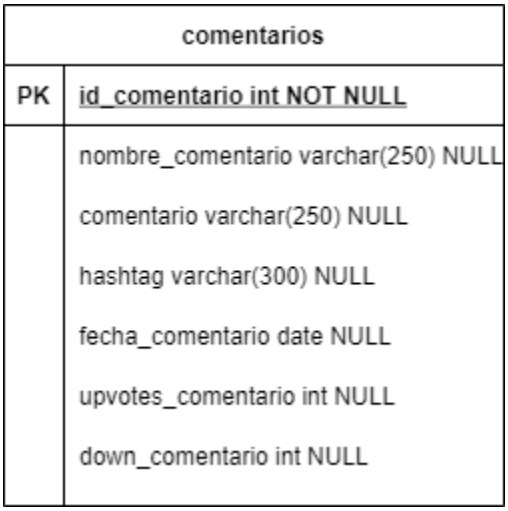
Manual Tecnico

Modelo Base de datos



Descripción de Herramientas

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.

La ejecución de Docker en AWS les ofrece a desarrolladores y administradores una manera muy confiable y económica de crear, enviar y ejecutar aplicaciones distribuidas en cualquier escala.

Docker le proporciona una manera estándar de ejecutar su código. Docker es un sistema operativo para contenedores. De manera similar a cómo una máquina virtual virtualiza (elimina la necesidad de administrar directamente) el hardware del servidor, los contenedores virtualizan el sistema operativo de un servidor. Docker se instala en cada servidor y proporciona comandos sencillos que puede utilizar para crear, iniciar o detener contenedores.

Doker-Compose

Compose es una herramienta para definir y ejecutar aplicaciones Docker de contenedores múltiples. Con Compose, utiliza un archivo YAML para configurar los

servicios de su aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde su configuración.

Compose funciona en todos los entornos: producción, puesta en escena, desarrollo, pruebas, así como flujos de trabajo de CI.

Usar Compose es básicamente un proceso de tres pasos:

- Defina el entorno de su aplicación con un Dockerfile para que pueda reproducirse en cualquier lugar.
- Defina los servicios que componen su aplicación docker-compose.yml para que puedan ejecutarse juntos en un entorno aislado.
- Ejecutar docker-compose up y Compose inicia y ejecuta toda su aplicación.

Containerd

Pensemos un momento en cómo es que funcionan los contenedores. Realmente, un contenedor es una abstracción de varias funcionalidades del kernel de linux. Para ejecutar un contenedor es necesario realizar llamadas al sistema (denominadas syscalls) para crear un ambiente *containerizado*.

Estas llamadas y configuraciones varían de versión en versión, y de plataforma en plataforma. Containerd abstrae esta funcionalidad de tan bajo nivel y funciona como una capa más amigable para utilizar contenedores, para que se puedan utilizar a partir de más software.

Anteriormente Kubernetes tenía unicamente dos opciones: seguir utilizando el Dockershim que tenía la interfaz de Docker, o interactuar con el kernel de linux directamente. Al sacar containerd de Docker, una alternativa nueva surgió: Utilizar containerd sin mezclar a Docker y toda la funcionalidad que no era necesaria.

¿Cómo se combina todo?

Docker: Docker es un software orientado a desarrolladores, que tiene un nivel muy alto de abstracción en su interfaz y permite crear y ejecutar contenedores desde su cliente. Actualmente utiliza a Containerd como su capa baja de ejecución de contenedores.

Kubernetes: Es un orquestador de contenedores que permite trabajar con múltiples interfaces de ejecución de contenedores (entre ellas, containerd). Está enfocado en ser utilizado para la escalabilidad y despliegue de contenedores a través de múltiples sistemas. Anteriormente, estaba totalmente ligado con Docker.

Containerd: Es una abstracción de todas las funcionalidades del kernel de linux necesarias que son necesarias para ejecutar contenedores. Está diseñado para ser una base de otras herramientas para el manejo de contenedores.

Prometheus

Prometheus es un sistema de monitorización y de alerta open source cuyo origen se remonta en el 2012 en la compañía SoundCloud, y desde allí (conjunto con el crecimiento de k8s) ha sido adoptado por muchas empresas. Prometheus se unió al Cloud Native Computing Foundation en el 2016.

El ecosistema de Prometheus consiste de varios componentes, los principales son los siguientes:

- El servidor de Prometheus para consultar y almacenar la series de datos.
- Un Pushgateway para permitir que los trabajos efímeros y por lotes expongan sus métricas a Prometheus.
- Exporters útil para casos donde no es factible instrumentar un sistema dado con métricas Prometheus directamente.
- Un sistema de manejo de alarmado.
- Un sistema de discovery.

