19 de abril de 2022

Ariel Rubelce Macario Coronado

universidad de san carlos de guatemala

manual tecnico

Practica 2

# Herramientas utilizadas.

* Sistemas Operativos.
  + Ubuntu.
  + Windows.
* Lenguajes
  + Go
  + Node
* Bases de datos.
  + Mongo DB
* Brokers.
  + Kafka.
* GCP
* Docker
* GRCP

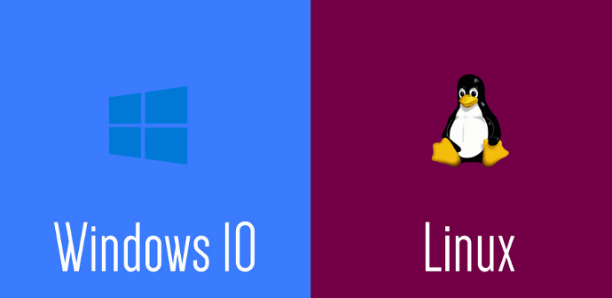
# Arquitectura.

Diagram

Description automatically generated

# Sistemas Operativos.

Se utilizaron dos sistemas operativos para realizar el proyecto, ya que realizar la programación de los distintos módulos algunos se realizaron en Windows y otros en Linux, también se utilizó Linux en las maquinas virtuales ya que se encuentra mayor facilidad para realizar los distintos comandos.



# Lenguajes.

Se usó Go y Node en la mayoría de los módulos del proyecto, para realizar el cliente y el servidor de grcp y go también se uso para el subsriber de Kafka.



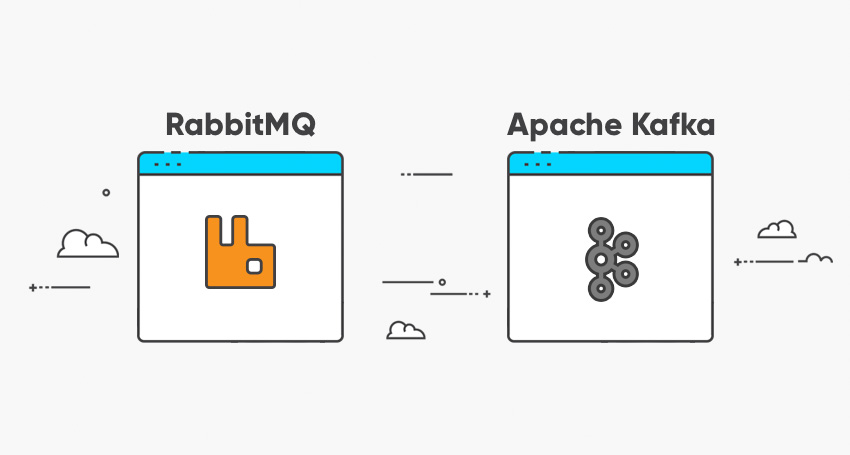
# Bases de datos.

Se usó 1 bases de datos las cuales son, mongo db, la cual es una base de datos de documentos la cual nos sirve para guardar nuestros logs.



