

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
ING. SERGIO ARNALDO MENDEZ AGUILAR
AUX. LEONEL AGUILAR
AUX. CARLOS RAMIREZ



PROYECTO #1

OLYMPICS GAME NEWS

GUATEMALA, AGOSTO DE 2021

OBJETIVOS

GENERAL

Realizar un sistema computacional distribuido, cloud native, utilizando diferentes servicios de Google Cloud Platform, virtualización a nivel de sistema operativo con Docker y Containerd y generadores de tráfico para ser aplicado a un tema actual.

ESPECÍFICOS

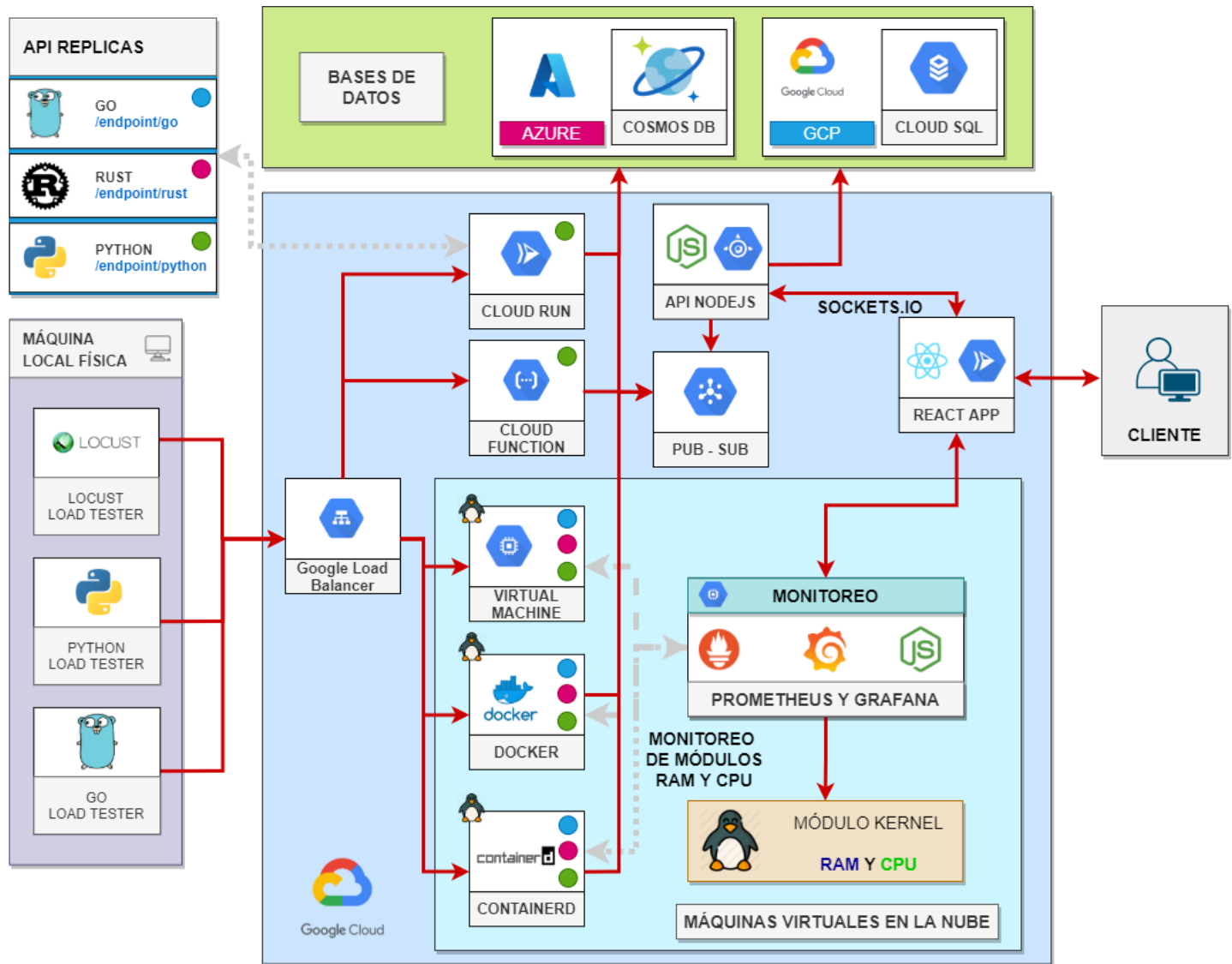
- Aplicar conocimiento de contenedores, imágenes, archivos de composición y redes entre contenedores.
- Experimentar y utilizar tecnologías nativas de la nube que ayudan a desarrollar sistemas distribuidos modernos.
- Generar tráfico y dividirlo en la red utilizando herramientas específicas como Locust y Cloud Load Balancer.
- Crear una app web utilizando React, uno de los frameworks con más demanda laboral en el mercado.
- Utilizar una amplia gama de servicios de la nube IaaS, PaaS y SaaS del proveedor GCP.
- Utilizar lenguajes modernos para la creación de una aplicación distribuida.
- Comparar dos diferentes bases de datos en la nube: Cosmos DB y Cloud SQL.

