UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
PRIMER SEMESTRE 2021
ING. SERGIO ARNALDO MÉNDEZ AGUILAR
AUX. SEBASTIÁN SÁNCHEZ TÚCHEZ
AUX. LUIS LEONEL AGUILAR SÁNCHEZ

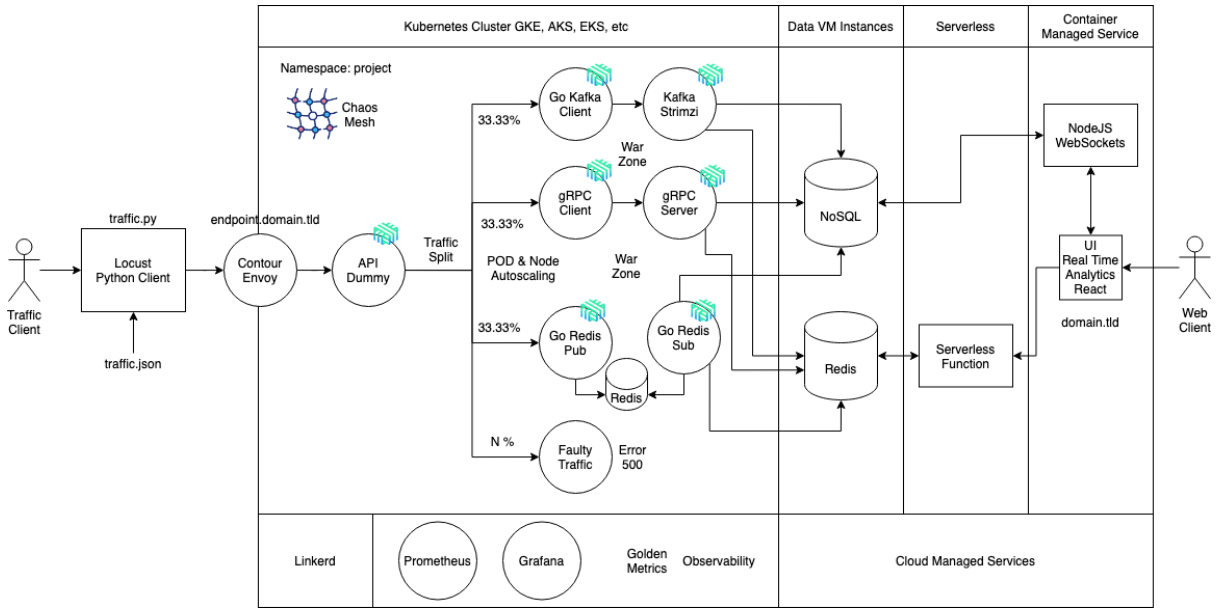
PROYECTO 2

Se solicita construir un sistema genérico de arquitectura distribuida que muestre estadísticas en tiempo real utilizando Kubernetes y service mesh como Linkerd y otras tecnologías Cloud Native. En la última parte se utilizará una service mesh para dividir el tráfico. Adicionalmente, se generará faulty traffic con Linkerd y Chaos Mesh para implementar Chaos Engineering. Este proyecto se utilizará para visualizar las personas vacunadas contra la COVID-19 alrededor del mundo.

Objetivos

- Comprender la teoría de la concurrencia y el paralelismo para desarrollar sistemas distribuidos.
- Experimentar y probar con las tecnologías Cloud Native útiles para desarrollar sistemas distribuidos modernos.
- Diseñar estrategias de sistemas distribuidos para mejorar la respuesta de alta concurrencia.
- Monitorear procesos distribuidos utilizando tecnologías asociadas a la observabilidad y la telemetría.
- Implementar contenedores y orquestadores en sistemas distribuidos.
- Medir la fidelidad y el desempeño en sistemas con alta disponibilidad.
- Implementar la Chaos Engineering.

