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

Título del Proyecto:

Arquitectura Distribuida en la Nube con Kubernetes

PONDERACIÓN: 50

Horas Aproximadas: 35

Índice

1. Resumen Ejecutivo

El proyecto "Tweets del Clima" tiene como propósito aplicar los conocimientos adquiridos en las unidades 1 y 2 del curso, enfocándose en la implementación de una arquitectura de sistema distribuido y escalable en Google Cloud Platform (GCP) utilizando Google Kubernetes Engine (GKE). Se construirá un sistema que simula la recepción y procesamiento de "tweets" sobre el clima mundial. Este sistema involucrará la generación de tráfico con Locust, una API REST en Rust, servicios en Go para procesamiento y publicación en message brokers (Kafka y RabbitMQ), consumidores para procesar estos mensajes, y bases de datos en memoria (Redis y Valkey) para almacenamiento, culminando en la visualización de datos con Grafana. El proyecto busca demostrar la comprensión y aplicación de tecnologías de contenedores, orquestación, comunicación entre microservicios, manejo de concurrencia y comparación de rendimiento de diferentes tecnologías de mensajería y almacenamiento.

2. 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, RabbitMQ) para la comunicación asíncrona entre servicios.
- Integrar y gestionar bases de datos en memoria (Redis, Valkey) para el almacenamiento de datos de alta velocidad.
- Implementar y utilizar un Container Registry (Harbor) 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.

3. Objetivos del Aprendizaje

3.1 Objetivo General

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

3.2 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.
   Ejemplo: El estudiante desplegará los servicios de API, workers de Go, message brokers, bases de datos y Grafana en un clúster de GKE, exponiendo la API a través de un Ingress Controller.
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.
   Ejemplo: El servicio en Go utilizará goroutines para manejar peticiones gRPC y publicar mensajes en Kafka y RabbitMQ de forma eficiente. La API en Rust manejará múltiples peticiones HTTP concurrentes.
3. **Crear, desplegar y utilizar contenedores y un Container Registry:** Empaquetar todas las aplicaciones en imágenes Docker, publicarlas en un Container Registry privado (Harbor) y desplegarlas en GKE.
   Ejemplo: El estudiante construirá imágenes Docker para la API en Rust y los servicios en Go, las subirá a Harbor alojado en una VM de GCP, y Kubernetes las descargará desde Harbor para crear los deployments.
4. **Entender y operar message brokers (Kafka y RabbitMQ):** Configurar, desplegar y utilizar Kafka y RabbitMQ para la comunicación asíncrona de mensajes, y comparar su rendimiento bajo carga.
   Ejemplo: El sistema enviará mensajes a través de ambos brokers, y se analizará cuál maneja mejor el volumen de mensajes en términos de latencia y throughput.
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.
   Ejemplo: El sistema será capaz de procesar 10,000 peticiones generadas por Locust con 10 usuarios concurrentes, demostrando la escalabilidad y robustez de la arquitectura.
6. **Integrar y utilizar bases de datos en memoria (Redis y Valkey) y visualización (Grafana):** Almacenar datos procesados en Redis y Valkey, y visualizar esta información y métricas del sistema en un dashboard de Grafana.
   Ejemplo: Grafana mostrará contadores de "tweets" por país y el total de mensajes procesados, consultando los datos almacenados en Redis y Valkey.

