UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
PRIMER SEMESTRE 2022
ING. SERGIO ARNALDO MENDEZ AGUILAR
AUX. CARLOS DAVID RAMIREZ ALTAN

PROYECTO FASE 2

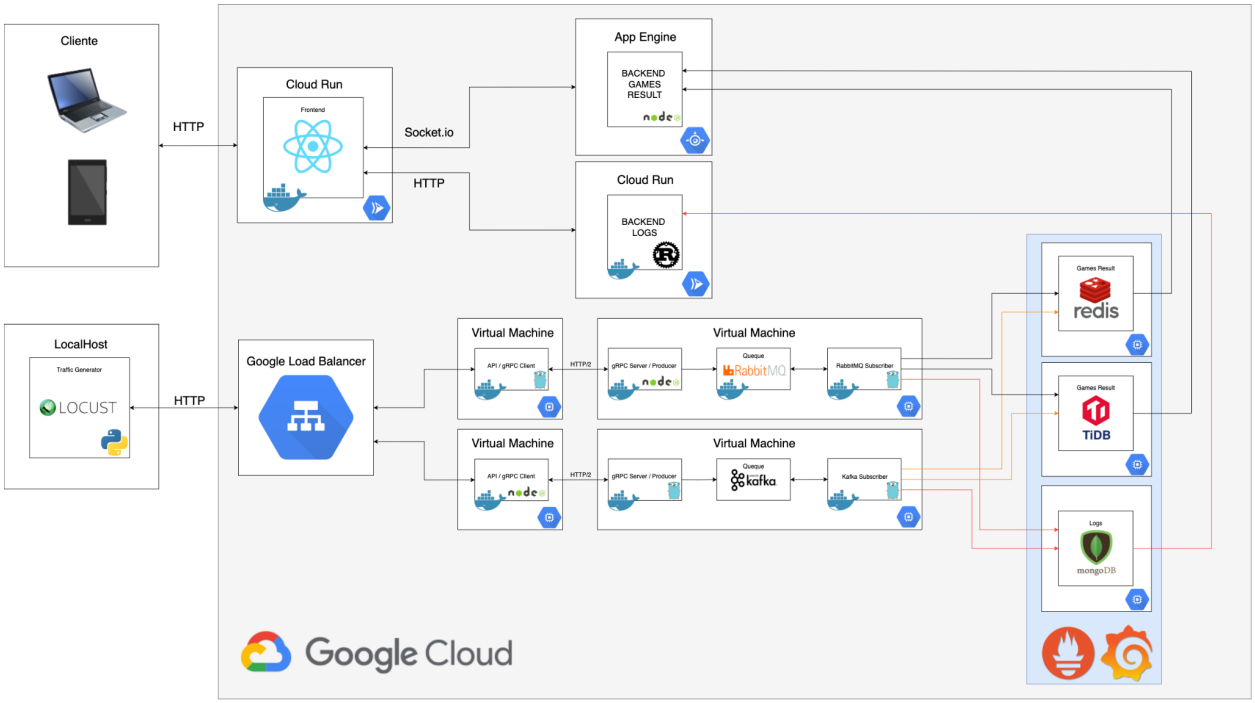
Se solicita construir un sistema genérico de arquitectura distribuida que muestre estadísticas en tiempo real utilizando diferentes servicios de Google Cloud Platform, virtualización a nivel de sistema operativo con Docker.

Este proyecto se aplicará a la visualización de los resultados de juegos implementados por los estudiantes.

OBJETIVOS

- Comprender la teoría de la concurrencia y paralelismo para desarrollar sistemas distribuidos.
- Experimentar y probar tecnologías Cloud Native que ayudan a desarrollar sistemas distribuidos modernos.
- Diseñar estrategias de sistemas distribuidos para mejorar la respuesta de alta concurrencia.
- Monitorear el procesamiento distribuido utilizando tecnologías asociadas a la observabilidad y la telemetría.
- Implementar contenedores en sistemas distribuidos.

ARQUITECTURA



Explicación:

El balanceador de carga recibe el tráfico (información del juego) y este se redirige a una API(gRPC client) que le pasa la data al servidor en el cual según la data obtenida se ejecuta uno de los algoritmos de los juegos implementados por el estudiante y así elegir el ganador del juego actual, luego se pasan los resultados del juego por medio de un Queue (RabbitMQ o Kafka). El Suscriber del Queue leerá estos datos y los almacenará en las bases de datos: Redis, Tidis y MongoDB.

Redis y Tidis: Los datos almacenados en estas bases de datos servirán para mostrar los paneles en tiempo real en el frontend de los resultados de los juegos

MongoDB: Esta base de datos se utiliza para llevar un log de los registros de transacciones.

Juegos:

Los juegos serán algoritmos sencillos implementados por los estudiantes estos algoritmos deben estar implementados en el servidor de gRCP de manera que al recibir los datos del juego en este servidor se seleccionara al azar uno de los juegos implementados por el estudiante de manera que al finalizar el juego y tener un ganador se procede a mandar los resultados del juego a la cola (Queue) correspondiente.

