MANUAL TÉCNICO

SISTEMAS OPERATIVOS 1

INTEGRANTES

Nombre	Carnet
Selvin Lisandro Aragón Pérez	201701133
Randy Alexander Can Ajuchan	201801527
Luis Pedro Pineda Gonzalez	201700995
Cristian Manasés Juárez Juárez	201700529

Módulos Kernel

Ver la versión del kernel

```
uname -r
```

Revisar módulos instalados, esto listará todos los módulos instalados en el kernel

```
lsmod
```

Instalar dependencias

Descargar headers del módulo específico que tenemos

```
sudo apt-get install linux-headers-$(uname -r)
```

Descargar build essentials, para compilar el código c

```
sudo apt-get install build-essential
```

Instalar modulo

Compilar archivos

```
make
```

Instalar el módulo con el comando *insmod*

https://linux.die.net/man/8/insmod#:~:text=insmod%20is%20a%20trivial%20program,is%20taken%20from%20standard%20input

Revisar los logs de los modulos

```
sudo dmesg
```

Ahora, revisamos el documento generado, cada vez que lo revisemos se reescribirá

```
cat /proc/[nombre_modulo]
```

para construir la imagen

```
sudo docker-compose up -d
```

para levantarlo es

```
sudo docker-compose up
```

Eliminar el modulo

No se espera ninguna salida cuando este comando es exitoso.

```
sudo rmmod "[nombre_modulo]"
```

Pasos a seguir para montar un módulo

```
# Instalar linux headers
$ sudo apt-get install linux-headers-$(uname -r)

# Instalar dependencias para compilar c y c++
$ sudo apt-get install build-essential
```

Para ver el directorio actual, y la versión del kernel, podemos usar los siguientes comandos.

Ya después que tenemos un módulo escrito, en el archivo [nombre_modulo].c:

```
# Ir al directorio del modulo
$ cd [directorio_modulo]

# Compilar archivos
$ make
```

Revisamos la información que generó el make.

```
# Revisar archivos salidos
$ ls -l

# -rw-rw-r-- 1 [usr] [usr]   169 [fecha y hora] Makefile
# -rw-rw-r-- 1 [usr] [usr]    74 [fecha y hora] modules.order
# -rw-rw-r-- 1 [usr] [usr]     0 [fecha y hora] Module.symvers
# -rw-rw-r-- 1 [usr] [usr]  2895 [fecha y hora] [nombre_modulo].c
# -rw-rw-r-- 1 [usr] [usr]  6080 [fecha y hora] [nombre_modulo].ko
# -rw-rw-r-- 1 [usr] [usr]   603 [fecha y hora] [nombre_modulo].mod.c
# -rw-rw-r-- 1 [usr] [usr]  2584 [fecha y hora] [nombre_modulo].mod.o
# -rw-rw-r-- 1 [usr] [usr]  5736 [fecha y hora] [nombre_modulo].o
```

Procedemos a montar el módulo

```
# Instalar el módulo con el comando insmod

https://linux.die.net/man/8/insmod#:~:text=insmod%20is%20a%20trivial%20program,is%20taken%20from%20standard%20input
# No se espera ninguna salida cuando este comando es exitoso.

$ sudo insmod [nombre_modulo].ko
```

Ahora, revisamos que el módulo esté instalado correctamente. Para ver el listado de módulos cargados:

```
  # Revisar los logs de los modulos
  $ sudo dmesg

  # Ahora, revisamos el documento generado, cada vez que lo revisemos se
reescribirá
  $ cat /proc/[nombre_modulo]
```

El resultado de este último comando sera escrito en la pantalla.

