UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1 SECCIÓN A
ING. SERGIO ARNALDO MENDEZ AGUILAR
AUX. FERNANDO ALBERTO MAZARIEGOS TAJIBOY
VACACIONES DICIEMBRE 2020

PROYECTO 2 v2.0

VISUALIZADOR EN TIEMPO REAL COVID-19

OBJETIVOS

- Comprender la simultaneidad y la teoría del paralelismo para desarrollar sistemas distribuidos.
- Experimente y pruebe con tecnologías nativas de la nube que ayudan a desarrollar sistemas distribuidos modernos.
- Diseñar estrategias de sistemas distribuidos para mejorar la respuesta de alta simultaneidad.
- Supervise el procesamiento distribuido mediante tecnologías asociadas a la observabilidad y la telemetría.
- Implementar contenedores y orquestadores en sistemas distribuidos.
- Mida la relialabilidad y el rendimiento en sistemas de alta disponibilidad.

DESCRIPCIÓN

Cree una arquitectura de sistema distribuida genérica que muestre estadísticas en tiempo real mediante Kubernetes y tecnologías nativas en la nube. Y proporcionar un despliegue blue/green, Canary o división de tráfico. Este proyecto se aplicará a los casos infectados actuales de COVID-19 en todo el mundo.

PRIMERA PARTE (Generador de tráfico Locust.io)

Esta parte consiste en la creación de una herramienta que genera tráfico utilizando Locust y Python como lenguaje de programación. Este tráfico será recibido por el balanceador de carga (k8s ingress) en este caso:

- Nginx(load.domain.tld)

Esta es la funcionalidad:

- Generar un archivo con el nombre de **traffic.json**
- El contenido de **traffic.json** serán los parámetros para enviar aleatoriamente en cada solicitud

Configuración de los parámetros de concurrencia en Locust:

- Número de usuarios totales a simular
- Tasa de generación (usuarios generados/segundo)
- Host

El archivo de aplicación o prueba de Locust debe denominarse como **traffic.py**

Ejemplo de archivo:

```
[
   {
       "name":"Pablo Mendoza"
       "location":"Guatemala City"
       "age":35
       "infected_type":"communitary"
       "state": "asymptomatic"
   }
]
```

SEGUNDA PARTE (Docker, Kubernetes y Load Balancers)

Esta parte contiene el uso de Git y Docker, la instalación del clúster de Kubernetes y la configuración de Load Balancers.

GIT, DOCKER Y KUBERNETES

Git: Git será la forma de almacenar y versionar código en github. Git/Github se utilizará como una herramienta para crear un entorno de desarrollo colaborativo para estudiantes.

Docker: Docker se usará para empaquetar las aplicaciones dentro de contenedores, será preferible utilizar la técnica sin distribución para crear el tamaño de imagen más pequeño si es posible. Docker será la herramienta para crear un entorno local para realizar pruebas antes de que una imagen de Docker se implemente en Kubernetes.

Kubernetes: Antes de la máquina cliente que genera el tráfico, el proyecto implementa un clúster de Kubernetes que se utilizará para implementar diferentes objetos:

- **Namespace:** Es una buena práctica para organizar toda la aplicación. En nuestro caso se debe tener el namespace ***project***.
- **Ingress controller:** Para exponer diferentes partes de aplicaciones fuera del clúster.
- **Deployments y servicios:** Para implementar y comunicar diferentes secciones de la aplicación.
- **Pods:** Si es necesario. Es común usar una abstracción más alta como deployments en su lugar.

Kubernetes se encargará de la orquestación del contenedor de las diferentes partes de la aplicación. Usando este tipo de tecnologías, el proyecto está diseñado para crear un entorno y una aplicación básicos nativos de la nube.

BALANCEADORES DE CARGA

Esta parte está relacionada con la configuración de Load Balancers (Kubernetes Ingress) de Capa 7 en el clúster de Kubernetes mediante helm o kubectl. El balanceador de carga será:

- Nginx

Esta parte es la manera de exponer la aplicación al mundo exterior.