Grafana

Grafana es un software libre basado en licencia de Apache 2.0, que permite la visualización y el formato de datos métricos. Permite crear cuadros de mando y gráficos a partir de múltiples fuentes, incluidas bases de datos de series de tiempo como Graphite, InfluxDB y OpenTSDB. Originalmente comenzó como un componente de Kibana y que luego le fue realizado una bifurcación.

Rust

Rust es un lenguaje de programación compilado, de propósito general y multiparadigma que está siendo desarrollado por Mozilla. Ha sido diseñado para ser "un lenguaje seguro, concurrente y práctico". Es un lenguaje de programación multiparadigma que soporta programación funcional pura, por procedimientos, imperativa y orientada a objetos.

Según la política de Mozilla, Rust es desarrollado de forma totalmente abierta y busca la opinión y contribución de la comunidad. El diseño del lenguaje se ha ido perfeccionando a través de las experiencias en el desarrollo del motor de navegador Servo, y el propio compilador de Rust. Aunque es desarrollado y patrocinado por Mozilla y Samsung, es un proyecto comunitario. Una gran parte de las contribuciones proceden de los miembros de la comunidad.

Para el 2020 es uno de los lenguajes de programación más usados a la hora de trabajar con criptomonedas y crear nodos para minar criptoactivos.

Cloud Run

Desarrolla e implementa aplicaciones en contenedores altamente escalables en una plataforma completamente administrada y sin servidores.

- Codifica a tu manera con los lenguajes que más te gusten (Go, Python, Java, Ruby, Node.js y muchos más)
- Simplifica la administración de la infraestructura para ofrecer una experiencia sencilla a los desarrolladores
- Compilado en el contenedor y los estándares abiertos de Knative, lo que permite la portabilidad de tus aplicaciones

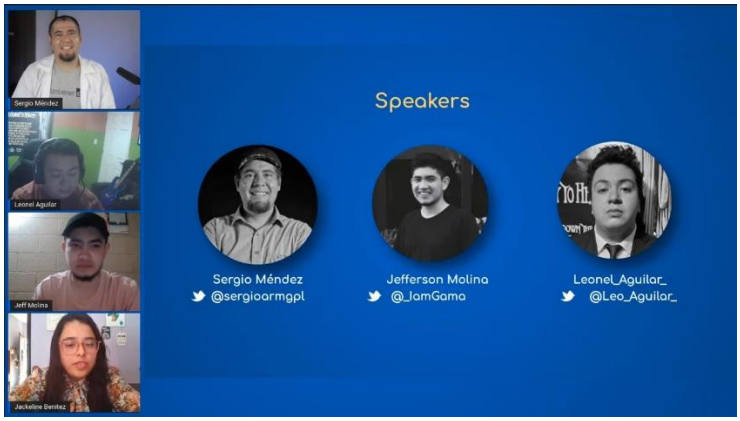
Cloud Functions

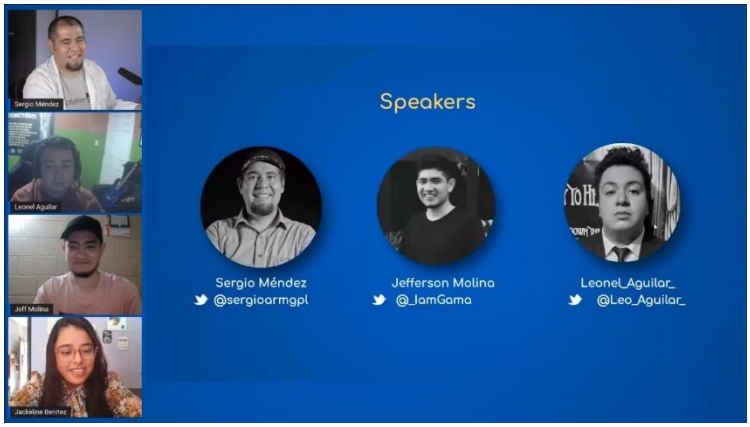
Cloud Functions ofrece al desarrollador una experiencia intuitiva y sencilla. Solo escribe tu código y deja que Google Cloud controle la infraestructura operativa. Desarrolla más rápido mediante la escritura y ejecución de fragmentos pequeños de código que responden a los eventos. Conéctate a Google Cloud o a servicios de nube de terceros mediante activadores para optimizar problemas de organización complejos.

