UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE CIENCIAS Y SISTEMAS

SISTEMAS OPERATIVOS 1 SECCIÓN N

ING. JESUS GUZMAN POLANCO

AUX. ALVARO NORBERTO GARCÍA

AUX. SERGIO ALFONSO FERRER GARCÍA

SEGUNDO SEMESTRE 2024

# PROYECTO 1

# Gestor de contenedores

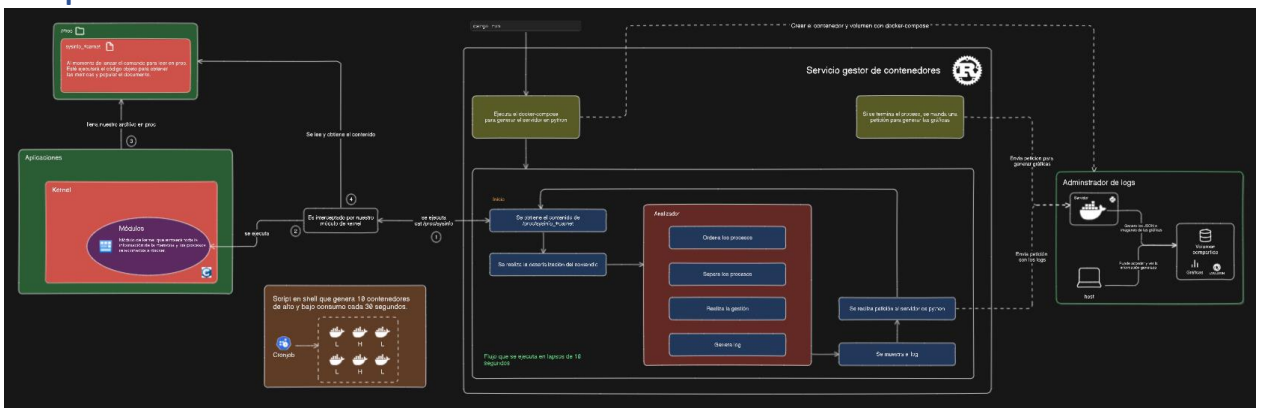
## Objetivos

- Conocer el Kernel de Linux mediante módulos de C.
- Hacer uso del lenguaje de programación Rust para la gestión del sistema.
- Comprender el funcionamiento de los contenedores usando Docker.
- Comprender el funcionamiento de los scripts de bash para la automatización de procesos.

## Introducción

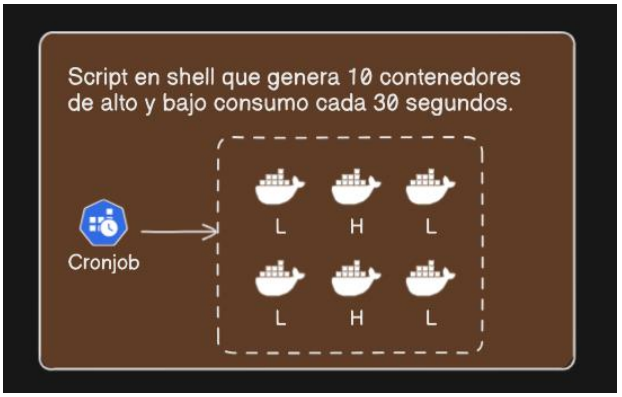
El objetivo de este proyecto es aplicar todos los conocimientos adquiridos en la unidad 1, con la implementación de un gestor de contenedores mediante el uso de scripts, módulos de kernel, lenguajes de programación y la herramienta para la creación y manejo de contenedores más popular, Docker. Con la ayuda de este gestor de contenedores se podrá observar de manera más detallada los recursos y la representación de los contenedores a nivel de procesos de Linux y como de manera flexible pueden ser creados, destruidos y conectados por otros servicios.

# Arquitectura




Enlace para acceder al recurso: https://app.eraser.io/workspace/xoEcZUDSyrCmcsEJidLd?origin=share

## Script creador de contenedores




Aplicando los conocimientos adquiridos sobre la creación de imágenes y contenedores de Docker mediante el uso de la CLI y la creación de scripts de bash. Se tendrá un cronjob el cual se estará ejecutando en lapsos de 30 segundos con la siguiente funcionalidad:

1. De manera aleatoria deberá generar 10 contenedores de las imágenes explicadas en la sección de notas.

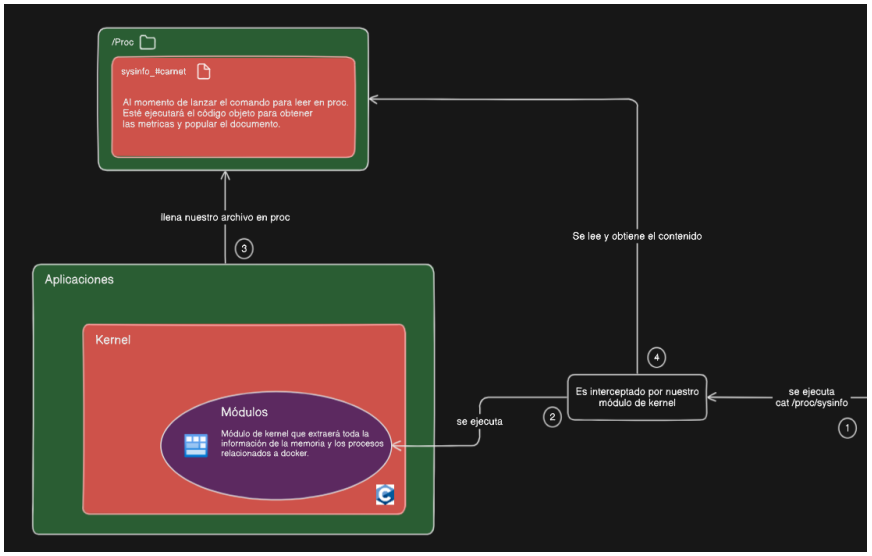
**Notas:**

- Usted como estudiante deberá generar 4 imágenes de docker separadas en dos grupos:
    - Contenedores de alto consumo
        - Consumo de ram
        - Consumo de cpu
    - Contenedores de bajo consumo
- Las tecnologías o métodos por utilizar para la creación de estas imágenes quedan a discreción del estudiante, pero es de suma importancia que se pueda diferenciar.

**Sugerencias:**

- Utilizar el módulo especial de Unix **/dev/urandom** para el nombre de los contenedores.

## Módulo de Kernel




Con los conocimientos adquiridos en la creación de módulos del kernel. Deberá crear un módulo el cual capturará las métricas de necesarias que el servicio necesitará para el análisis de la memoria y contenedores. A Continuación, se detallará la información que debe ser capturada y guardada en la carpeta **/proc.**

1. Capturar en MB o KBs:
    a. Total, de memoria RAM
    b. Memoria RAM libre
    c. Memoria RAM en uso
2. Todos los procesos relacionados a los contenedores generados por el script
    a. PID
    b. Nombre
    c. Línea de comando que se ejecutó o id del contenedor
    d. Vsz (Tamaño de la memoria virtual) en KBs
    e. Rss (Tamaño de memoria física) en KBs
    f. Porcentaje de memoria utilizada
    g. Porcentaje de CPU utilizado

