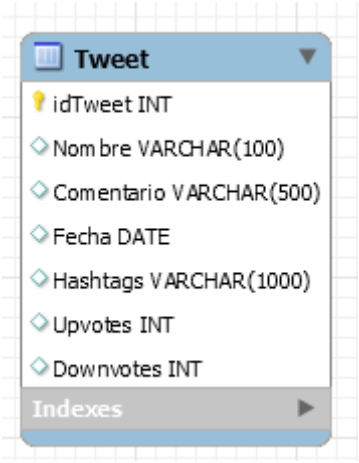
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
ING. SERGIO ARNALDO MENDEZ AGUILAR
AUX. LEONEL AGUILAR
AUX. CARLOS RAMIREZ

Manual Técnico

Nombre	Carné
Cristian Francisco Meoño Canel	201801397
Josue Guillermo Orellana Cifuentes	201801366
César Alejandro Sosa Enríquez	201800555

Contenido

Modelo de Base de Datos



Preguntas de Reflexión

- ¿Qué generador de tráfico es más rápido? ¿Qué diferencias hay entre las implementaciones de los generadores de tráfico?
    - Locust: Muestra una gran velocidad y la flexibilidad que tiene para configurar la cantidad de usuarios y el tiempo entre peticiones es una ventaja destacable, además de las estadísticas que brinda las cuales son de muchísima utilidad.
    - Go Load Tester: Presenta una ralentización a la hora de leer los datos desde el archivo JSON que se enviarán.
    - Python Load Tester: Presenta una velocidad constante y veloz para leer los datos desde el archivo JSON.

    Concluimos que el mejor generador de tráfico es Locust por las ventajas mencionadas anteriormente.

- ¿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?
    - PYTHON:

	python	python docker	pytho containerD	python cloud run
mongo	0.09633803	0.121267606	0.093234848	0.088472222
mysql	0.00671972	0.006164789	0.005872727	0.007536111

    GO:

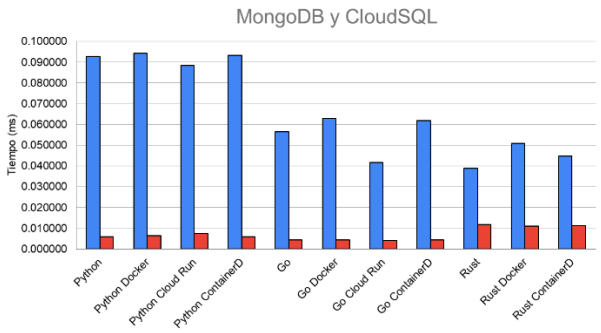
	go	go docker	go containerD	go cloud run
mongo	0.058309859	0.09525424	0.061774194	0.0415
mysql	6.85915E-05	0.00120339	7.54839E-05	0.000914

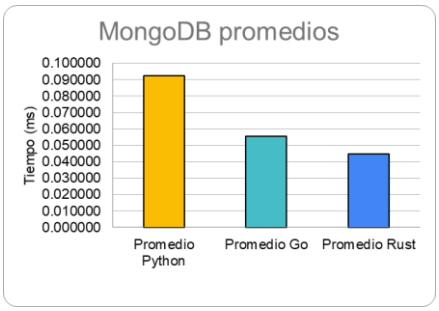
    - El lenguaje para las apis que fue más óptimo fue Go.
    - El lenguaje con el performance ménos óptimo fue Python.
- ¿Cuál de los servicios de Google Cloud Platform fue de mejor para la implementación de las APIs? ¿Cuál fue el peor? ¿Por qué?
    - El mejor servicio fue python con containerD y Go sin contenedores pero igual en engine cloud, en estas máquinas virtuales las APIs se desempeñaron mejor

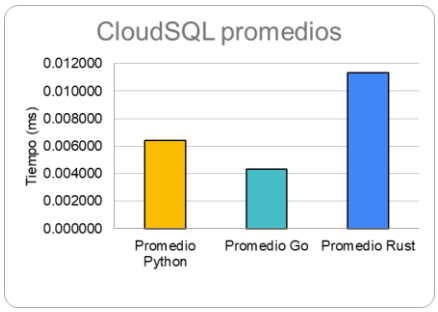
en ambos lenguajes. Mientras que la peor fue para python fue engine cloud con la API en docker y el mismo caso para Go.
Esto lo determinamos viendo los tiempos que se calcularon sobre los datos cargados y así obtuvimos un parámetro para poder comparar las diferentes implementaciones.

