UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

SISTEMAS OPERATIVOS 1 SECCIÓN N

ING. SERGIO ARNALDO MENDEZ AGUILAR

AUX. GERMAN JOSÉ PAZ CORDÓN

SEGUNDO SEMESTRE 2022

Usactar World Cup 2022

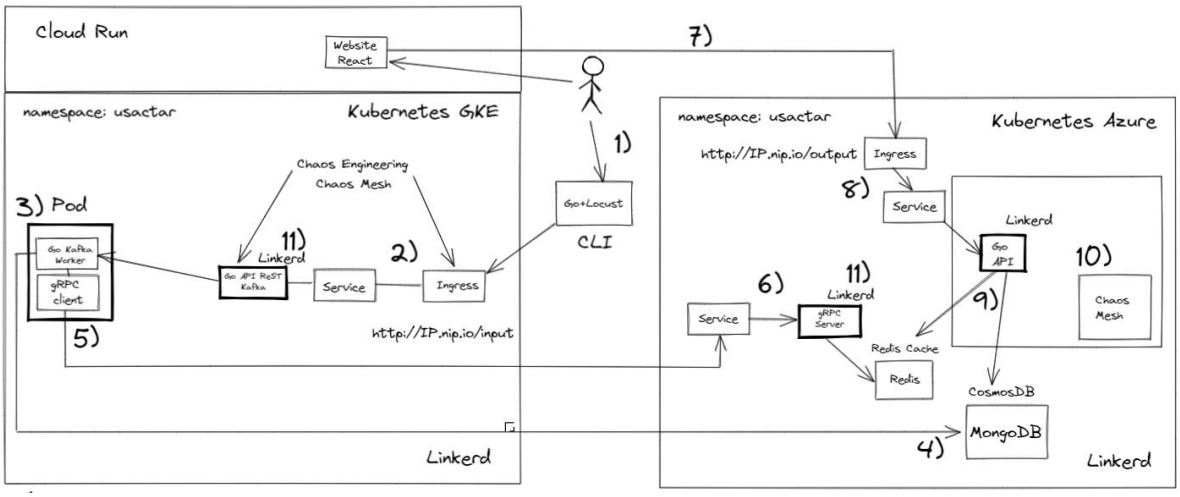
OBJETIVOS

- Comprender la teoría de la concurrencia y el paralelismo para desarrollar sistemas distribuidos.
- Experimentar con tecnologías nativas de la nube 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 y orquestadores en sistemas distribuidos.
- Medir la confiabilidad y el rendimiento en sistemas de alta disponibilidad.
- Implementar ingeniería de caos.

DESCRIPCIÓN

Se requiere la construcción una arquitectura de sistema distribuido genérica que muestre las predicciones de las personas sobre los partidos de futbol del mundial de Qatar 2022. Estos datos se deben procesan mediante una conexión multicluster, uno de Google Cloud y el otro de Microsoft Azure. Esta conexión divide el procesamiento de datos en datos y registros en vivo. Para este proyecto se utilizará Linkerd para interconectar los clusters y Chaos Mesh para Ingeniería de Caos.

ARQUITECTURA



IMPLEMENTACIÓN

PASO 1 (Locust + Go)

Esta parte consiste en la creación de una herramienta que genera tráfico de entrada utilizando Locust y Go. Este tráfico será recibido por balanceadores de carga (entradas de k8s). Para este caso:

- http://IP.nip.io/input, este dominio se expone mediante un controlador de entrada utilizando NGINX.

Funcionalidad: Ejecutar el comando para enviar datos:

run -f data.json --concurrence 10 -n 10000 --timeout 3m

Se deberá leer un archivo JSON con el siguiente formato:

{

"team1": "Guatemala",

"team2": "Argentina",

"score":"2-5",

"phase":<1,2,3,4>

}

Esto contiene el equipo que está en el partido y la fase del torneo.

- 1 -> Octavos
- 2 -> Cuartos
- 3 -> Semifinal
- 4 -> Final

A continuación, los parámetros se utilizan de la siguiente manera:

- **--concurrencia**, son las solicitudes simultáneas o en paralelo
- **-n**, el total de solicitudes enviadas
- **--timeout**, si las solicitudes llegan a ralentizar estos parámetros suspenderán la ejecución.

Este comando se compila con Go y llama a Locust para generar el tráfico.

PASO 2 (Cola de Kafka + MongoDB)

En esta parte una API creada con Go, recibe las solicitudes y la inserción esta información utilizando una cola de Kafka. En paralelo hay un pod que consta de dos partes. Uno que lee los datos de Kafka usando Go, para finalmente escribir la información en MongoDB.