Arquitectura



Primera parte (generador de tráfico con Locust)

Debe crearse una herramienta que genere tráfico utilizando Locust y el lenguaje de programación Python. El tráfico será recibido por un balanceador de carga público (k8s ingress) con el dominio: *endpoint.domain.tld*; este dominio será expuesto utilizando un ingress controller.

El generador de tráfico leerá un archivo llamado *traffic.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 host donde serán recibidas las peticiones. La aplicación o el archivo de prueba para Locust se llamará *traffic.py*.

```
[
    {
        "name": "Pablo Mendoza"
        "location": "Guatemala City"
        "gender": "male"
        "age": 35
        "vaccine_type": "Sputnik V"
    }
]
```

Ejemplo del archivo *traffic.json*.

Segunda parte (Docker, Kubernetes y balanceadores de carga)

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

Git, Docker y Kubernetes

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. Si los estudiantes consideran necesario colocar el repositorio en privado deben agregar a los tutores como colaboradores (*leoaguilar97* y *sebastiansnz*).

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 en el clúster de Kubernetes.

Kubernetes: Kubernetes será el encargado de la orquestación de los contenedores. Antes de que el cliente genere el tráfico, el proyecto implementará un clúster de Kubernetes que se utilizará para desplegar distintos objetos:

- **Ingress controllers:** para exponer distintas partes de la aplicación.
- **Deployments y services:** para desplegar y comunicar distintas secciones de la aplicación.
- **Pods:** si es necesario, pero es común utilizar otros objetos con un nivel más alto de aplicación como los deployments.

Se sugiere utilizar un namespace separado llamado *project*, ya que es una buena práctica para organizar toda la aplicación.

Balanceadores de Carga

Se debe configurar un balanceador de carga de capa 7 (como Kubernetes Ingress) en el clúster de Kubernetes utilizando Helm o Kubectl. Este balanceador expondrá la aplicación al mundo exterior. Para el proyecto se seleccionará uno de las siguientes opciones:

- nginx-ingress
- Gloo
- Contour
- Traefik

Ingreso

La meta es comparar el tiempo de respuesta y el desempeño de las distintas rutas, la primera utilizando gRPC con Go, la segunda utilizando Redis Pub/Sub y la tercera utilizando Kafka. Toda la información de entrada pasará a través del ingress controller.

Primer ruta:

- Generador de tráfico.
- Ingress.
- Cliente Go gRPC.
- Servidor Go gRPC.
- Escritura en las bases de datos NoSQL.

Segunda ruta:

- Generador de tráfico.
- Ingress.
- Cliente Redis Pub/Sub.
- Servidor Redis Pub/Sub.
- Escritura en las bases de datos NoSQL.

Tercera ruta:

- Generador de tráfico.
- Ingress.
- Cliente Go Kafka.
- Servidor Kafka con Strimzi.
- Escritura en las bases de datos NoSQL.

Se desea que se implemente autoscaling vertical y horizontal, corrutinas e hilos de acuerdo a la naturaleza del servicio y la implementación. La implementación queda a elección paro debe ser justificada según el contexto y las buenas prácticas.

Traffic splitting

Se debe implementar un traffic splitter utilizando Linkerd. Debe dividirse el tráfico en las tres rutas en un tercio de tráfico para cada ruta (33.33% para la primera ruta, 33.33% para la ruta y 33.33% para la tercera). Para implementarlo Linkerd utiliza un dummy service que puede ser la copia de un servicio de cualquiera de las rutas y debe utilizarse nginx para esta funcionalidad.

Adicionalmente deben de realizarse las siguientes pruebas con faulty traffic:

- Cola #1 50%, cola #2 50%
- Cola #1 33.33%, cola #2 33.33%, cola #3 33.33%
- Cola #1 33.33%, cola #2 33.33%, faulty traffic 33.33%
- Cola #1 50%, faulty traffic 50%

Deben responderse las siguientes preguntas:

- Cómo funcionan las golden metrics, cómo pueden interpretarse las cuatro pruebas de faulty traffic utilizando de base las gráficas y métricas que muestra Linkerd y Grafana en los dashboards.
- Mencionar al menos tres patrones de conducta que fueron descubiertos.