INGRESS

El objetivo es comparar la respuesta de tiempo y el rendimiento para las diferentes rutas, una usando Go y la otra usando Redis Pub/Sub. Toda la información de entrada pasa a través de un controlador de entrada (ingress controller).

Primera ruta de acceso:

- Generador de Tráfico
- Ingress
- go-grpc-client
- go-grpc-server
- Escribir en la base de datos NoSQL

Segunda ruta de acceso:

- Generador de Tráfico
- Ingress
- redis-pub
- redis-sub
- Escribir en la base de datos NoSQL

Nota: Se desea implementar el escalado automático vertical y horizontal, corrutinas y subprocesos de acuerdo con la naturaleza del servicio a implementar. Esta implementación es abierta, pero debe justificarse en el contexto de las mejores prácticas.

DIVISIÓN DE TRÁFICO (TRAFFIC SPLITTING)

Para la división de tráfico en el proyecto se utilizará Linkerd con la idea de dividir el 50% de tráfico a la primera ruta de acceso y el otro 50% a la segunda ruta de acceso. Para implementar este Linkerd se utilizará un servicio ficticio que puede ser la copia del servicio de una de las rutas, y tiene que ser utilizado **Nginx** para esta funcionalidad. Ahora mismo es la opción estable y probada para este proyecto.

TERCERA PARTE (RPC, Brokers y Bases de Datos NoSQL)

La idea principal en esta parte es crear una forma de alto rendimiento para escribir datos en bases de datos NoSQL, utilizando la comunicación RPC frente a los intermediarios (brokers). El objetivo es comparar el performance de las rutas. La implementación consiste en la primera ruta de acceso usará Redis para recibir datos que se escribirán en bases de datos NoSQL, y el otro utiliza un RPC de alto rendimiento, gRPC.

- **Redis Pub/Sub**: Una forma de crear un sistema de cola, mensajería para aplicaciones nativas en la nube y arquitecturas de microservicios.
- **gRPC:** Es un framework RPC de alto rendimiento, que se puede ejecutar en cualquier entorno. Se usa principalmente para conectar servicios back-end.

CUARTA PARTE (Base de Datos NoSQL)

Se utilizarán bases de datos NoSQL, debido a la naturaleza del sistema y los datos sin esquema. MongoDB podría utilizarse para almacenar datos persistentes y Redis para implementar contadores y algunas cachés para mostrar datos o análisis en tiempo real. Es la decisión del estudiante de cómo implementarlo.

- **MongoDB:** Es una base de datos de documentos NoSQL que almacena la información mediante el formato de datos JSON.
- **Redis:** Es una base de datos NoSQL Key-Value que implementa diferentes tipos de datos como list, sets, conjuntos ordenados, etc.

Estas bases de datos se instalarán en una instancia que tenga que ser accesible en la VPC del clúster de Kubernetes.

QUINTA PARTE (Sitio Web o Página Principal)

En la última parte se debe crear un sitio web para mostrar en tiempo real los datos insertados, utilizando una página principal desarrollado con NodeJS, React u otro framework Javascript progresivo. Puede usar websockets en NodeJS u otro lenguaje para mostrar la fecha en tiempo real. Esta página principal tiene que mostrar los siguientes datos:

Secciones de datos:

- Recopilación de datos almacenados en MongoDB.
- Top 3 de áreas infectadas en MongoDB.
- Gráfico circular del porcentaje infectado de estados, departamentos, etc., en MongoDB
- Los últimos 5 casos infectados almacenados en Redis.
- Gráfico de barras de rango de edad de infectados en Redis.

SERVICIOS ADMINISTRADOS EN LA NUBE (CLOUD MANAGED SERVICES)

El objetivo de esta parte es que los estudiantes experimenten con servicios administrados en la nube para observar el funcionamiento como servicio, bases de datos, máquinas virtuales, DNS y servicios administrados por contenedores, estos servicios podrían cambiar entre el proveedor de la nube seleccionado para este proyecto.