DESCRIPCIÓN GENERAL

A partir de la finalización de los juegos olímpicos Tokio 2021, se tiene la iniciativa por parte de su grupo de estudiantes de sistemas operativos 1 de realizar un visualizador de las noticias y comentarios de las olimpiadas que están realizando los espectadores.

El sistema será totalmente en la nube, utilizando diferentes servicios de Google Cloud Platform y una base de datos de Cosmos DB en Microsoft Azure. Contará con una carga masiva de datos a partir de diferentes generadores de tráfico, la información a mandar será detallada más adelante. Además de este sistema, se contará con un modo “Administrador” en el cual se podrán visualizar gráficas y métricas relevantes de las noticias, y de la información de la RAM y procesos de las máquinas virtuales que se tendrán en la nube.

DIAGRAMA GENERAL DEL SISTEMA



Para una mejor resolución de la imagen y más información, consultar repositorio del laboratorio.

<https://github.com/leoaguilar97/so1-course/>

EXPLICACIÓN DEL FLUJO GENERAL

El sistema contará con diferentes partes, cada una de ellas es crucial para que el sistema funcione correctamente. Estas partes están divididas por colores en el diagrama.

MÁQUINA LOCAL FÍSICA (RECTÁNGULO MORADO)

Los componentes en esta parte estarán guardados en la máquina local de los integrantes, **estos no necesitan estar en la nube.**

La finalidad de las aplicaciones que están en esta parte es enviar tráfico al Load Balancer de Google Cloud (detallado más adelante). Básicamente estas aplicaciones tienen que tener las siguientes funcionalidades:

- Leer un archivo .json que contenga un array de datos (los datos respectivos a cada generador de tráfico).
- Procesar el archivo y enviar tráfico a un endpoint.
- Mostrar cuantos datos fueron enviados y cuantos datos tuvieron error.

La información a enviar por los tres generadores de tráfico es la siguiente:

```
{
  "nombre": "Leonel Aguilar", // Nombre del publicador de la noticia o comentario
  "comentario": "El día de hoy el atleta [...]", // Comentario realizado
  "fecha": "24/07/2021", // Fecha en la que se realizo el comentario
  "hashtags": ["remo", "atletismo", "natacion"], // Hashtags o etiquetas del comentario
  "upvotes": 100, // Cantidad de personas a las que les gusto el comentario
  "downvotes": 30 // Cantidad de personas a las que no les gusto el comentario
}
```

GOOGLE LOAD BALANCER

Se debe configurar un Cloud Load Balancer <https://cloud.google.com/load-balancing> el cual recibirá todo el tráfico creado por Locust y los otros dos generadores de tráfico, este redireccionará el tráfico a los diferentes servicios a implementar.

API RÉPLICAS (RECTÁNGULO AZUL)

Se realizarán tres APIs que funcionarán de exactamente la misma manera (por ello el “réplica”) pero serán realizadas en diferentes lenguajes. La funcionalidad de estas APIs es recibir el tráfico que se genere y pase por el Load Balancer, para luego pasar esta información a la base de datos y crear una notificación utilizando Google PubSub. El estudiante deberá considerar qué rutas son las necesarias para pasar este tráfico, pero como inicio y sugerencia se tienen las siguientes:

- **/iniciarCarga:** Una ruta que podrá conectarse a la base de datos y esperar los datos a cargar.
- **/publicar:** Una ruta que tendrá la opción de publicar la información a la base de datos.
- **/finalizarCarga:** Una ruta que cerrará la conexión a las bases de datos y mandará una notificación a través de Google PubSub.