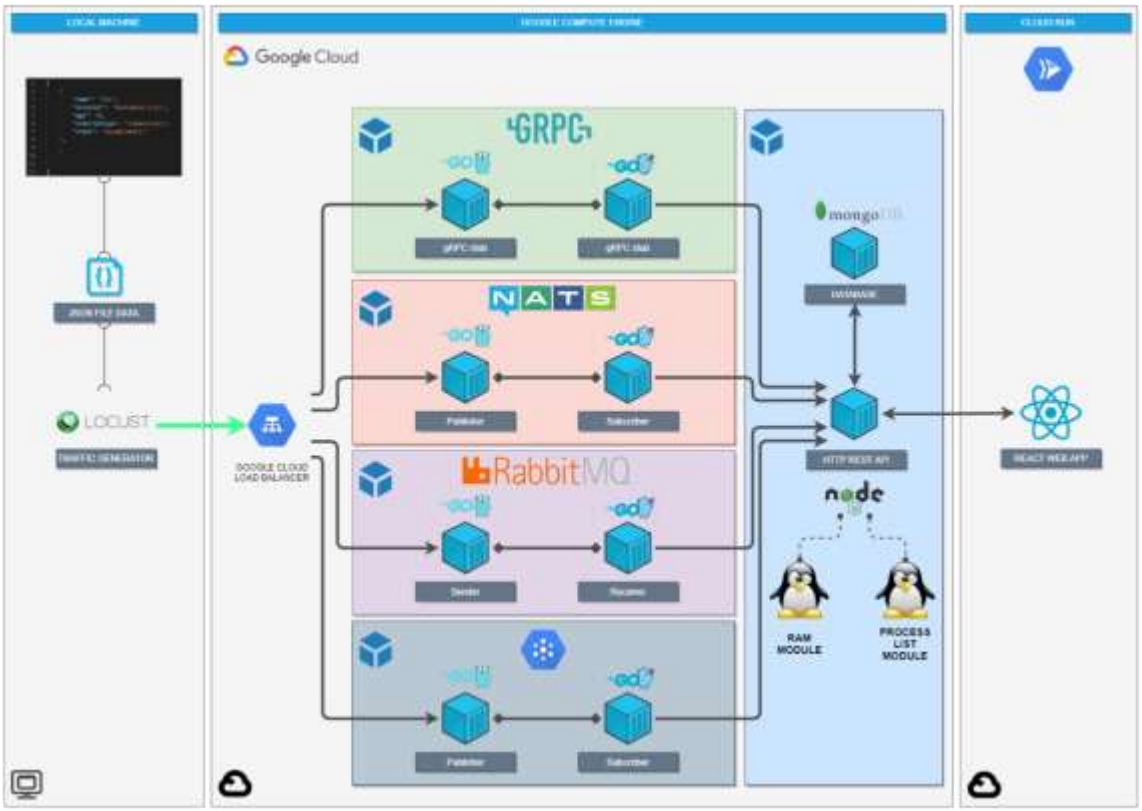
Para eliminar el módulo (para volver a cargarlo o simplemente eliminarlo).

```
  # Eliminar el modulo
  # No se espera ninguna salida cuando este comando es exitoso.

  $ sudo rmmod "[nombre_modulo]"
```

Flujo general del programa

El flujo del programa se puede visualizar en el siguiente diagrama:



gRPC

gRPC es un sistema de llamada a procedimiento remoto de código abierto desarrollado inicialmente en Google. Utiliza como transporte HTTP/2 y Protocol Buffers como lenguaje de descripción de interfaz.

Documentación: https://grpc.io/

NATS

Traducción del inglés-NATS es un sistema de mensajería de código abierto. El servidor NATS está escrito en el lenguaje de programación Go. Las bibliotecas cliente para interactuar con el servidor están disponibles para docenas de lenguajes de programación principales.

Documentación: https://docs.nats.io/

RabbitMQ

RabbitMQ es un software de negociación de mensajes de código abierto que funciona como un middleware de mensajería. Implementa el estándar Advanced Message Queuing Protocol.

Documentación: https://www.rabbitmq.com/documentation.html

¿Qué es Pub/Sub?

Pub/Sub es un servicio de mensajería asíncrona que separa los servicios que producen eventos de servicios que procesan eventos. Puedes usar Pub/Sub, como middleware orientado a la mensajería o transferencia y entrega de eventos para las canalizaciones de estadísticas de transmisión.

Documentación: https://cloud.google.com/pubsub/docs/overview

Docker

Docker build

Necesitamos realizar un build a nuestro Dockerfile, basicamente necesitamos crear una imagen.

Docker puede construir imágenes automáticamente, leyendo las instrucciones indicadas en un fichero Dockerfile. Se trata de un documento de texto que contiene todas las órdenes a las que un usuario dado puede llamar, desde la línea de comandos, para crear una imagen.

