hasta su procesamiento y almacenamiento.

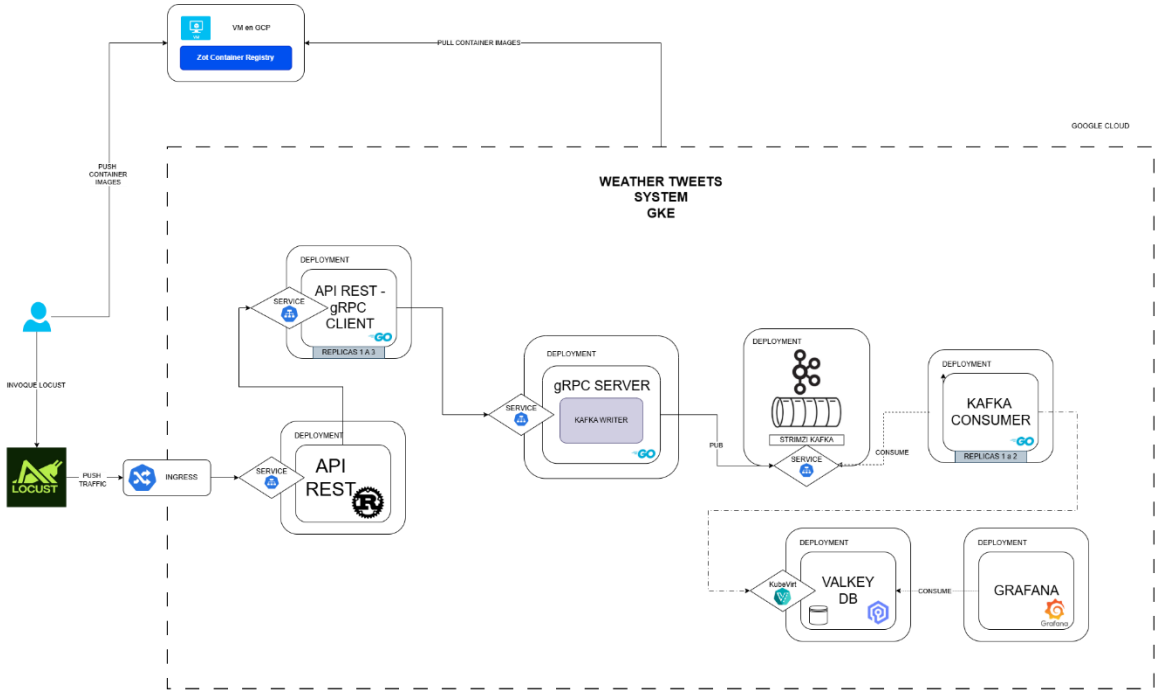
6. **Integrar y utilizar bases de datos en memoria (Valkey) y visualización (Grafana):** Almacenar datos procesados en Valkey , y visualizar esta información y métricas del sistema en un dashboard de Grafana.

Enunciado del Proyecto

Descripción del problema a resolver

En la era digital, la cantidad de datos generados es masiva y requiere sistemas capaces de procesarlos en tiempo real o cuasi-real. Este proyecto simula un escenario donde se recibe información de ventas de productos durante la época de black friday de diferentes categorías. El desafío es diseñar e implementar una arquitectura distribuida, escalable y resiliente que pueda ingerir estos datos, procesarlos, almacenarlos y visualizarlos, utilizando tecnologías modernas de nube y contenedores. Se busca no solo la funcionalidad, sino también la capacidad de analizar y comparar el rendimiento de diferentes componentes, como message brokers y bases de datos en memoria.

Alcance del proyecto



- **Componentes clave incluyen:** Locust (generador de carga), Ingress NGINX, API REST (Rust), Servicios gRPC (Go) para publicación en message brokers, Kafka, Consumidores (Go), Valkey (DB en memoria con KubeVirt), Grafana para los dashboards y Zot (Container Registry).

