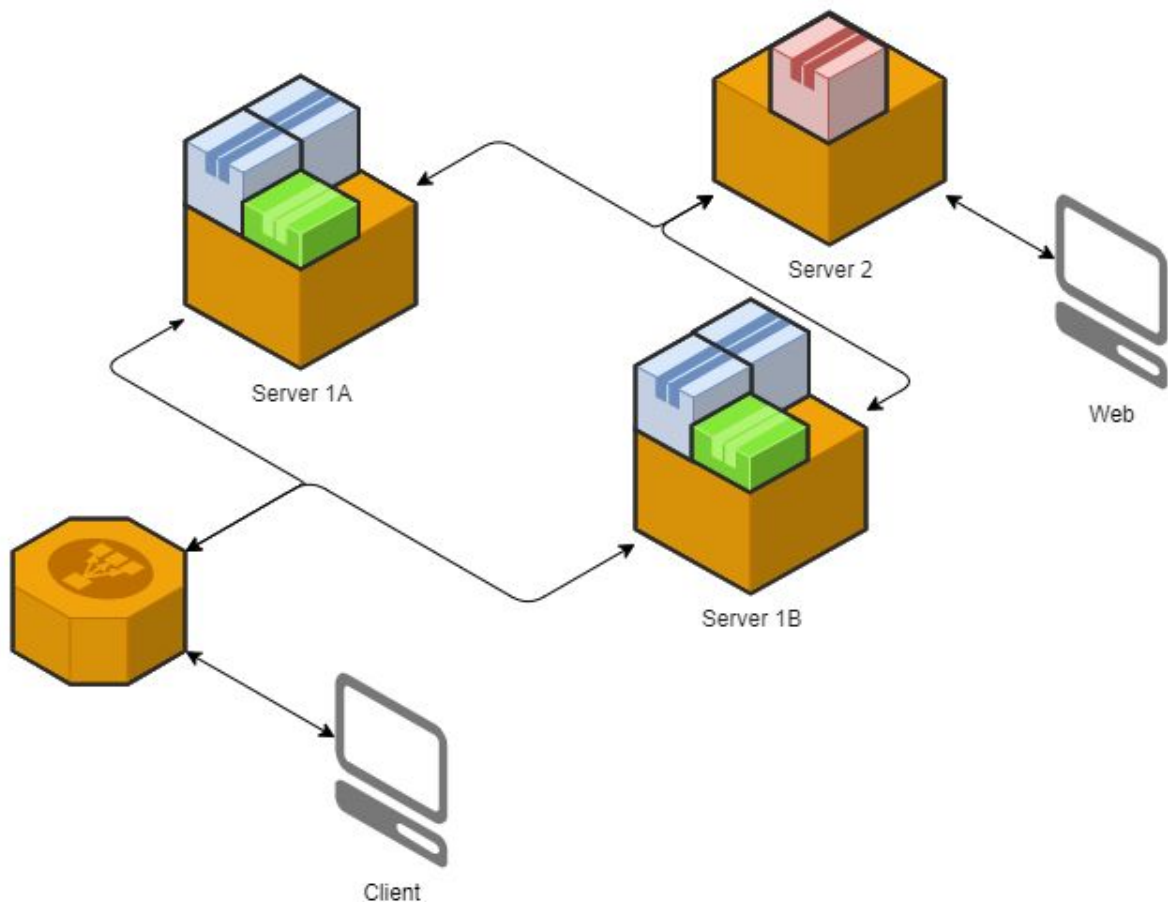
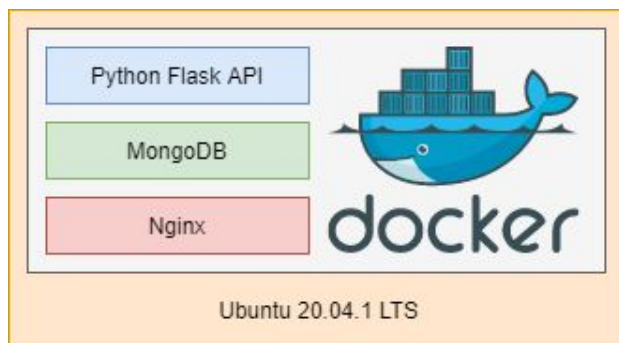