Los tres lenguajes a implementar serán:

- **Python**
- **Go**

- Rust

En cada uno de estos lenguajes deberán crear la misma funcionalidad de la API para conectarse a la base de datos y enviar información enviada desde los generadores de tráfico. Además, se debe crear una notificación de cuantos datos fueron cargados en Google Pub/Sub.

SERVICIOS EN GOOGLE CLOUD PLATFORM

CLOUD RUN – APIS

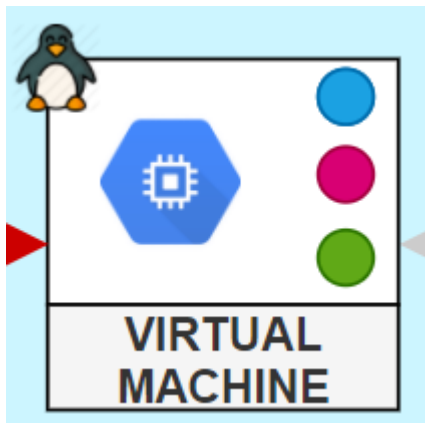
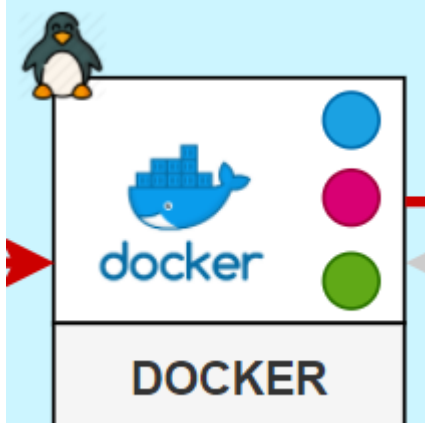
En Google Cloud Run se cargará el servicio de la API de Python y de Go.

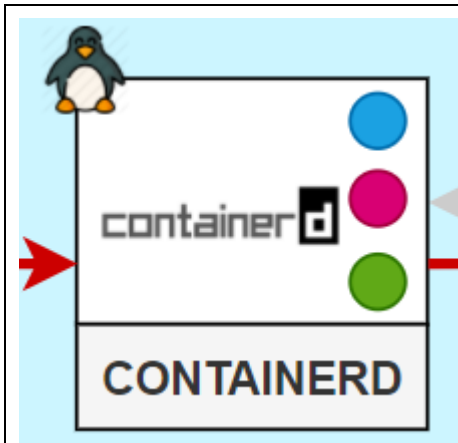
CLOUD FUNCTIONS

En Google Functions se cargará el servicio de la API de Python y de Go.

MÁQUINAS VIRTUALES EN LA NUBE

Se tendrán tres máquinas virtuales en la nube en Google Compute Engine, cada una con las mismas APIs que se detallaron anteriormente. La única diferencia es la forma de instalar y configurar las APIs, que se detalla a continuación:

	<p>En esta máquina Virtual se tendrán las tres APIs: Python, Rust y Go. Esta máquina Virtual utilizará el sistema operativo Debian.</p> <p>En esta máquina virtual NO se tendrá ningún tipo de virtualización a nivel de sistema operativo, esto quiere decir que se configurarán los ambientes de las tres APIs de manera independiente y en el sistema operativo afitrión.</p> <p>Notar como hay un símbolo de Linux en la parte superior izquierda, esto quiere decir que en esta máquina virtual se instalarán módulos Kernel.</p>
	<p>En esta máquina Virtual se tendrán las tres APIs: Python, Rust y Go. Esta máquina Virtual utilizará el sistema operativo CentOS.</p> <p>En esta VM se utilizará Docker como controlador de containers de las tres APIs anteriormente mencionadas. Esto quiere decir que tendrán un contenedor con Python, otro con Rust y otro con Go.</p> <p>Notar como hay un símbolo de Linux en la parte superior izquierda, esto quiere decir que en esta máquina virtual se instalarán módulos Kernel.</p>