- No tendrás que aprovisionar, administrar ni actualizar servidores
- Escala automáticamente según la carga
- Funciones integradas de supervisión, registro y depuración
- Seguridad integrada a nivel de funciones y por función que se basa en el principio de privilegio mínimo
- Capacidades de red clave para situaciones híbridas y de múltiples nubes

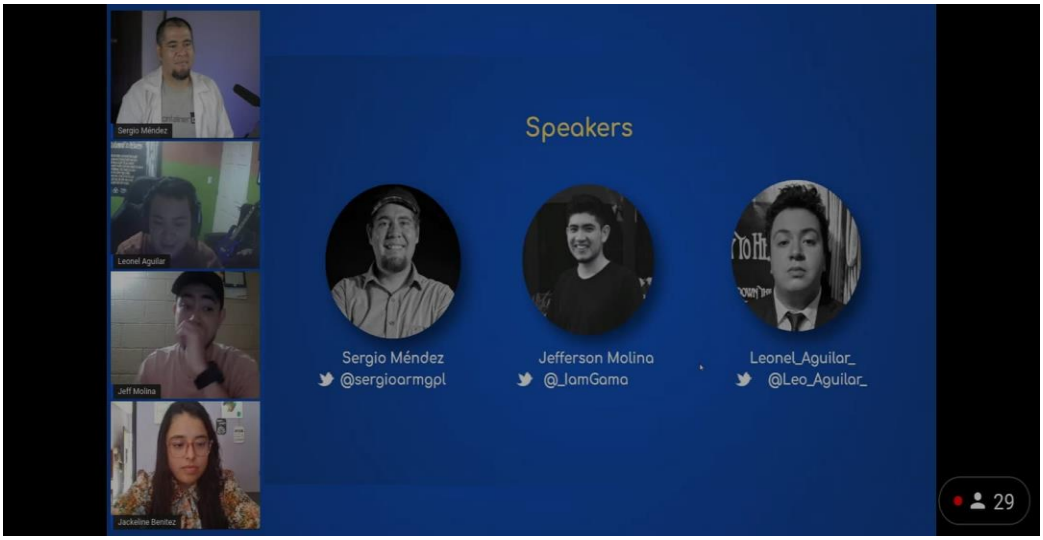
Capturas conferencia

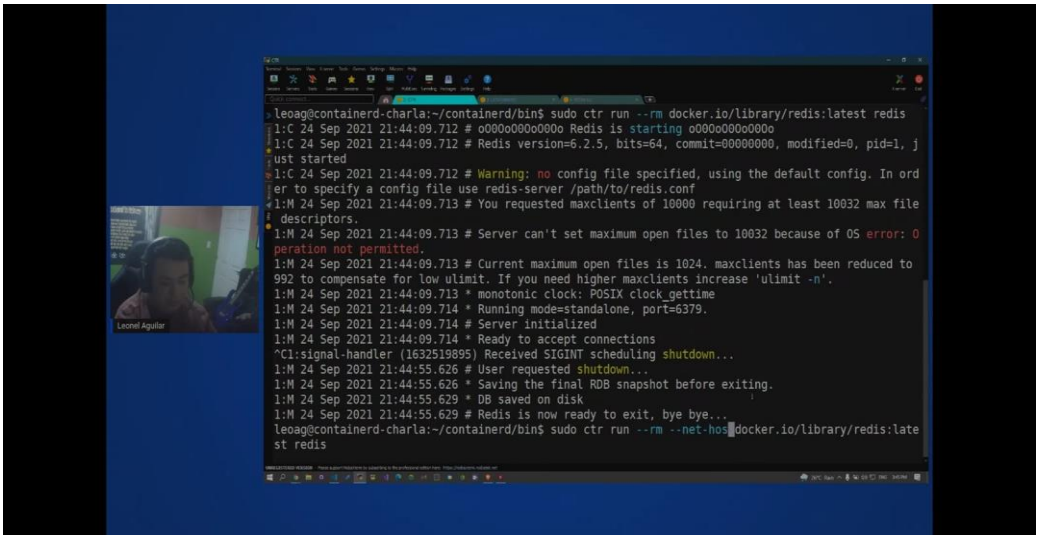
Juan Pablo

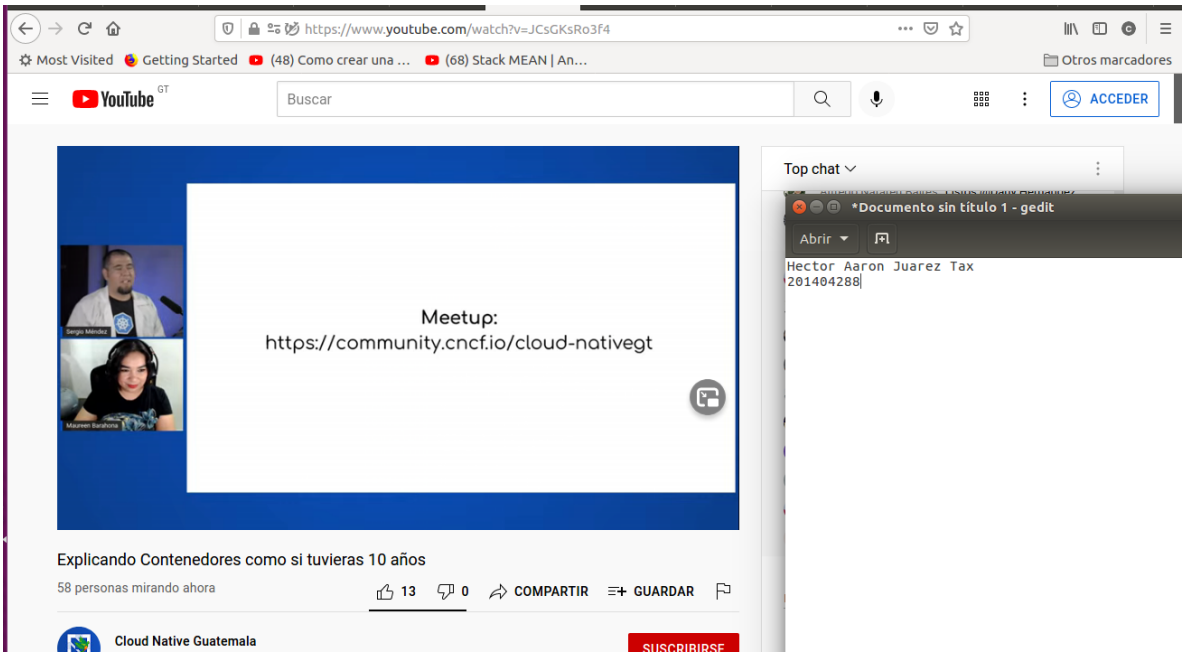


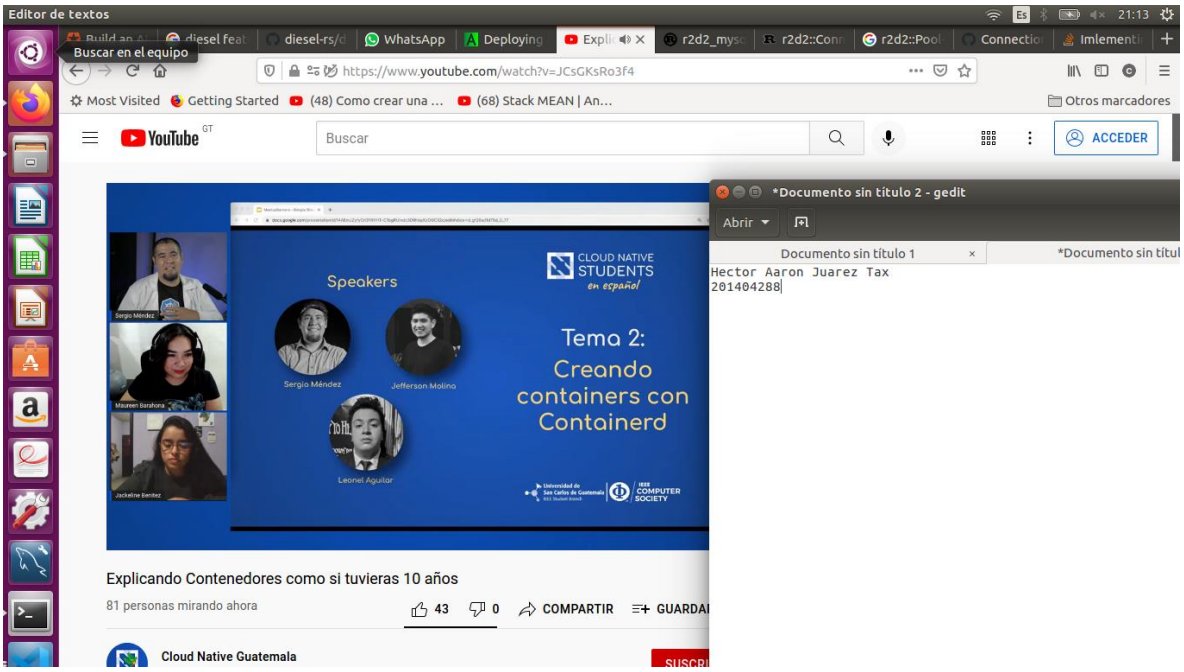


Aaron Juarez