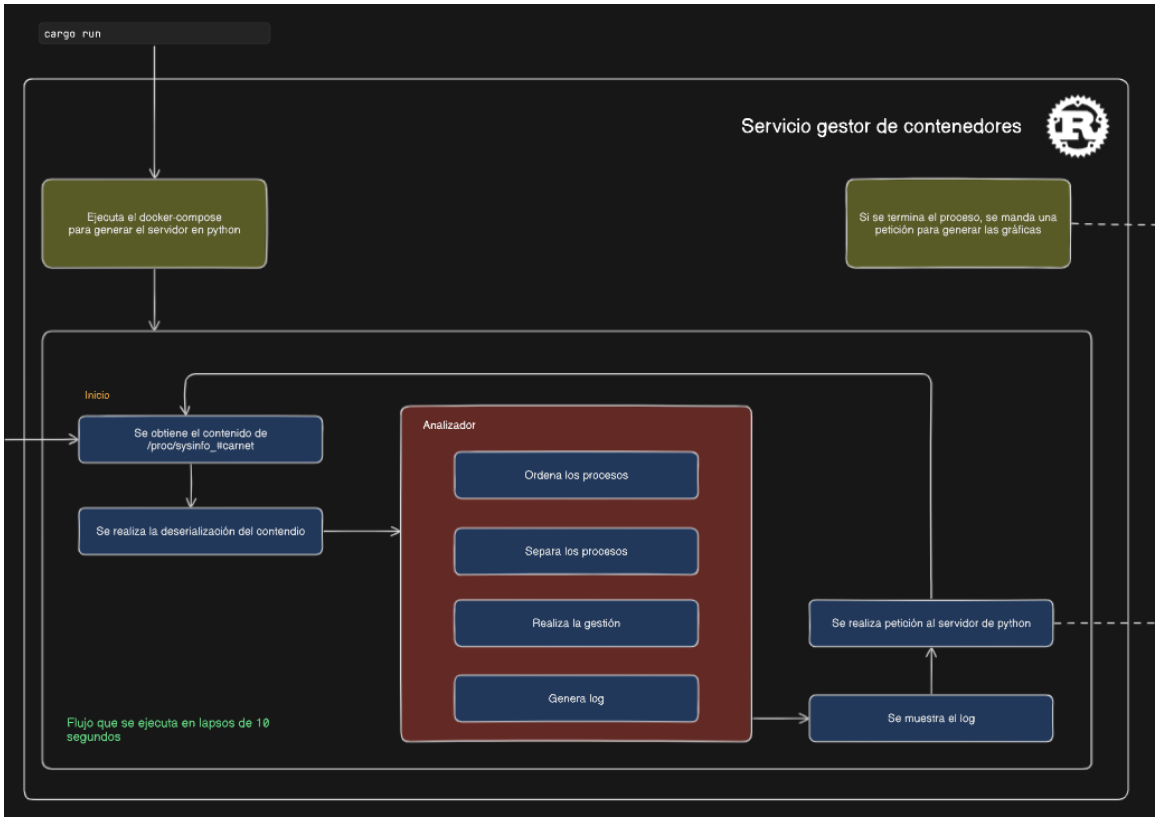
**Notas:**

1. El nombre del archivo que se generará en la carpeta /proc debe de llevar el siguiente nombre **<sysinfo_#carnet>.**
2. Manejar una estructura JSON.

**Sugerencias:**

- Utilizar la estructura task_struct para poder filtrar de manera correcta únicamente los procesos relacionados a los contendores y extraer la información necesaria.

## Servicio en Rust




Este servicio es el corazón del proyecto en el cual se llevará la comunicación con diferentes partes y funcionalidades. Por el manejo seguro de memoria y características únicas, se le has solicitado que construya el gestor de contenedores en el lenguaje de programación Rust el cual estará encargado del análisis, ejecución y comunicación de diferentes flujos durante toda la ejecución del servicio. A continuación, se describe el comportamiento de este.

1. Al iniciar el servicio, este deberá crear un contenedor el cual estará recibiendo todas las peticiones http que realice el servicio durante su tiempo de vida para llevar un registro.
2. Después de haber creado el contenedor administrador de registros, se estará ejecutando dentro de un loop infinito que será finalizando al emitir una señal, como por ejemplo Ctrl + C, una serie de procedimientos y análisis cada 10 segundos, estos son:
    a. Lectura del archivo en **/proc/<sysinfo_#carnet>**
    b. Deserialización del contenido
    c. El análisis respectivo para la gestión de los contenedores
    d. Generación de logs
    e. Peticiones http al contenedor encargado de administrar los logs
3. Al finalizar el servicio se deberá lanzar una última petición http al contenedor que administra los logs para la generación de las gráficas y eliminar el cronjob.

**Restricciones**

- Debe siempre haber 3 contenedores de bajo consumo y estos deben ser los últimos 3
- Debe siempre haber 2 contenedores de alto consumo y estos deben ser los primeros 2
- No eliminar el contenedor que gestiona los logs