Referencia docker build: https://docs.docker.com/engine/reference/builder/

Los pasos principales para crear una imagen a partir de un fichero Dockerfile son:

1. Crear un nuevo directorio que contenga el fichero, con el guión y otros ficheros que fuesen necesarios para crear la imagen.
2. Crear el contenido.
3. Construir la imagen mediante el comando docker build.

```
# Sintaxis básica de docker build

$ docker build [opciones] RUTA | URL | -

# En nuestro caso:
# Ir al directorio donde esta el Dockerfile
$ cd [PATH_AL_DOCKERFILE]

# Crear imagen utilizando build, notar el -t
# -t define un tag con el que nos referiremos a la imagen
$ docker build . -t node:14
```

Docker run

El comando docker run crea un contenedor desde una imagen, e inicia el contenedor.

```
# Sintaxis básica de docker run
$ docker run [OPTIONS] IMAGE [COMMAND] [ARG...]

# En nuestro caso:
# -dit dice a docker que haga detach (correr en segundo plano), y que el
contenedor podra ser interactivo con una consola de tipo tty.
# --name le asigna un nombre al container
# -v copia al volumen del contenedor /elements/procs el contenido de la carpeta
/procs en el sistema operativo
# -p dice que puerto exponer al mundo
# por ultimo ponemos el tag que le definimos a la imagen
$ docker run -dit --name node-api -v /proc/:/elements/procs/ -p 80:80 node:14
```

API Node

Luego de tener instalado el módulo kernel, podemos acceder a él en el archivo "/procs/[nombre_modulo]", en el caso específico del ejemplo de clase, en /procs/timestamps.

Este archivo puede ser leído por *cualquier librería de archivos*, esto quiere decir que no necesitamos de una arquitectura ni lenguaje específico, unicamente necesitamos estar en el grupo de permisos de lectura del archivo.

En este caso, utilizaremos NodeJS para realizar una muy pequeña API que únicamente nos devolverá el timestamp en formato HTML y como un string.

NodeJS

Node.js es un entorno de tiempo de ejecución de JavaScript (de ahí su terminación en .js). Este entorno de tiempo de ejecución en tiempo real incluye todo lo que se necesita para ejecutar un programa escrito en JavaScript.

Nodejs facilita la realización de APIs y de servidores web, debido a la gran versatilidad de JavaScript.

En este caso, realizaremos la API utilizando un framework llamado express.

Node.js utiliza un modelo de entrada y salida sin bloqueo controlado por eventos que lo hace ligero y eficiente (con entrada nos referimos a solicitudes y con salida a respuestas). Puede referirse a cualquier operación, desde leer o escribir archivos de cualquier tipo hasta hacer una solicitud HTTP.

La idea principal de Node.js es usar el modelo de entrada y salida sin bloqueo y controlado por eventos para seguir siendo liviano y eficiente frente a las aplicaciones en tiempo real de uso de datos que se ejecutan en los dispositivos. Es una plataforma que no dominará el mundo del desarrollo web pero si que satisface las necesidades de una gran mayoría de programadores.

La finalidad de Node.js no tiene su objetivo en operaciones intensivas del procesador, de hecho, usarlo para programación de más peso eliminará casi todas sus ventajas. Donde Node.js realmente brilla es en la creación de aplicaciones de red rápidas, ya que es capaz de manejar una gran cantidad de conexiones simultáneas con un alto nivel de rendimiento, lo que equivale a una alta escalabilidad.

Express

Express.js es un framework para Node.js que sirve para ayudarnos a crear servicios web, e incluso aplicaciones web. Provee de todas las herramientas necesarias para cumplir con esta misión (batteries-included), como routers, manejo de sesiones, cookies; y también mucha adaptabilidad utilizando middlewares.

En el caso de este ejemplo, lo utilizaremos únicamente para realizar una API que nos retornará el timestamp y posteriormente lo guardará.

https://expressjs.com/

MongoDB

MongoDB es una base de datos NoSQL que utiliza documentos para guardar la información, ofrece muy buena escalabilidad y flexibilidad. En este caso, utilizaremos un servicio SaaS llamado Mongo Atlas, que será utilizado para tener mongodb en la nube.

Página de MongoAtlas: https://cloud.mongodb.com/

Se creará un cluster en mongoatlas para su utilización de forma fácil.

Mongoose

Mongoose es una librería para Node. js que nos permite escribir consultas para una base de datos de MongooDB, con características como validaciones, construcción de queries, middlewares, conversión de tipos y algunas otras, que enriquecen la funcionalidad de la base de datos.

https://mongoosejs.com/