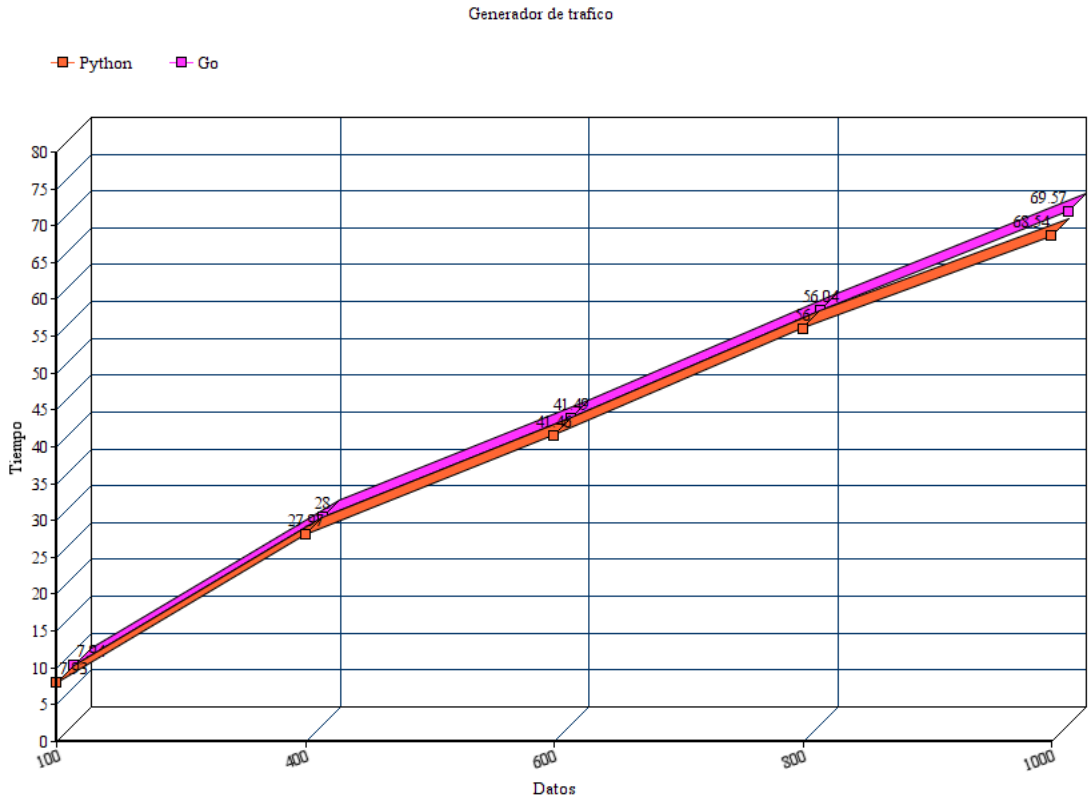



https://www.youtube.com/watch?v=JCsGKsRo3f4
Most Visited
Getting Started
(48) Como crear una ...
(68) Stack MEAN | An...
Otros marcadores
YouTube GT
Buscar
ACCEDER
Meetup:
https://community.cncf.io/cloud-nativegt
Top chat
*Documento sin título 1 - gedit
Abrir
Hector Aaron Juarez Tax
201404288
Explicando Contenedores como si tuvieras 10 años
58 personas mirando ahora
13
0
COMPARTIR
GUARDAR
Cloud Native Guatemala
SUSCRIBIRSE


Editor de textos
Buscar en el equipo
https://www.youtube.com/watch?v=JCsGKsRo3f4
Most Visited
Getting Started
(48) Como crear una ...
(68) Stack MEAN | An...
Otros marcadores
YouTube GT
Buscar
ACCEDER
Speakers
CLOUD NATIVE STUDENTS en español
Tema 2:
Creando containers con Containerd
Sergio Méndez
Jefferson Molino
Leoniel Aguilar
*Documento sin título 2 - gedit
Abrir
Documento sin título 1
Hector Aaron Juarez Tax
201404288
Explicando Contenedores como si tuvieras 10 años
81 personas mirando ahora
43
0
COMPARTIR
Cloud Native Guatemala

Preguntas

1. ¿Qué generador de tráfico es más rápido? ¿Qué diferencias hay entre las implementaciones de los generadores de tráfico?



