UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
SEGUNDO SEMESTRE 2020
ING. SERGIO MÉNDEZ
ING. ALLAN MORATAYA
AUX. DAVID ALVARADO
AUX. SEBASTIÁN SÁNCHEZ

PROYECTO 1

OBJETIVOS:

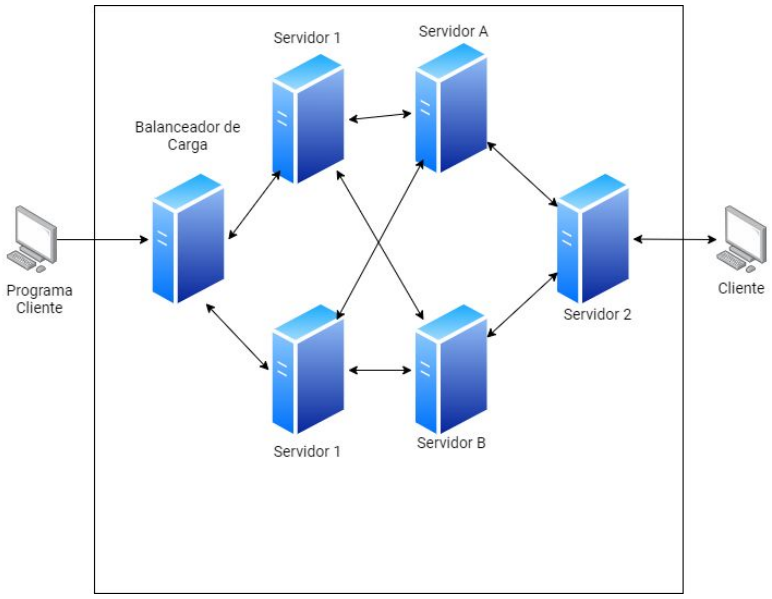
- Poner en práctica los conocimientos adquiridos en el curso sobre la estructura del sistema operativo, virtualización y cloud computing.

DESCRIPCIÓN:

El Cloud Computing es la entrega de servicios (como servidores, almacenamiento, bases de datos, software, etc.) a través de Internet. Desde que se estableció como una tecnología habitual al inicio de la década representa un gran cambio a la forma tradicional en que las empresas piensan sobre los recursos de computación.

Tomando en cuenta todas las ventajas de la computación en la nube y las herramientas vistas en el curso, deben realizar una aplicación que monitorizará los recursos y mostrará los elementos almacenados en la base de datos de dos servidores.

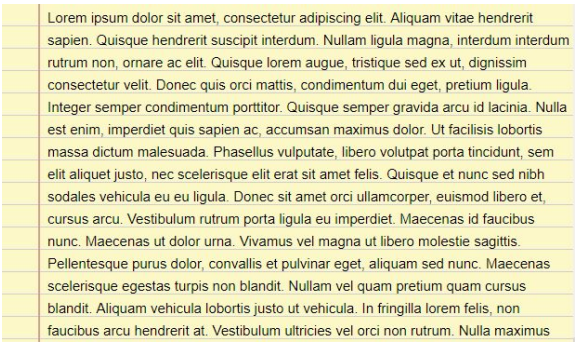
ARQUITECTURA:



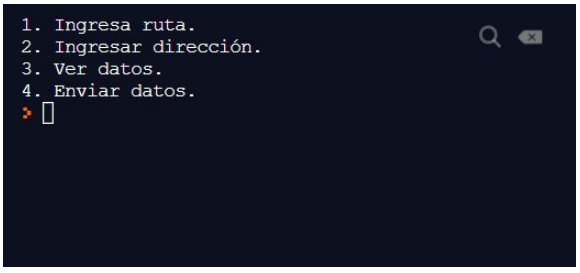
PROGRAMA CLIENTE:

Se deberá crear un programa escrito en Python (no es necesaria una interfaz gráfica). Este deberá leer el contenido de un archivo de texto, dividirlo en oraciones, luego creará un objeto por cada oración y le asociará un usuario (la asignación del usuario queda a criterio del estudiantes). Cada uno de estos elementos debe ser enviado al *Balanceador de carga* como una petición POST, debe hacerse una petición por cada objeto. El programa solicitará la ruta del archivo a dividir y la dirección a donde realizará las peticiones (la dirección del *Balanceador*), almacenará los objetos obtenidos del último análisis y podrá mostrarlos. Se sugiere el siguiente formato para los elementos a enviar:

```
{
      "autor": "juanito",
      "nota": "Buenos días, usuarios de internet!"
}
```



Ejemplo del archivo de texto.