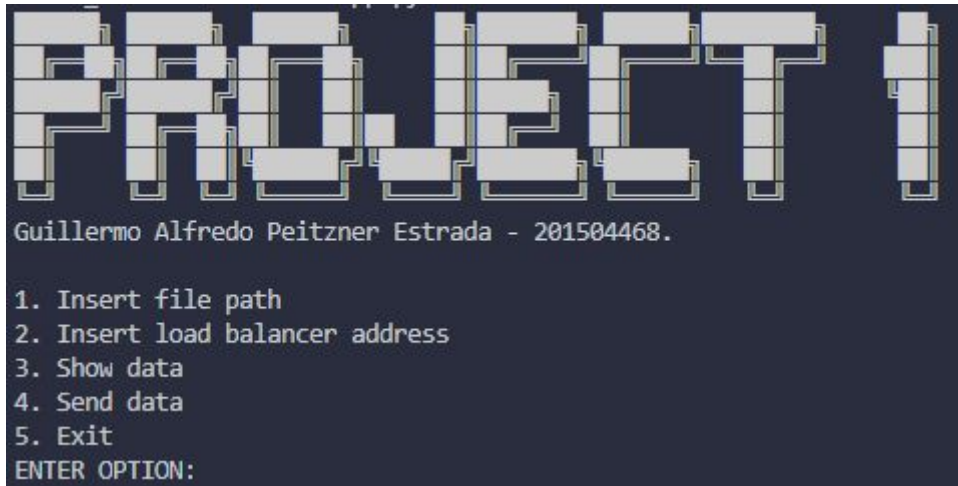
Manual Técnico

Arquitectura



Cliente

Su función es cargar archivos de texto plano, extraer las oraciones que se encuentren en el archivo, convertirlas en objetos de formato JSON con atributos de autor y oración para poder enviarlas a un balanceador por medio de peticiones POST del protocolo HTTP. El lenguaje utilizado es Python.



```
Guillermo Alfredo Peitzner Estrada - 201504468.

1. Insert file path
2. Insert load balancer address
3. Show data
4. Send data
5. Exit
ENTER OPTION:
```

Estructura del objeto JSON

```
[{
  "author": "User1",
  "sentence": "Massa id neque aliquam vestibulum morbi"
}]
```

Código fuente

Imports

Código	Descripción
<pre>import requests</pre>	Librería utilizada para realizar peticiones HTTP.
<pre>import random</pre>	Librería utilizada para generar números aleatorios.

Variables globales

Código	Descripción
<pre>data = []</pre>	Almacena los objetos con el autor y oración.
<pre>load_balancer_address = "http://so1-1607494048.us-east-2.elb.amazonaws.com:4999/sentence"</pre>	Almacena la dirección del balanceador de carga.
<pre>users = ["User1", "User2", "User3", "User4", "User5"]</pre>	Almacena los autores que pueden estar presentes en los objetos que se van a enviar.

Métodos

Código
<pre>def send_data(): print("") print(load_balancer_address) if data: if load_balancer_address: for sentence in data: try: r = requests.post(load_balancer_address, json=sentence) print("RESULT: ", r.json()) except: print("ERROR: sending sentence ", sentence) else: print("ERROR: load balancer not defined") else: print("ERROR: data not loaded")</pre>
Descripción
Envía los objetos almacenados en "data" al balanceador de carga "load_balancer_address" y muestra los resultados de cada petición.

Código

```
def show_data():
    print("")
    if data:
        for sentence in data:
            print(sentence)
    else:
        print("ERROR: data not loaded")
```

Descripción

Muestra los objetos almacenados en “data”.

Código

```
def insert_load_balancer_address():
    print("")
    print("ENTER LOAD BALANCER ADDRESS:")
    global load_balancer_address
    load_balancer_address = input()
```

Descripción

Guarda la dirección ingresada del balanceador de carga en “load_balancer_address”.