Grafica Generadores de trafico

Python			Go	
Datos	Tiempo		Datos	Tiempo
100	7.93		100	7.94
400	27.97		400	28
600	41.46		600	41.49
800	56		800	56.04
1000	68.54		100	69.57

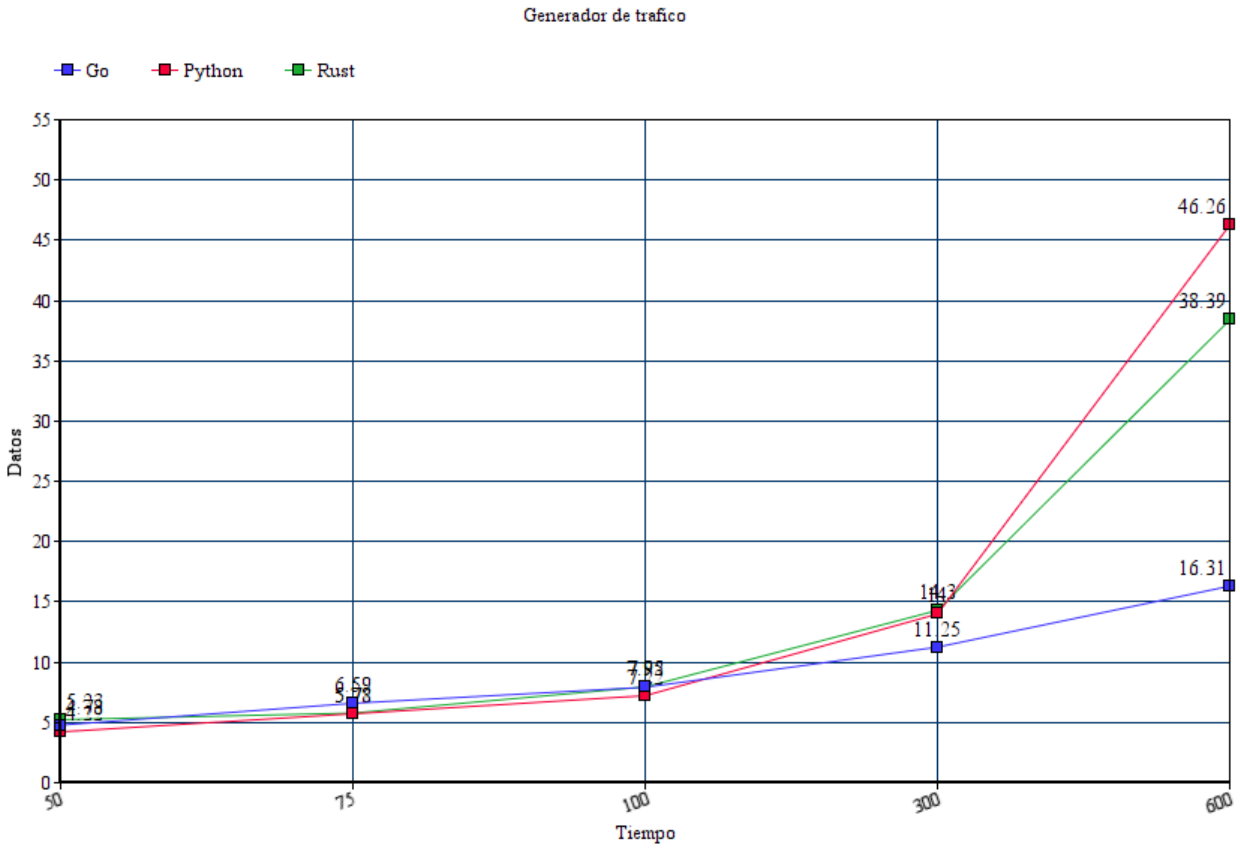
Tabla de datos

```
==========TIEMPO==============
68.54
```

Captura de consola de prueba

La única diferencia en la implementación de los generadores de tráfico es el lenguaje, ya que la lógica de programación en ambas es la misma, se puede notar que los tiempos varían únicamente en centésimas de segundo.