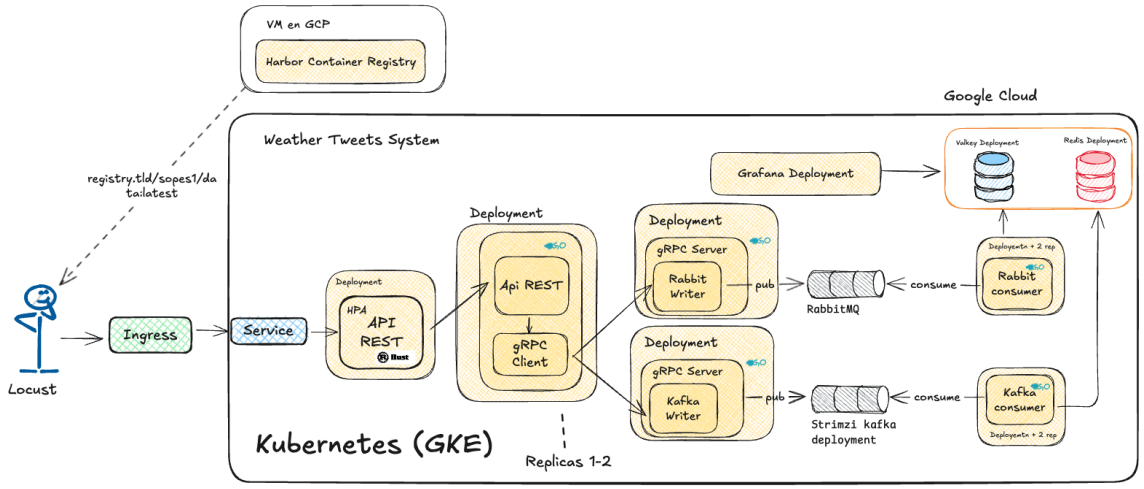
4. Enunciado del Proyecto

Esta sección define de manera clara y detallada los aspectos específicos que los estudiantes deberán abordar en el proyecto. Se incluye el problema a resolver, la arquitectura general, el alcance, y los entregables esperados.

4.1 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 reciben "tweets" sobre el clima de diferentes partes del mundo. 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.

4.3 Alcance del proyecto



- **Componentes clave incluyen:** Locust (generador de carga), Ingress NGINX, API REST (Rust), Servicios gRPC (Go) para publicación en message brokers, Kafka, RabbitMQ, Consumidores (Go), Redis, Valkey, Grafana, y Harbor (Container Registry).

4.3 Alcance del proyecto

Establece los límites del proyecto para que los estudiantes sepan hasta dónde deben llegar.

- **Alcance obligatorio (basado en el documento provisto):**
  - **Locust:** Generación de tráfico con la estructura JSON especificada (description, country, weather) 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 y RabbitMQ.
        - **Deployments 2 y 3 (gRPC Server y Writers):** Publican mensajes en Kafka y RabbitMQ respectivamente. Pruebas con 1 y 2 réplicas.
    - **Deployments de Kafka y RabbitMQ:** Almacenan y distribuyen mensajes. Comparar rendimiento.
    - **Deployments de Consumidores:** Dos deployments (uno para Kafka, otro para RabbitMQ) que consumen mensajes y almacenan datos extraídos en Redis y Valkey respectivamente.
    - **Deployments Redis y Valkey:** Almacenan datos procesados, con persistencia asegurada. Comparar y justificar uso de HPA o 2 réplicas.
    - **Deployment Grafana:** Visualiza datos de Redis y Valkey en un dashboard (contadores por país, total mensajes). Instalación recomendada con Helm.
    - **Harbor:** Implementado en una VM de GCP fuera del clúster K8s. Todas las imágenes Docker de los componentes se publican y se descargan desde Harbor.
    - **OCI Artifact:** Descarga de archivo de entrada desde el registry como un OCI Artifact (se debe especificar qué archivo y cómo se usa).
    - **Documentación:** Responder preguntas específicas sobre Kafka, Valkey/Redis, gRPC/HTTP, 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, información de Locust, uso de GKE, no habrá prórroga.
    - **Github:** Repositorio privado (Proyecto2), agregar auxiliares.
- **Alcance opcional (No puntuable, para exploración):**
    - Implementar métricas de Prometheus y visualizarlas en Grafana.
    - Añadir pruebas de resiliencia (ej. simular caída de un pod).
    - Optimizar aún más el rendimiento de los componentes.

4.4 Requerimientos técnicos

- **Cloud:** Google Cloud Platform (GCP), Google Kubernetes Engine (GKE).
- **Lenguajes de Programación:** Python (para Locust), Rust, Go.
- **Contenerización:** Docker.
- **Orquestación:** Kubernetes.
- **Message Brokers:** Apache Kafka, RabbitMQ.
- **Bases de Datos en Memoria:** Redis, Valkey.
- **Visualización:** Grafana.
- **Container Registry:** Harbor.
- **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.

4.5 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. Organizado en la carpeta Proyecto3 en GitHub.
Informe Técnico	En formato Mark Down o PDF. Debe incluir: Documentación de los deployments y una breve explicación con ejemplos. Respuestas a las preguntas planteadas en la sección "Documentación" del enunciado del proyecto. 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 vs RabbitMQ, Redis vs Valkey, impacto de réplicas).

- **Informe técnico**: Un informe de no más de 10 páginas que describa el proceso de desarrollo, los retos encontrados y cómo se resolvieron.
- **Documentación del usuario**: Manual de uso de la aplicación con capturas de pantalla.