Ejemplo del programa ejecutándose.

BALANCEADOR DE CARGA:

Se debe implementar el balanceador de carga nativo de Google Cloud o AWS para distribuir la carga del Servidor 1. Al ser dos servidores, el balanceador debe apuntar a las direcciones de los dos Servidores 1; para probar esto se debe soportar la caída de uno de los dos Servidores 1: si uno de los servidores es apagado, el otro soportará la carga enviada.

SERVIDOR A Y B:

La configuración para ambos servidores es la misma, debe instalarse cualquier distribución de Linux y deben ser máquinas virtuales del proveedor de servicios de computación en la nube a su elección. Se solicita lo siguiente:

- **Módulos del Kernel**
  Deben programarse dos módulos, estos consultarán el porcentaje de

memoria principal (RAM) utilizado en el servidor y el porcentaje de uso de CPU.

- **Bases de Datos**
  Se debe implementar una base de datos NoSQL mediante un contenedor de Docker. El motor de base de datos a utilizar será MongoDB.

- **API REST en Python**
  Se debe programar una API REST en Python, la ejecución de esta se realizará dentro de un contenedor de Docker.

La API consultará la información de la base de datos y los archivos generados por los módulos cargados previamente, si otros servidores o servicios necesitan información sobre el rendimiento puede consultarlo a través de peticiones a esta API. Para iniciarse ambos contenedores debe utilizarse Docker Compose.

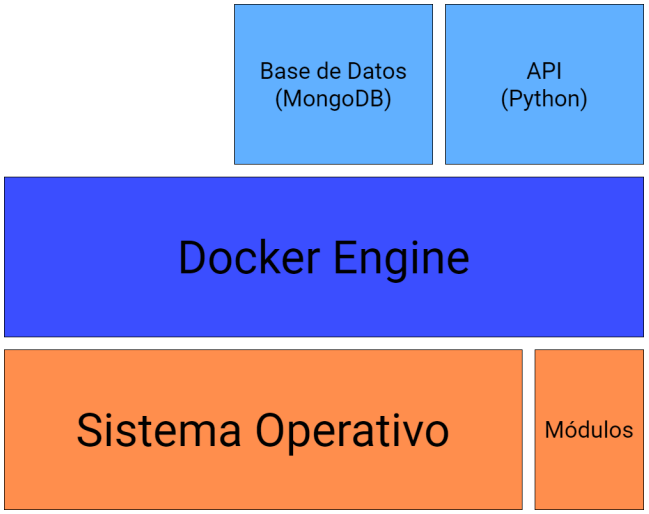


Diagrama de capas del Servidor A y B.

SERVIDOR 1:

En este servidor se ejecutará una API REST dentro de un contenedor de Docker. Su funcionamiento será similar al de un balanceador de carga. Cuando reciba la petición POST del balanceador de carga este debe elegir el servidor al que mandará el elemento, consultará el porcentaje de RAM, de CPU y la cantidad de elementos almacenados en A y B. Tomando en cuenta los siguientes criterios:

- Si la base de datos de A tiene más elementos que B, se insertará en B, en caso contrario se insertará en A.

- Si ambos tienen la misma cantidad de elementos almacenados, se comparará el porcentaje de RAM utilizada, el servidor con menor porcentaje será el escogido.
- Si ambos tienen el mismo porcentaje de RAM, debe realizarse la comparación con el porcentaje de CPU.
- Si el porcentaje de CPU es el mismo, debe enviarse al Servidor A.

SERVIDOR 2:

Este servidor se utilizará para publicar la aplicación web que mostrará la información de ambos servidores (los datos almacenados en la base de datos y los datos del sistema). Se deberá utilizar un contenedor de Docker para ejecutar el servidor web, queda a elección de los estudiantes cuál utilizar.