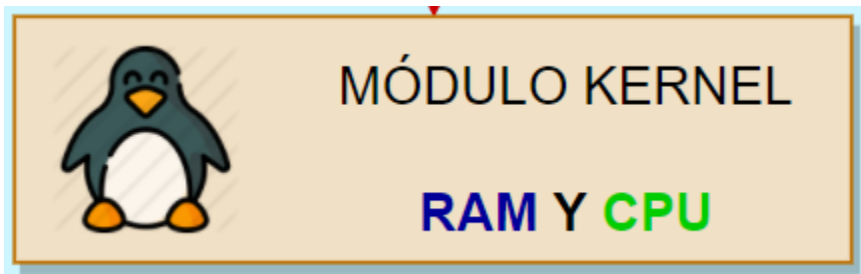


En esta máquina Virtual se tendrán las tres APIs: Python, Rust y Go. Esta máquina Virtual utilizará el sistema operativo **Ubuntu**.

En esta VM se utilizará Containerd como controlador de containers de las tres APIs anteriormente mencionadas. Esto quiere decir que tendrán un contenedor con Python, otro con Rust y otro con Go.

Notar como hay un símbolo de Linux en la parte superior izquierda, esto quiere decir que en esta máquina virtual se instalarán módulos Kernel.

MÓDULO RAM



Montar un módulo que lea la RAM del sistema. Este módulo debe de proveer de la siguiente información:

- Memoria RAM total (en MB)
- Memoria RAM en uso (en MB)
- Memoria RAM libre (en MB)
- Porcentaje de memoria RAM siendo utilizada.

Para validar que sea correcto, debe mostrar los mismos valores que el comando:

```
$ free --mega -t
```

MÓDULO LISTA DE PROCESOS

Montar un módulo que lea la información del CPU del sistema. Este módulo debe proveer la siguiente información:

- Porcentaje de CPU utilizado
- Cantidad de procesos ejecutándose

Para validar que sea correcto, debe mostrar valores parecidos a los obtenidos con el comando:

```
$ free --mega -t
```

Los módulos Kernel a montar en el sistema operativo deberán ser desarrollados en lenguaje C como fue visto en el laboratorio. Deberán ser montados en las tres máquinas virtuales que contienen las APIs. La forma de leer este archivo y mandarlo al módulo de monitoreo queda a discreción del estudiante. Se recomienda que:

- Se implemente un endpoint en la API de su elección (la más sencilla sería en Python) para leer el contenido de los archivos desde el directorio /proc y enviarlo por una petición HTTP al módulo de monitoreo.

MÓDULO DE MONITOREO



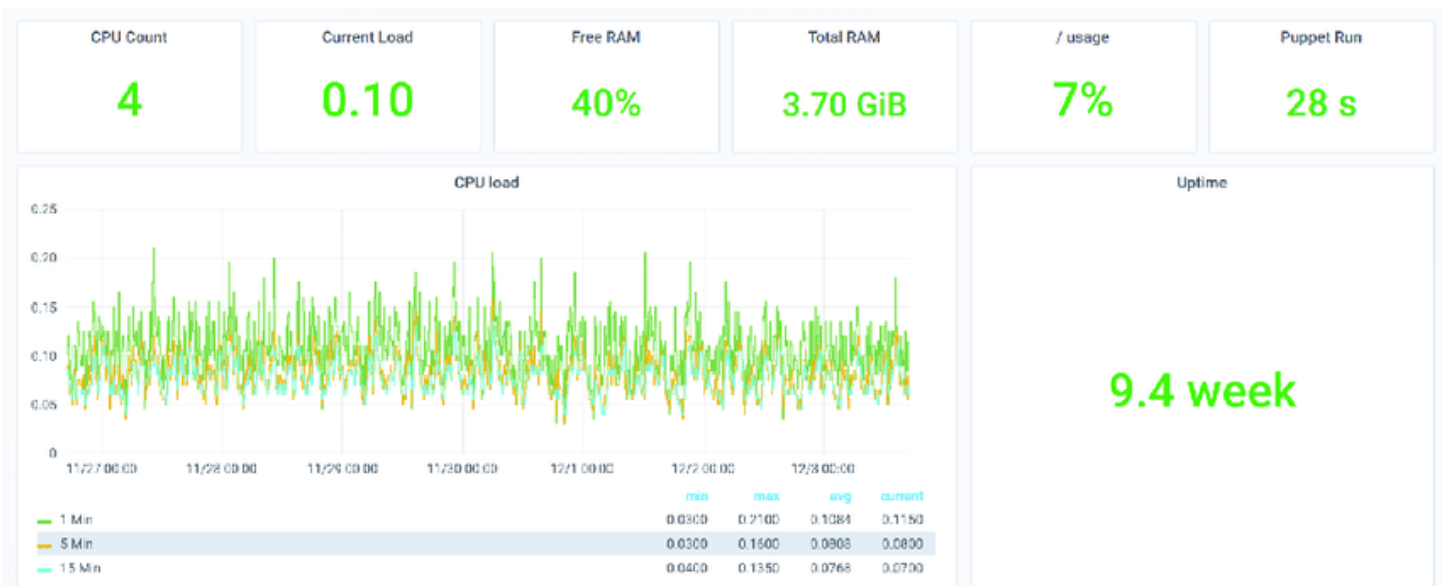
En este módulo se pretende generar un Dashboard utilizando Prometheus y Grafana donde se observe la información de la RAM (toda la que aparece en el Kernel) y la información del CPU y cantidad de procesos ejecutándose. La manera de obtener esta información queda a discreción del estudiante. Se recomienda:

- Implementar una API NodeJS que pida la información por medio de una petición HTTP a las máquinas virtuales donde se encuentran los módulos Kernel y leer la información desde NodeJS hasta prometheus.

Para más información consultar acá: <https://codersociety.com/blog/articles/nodejs-application-monitoring-with-prometheus-and-grafana>

El módulo de monitoreo debe estar en una máquina virtual, ya sea una dedicada únicamente para esto, o una de las que ya tienen de las APIs. La manera de implementar el dashboard de grafana queda a discreción del estudiante, sin embargo se recomienda que se utilicen Dashboards previamente realizados y únicamente acoplarlos a la información manejada de la RAM y CPU.

Un ejemplo de cómo se puede ver este reporte:



GOOGLE PUB SUB

Se utilizará Google Pub/Sub para enviar notificaciones acerca de la carga de información desde las APIs hasta la API principal de NodeJS. La información de esta notificación es la siguiente:

```
{
  "guardados": 1000, //cantidad de registros almacenados
  "api": "Python", //desde qué API se guardó la info, puede ser RUST, PYTHON O GO
  "tiempoDeCarga": 10, //Tiempo que tardo la transaccion en segundos
  "bd": "CosmoDB", //Base de datos en la que se guardo la info, CosmoDB o CloudSQL
}
```

La notificación se disparará cada vez que se finalice la carga de uno de los generadores de tráfico, es decir, no llegará una notificación por cada registro guardado, sino una notificación por el total de los registros. Si desde el generador envió 200 registros, se deberá mostrar una única notificación con el total de guardados de 200.

APP ENGINE

Esta API estará escrita con Javascript utilizando NodeJS. La finalidad de esta API será recolectar los datos para ser mostrados en la aplicación del cliente. Además de realizar reportes con la información que se tiene almacenada (detallado posteriormente). **Esta API estará publicada en el APP ENGINE de Google Cloud.**

Importante: Esta API se conectará a la APP a través del módulo **Sockets.IO**. para simular una aplicación en tiempo real.