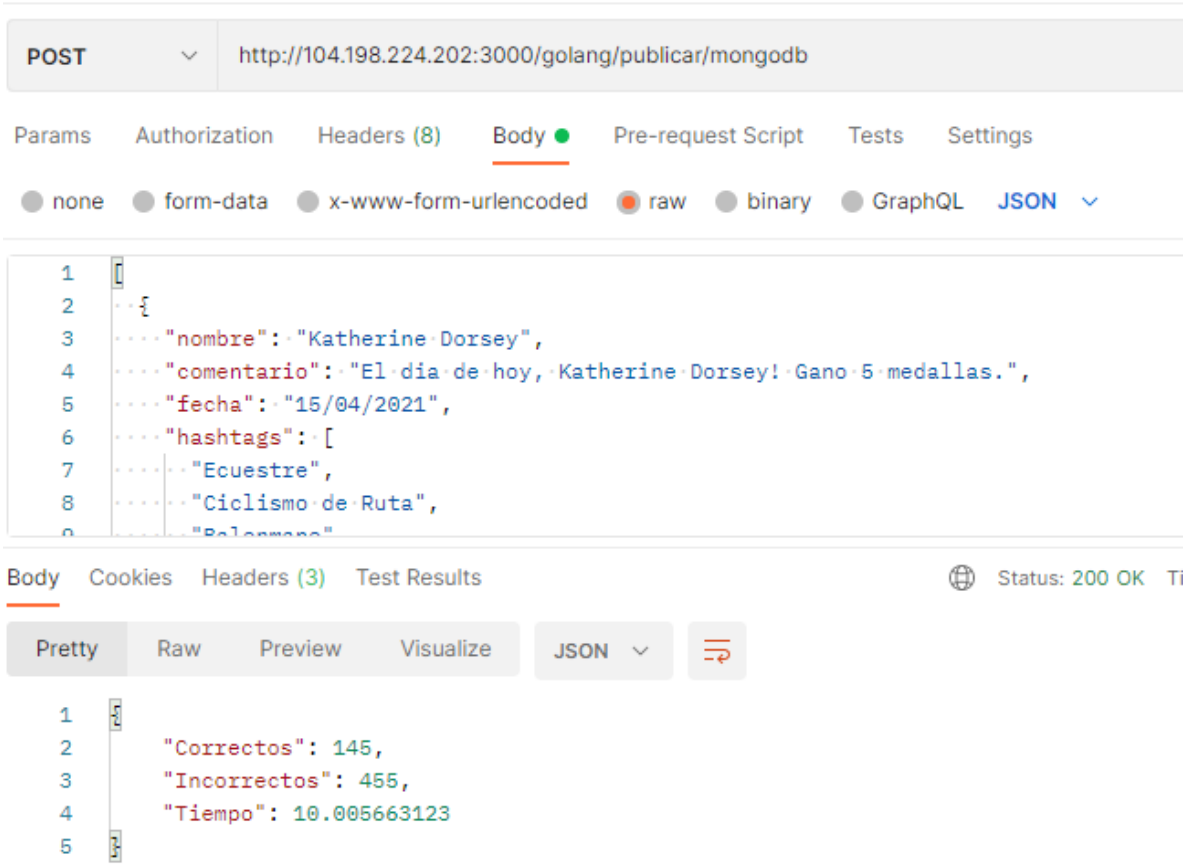
2. ¿Qué lenguaje de programación utilizado para las APIs fue óptimo con relación al tiempo de respuesta entre peticiones? ¿Qué lenguaje tuvo el performance óptimo?



	Go	Python	Rust
Datos	Tiempo	Tiempo	Tiempo
50	4.78	4.22	5.23
75	6.59	5.72	5.78
100	7.92	7.22	7.95
300	11.52	14	14.3
600	16.31	46.26	38.39

La gráfica y los datos anteriores, muestran la relación entre el tiempo y los datos enviados por cada api, de lo cual podemos concluir que la de mejor tiempo de respuesta es el api de Python, y la menos optima es la de Go, debido que durante las pruebas devolvió varios datos fallidos para la

inserción de CosmosDB, por lo cual se muestra un menor tiempo de respuesta.