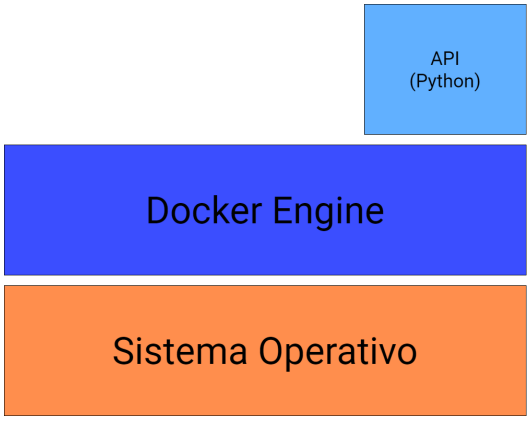


Diagrama de capas del Servidor 1.

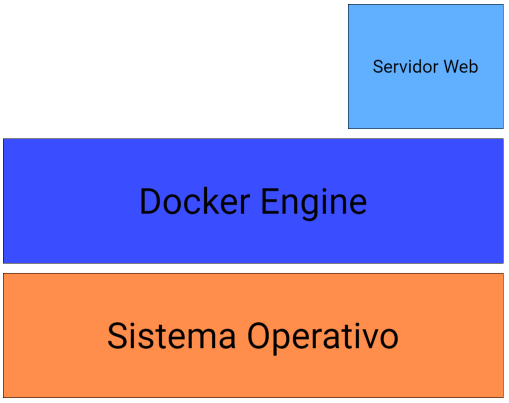
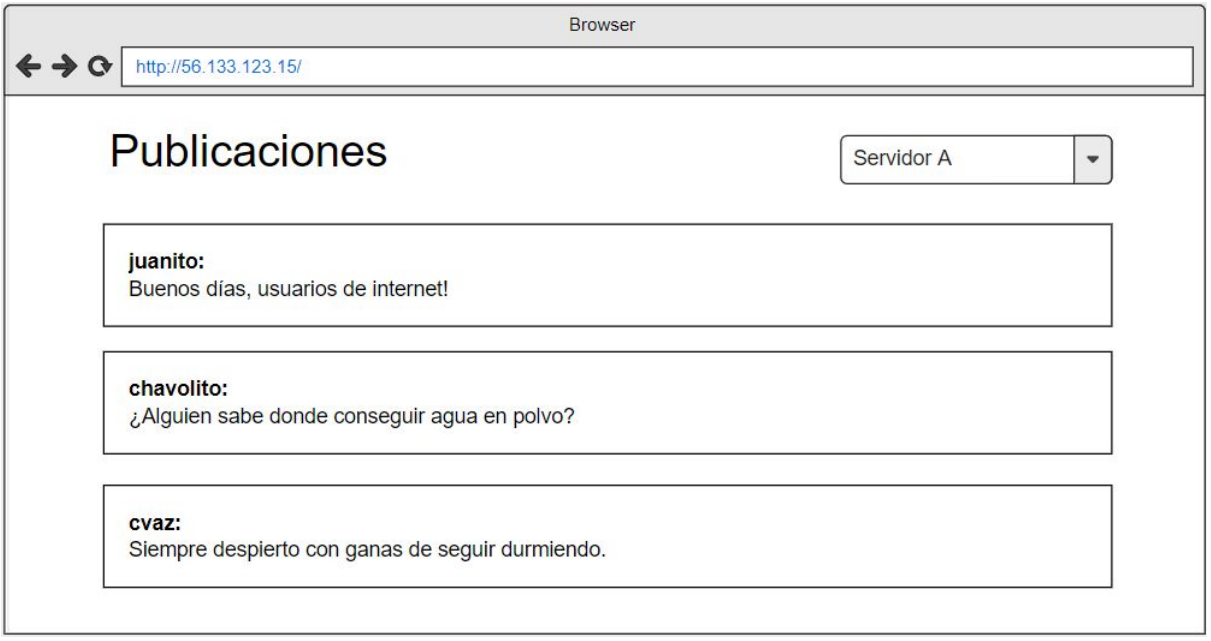