Por ejemplo, un juego tendrá las siguientes reglas:

Genera 10 números

Elija al azar un número como ganador y el otro perderá el juego.

Nota: Se deberán implementar 5 juegos diferentes.

PRIMERA PARTE (GENERADOR DE TRÁFICO CON LOCUST)

Debe generar tráfico utilizando Locust y el lenguaje de programación Python. El tráfico será recibido por un balanceador de carga.

El generador de tráfico leerá un archivo llamado games.json que contendrá datos que se mandarán aleatoriamente en cada petición. Se deberán configurar los parámetros de concurrencia: el número de usuarios a simular, la tasa de generación de usuarios (usuarios generados/segundo) y la dirección del loadbalancer donde serán recibidas las peticiones.

La aplicación de Locust se llamará traffic-games.py.

A continuación se muestra un ejemplo de cómo será el archivo de entrada, y los datos que contendrá cada objeto.

```
{
    "game_id": 2, // Id del juego que se va a ejecutar
    "players": 23 // Numero de jugadores que van a participar en el juego
},
```

El número de game_id solo será de 1 a 5 ya que el estudiante debe implementar 5 juegos y dependiendo del número que se envía desde el generador el estudiante seleccionará uno de sus 5 juegos a ejecutar.

Por ejemplo: El estudiante implementó el Juego1, Juego2, Juego3, Juego4, Juego5. Si desde el generador de tráfico recibe game_id = 1 eso quiere decir que se debe de ejecutar el juego1, el nombre los juegos queda a discreción del estudiante sin embargo cada uno de los juegos que simplemente se debe de relacionar con un id de 1 a 5 ya que esa será la forma de saber el número de juego que el generador está solicitando que se juegue.

SEGUNDA PARTE (DOCKER Y BALANCEADORES DE CARGA)

Esta parte contiene el uso de Git, Docker y la configuración de los balanceadores de carga.

GIT:

Será la herramienta para gestionar las versiones del código del proyecto. Se utilizará como herramienta para que los estudiantes desarrollen colaborativamente el proyecto. Los estudiantes deben tener un repositorio remoto donde se presentará el proyecto, se recomienda utilizar Github o Gitlab.

DOCKER:

Se utilizará para empaquetar la aplicación en contenedores, donde se sugiere utilizar técnicas distroless para crear imágenes pequeñas si es posible. Docker será la herramienta para crear un entorno local de pruebas antes de desplegar las imágenes de los contenedores.

BALANCEADORES DE CARGA:

Se debe configurar un balanceador de carga de capa que distribuya el tráfico. Este balanceador expondrá la aplicación al mundo exterior.

INGRESO:

El objetivo es comparar el tiempo de respuesta y el desempeño de las distintas rutas, la primera utilizando Kafka como broker, la segunda usando RabbitMQ. Toda la información de entrada pasará a través del balanceador de carga.

PRIMERA RUTA (RabbitMQ):

- Generador de Trafico
- Load Balancer
- API / gRPC Client (Go)
- grPC Server (NodeJS) / Producer RabbitMQ Queue
- RabbitMQ
- RabbitMQ Subscriber
- Escribir en las bases de datos NoSQL (Redis,Tidis y MongoDB)

SEGUNDA RUTA (kafka):

- Generador de Trafico
- Load balancer
- API / gRPC Client (NodeJS)
- grPC Server (Go) / Producer Kafka Queue
- Kafka
- Kafka Subscriber
- Escribir en las bases de datos NoSQL (Redis,Tidis y MongoDB)

TERCERA PARTE (RPC AND BROKERS)

La principal idea es crear una manera de escribir datos en bases de datos NoSQL con alto desempeño utilizando la comunicación por RPC, message brokers y las colas de mensajería. La meta es comparar el desempeño de las rutas, Consulte el diagrama de arquitectura.

gRPC: Es un framework de RPC de alto desempeño que puede ser ejecutado en cualquier ambiente, usado principalmente para conectar servicios de backend.

Kafka: Es un sistema de colas de alta disponibilidad para la transmisión de datos en aplicación en tiempo real.

RabbitMQ: Es un modo de sistema de cola de la vieja escuela para funcionar como intermediario o procesamiento de tareas.

Debe responderse las siguientes preguntas:

- ¿Qué sistema de mensajería es más rápido?
- ¿Cuáles son las ventajas y desventajas de cada sistema?
- ¿Cuál es el mejor sistema?

Nota: gRCP debe ser implementado en 2 lenguajes de programación diferente go y nodejs.

CUARTA PARTE (NOSQL DATABASES)

El proyecto está basado en la estructura de la arquitectura de Instagram, debido a la naturaleza del sistema y a la ausencia de datos estructurados es mejor utilizar bases de datos NoSQL. MongoDB podría utilizarse para almacenar datos persistentes y Redis para implementar contadores y caché para mostrar datos y analíticos en tiempo real.