CLOUD RUN - APP

APP WEB

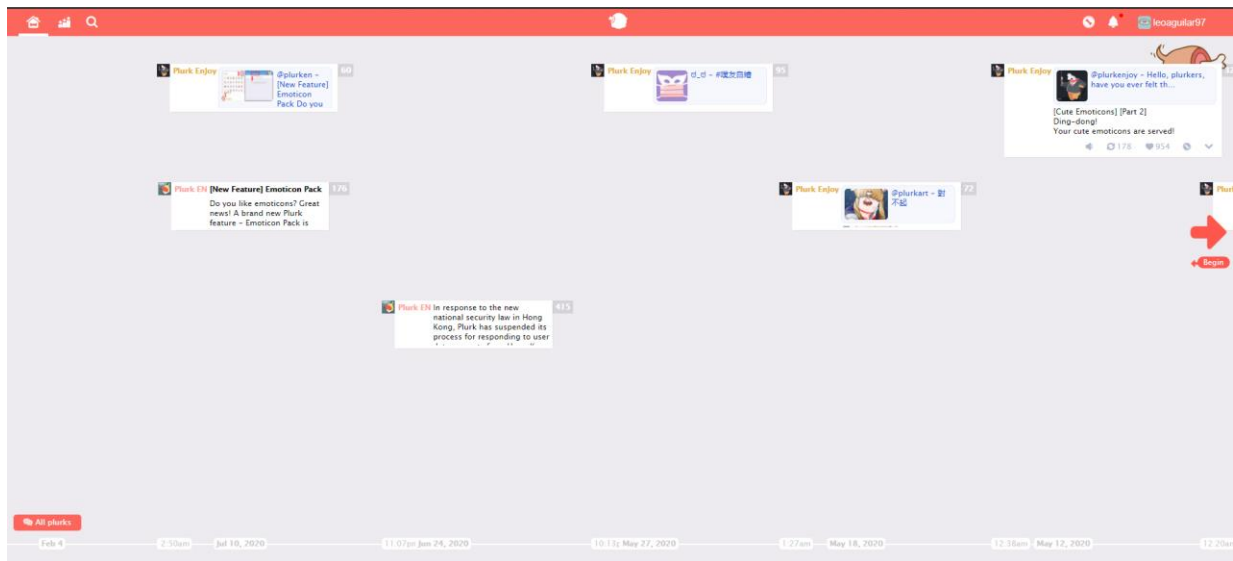
Realizar una app web utilizando el framework React. Esta mostrará las diferentes métricas a partir de los datos guardados en el servidor de CosmosDB y de CloudSQL. Se tomará en cuenta en la calificación la aplicación de conceptos de UX/UI y de responsiveness para el diseño de la app. Por último, la APP realizada en React debe ser publicada utilizando Google Cloud Run.

VISTA DE INFORMACIÓN

La información debe ser mostrada en su totalidad. Las vistas en las que se mostrará la información queda a discreción del estudiante, sin embargo tomar en cuenta que se calificará la estética y usabilidad de esta. Para darse una idea de cómo mostrar la información puede usar como referencia la página <http://twitter.com>.

Algunas referencias de como manejar las vistas se detallan a continuación:

VISTA DE NOTICIAS Y MENSAJES



VISTA DE UNA NOTICIA

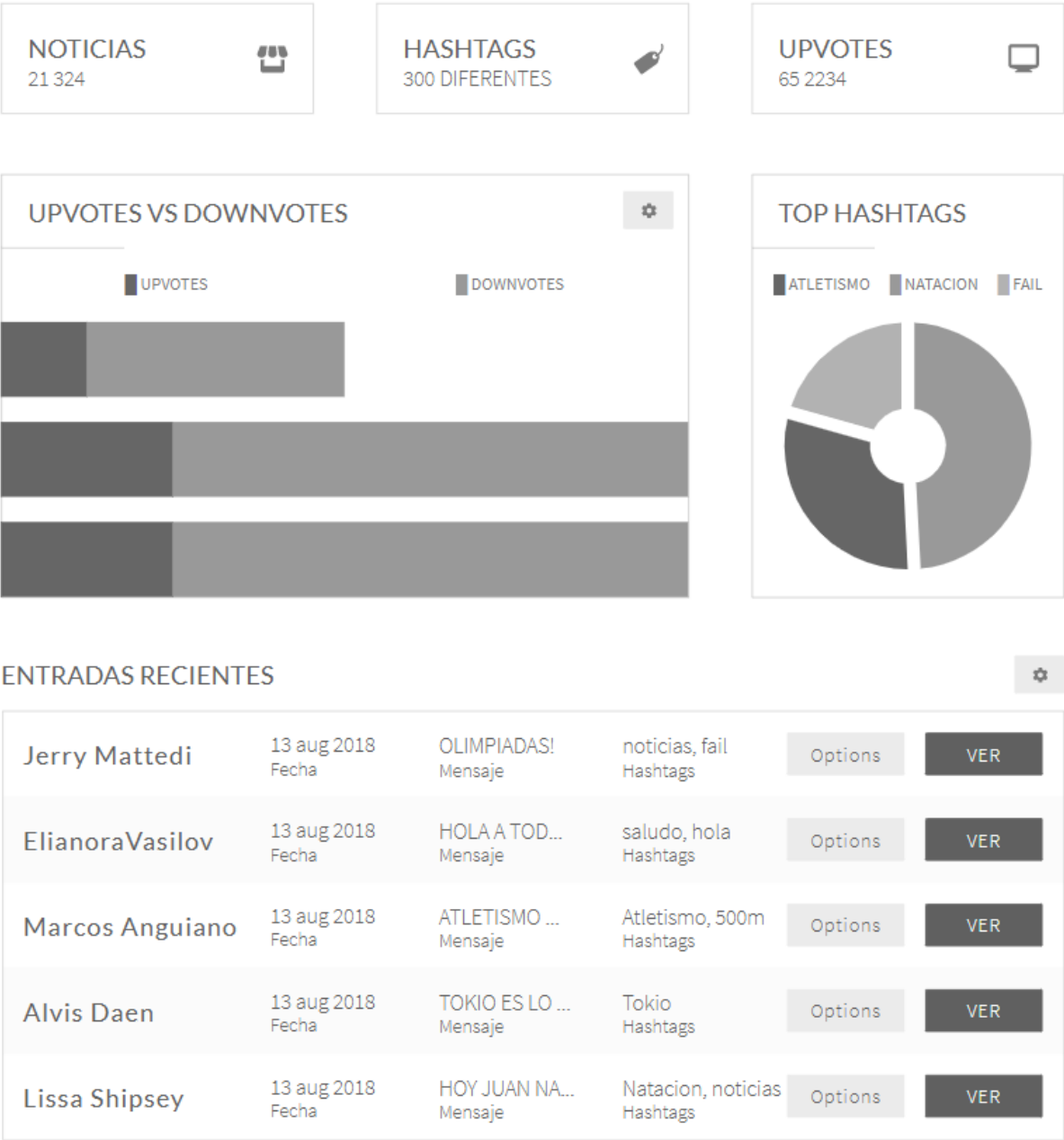


VISTA DE REPORTES

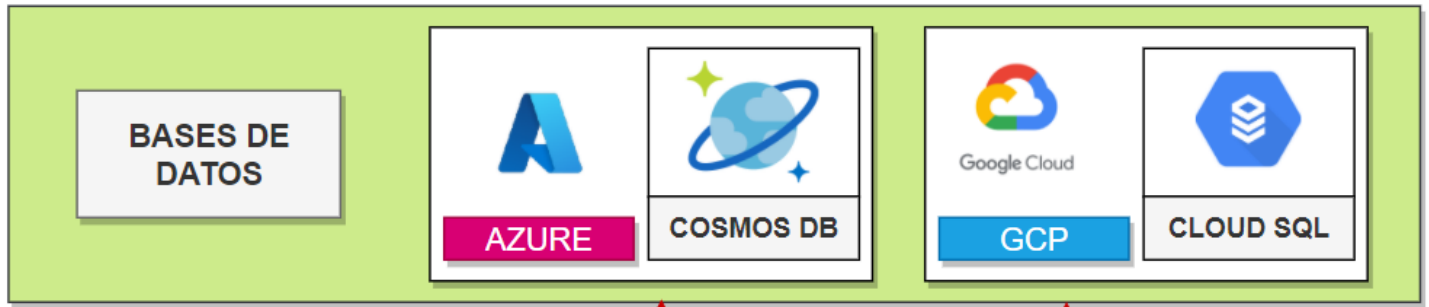
Aparte de la información mostrada de la base de datos, se le pide realizar los siguientes reportes en forma de tablas, estos reportes **deberán de ser en tiempo real utilizando Sockets.IO**.

- Mostrar el total de noticias, upvotes y hashtags (diferentes) que hay en el sistema.
- Tabla con los datos que están en el sistema.
- Gráfica circular del top 5 de hashtags. Se calculará a partir de la cantidad de upvotes que tiene cada hashtag.
- Gráfica (de barras, líneas, stackeada, etc...) que compare la cantidad de upvotes y downvotes por día.