Código

```
def insert_file_path():
    print("")
    print("ENTER FILE PATH:")
    file_path = input()
    try:
        f = open(str(file_path), "r")
        file_data = f.read()
        file_data = file_data.split(". ")
        data.clear()
        for sentence in file_data:
            data.append(
                {"author": users[random.randint(0, 4)], "sentence":
sentence})
    except:
        data.clear()
        print("ERROR: invalid file path")
```

Descripción

Carga el archivo indicado en la dirección ingresada, extrae las oraciones y las convierte en objetos con atributos de autor y oración. Cada autor es seleccionado de manera aleatoria.

Código

[illegible]

Descripción

Muestra el encabezado de la aplicación.

Código

```
def menu():
    header()
    while(True):
        print("")
        print("1. Insert file path")
        print("2. Insert load balancer address")
        print("3. Show data")
        print("4. Send data")
        print("5. Exit")
        print("ENTER OPTION:")
        option = input()
        if option == "1":
            insert_file_path()
        elif option == "2":
            insert_load_balancer_address()
```

```
elif option == "3":
    show_data()
elif option == "4":
    send_data()
elif option == "5":
    break
else:
    print("ERROR: invalid option")
```

Descripción

Menú principal de la aplicación.

Código

```
if __name__ == "__main__":
    menu()
```

Descripción

Inicio de la aplicación.

Server 1

Este servidor ejecuta una API REST dentro de un contenedor de Docker. Su funcionamiento es como un balanceador de carga, cuando recibe una petición POST con un objeto con atributos de autor y oración, este elige al servidor que mandara el elemento dependiendo el porcentaje de RAM, CPU y cantidad de elementos tomando en cuenta los siguientes criterios:

- Si la base de datos A contiene más elementos que B, se inserta en B, caso contrario se inserta en A.
- Si ambos tienen la misma cantidad de elementos se compara el porcentaje de RAM y el servidor con menor porcentaje es elegido.
- Si ambos tienen el mismo porcentaje de RAM se compara el porcentaje de CPU y servidor con menor porcentaje es elegido.
- Si el porcentaje de RAM es el mismo, se envía al servidor A.

Endpoints

Dirección [TIPO]	JSON
/sentence [POST]	<pre>{ "author": "User1", "sentence": "Massa id neque aliquam vestibulum" }</pre>
Respuesta	<pre>{ "author": "User1", "date": "2020-10-08 16:51:30.506529", "sentence": "Massa id neque aliquam vestibulum" }</pre>

Imports

Código	Descripción
<pre>import os</pre>	Librería para acceder a propiedades del sistema.
<pre>import requests</pre>	Librería utilizada para realizar peticiones HTTP.
<pre>from flask import Flask, jsonify, request</pre>	Librería para realizar una API REST.
<pre>from flask_cors import CORS</pre>	Librería para configurar los CORS de la API REST.

Variables globales

Código	Descripción
<pre>app = Flask(__name__)</pre>	Inicia la API REST.
<pre>CORS(app)</pre>	Configura los CORS de la API REST.
<pre>serverA = "http://18.222.150.51:5000/"</pre>	Dirección del servidor A.
<pre>serverB = "http://3.14.28.178:5000/"</pre>	Dirección del servidor B.