5. Metodología

1. **Fase 1: Configuración y Componentes Base (Semana 1-2)**
    - Configurar cuenta de GCP y crear clúster de GKE.
    - Instalar y configurar Harbor en una VM en GCP.
    - Desarrollar y contenerizar la API REST en Rust. Publicar en Harbor.
    - Desarrollar y contenerizar el Go Deployment 1 (API REST y gRPC Client). Publicar en Harbor.
    - Configurar Locust para generar tráfico básico.
    - Desplegar NGINX Ingress Controller.
    - Pruebas iniciales de flujo: Locust -> Ingress -> API Rust -> Go Deployment 1.
2. **Fase 2: Message Brokers y Writers (Semana 3-4)**
    - Desplegar Kafka (Strimzi) y RabbitMQ en GKE.
    - Desarrollar y contenerizar los Go Deployments 2 y 3 (gRPC Server y Writers para Kafka y RabbitMQ). Publicar en Harbor.
    - Integrar Go Deployment 1 con los Writers.

   - Pruebas de publicación de mensajes en ambos brokers.
3. **Fase 3: Consumidores y Bases de Datos en Memoria (Semana 5-6)**
   - Desarrollar y contenerizar los Consumidores (Go) para Kafka y RabbitMQ. Publicar en Harbor.
   - Desplegar Redis y Valkey en GKE, asegurando persistencia.
   - Integrar Consumidores para almacenar datos en Redis y Valkey.
   - Pruebas de consumo y almacenamiento.
4. **Fase 4: Visualización y Pruebas de Carga (Semana 7)**
   - Desplegar Grafana (Helm recomendado) en GKE.
   - Configurar dashboards en Grafana para visualizar datos de Redis y Valkey.
   - Realizar pruebas de carga completas con Locust (10,000 peticiones, 10 usuarios concurrentes).
   - Implementar y probar HPA para el deployment de Rust.
   - Analizar rendimiento con 1 y 2 réplicas para los deployments de Go Writers.
5. **Fase 5: Documentación y Entrega (Semana 8)**
   - Redactar el manual técnico, incluyendo respuestas a las preguntas y análisis de rendimiento.
   - Asegurar que todos los componentes estén correctamente configurados y documentados en el repositorio.
   - Preparar la entrega final según los requisitos.

6. Desarrollo de Habilidades Blandas

6.1 Proyectos Individuales

Para complementar el desarrollo técnico, esta sección se centra en las habilidades blandas que los estudiantes deberán mejorar a lo largo del proyecto. Dado que este proyecto se plantea como individual, el enfoque es similar al Proyecto 1 pero con mayor énfasis en la gestión de complejidad.

- **6.1.1 Autogestión del Tiempo y Complejidad:** El estudiante deberá gestionar un proyecto con múltiples componentes interdependientes y tecnologías diversas, requiriendo una excelente planificación y priorización para cumplir con los plazos.
- **6.1.2 Responsabilidad y Compromiso Técnico:** Asumir la responsabilidad completa de una arquitectura compleja, desde el diseño conceptual hasta la implementación funcional y el análisis de rendimiento.
- **6.1.3 Resolución Avanzada de Problemas y Depuración Distribuida:** Identificar y solucionar problemas en un sistema distribuido, donde los errores pueden ser difíciles de rastrear a través de múltiples servicios y tecnologías.
- **6.1.4 Investigación y Aprendizaje Autónomo:** Investigar y aprender sobre múltiples tecnologías (Kubernetes, Kafka, RabbitMQ, Go, Rust, Grafana, etc.) y sus mejores prácticas de implementación e integración.
- **6.1.5 Documentación Técnica y Comunicación de Resultados:** Elaborar un manual técnico claro y conciso que no solo describa el sistema, sino que también analice y justifique decisiones de diseño y compare el rendimiento de tecnologías.

7. Cronograma

El cronograma describe las etapas clave del proyecto, los plazos estimados para cada una, y el proceso de asignación, elaboración y calificación de las tareas. Los estudiantes deberán seguir este plan para asegurar que el proyecto avance de manera organizada y cumpla con los plazos establecidos. Cada fase incluye la asignación de tareas, el tiempo estimado para su elaboración, y el momento de su calificación.

Tipo	Fecha Inicio	Fecha Fin
Asignación de Proyecto	Semana	Semana
Elaboración	Semana	Semana
Calificación	Semana	Semana

8. Rúbrica de Calificación

8.1 Requisitos para optar a la calificación

Antes de la evaluación del proyecto, los estudiantes deben cumplir con los requisitos que se indiquen en esta sección.

Tema	Descripción	Cumple (Si/No)
Entrega Completa	Se entrega el enlace al repositorio privado de GitHub (con acceso a los auxiliares) conteniendo todo el código fuente, Dockerfiles, YAMLs de Kubernetes, y el manual técnico.	
Funcionalidad del Clúster en GCP (GKE)	El sistema se puede desplegar y es funcional en un clúster de GKE.	