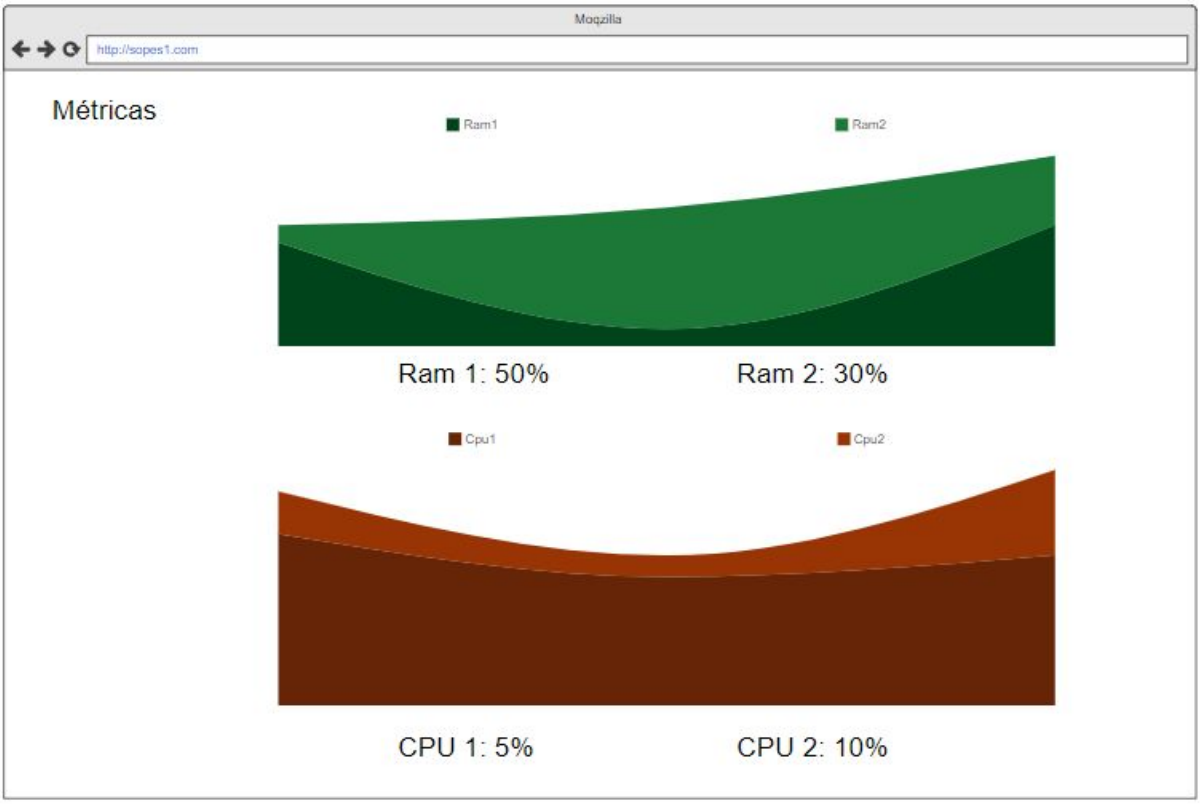
Diagrama de capas del Servidor 2.

APLICACIÓN WEB:

La aplicación tendrá dos secciones importantes. La primera sección debe mostrar todas los elementos almacenados en ambas bases de datos, estos elementos se actualizarán en tiempo real, con la opción de cambiar entre servidores.



La segunda sección debe mostrar el porcentaje de RAM y CPU utilizado en tiempo real (la gráfica y los datos deben actualizarse automáticamente cada 5s):



ENTREGABLES:

- Código de la aplicación cliente.
- Código de la API del Servidor 1 y el Dockerfile.
- Código de la API de los Servidores A y B, código de los módulos del kernel, Dockerfile de los contenedores y el archivo yml de Docker Compose.

- Código de la aplicación web y el Dockerfile del contenedor del Servidor 2.
- Manual técnico documentando todos los elementos del sistema.

Todo irá comprimido en un rar con nombre [SO1]Proyecto1_Carné.

CONSIDERACIONES:

- La aplicación cliente y las APIs deben estar escritas en Python.
- La aplicación web de S2 será implementada en React.
- Los cinco servidores deben ser máquinas virtuales en la nube.
- La nube a utilizar es Google Cloud para la sección A, AWS para la sección B.

FECHA DE ENTREGA:

08 de octubre de 2020.