Métodos

Código
<pre>@app.route("/sentence", methods=["POST"]) def add_sentence(): sentence = { "author": request.json["author"], "sentence": request.json["sentence"] } try: r_server1 = requests.get(serverA+"usage") except: try: r = requests.post(serverB+"sentence", json=sentence) return jsonify({"server": "B", "filter": "down", "data": r.json()}) except: return jsonify({"message": "down"}) try: r_server2 = requests.get(serverB+"usage") except: try: r = requests.post(serverA+"sentence", json=sentence) return jsonify({"server": "A", "filter": "down", "data": r.json()}) except: return jsonify({"message": "down"}) r_dictionary_server1 = r_server1.json() r_dictionary_server2 = r_server2.json() if r_dictionary_server1["database"] > r_dictionary_server2["database"]:</pre>


```

        r = requests.post(serverB+"sentence", json=sentence)
        return jsonify({"server": "B", "filter": "database", "data":
r.json()})

        elif      r_dictionary_server1["database"]      <
r_dictionary_server2["database"]:
        r = requests.post(serverA+"sentence", json=sentence)
        return jsonify({"server": "A", "filter": "database", "data":
r.json()})
    else:
        if r_dictionary_server1["ram"] > r_dictionary_server2["ram"]:
            r = requests.post(serverB+"sentence", json=sentence)
            return jsonify({"server": "B", "filter": "ram", "data":
r.json()})

            elif      r_dictionary_server1["ram"]      <
r_dictionary_server2["ram"]:
            r = requests.post(serverA+"sentence", json=sentence)
            return jsonify({"server": "A", "filter": "ram", "data":
r.json()})
        else:
            if      r_dictionary_server1["cpu"]      >
r_dictionary_server2["cpu"]:
                r = requests.post(serverB+"sentence", json=sentence)
                return jsonify({"server": "B", "filter": "cpu",
"data": r.json()})

                elif      r_dictionary_server1["cpu"]      >
r_dictionary_server2["cpu"]:
                r = requests.post(serverA+"sentence", json=sentence)
                return jsonify({"server": "A", "filter": "cpu",
"data": r.json()})
            else:
                r = requests.post(serverA+"sentence", json=sentence)
                return jsonify({"server": "A", "filter": "A", "data":
r.json()})

```

Descripción

Código utilizado para realizar el balanceo de carga en los dos servidores, A y B.

Dockerfile

Código	Descripción
<pre>FROM python:3.7-alpine WORKDIR /code ENV FLASK_APP=app.py ENV FLASK_RUN_HOST=0.0.0.0 ENV FLASK_RUN_PORT=4999 RUN apk add --no-cache gcc musl-dev linux-headers COPY requirements.txt requirements.txt RUN pip install -r requirements.txt EXPOSE 4999 COPY .. CMD ["flask", "run"]</pre>	<p>Configuración para la creación de la imagen de docker. Utiliza como base una imagen con Python 3.7, establece el nombre del directorio de trabajo de la imagen. Establece las variables de entorno de la aplicación e instala las dependencias necesarias definidas en el archivo "requirements.txt". Expone el puerto 4999 y copia todo el contenido del directorio actual al directorio de trabajo, para finalizar inicia la aplicación.</p>

Docker Compose

Código	Descripción
<pre>version: "3.8" services: web: build: . ports: - "4999:4999" volumes: - ./code environment: FLASK_ENV: development</pre>	<p>Configuración para la creación del contenedor. Agrega todo el contenido del directorio actual, crea un enlace del puerto 4999 entre el servidor host y el contenedor, define el directorio actual como volumen principal y establece el contenedor en modo de desarrollo.</p>

Server A y B

Estos servidores tienen los siguientes módulos:

- Módulos del kernel: se tiene un módulo programado para mostrar el porcentaje de consumo de RAM y CPU del servidor.
- Bases de datos: se tiene una base de datos no relacional (MongoDB) para almacenar los objetos con atributos de usuario y oración.
- API REST: utilizada para ingresar los objetos a la base de datos.

Endpoints

Dirección [TIPO]	JSON
/usage [GET]	Vacío
Respuesta	<pre>{ "cpu": "0", "database": 36, "ram": "61" }</pre>

Dirección [TIPO]	JSON
/sentence [POST]	<pre>{ "author": "User1", "sentence": "Massa id neque aliquam vestibulum" }</pre>
Respuesta	<pre>{ "author": "User1", "date": "2020-10-08 21:58:29.479944", "sentence": "Massa id neque aliquam vestibulum" }</pre>

Dirección [TIPO]	JSON
/sentence [GET]	Vacío
Respuesta	<pre>[{ "author": "User5", "date": "2020-10-08 00:03:51.290330", "sentence": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua" }]</pre>

	<pre> }, { "author": "User1", "date": "2020-10-08 00:03:51.690781", "sentence": "Faucibus scelerisque eleifend donec pretium vulputate" }] </pre>
--	--

Código fuente

Imports

Código	Descripción
<pre>import os</pre>	Librería utilizada para utilizar funciones del sistema.
<pre>from flask import Flask, jsonify, request</pre>	Librería utilizada para crear la API REST
<pre>from flask_cors import CORS</pre>	Librería utilizada para la configuración de CORS de la API REST.
<pre>from pymongo import MongoClient</pre>	Librería utilizada para la comunicación con MongoDB.
<pre>from datetime import datetime</pre>	Librería utilizada para obtener el tiempo actual.