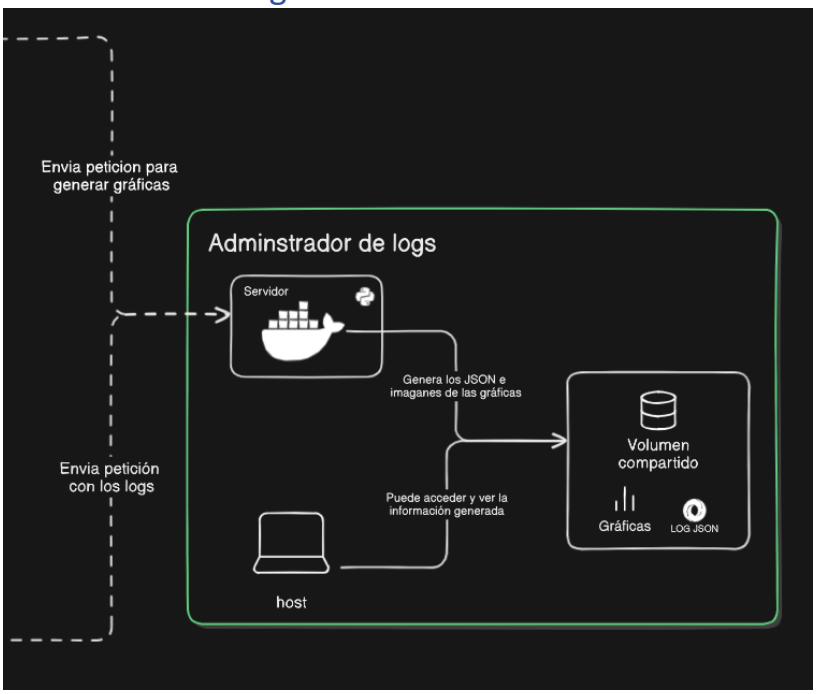
**Notas**

- Guardar el id del contendor encargado de las administraciones de los logs para no provocar su eliminación.
- Dentro del análisis de la gestión de contenedores debe aplicar una serie de ordenamientos, esto debido a que habrá contenedores con características similares pero que en alguno de estos 4 atributos existe una variación.
    - Ordenar por uso de cpu
    - Ordenar por uso de ram
    - Ordenar por vsz
    - Ordenar por rss
- El servicio **deberá ser capaz de decidir** que contenedores **eliminar** y cuales **mantener** en ejecución utilizando como referencia las restricciones.
- Al momento de finalizar el servicio, se recomienda eliminar el cronjob para evitar la creación de contenedores desmedida.
- Deberá imprimir en consola de manera ordenada y estilizada:
    - La información de la memoria
    - Los contenedores de alto bajo consumo
    - Los contenedores de bajo consumo
    - Los contenedores eliminados
- Existen dos tipos de logs en cuales además de agregar la respectiva información, debe de registrare la fecha y hora.
    - Los logs de la memoria
    - Los logs de los procesos

**Sugerencias:**

- Utilizar listas (Vectores)
- Utilizar una terminal en modo superusuario
- Utilizar Structs
- Se recomienda, pero no es obligatorio el uso de threads (hilos) para agilizar la eliminación de contenedores.

## Contenedor administrador de logs




Con los conocimientos adquiridos sobre Docker, se le solicita crear un contenedor que actúe como servidor en el lenguaje de programación Python y que tenga como finalidad, recibir las peticiones generadas por el servicio de Rust y almacenar en formato JSON todos los registros generados y que estos puedan ser accesibles tanto por la máquina host como por el contenedor con el uso de volúmenes. Adicional a esto, existiría una funcionalidad que será llamada por servicio de Rust al finalizar y su objeto será realizar gráficas con todos los logs almacenados.

**Notas:**

- Deberá poder guardar y agregar nuevos registros generados por el servicio de rust en un archivo JSON compartido
- Uso de volúmenes para permitir un **binding**
- Deberá generar 2 gráficas con la información de los logs respectivos, estas gráficas quedan a discreción del estudiante, pero es fundamental que sean representativas, demostrativas y lógicas.

**Sugerencias:**

- Utilizar matplotlib
- Utilizar fastapi

## Requisitos Mínimos

- Documentación de modo de uso, instalación y una breve explicación con ejemplos.
- Scripts para la creación de contenedores
- Módulo de kernel
- Servicio de Rust (hasta la eliminación de contenedores)

## Restricciones

- El proyecto se realizará de forma individual.
- La obtención de la información se hará a través de los módulos de Kernel en C.
- El servicio se realizará con Rust.
- Trabajar en el sistema operativo Linux (Físico o virtualizado).

## Github

- El código fuente debe de ser gestionado por un repositorio privado de Github
- Crea una Carpeta en el repositorio llamado: Proyecto1
- Agregar a los auxiliares al repositorio de GitHub: **AlvaroG13191704** y **SergioFerrer9**

## Calificación

- Al momento de la calificación se verificará la última versión publicada en el repositorio de GitHub
- Cualquier copia parcial o total tendrán nota de 0 puntos y serán reportadas al catedrático y a la Escuela de Ciencias y Sistemas.
- Si el proyecto se envía después de la fecha límite, se aplicará una penalización del 25% en la puntuación asignada cada 12 horas después de la fecha límite. Esto significa que, por cada período de 12 horas de retraso, se reducirá un 25% de los puntos totales posibles. La penalización continuará acumulándose hasta que el trabajo alcance un retraso de 48 horas (2 días). Después de este punto, si el trabajo se envía, la puntuación asignada será de 0 puntos.

## Entregables

- La entrega se debe realizar antes de las 23:59 del 08 de septiembre de 2024.
- La forma de entrega es mediante UEDI subiendo el enlace del repositorio.
- Manual técnico en formato Mark Down o PDF.

## Referencias

- https://github.com/AlvaroG13191704/SO1_S2_2024