- ¿Considera que es mejor utilizar Containerd o Docker y por qué?
    - Nos pareció que Docker presenta una ventaja sobre containerD al estar mejor documentado y ser más sencillo de comenzar a implementar. Pero no ignoramos la ventaja que tiene containerD al ser el estándar de la indurstia, por lo que creemos que es mejor usar containerD pero que es bueno conocer Docker antes para tener un conocimiento previo del funcionamiento de los contenedores.
- ¿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?

Tiempo por dato subido en milisegundos		
	MongoDB	CloudSQL
Python	0.092870	0.005859
Python Docker	0.094381	0.006308
Python Cloud Run	0.088488	0.007536
Python ContainerD	0.093235	0.005873
Go	0.056471	0.004360
Go Docker	0.062951	0.004410
Go Cloud Run	0.041538	0.004220
Go ContainerD	0.061774	0.004350
Rust	0.038785	0.011682
Rust Docker	0.050748	0.011028
Rust ContainerD	0.044766	0.011355







    - La que soportó mejor la carga y tuvo la menor latencia fue CloudSQL.
    - Nuestra recomendación se inclina por CloudSQL ya que la diferencia MongoDB es muy significativa, dentro de las cifras que se manejan con estas tecnologías.
- Considera de utilidad la utilización de Prometheus y Grafana para crear dashboards, ¿Por qué?
    - Sí porque cuando se comprende cómo generar las métricas se hace muy fácil poder generar gráficas y esto ayuda a analizar datos de una manera más sencilla.

Descripción de herramientas

Go

Go es un lenguaje de programación creado en el año 2007 por Google. Go nació sobre todo para mejorar la concurrencia que otros lenguajes ya existentes como Python, Java o C/C++ no eran capaces de manejar correctamente. El software es cada vez más complejo y tiene que hacer más cosas simultáneamente.

Rust

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

Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Locust

Locust es una herramienta de prueba de rendimiento escalable, programable y fácil de usar.

Google Load Balancing

Cloud Load Balancing es una herramienta que puede escalar aplicaciones de Compute Engine desde cero hasta pleno rendimiento, y sin necesidad de prepararlas. Distribuye los recursos de computación con balanceo de carga en una o varias regiones, cerca de tus usuarios y para cumplir tus requisitos de alta disponibilidad.

Google Compute Engine

Un servicio de computación seguro y personalizable con el que puedes crear y ejecutar máquinas virtuales en la infraestructura de Google.

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.

ContainerD

Un estándar de la industria para la ejecución de contenedores con énfasis en simplicidad, robustes y portabilidad.

Prometheus

Prometheus es un sistema de monitoreo de código abierto basado en métricas. Recopila datos de servicios y hosts mediante el envío de solicitudes HTTP en puntos finales de métricas. Luego, almacena los resultados en una base de datos de series de tiempo y los pone a disposición para análisis y alertas.

Grafana

Grafana es una herramienta para visualizar datos de serie temporales. A partir de una serie de datos recolectados obtendremos un panorama gráfico de la situación de una empresa u organización.

Módulos Kernel

Un módulo del kernel es un fragmento de código o binarios que pueden ser cargado y eliminados del kernel según las necesidades de este. Tienen el objetivo de extender sus funcionalidades son fragmentos de código que pueden ser cargados y eliminados del núcleo bajo demanda.

Azure Cosmos DB

Azure Cosmos DB es una base de datos NoSQL totalmente administrada para el desarrollo de aplicaciones modernas. Los tiempos de respuesta de milisegundos de un solo dígito y la escalabilidad automática e instantánea garantizan la velocidad a cualquier escala

GCP Cloud SQL

Cloud SQL para MySQL es un servicio de base de datos totalmente gestionado que le facilita la configuración, el mantenimiento y la gestión de bases de datos MySQL relacionales en Cloud Platform.

PUB/SUB

Pub/Sub permite que los servicios se comuniquen de forma asíncrona, con latencias de alrededor de 100 milisegundos. Se usa para estadísticas de transmisión y canalizaciones de integración de datos con el fin de transferir y distribuir datos.