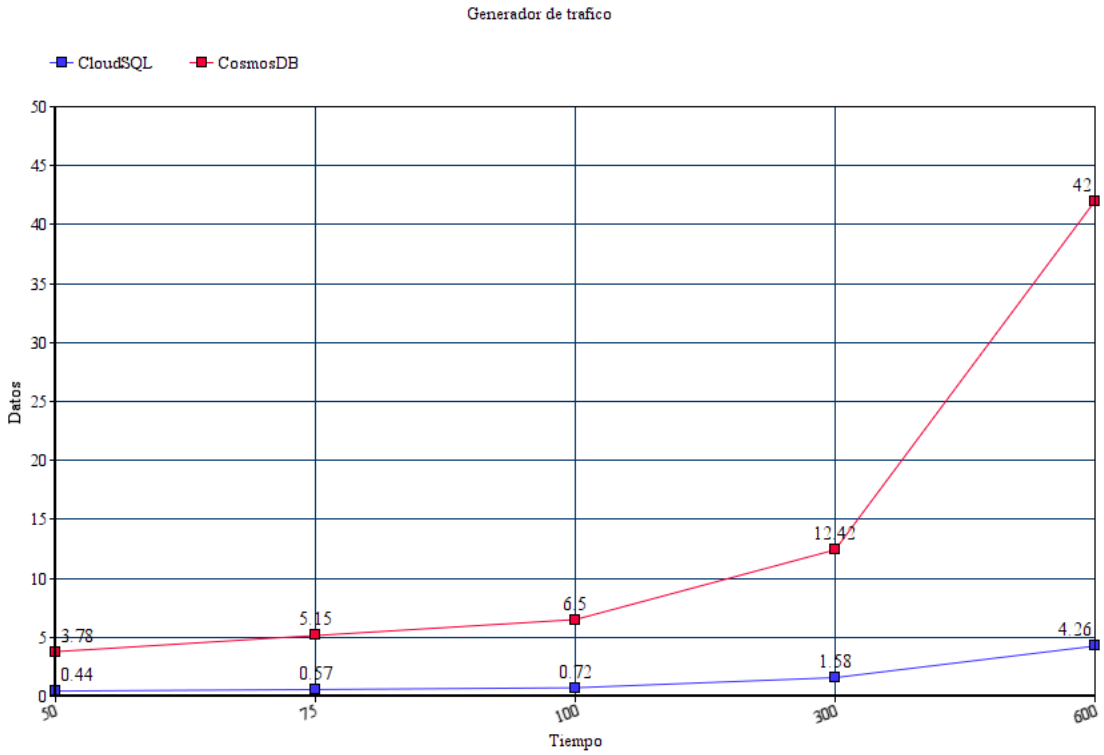


3. ¿Cuál de los servicios de Google Cloud Platform fue de mejor para la implementación de las APIs? ¿Cuál fue el peor? ¿Por qué?
   R// Tomando en cuenta el tiempo de respuesta y la complejidad para utilizarlo, consideramos que el mejor servicio es el de Cloud Run, esto debido a las pocas configuraciones que se deben realizar y a un tiempo de respuesta que mejora al de servicios como el de Cloud Functions, además comparado con este mismo servicio, únicamente se debe importar la imagen de Docker, sin necesidad de modificar el código. O bien realizar complejas instalaciones o saber utilizar la consola de Linux como es el caso de utilizar una máquina virtual en Google Cloud.

4. ¿Considera que es mejor utilizar Containerd o Docker y por qué?
   Teniendo en cuenta que actualmente al utilizar Docker se esta empleando Containerd por defecto, se recomienda utilizar Docker, siendo el único motivo que es más fácil de implementar que Containerd, ya que Containerd esta hecho para servir de base de otras herramientas para el manejo de contenedores, al final no es tan amigable para el usuario. Además, se pudo

observar que en las maquinas en las que corría Containerd netamente, ralentizaban su funcionamiento, no ocurriendo lo mismo en Docker

5. ¿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?



	cloudSQL	CosmosDB
Datos	Tiempo	Tiempo
50	0.44	3.78
75	0.57	5.15
100	0.72	6.5
300	1.58	12.42
600	4.26	42

La gráfica y la tabla anterior muestran la cantidad de datos y el tiempo de respuesta de cada tecnología implementada para las bases de datos, de lo cual podemos recomendar utilizar cloudSQL para un proyecto de esta índole, debido a sus mejores tiempos de inserción.

6. Considera de utilidad la utilización de Prometheus y Grafana para crear dashboards, ¿Por qué?

Si, en una aplicación en el ámbito real, es importante el monitoreo de esta con el fin de saber que nuestros servicios no fallen y si lo hacen saber cómo solucionarlo de mejor manera y Prometheus nos ofrece múltiples opciones de gran utilidad, como alertas, seguimiento de tráfico, saturación entre otras. Además, es de código abierto, lo cual le da un plus. Se integra perfectamente con Grafana, lo cual nos permite apreciar de una manera entendible los datos a través de graficas y dashboards visualmente atractivos, con lo cual cualquier persona podría entenderlos.