# Brokers.

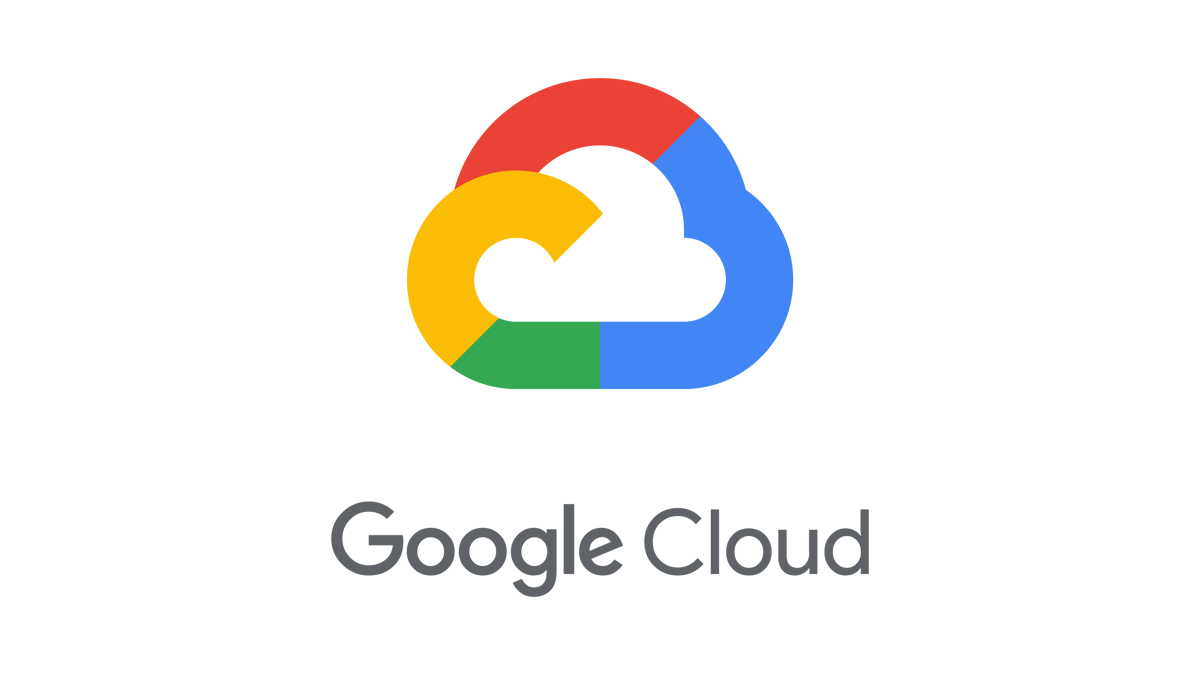
Kafka que es una plataforma distribuida de transmisión de datos que permite publicar, almacenar y procesar flujos de registros, así como suscribirse a ellos de forma inmediata.



# GCP.

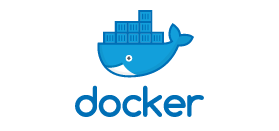
Se trata de la suite de infraestructuras y servicios que Google utiliza a nivel interno y ahora y disponible para cualquier empresa de tal forma que sea aplicable a multitud de proceso empresariales.

Básicamente nos aparta todas las herramientas necesarias para diseñar hacer testing y lanzar aplicaciones desde gcloud con mucha mas seguridad y escalabilidad que cualquiera herramienta gracias a la propia infraestructura con la que Google cuenta.



# Docker.

Docker es una plataforma de software que le permite crear, probar e implementar aplicaciones rápidamente. Docker empaqueta software en unidades estandarizadas llamadas contenedores que incluyen todo lo necesario para que el software se ejecute, incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución. Con Docker, puede implementar y ajustar la escala de aplicaciones rápidamente en cualquier entorno con la certeza de saber que su código se ejecutará.



# Creación del cluster.

Graphical user interface, application, Teams

Description automatically generated

Debemos de seleccionar una GKE Standard para crear nuestro cluster.

Graphical user interface, text, application, email

Description automatically generated

Se selecciona el nombre del cluster y la región en la que deseemos el cluster.

Graphical user interface, text, application, email

Description automatically generated

En versión de plano de control seleccionamos versioin estática y la versión predeterminada.

Graphical user interface, text, application, email

Description automatically generated

Seleccionamos en default pool seleccionamos la cantidad de nodos que deseemos para nuestro cluster.

Graphical user interface, text, application, email

Description automatically generated

Seleccionamos el tipo de máquina que deseamos y luego precedemos a crear nuestro cluster, tardara alrededor de 5 minutos.