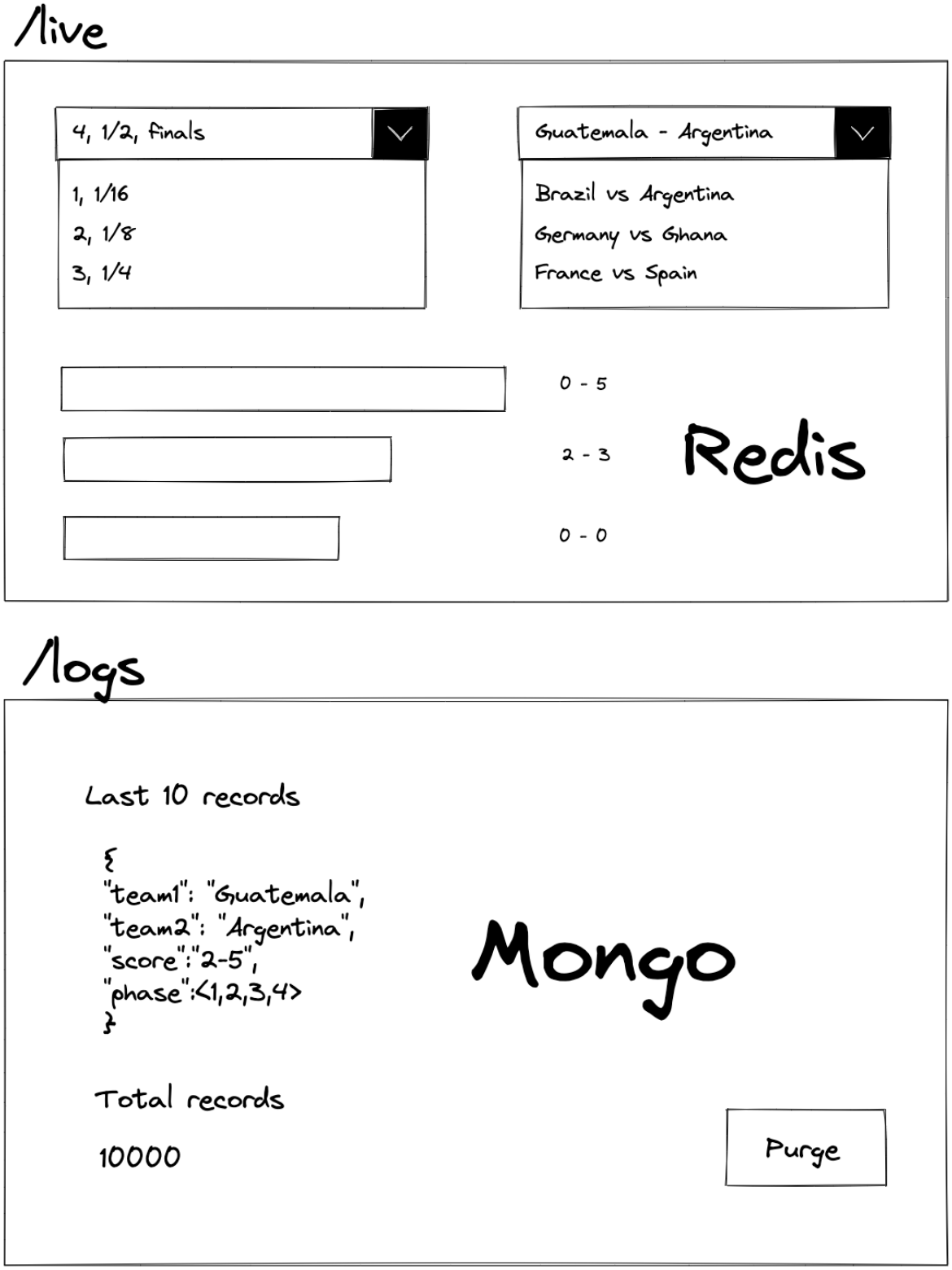
PASO 5-6 (gRPC + Redis)

En segundo lugar, se recibe la información que se transforma al formato Protobuf que será leída por el servidor gRPC que escucha en el otro clúster, para ello, es necesario un servicio espejo utilizando la función linkerd multiclúster, que refleje el servidor en el orden clúster con el fin de llamar al servicio en el otro clúster como si este existiera en el mismo clúster. Finalmente, este servidor recibe la información e instala y almacena la información en Redis, utilizando Hashes y Sets para la página de datos /live.

PASO 7-9 (Cloud Run)

En esta parte hay una aplicación Frontend que consta de 2 páginas:

- **/live**, que muestra las predicciones de los fans.
- **/logs**, que muestra los registros de los datos recibidos.



Esta aplicación debe ser programada en React y se implementa mediante Google Cloud Run y llama a otra API programada en Go en el clúster de Azure. Esta API lee directamente la información de Redis Cache y CosmosDB usando MongoDB.