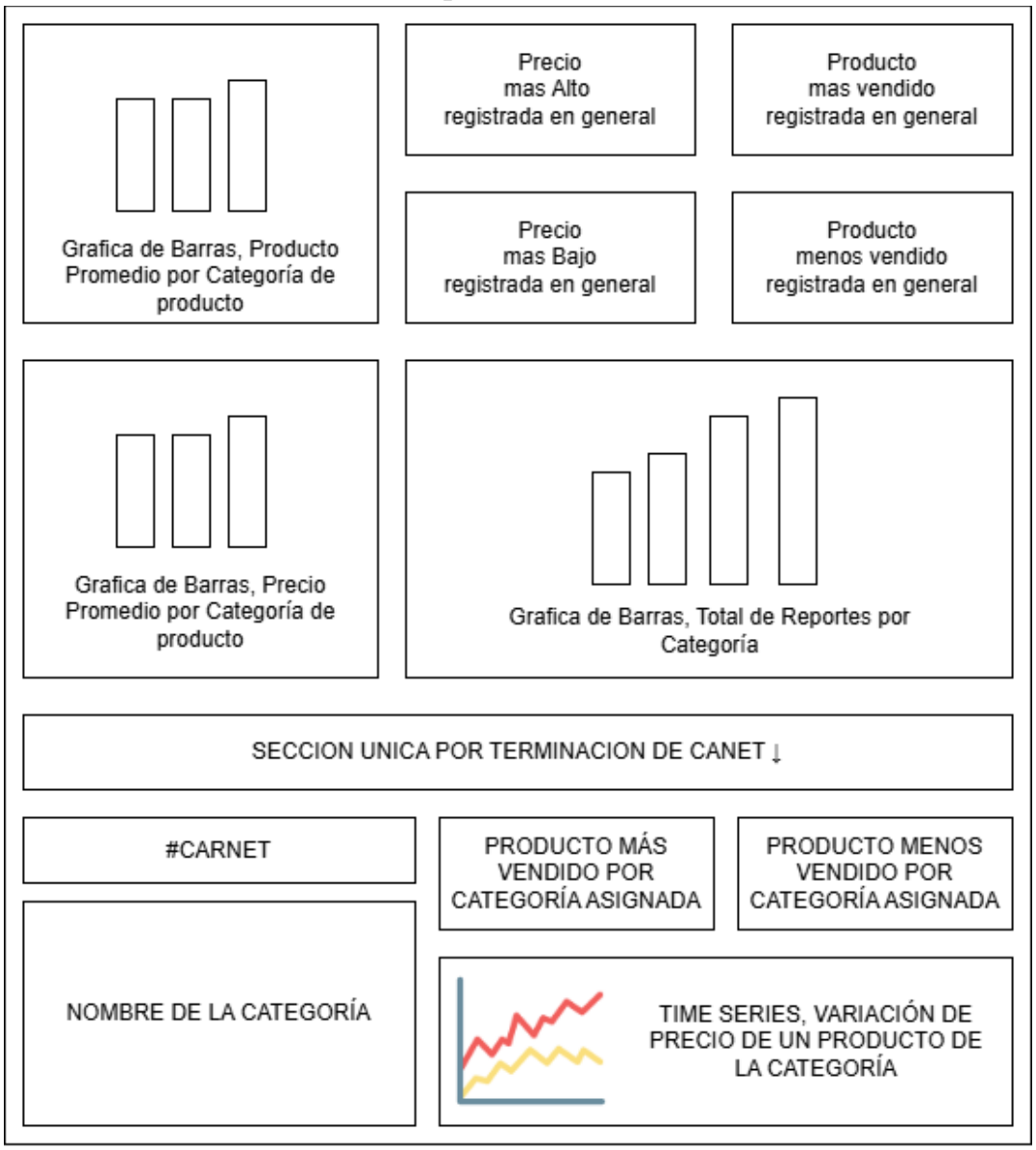
4.3 Alcance del proyecto

- **Alcance obligatorio (basado en el documento provisto):**
  - **Locust:** Generación de tráfico con la estructura JSON especificada (categoría, producto, precio, cantidad_vendida) hacia el Ingress Controller.
  - **Deployments de Rust:** API REST que recibe peticiones de Locust, envía a un Deployment de Go, soporta alta carga y escala con HPA (1-3 réplicas, CPU > 30%).
  - **Deployments de Go:**
    - **Deployment 1 (API REST y gRPC Client):** Recibe de Rust, actúa como cliente gRPC, invoca funciones para publicar en Kafka.
    - **Deployments 2 y 3 (gRPC Server y Writers):** Publican mensajes en Kafka. Pruebas con 1 y 2 réplicas.
  - **Deployment de Kafka:** Almacenan y distribuyen mensajes.
  - **Deployment de Consumidor:** Un deployment para Kafka que consume mensajes y almacenan datos extraídos en Valkey respectivamente.
  - **Deployments Valkey :** Almacenan datos procesados, con persistencia asegurada utilizando 2 réplicas por defecto,
  - **Deployment Grafana:** Visualiza datos de Valkey en un dashboard. *Instalación recomendada con Helm.*
  - **Zot:** Implementado en una VM de GCP fuera del clúster K8s. Todas las imágenes Docker de los componentes se publican y se descargan desde Zot.
  - **OCI Artifact:** Descarga de archivo de entrada desde el registry como un OCI Artifact (se debe especificar qué archivo y cómo se usa en la documentación)
  - **Documentación:** Responder preguntas específicas sobre Kafka, Valkey, gRPC/HTTP, HPA, VPA, y mejoras con replicación.
  - **Sugerencias Generales:** Uso de namespaces, NGINX Ingress Controller, creación propia de imágenes Docker.
  - **Requisitos Mínimos:** Clúster de Kubernetes en GCP.
  - **Restricciones:** Proyecto individual, uso obligatorio de Locust y GKE
  - **Github:** Repositorio privado, carpeta o repositorio llamado ‘proyecto2’, agregar auxiliar.
- **Alcance opcional (No puntuable, pero recomendado para evitar saturación de su sistema):**
  - Optimizar el rendimiento de los componentes con requests y limits.
  - Agregar un tiempo de expiración para los datos en la db Valkey para evitar saturacion

4.3.1 Estructura de los Ventas de Productos:

```
syntax = "proto3";
package blackfriday;

option go_package = "./proto";

// Mensaje con información de venta de producto durante Black Friday