# Archivos YAML.

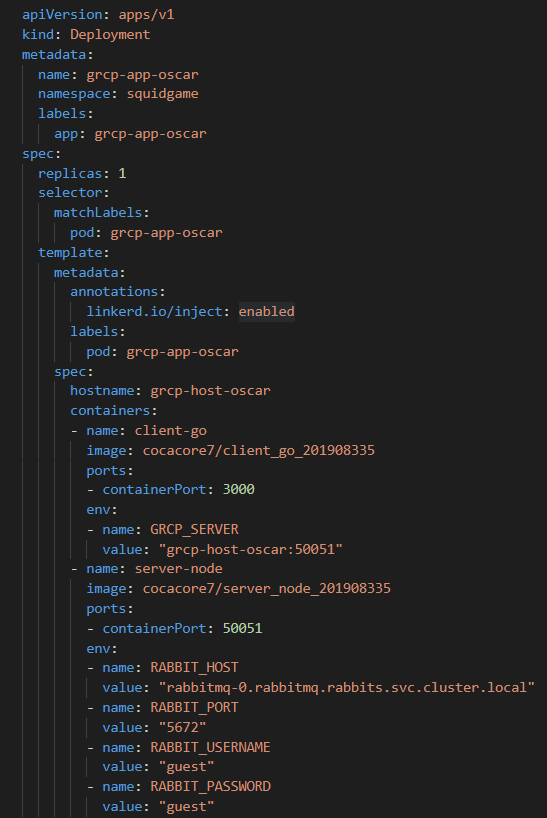
## clients-servers

Deployment de aplicación grcp con servidor en go, y cliente en node, con conexión a Kafka stimzi

Texto

Descripción generada automáticamente

Deployment de aplicación grcp con servidor en node, y cliente en go, con conexión a RabbitMQ



Se crean los servicios de tipo ClusterIP para cada aplicación grcp

Texto

Descripción generada automáticamente

Se crea el servicio ingress inyectando el ingress controller que nos ayudara a implementar traffic spliting entre los servicios clusterip

Texto

Descripción generada automáticamente

Se configura un servicio de faulty traffic para simular perdida de trafico en nuestro cluster con traffic spliting

Texto

Descripción generada automáticamente

## Rabbit-conf

Se configuran y deployan 4 pods de rabbit

Texto

Descripción generada automáticamente

## Subscribers

Se crean dos deployments con las aplicaciones de subscribers para cada sistema de mensajeria implementado

Texto

Descripción generada automáticamente

## Traffic-splitting

Se implementa un archive de configuracion de carga para nuestros servicios

Texto

Descripción generada automáticamente

## Front-conf

Se implementa el deployment de la aplicación de rust con su servicio Load Balancer

Texto

Descripción generada automáticamente

Se implementa el deploy de la aplicación de node y su respectivo servicio de tipo load balancer

Texto

Descripción generada automáticamente

Se implementa el deploy de la aplicación frontend con su respectivo servicio ClusterIP

Texto

Descripción generada automáticamente

Se implementa el servicion ingress con inyección de ingress controller para acceder al servicio de la aplicación frontend

Texto

Descripción generada automáticamente

# Preguntas.

## RPC AND BROKERS

¿Qué sistema de mensajería es más rápido?

Rabbit

¿Cuántos recursos utiliza cada sistema? (Basándose en los resultados que muestra el Dashboard de Linkerd)

¿Cuáles son las ventajas y desventajas de cada sistema?

Las ventajas de Kafka esta fase fue que la instalación de este sistema de mensajería es muy rápida ya que existe strimzi además podemos observa que las cola es mas eficiente, una de las desventajas es que es algo complejo y se necesitan varios días para aprender a utilizarla.

Una ventaja de rabbit es que es un fácil de utilizar y aprender y una desventaja de rabbit es que en un cluster debemos de realizar unos pasos distintos para poder deployarlo.

¿Cuál es el mejor sistema?

El mejor es Rabbit ya que es muy fácil de aprender y fácil de instalar también también por las resultados podemos observar que es mas eficiente.

## NOSQL.

¿Cuál de las dos bases (Redis y Tidis) se desempeña mejor y por qué?

Redis, ya que según nuestros resultados se puede observar que los datos llevan mucho mas rápido vs tidis db también ya que tidis es basada en redis.