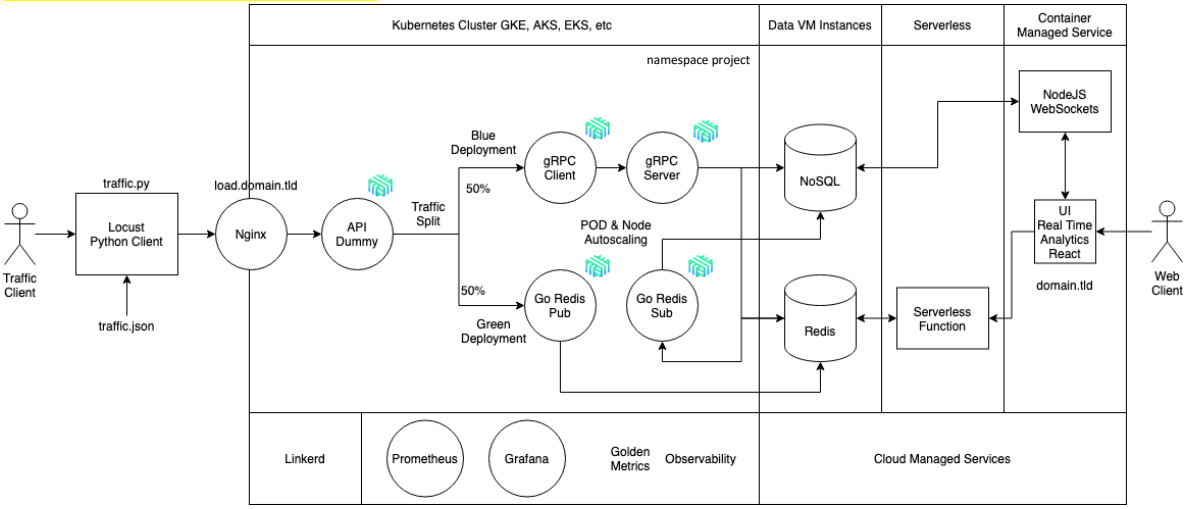
OBSERVABILIDAD Y MONITOREO

El estudiante tiene que decidir los lugares para implementar la observabilidad y las métricas doradas usando Linkerd.

Linkerd: El proyecto tiene que implementar la observabilidad en la red y las respuestas asociadas a los diferentes pods o implementaciones implementadas en el proyecto. En este proyecto, el proyecto implementa una supervisión en tiempo real de las métricas doradas.

Prometheus: El proyecto tiene que implementar la supervisión del estado de los servicios mediante Prometheus, por ejemplo, puede utilizar Prometheus para supervisar bases de datos NoSQL y visualizar la información mediante Grafana.

ARQUITECTURA FINAL



RESTRICCIONES

- El proyecto se desarrollará en grupos de 3 estudiantes
- Implementar con las entradas y los lenguajes mencionados
- Desarrollar un manual técnico y de usuario en formato PDF
- Si se encuentran copias, el grupo recibirá una puntuación de 0 puntos y será reportado
- No se aceptarán entregas tarde

ENTREGABLES

- Código fuente en GitHub
- Manual técnico
- Manual de usuario

FORMA DE ENTREGA

Es indispensable que se realice la entrega de 2 formas:

- Mediante UEDI, subiendo el enlace del repositorio con el código fuente y el manual técnico y de usuario, en una carpeta comprimida [SO1]Proyecto2_grupo<<no_grupo>>.rar.
- Por medio de un repositorio de GitHub, el cual debe ser privado con el nombre: proyecto2_grupo<<no_grupo>>_so1. Se deberá agregar al usuario: **AuxFernandoMazariegos**.

La entrega se debe realizar antes del lunes 4 de enero de 2021