• MongoDB: es una base de datos NoSQL documental que almacena la información utilizando el formato de datos JSON. Un ejemplo de log seria:

```
{
    "game_id": 2,  // Id del juego
    "players": 23, // Cantidad de jugadores que participaron en el juego
    "game_name": "Random", // Nombre del juego
    "winner": 2, // Jugador que gano el juego
    "queue": "RabbitMQ" // Queue por la que paso el mensaje al subscriber
},
```

• Redis y Tidis: son bases de datos NoSQL de clave-valor que implementa distintos tipos de estructuras de datos como listas, conjuntos, conjuntos ordenados, etc. Estas bases de datos almacenarán los resultados de los juegos y dicha información la determinará el estudiante ya que en base a los reportes que debe generar el en frontend debe analizar qué información es necesaria almacenar.

Estas bases de datos serán instaladas en instancias de máquina virtual en gcp (una VM por cada base de datos).

Deben responderse la pregunta:

- ¿Cuál de las dos bases se desempeña mejor y por qué?

QUINTA PARTE (PÁGINA WEB)

Debe crearse un sitio web que muestre en tiempo real los datos insertados, de los resultados de los juegos el sitio web debe ser desarrollado utilizando React.

Backend Logs: Debe de implementar un backend para acceder a los logs almacenados en MongoDB, este debe ser desarrollado utilizando Rust como lenguaje de programación y este debe ser consumido por el frontend por medio de peticiones http.

Backend Games Results: Debe de implementar un backend para acceder a los registros en las bases de datos Redis, Tidis y este debe ser desarrollado utilizando NodeJS como lenguaje de programación y este debe ser consumido por el frontend por medio de sockets.

página principal debe mostrar los siguientes reportes:

Reportes con datos de MongoDB:

- Tabla con los logs almacenados.
- Gráfica del top 3 de juegos.
- Gráfica que compara a los 2 Subscribers de go (la cantidad de inserciones que hizo cada Subscriber).

Reportes con datos de Redis y Tidis:

- Últimos 10 juegos.
- Los 10 mejores jugadores.
- Estadísticas del jugador en tiempo real.

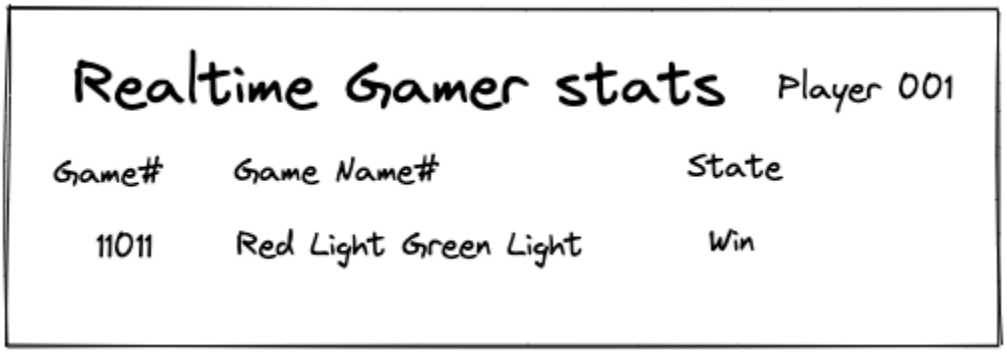
NOTA:

- **La forma en que se muestran los reportes queda a discreción del estudiante sin embargo debe tener por separado los reportes de Redis, los reportes de Tidis y los Reportes de MongoDB.**
- **Redis y Tidis almacenan la misma información esto para poder comparar cual base de datos se desempeña mejor.**

Página Principal:

Estadísticas en tiempo real de X jugador:

Para esta vista se debe poder elegir el jugador del que se desean ver sus estadísticas en tiempo real.



OBSERVABILITY AND MONITORING

Prometheus: The project have to implement monitoring for the state of the services using Prometheus, for example you can use Prometheus to monitor NoSQL Databases and visualize the information using Grafana

Con el uso de prometheus y grafana se deben de monitorear las 3 vms que contienen las bases de datos.

OBSERVACIONES

- Todo debe implementarse utilizando el lenguaje o la herramienta especificada.
- La fase del proyecto debe realizarse en grupos con un máximo de 3 integrantes.
- El uso de GCP es obligatorio para desplegar todo lo solicitado.
- Deben escribir un manual de usuario y un manual técnico en el repositorio del proyecto utilizando Markdown.
- Las copias detectadas tendrán una nota de cero puntos y serán reportadas a la Escuela de Ciencias y Sistemas.
- No se aceptarán entregas después de la fecha y hora especificada.
- Se deberá agregar al usuario **racarlosdavid** al repositorio del proyecto.

ENTREGABLES

- Link del repositorio, debe incluir todo el código fuente con los archivos de manifiesto o cualquier archivo adicional de configuración.
- Manuales.

FECHA DE ENTREGA

6 de abril antes de las 23:59 a través de UEDi. No hay prórrogas.