También se debe agregar un botón o filtro con el cual se pueda decidir desde qué base de datos realizar los reportes (es decir, deben implementar los reportes en ambas bases de datos). Se recomienda utilizar la siguiente librería: <https://canvasis.com/react-charts>



BASES DE DATOS



Uno de los objetivos de este proyecto es la comparativa entre la base de datos CosmosDB y utilizar el servicio de CloudSQL, realizando también un proyecto multi-cloud.

Se deberá configurar una base de datos Cosmos DB utilizando Microsoft Azure y otra base de datos de su elección (mysql, postgresql, sql) utilizando CloudSQL en Google Cloud Platform.

Ambas bases de datos contendrán la misma información en cada una. La información llegará a ellas a través de las API Réplicas descritas anteriormente, y serán consumidas por la API Rest para ser procesada y posteriormente ser visualizada en la app web en React. La manera en la que se almacenará la información queda a discreción del estudiante.

PREGUNTAS DE REFLEXIÓN

Las preguntas constituirán una ponderación elevada del punteo total del proyecto, y solamente se calificarán las respuestas que tengan una implementación que la respalde, es decir, se debe de realizar el módulo de Prometheus y Grafana para optar a responder la pregunta de ese módulo. Todas las preguntas deben ser entregadas con su respectiva respuesta en el manual técnico, y se preguntarán en la calificación. Las preguntas a responder son las siguientes:

- ¿Qué generador de tráfico es más rápido? ¿Qué diferencias hay entre las implementaciones de los generadores de tráfico?
- ¿Qué lenguaje de programación utilizado para las APIs fue más óptimo con relación al tiempo de respuesta entre peticiones? ¿Qué lenguaje tuvo el performance menos óptimo?
- ¿Cuál de los servicios de Google Cloud Platform fue de mejor para la implementación de las APIs? ¿Cuál fue el peor? ¿Por qué?
- ¿Considera que es mejor utilizar Containerd o Docker y por qué?
- ¿Qué base de datos tuvo la menor latencia entre respuestas y soportó más carga en un determinado momento? ¿Cuál de las dos recomendaría para un proyecto de esta índole?
- Considera de utilidad la utilización de Prometheus y Grafana para crear dashboards, ¿Por qué?

OBSERVACIONES

- El proyecto será realizado en grupos no mayores a 3 personas.
- Se deberá crear un repositorio de GitHub o Gitlab **privado** donde realizarán colaboraciones todos los integrantes del equipo.
- Se revisarán los commits y que cada integrante haya realizado aportes al repositorio.

- Solamente se tomarán en cuenta las implementaciones con los lenguajes y restricciones mencionadas.
- Se entregará un archivo de entradas el día de la calificación, sin embargo un archivo con la misma estructura será entregado desde el inicio para que puedan realizar pruebas.
- Si se encuentran copias, el estudiante recibirá una puntuación de 0 puntos y será reportado a la escuela de ciencias y sistemas.
- No se aceptarán entregas tarde.

ENTREGABLES

El modo de entrega será via **UEDI**, únicamente adjuntar el link a su repositorio, la entrega será por grupo así que una única persona deberá entregar, se habilitará el espacio en días cercanos a la fecha de entrega. Únicamente deben entregar los grupos que creen terminar para la fecha estipulada.

REPOSITORIO DE GITHUB

Aparte de agregar a cada auxiliar, el repositorio debe contener los siguientes archivos: (En la organización de carpetas que mejor le parezca al estudiante).

- Dockerfiles
- Docker-compose files
- Código fuente React
- Locust file
- Generador de tráfico de Python
- Generador de tráfico de Go
- Código de la API en Python
- Código de la API en Rust
- Código de la API en Go
- Código utilizado con Google Pub Sub
- Código de JavaScript utilizado para la API NodeJS
- Código de C del módulo RAM
- Código de C del módulo de Procesos
- Cualquier otro archivo necesario de configuración
- Modelos de entidad relación o de objetos de la base de datos
- **Manual técnico, con las respuestas a las preguntas de reflexión.**
- **Manual de usuario**

FECHA DE ENTREGA

VIERNES 24 DE SEPTIEMBRE DEL 2021