React

React es una biblioteca escrita en JavaScript, desarrollada en Facebook para facilitar la creación de componentes interactivos, reutilizables, para interfaces de usuario

Módulos Kernel

RAM

La lectura de las cantidades de memoria se hizo a través de la siguiente porción de código:

```c
struct sysinfo inf;

static int get_data(struct seq_file * file, void *v){
    si_meminfo(&inf);
    unsigned long total = (inf.totalram*4);
    unsigned long libre = (inf.freeram*4);
    seq_printf(file, "{\n");
    seq_printf(file,"\"total\": %lu,\n",total/1024);
    seq_printf(file,"\"libre\": %lu,\n", libre/1024);
    seq_printf(file,"\"en_uso\": %lu\n", ((total - libre)*100)/total);
    seq_printf(file, "}\n");
    return 0;
}
```

Se implementó la estructura sysinfo ya que esta cuenta con atributos que brindan la información que necesitamos de manera directa, los campos que trae esta estructura son los siguientes:

Data Fields

__kernel_long_t	uptime
__kernel_ulong_t	loads [3]
__kernel_ulong_t	totalram
__kernel_ulong_t	freeram
__kernel_ulong_t	sharedram
__kernel_ulong_t	bufferram
__kernel_ulong_t	totalswap
__kernel_ulong_t	freeswap
__u16	procs
__u16	pad
__kernel_ulong_t	totalhigh
__kernel_ulong_t	freehigh
__u32	mem_unit
char	_f [20-2 *sizeof(__kernel_ulong_t)-sizeof(__u32)]

Los datos de esta estructura son asignados en la función si_meminfo la cual se incluye desde la librería mm.h y está definida de la siguiente manera:

```c
void si_meminfo(struct sysinfo *val)
{
        val->totalram = totalram_pages();
        val->sharedram = global_node_page_state(NR_SHMEM);
        val->freeram = global_zone_page_state(NR_FREE_PAGES);
        val->bufferram = nr_blockdev_pages();
        val->totalhigh = totalhigh_pages();
        val->freehigh = nr_free_highpages();
        val->mem_unit = PAGE_SIZE;
}
```

CPU:

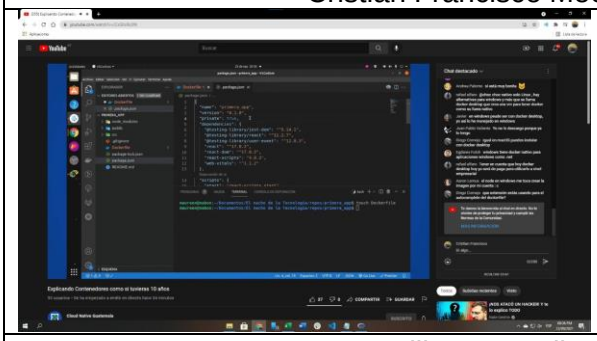
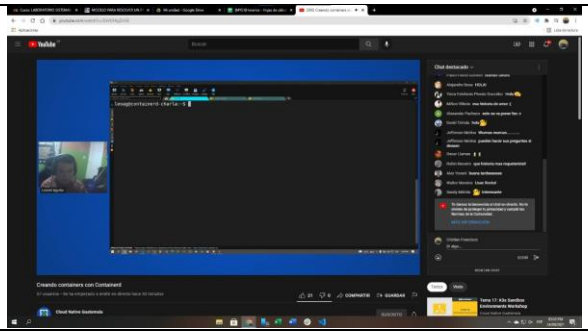
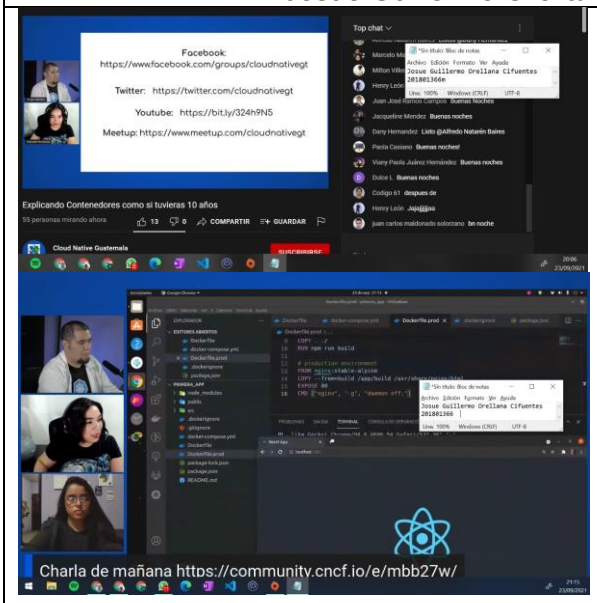
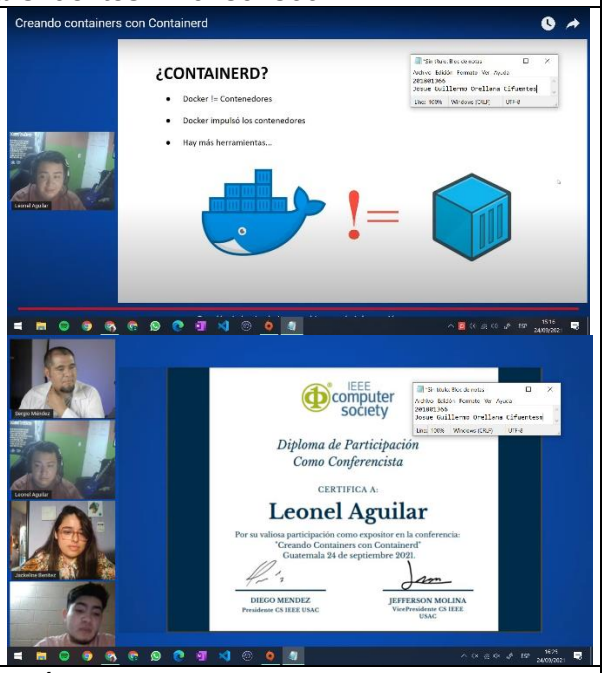
La lectura de la cantidad de procesos se hizo a través de la siguiente porción de código:

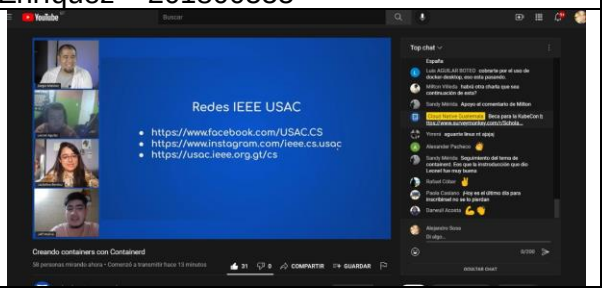
```
struct task_struct *task;

static int proc_cpu_msg(struct seq_file * file, void *v){
    int procesos = 0;
    seq_printf(file, "{\n");
    for_each_process(task){
        procesos++;
    }
    seq_printf(file, "\"procesos\":%d",(procesos));
    seq_printf(file, "\n}");
    return 0;
}
```

Se implementó la estructura task_struct ya que esta es la que se utiliza para hacer el conteo de los procesos a través del método for_each_process, tanto el struct como el método están definidos en la librería sched.h.

Screenshots de Participación

<table>
<tr><th>23 de septiembre</th><th>24 de septiembre</th></tr>
<tr><td colspan="2">Cristian Francisco Meoño Canel – 201801397</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td colspan="2">Josue Guillermo Orellana Cifuentes – 201801366</td></tr>
<tr><td> 
</td><td> 

</td></tr>
<tr><td colspan="2">César Alejandro Sosa Enríquez – 201800555</td></tr>
<tr><td></td><td> 
</td></tr>
</table>

Anexos

Notificacion #10

Guardados: 71
API: Python
Tiempo de Carga: 0.47719836235046387
Base de Datos: MySQL

Notificacion #11

Guardados: 71
API: Python
Tiempo de Carga: 6.842530250549316
Base de Datos: MongoDB

Notificacion #2

Guardados: 71
API: Go
Tiempo de Carga: 4.879107ms
Base de Datos: MySQL

Notificacion #3

Guardados: 71
API: Go
Tiempo de Carga: 4.147472547s
Base de Datos: MongoDB

Notificacion #0

Guardados: 71
API: Python Docker
Tiempo de Carga: 0.43771815299987793
Base de Datos: MySQL

Notificacion #1

Guardados: 71
API: Python Docker
Tiempo de Carga: 8.612148523330688
Base de Datos: MongoDB

Notificacion #4

Guardados: 59
API: Go Docker
Tiempo de Carga: 7.105852ms
Base de Datos: MySQL

Notificacion #5

Guardados: 59
API: Go Docker
Tiempo de Carga: 5.626734743s
Base de Datos: MongoDB

Notificacion #6

Guardados: 66
API: Python ContainerD
Tiempo de Carga: 0.38762998580932617
Base de Datos: MySQL

Notificacion #7

Guardados: 66
API: Python ContainerD
Tiempo de Carga: 6.153530120849609
Base de Datos: MongoDB