PASO 10 (Chaos Mesh)

Esto consiste en ejecutar pruebas de ingeniería de caos utilizando Chaos Mesh, en este caso, se tiene que destruir la aplicación que devuelve los datos para la página /live en tiempo real, y opcionalmente para la API que recibe los datos antes de insertarlos en el tema Kafka.

PASO 11 (Linkerd)

Esta parte consiste en observar la API que inserta los datos en la cola de Kafka y observar la API Go que lee los datos de las páginas. Para ello se utiliza la malla de servicio Linkerd, para mostrar alguna información en tiempo real.

CONSIDERACIONES

Cloud and system: Deben de crear sus propias imágenes de contenedor utilizando Google, Azure and Kubernetes.

Databases: Utilizar Azure CosmosDB con compatibilidad con MongoDB para registros y Redis Cache para Redis, estas bases de datos deben tener acceso para ser utilizadas por el cluster de Google.

Multicluster Connection: Utilizar Linkerd para conectar el cluster de Google con el de Azure.

Namespaces: Utilizar namespace para organizar los objetos de Kubernetes.

Load Balancers: Utilizar NGINX Ingress Controller para exponer las APIs.

Esta parte es la manera de exponer la aplicación:

RPC and Brokers: La idea principal en esta parte es crear una manera de alto rendimiento para escribir datos en bases de datos NoSQL, utilizando la comunicación entre RPC (gRPC) y Brokers (Kafka).

gRPC: Es un framework de alto rendimiento que puede correrse en cualquier entorno. Es usado principalmente para conectar servicios de backend.

Kafka: Es un modo de sistema de colas de alta disponibilidad para transmitir datos para aplicaciones en tiempo real. Tomando en consideración las siguientes preguntas:

- ¿Cómo funciona Kafka?
- ¿Qué es el comportamiento de Kafka al procesar datos? ¿Es lento?

Sitio web: En la última parte hay que crear una web para mostrar en tiempo real los datos insertados, utilizando las diferentes páginas (Ver diagrama de arquitectura) desarrolladas con React. Puede usar websockets en NodeJS. Esta página principal tiene que mostrar los siguientes datos:

- Los registros deben almacenarse en MongoDB.
- Los datos en tiempo real deben almacenarse en Redis.
- Replicar datos reales basados en Quatar 2022.

Linkerd: El proyecto tiene que implementar la observabilidad en la red y las respuestas asociadas a los diferentes pods o deployments implementados. En esto, el proyecto implementa un monitoreo en tiempo real de golden metrics.

Prometheus: El proyecto tiene que implementar el monitoreo del estado de los servicios utilizando Prometheus, por ejemplo, se puede usar Prometheus para monitorear bases de datos NoSQL y visualizar la información usando Grafana.

Chaos Engineering

En esta parte el estudiante tiene componentes de muerte del clúster, al mismo tiempo muestra el comportamiento del caos en el clúster.

Chaos Mesh: Utilizar Chaos Mesh para implementar, Slow Network, Pod Kill, Pod Failture. El objetivo es monitorear el comportamiento del sistema mientras el Caos está en progreso.

Preparar los siguientes experimentos:

- Pod kill
- Pod failure
- Container kill

Responda a las siguientes preguntas:

- ¿Cómo se refleja cada experimento en el gráfico de Linkerd?, ¿Qué sucede?

- ¿Cómo cada experimento es diferente?
- ¿Qué experimento es el experimento más dañino?

Nota: La zona de guerra se refiere a una prueba de Ingeniería del Caos en ejecución.

RESTRICCIONES

- El proyecto debe ser realizado en grupos de 3.
- La aplicación debe tener un aspecto profesional.
- La interfaz gráfica debe ser realizada con React.
- No ser aceptarán entregas tardías.
- Implementar un cluster de Google Cloud y otro de Microsoft Azure.
- Cualquier copia parcial o total será reportada a la Escuela de Ciencias y Sistemas para que proceda como indica el reglamento.
- Utilizar un repositorio de GitHub, el cual debe ser privado con el nombre: **so1_proyecto_g<<#grupo>>.**
- El código fuente debe ser administrado por medio de un repositorio de github, al momento de la calificación se bajará la última versión.
- Agregar al auxiliar al repositorio: GermanJosePazCordon

ENTREGABLES

- Enlace del repositorio de GitHub.

- Manual de usuario y manual técnico. Estos deben ser realizado por separado y subidos a su repositorio en formato pdf.

FORMA DE ENTREGA

- Mediante UEDI subiendo el enlace del repositorio.

La entrega se debe realizar antes de las 23:59 del 04 de noviembre de 2022.