Tercera parte (RPC, brokers y bases de datos NoSQL)

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.

- **gRPC:** Es un framework de RPC de alto desempeño que puede ser ejecutado en cualquier ambiente, usado principalmente para conectar servicios de backend.
- **Redis Pub/Sub:** Es una manera de crear un sistema de colas de mensajería para aplicaciones nativas de cloud y arquitecturas de microservicios.
- **Kafka:** Es un sistema de colas de alta disponibilidad para la transmisión de datos en aplicación en tiempo real.

Debe responderse las siguientes preguntas:

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

Cuarta parte (base de datos NoSQL)

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.
- **Redis:** es una base de datos NoSQL de clave-valor que implementa distintos tipos de estructuras de datos como listas, conjuntos, conjuntos ordenados, etc. Se utilizará adicionalmente para implementar el servicio de pub/sub.

Estas bases de datos serán instaladas en una instancia que será accesible en la VPC del clúster de Kubernetes.

Deben responderse la pregunta:

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

Quinta parte (sitio web)

Debe crearse un sitio web que muestre en tiempo real los datos insertados, debe ser desarrollado utilizando NodeJS y React. Pueden utilizarse websockets en NodeJS para mostrar los datos en tiempo real, la página principal debe mostrar los siguientes reportes:

- Datos almacenados en la base de datos, en MongoDB.
- Los diez países con más vacunados, en Redis.
- Personas vacunadas por cada país, en Redis.
- Gráfica de pie de los géneros de los vacunados por país, en MongoDB.
- Los últimos cinco vacunados almacenados por país, en MongoDB.
- Gráfica de barras del rango de edades (de diez en diez) por cada país, en Redis.
- Mapa interactivo que muestre estos reportes.

Cloud Managed Services

Se busca que los estudiantes experimenten con los Cloud Managed services del proveedor de Cloud a su elección. El servidor de NodeJS y la aplicación web escrita con React deben estar desplegados a través de estas herramientas. Las consultas a la base de datos de MongoDB se harán a través del servidor de NodeJS y las consultas a la base de datos de Redis se harán a través de funciones serverless.

Observabilidad y Monitoreo

Debe implementarse los observabilidad y las golden metrics utilizando Linkerd en los pods o deployments del proyecto. El proyecto debe implementar el monitoreo del estado de los servicios usando Prometheus.

Chaos Engineering

Se debe implementar el faulty traffic al sistema, detener elementos en el cluster de Kubernetes y mostrar este comportamiento caótico con:

- **Linkerd:** Se utilizará para generar faulty traffic.
- **Chaos Mesh:** Se implementará para experimentar con: Slow Network, Pod Kill, Pod Failure y Kernel Fail.

La meta es monitorear el comportamiento del sistema mientras el caos está en progreso. Deben prepararse los siguientes experimentos:

- faulty traffic con Linkerd (de acuerdo a la sección de traffic splitting)
- Pod Kill
- Pod Failure
- Container Kill
- Network Emulation (Netem) Chaos
- DNS Chaos

Deben responderse las siguientes preguntas:

- ¿Cómo cada experimento se refleja en el gráfico de Linkerd, qué sucede?
- Cómo cada experimento es distinto,
- Cúal de todos los experimentos es el más dañino.

Restricciones

- El proyecto será desarrollado en grupo de máximo cuatro estudiantes.
- Todo debe implementarse utilizando el lenguaje o la herramienta especificada.
- Deben escribir un manual técnico explicando todos los componentes del proyecto y respondiendo las preguntas de cada sección.
- Deben escribir un manual de usuario.
- 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.

Entregables

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

Enlaces

- https://kubernetes.io/
- https://linkerd.io/
- https://grpc.io/
- https://www.mongodb.com/
- https://redis.io/
- https://strimzi.io/
- https://chaos-mesh.org/
- **https://github.com/sergioarmgpl/operating-systems-usac-course/**

Fecha de entrega

12 de mayo antes de las 23:59 a través de UEDi.