8.2 Resumen de Puntuaciones

Área	Puntos Totales	Puntos Obtenidos
1. Conocimiento técnico (85%)		
Arquitectura General y Despliegue en GKE/Harbor	20	
Servicios de Ingesta y Procesamiento (Rust, Go)	20	
Message Brokers y Consumidores (Kafka, RabbitMQ)	15	
Almacenamiento y Visualización (Redis,Valkey,Grafana)	15	
Sub-Total Conocimiento	**70**	
2. Análisis, Documentación y Cumplimiento (30%)		
Pruebas de Carga, HPA y Análisis de Réplicas	10	
Manual Técnico y Respuestas a Preguntas	15	

Cumplimiento de Especificaciones y Buenas Prácticas	5	
Sub-Total Habilidades	**30**	
TOTAL	**100**	

*La calificación debe incluir ambas áreas conocimientos y habilidades.

8.3 Detalle de la Calificación

Criterio	Descripción	Puntos Máximos	Puntuación Obtenida
1. Implementación funcional			
1. Funcionalidad del Sistema		**70**	
1.1 Arquitectura General y Despliegue en GKE/Harbor	Despliegue completo y funcional de todos los componentes en GKE. Uso correcto de Harbor para imágenes Docker. Configuración de Ingress, Services, Deployments, HPA.	20	
1.2 Servicios de Ingesta y Procesamiento (Rust, Go)	API REST en Rust funcionando y escalando. Servicios en Go (API/gRPC Client, gRPC Server/Writers) implementados correctamente, con manejo de concurrencia y comunicación gRPC.	20	
1.3 Message Brokers y Consumidores (Kafka, RabbitMQ)	Kafka y RabbitMQ desplegados y configurados. Los Writers publican mensajes correctamente. Los Consumidores leen de las colas/tópicos y procesan los mensajes.	15	
1.4 Almacenamiento y Visualización (Redis, Valkey, Grafana)	Redis y Valkey desplegados con persistencia. Los Consumidores almacenan datos en ellos. Grafana se conecta y muestra la información requerida en un dashboard funcional (contadores por país, total mensajes).	15	
Sub-Total de Puntos		**70**	

2.Análisis, Documentación y Cumplimiento			
2.1 Pruebas de Carga, HPA y Análisis de Réplicas	Locust genera la carga especificada. HPA para API Rust funciona según lo definido. Se realizan y documentan pruebas comparativas con 1 y 2 réplicas para Go Writers, analizando rendimiento.	**10**	
2.2 Manual Técnico y Respuestas a Preguntas	El manual es claro, completo, incluye todas las secciones requeridas (despliegue, ejemplos, arquitectura, conclusiones). Se responden de forma correcta y justificada todas las preguntas planteadas en el enunciado del proyecto.	**10**	
2.3 Cumplimiento de Especificaciones y Buenas Prácticas	Se cumplen todos los requisitos mínimos y restricciones. Uso de namespaces. Código fuente bien organizado y versionado en GitHub. Uso de OCI Artifact según lo especificado.	**5**	
Sub-Total de Puntos		**30**	
Total		**100**	

8.4 Valores

En el desarrollo del proyecto, se espera que cada estudiante demuestre honestidad académica y profesionalismo. Por lo tanto, se establecen los siguientes principios:

1. **Originalidad del Trabajo**
   - Cada estudiante debe desarrollar su propio código, configuraciones y/o documentación, aplicando los conocimientos adquiridos en el curso.
2. **Prohibición de Copias y Plagio**
   - (Según el documento del proyecto) Cualquier copia parcial o total tendrán nota de **0 puntos** y serán reportadas al catedrático y a la Escuela de Ciencias y Sistemas.
   - Esto incluye la reproducción de código entre compañeros, la reutilización de proyectos de semestres anteriores o el uso de código externo sin la debida referencia (ver punto 3).
3. **Uso Responsable de Recursos Externos**
   - El uso de librerías, frameworks, plantillas de configuración de Kubernetes, y ejemplos de código externos (ej. de tutoriales, documentación oficial) está

permitido para aprendizaje y como base, siempre y cuando se referencian correctamente (si se adapta una porción significativa) y se comprendan plenamente. El diseño y la integración de la arquitectura deben ser originales. (Consultar con el catedrático su política específica).

4. **Revisión y Detección de Plagio**
    - Se podrán utilizar herramientas automatizadas y revisiones manuales para identificar similitudes en los proyectos.
    - En caso de sospecha, el estudiante deberá justificar su código y demostrar su desarrollo individual. Si este extremo no es comprobable la calificación será de **0 puntos**.

Al detectarse estos aspectos se informará al catedrático del curso quien realizará las acciones que considere oportunas.
Penalización por Entrega Tardía: (Según el documento del proyecto) 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.

8.5 Comentarios Generales

_____

_____

_____

_____