message ProductSaleRequest {
    CategoriaProducto categoria = 1;
    string producto_id = 2;
    double precio = 3;
    int32 cantidad_vendida = 4;
}

// Lista de categorías de productos
enum CategoriaProducto {
    Electronica = 1;
    Ropa = 2;
    Hogar = 3;
    Belleza = 4;
}

// Respuesta del servidor
message ProductSaleResponse {
    string estado = 1;
}

// Servicio gRPC para procesamiento de ventas durante Black Friday
service ProductSaleService {
    rpc ProcesarVenta (ProductSaleRequest) returns
(ProductSaleResponse);
}
```

4.3.2 Estructura del Dashboard Requerido



Asignación de municipio en base al último dígito de su carnet:

- 0,1,2 = Electronica
- 3,4,5 = Ropa
- 6,7 = Hogar
- 8,9 = Belleza

Gráfica de Barra, Total de reportes por categoría se refiere al número de veces que se registró una categoría (`Electronica, Ropa, Hogar,etc`)

Requerimientos técnicos

- **Cloud:** Google Cloud Platform (GCP), Google Kubernetes Engine (GKE).
- **Lenguajes de Programación:** Python (para Locust), Rust, Go.

- **Contenerización:** Docker.
- **Orquestación:** Kubernetes
- **Gestión de VMs:** KubeVirt.
- **Message Brokers:** Strimzi Kafka.
- **Bases de Datos en Memoria:** Valkey.
- **Visualización:** Grafana.
- **Container Registry:** Zot.
- **Generación de Carga:** Locust.
- **Ingress Controller:** NGINX.
- **Herramientas de Desarrollo:** Git, GitHub, IDEs/editores a elección.
- **Sistema Operativo Local (para desarrollo y Locust):** Linux, macOS, o Windows con WSL2.

Entregables

Tipo	Descripción
Link de repositorio	Todos los archivos de código fuente de los diferentes componentes (Rust, Go, Python para Locust si se personaliza), archivos de configuración de Kubernetes (YAMLs para Deployments, Services, Ingress, HPA, etc.), Dockerfiles para cada componente, scripts de apoyo. Organizados en la carpeta Proyecto2 en GitHub.
Informe Técnico	En formato Markdown únicamente. Debe incluir: Documentación de los deployments y una breve explicación con ejemplos. Instrucciones claras para desplegar y probar todo el sistema. Arquitectura del sistema (puede referenciar el diagrama proporcionado o mejorarlo). Conclusiones sobre el rendimiento y comparativas (Kafka, Valkey con impacto de réplicas, API REST vs gRPC). Describa el proceso de desarrollo, los retos encontrados y cómo se resolvieron.