Variables globales

Código	Descripción
<pre>app = Flask(__name__)</pre>	Variable utilizada para inicializar la API REST.
<pre>CORS(app)</pre>	Configurando CORS de la API REST.
<pre>client = MongoClient('db', 27017)</pre>	Conexión con la base de datos en MongoDB.
<pre>db = client.tododb</pre>	Creación de la base de datos.

Métodos

Código

```
@app.route("/usage")
def get_usage():
    f = open("/procs/so_info", "r")
    response = str(f.read()).split(";")
    f.close()
    return jsonify({
        "ram": response[0],
        "cpu": response[1],
        "database": db.tododb.count()
    })
```

Descripción

Método utilizado para retornar las métricas actuales del servidor.

Código

```
@app.route("/sentence", methods=["POST"])
def add_sentence():
    sentence = {
        "author": request.json["author"],
        "sentence": request.json["sentence"],
        "date": str(datetime.now())
    }
    sentence_id = db.tododb.insert_one(sentence)
    new_sentence = db.tododb.find_one({"_id": sentence_id.inserted_id})
    result = {
        "author": new_sentence["author"],
        "sentence": new_sentence["sentence"],
        "date": new_sentence["date"]
    }
    return jsonify(result)
```

Descripción

Método utilizado para realizar la inserción de un registro a la base de datos.

Código
<pre>@app.route("/sentence", methods=["GET"]) def get_sentences(): sentences = [] for sentence in db.todosdb.find(): sentences.append({ "author": sentence["author"], "sentence": sentence["sentence"], "date": sentence["date"] }) return jsonify(sentences)</pre>
Descripción
Método utilizado para retornar todos los registros actuales en la base de datos.

Dockerfile

Código	Descripción
<pre>FROM python:3.7-alpine WORKDIR /code ENV FLASK_APP=app.py ENV FLASK_RUN_HOST=0.0.0.0 RUN apk add --no-cache gcc musl-dev linux-headers COPY requirements.txt requirements.txt RUN pip install -r requirements.txt EXPOSE 5000 COPY . . RUN mkdir -p /procs/ CMD ["flask", "run"]</pre>	<p>Configuración para la creación de la imagen de docker. Utiliza como base una imagen con Python 3.7, establece el nombre del directorio de trabajo de la imagen. Establece las variables de entorno de la aplicación e instala las dependencias necesarias definidas en el archivo "requirements.txt". Expone el puerto 4999 y copia todo el contenido del directorio actual al directorio de trabajo, crea una carpeta llamada "procs" y para finalizar inicia la aplicación.</p>

Docker Compose

Código	Descripción
<pre>version: "3.8" services: web: build: . ports: - "5000:5000" volumes: - ./code - /proc:/procs/ links: - db environment: FLASK_ENV: development db: image: mongo:latest</pre>	Configuración para la creación del contenedor. Agrega todo el contenido del directorio actual, crea un enlace del puerto 4999 entre el servidor host y el contenedor, define el directorio actual como volumen principal, comparte la carpeta /proc/ del host a la carpeta /procs/ del contenedor, crea un enlace entre el contenedor actual con el contenedor donde se encuentra la base de datos, establece el contenedor en modo de desarrollo y crea un contenedor con la última versión de MongoDB.

Módulo del Kernel

Código fuente

Variables
<pre>struct sysinfo info; long available, ram_usage; int i, j; u64 user, nice, system, idle, iowait, irq, softirq, steal, t_total, t_idle, t_usage, cpu_usage; u64 guest, guest_nice; u64 sum = 0; u64 sum_softirq = 0; unsigned int per_softirq_sums[NR_SOFTIRQS] = {0}; struct timespec64 boottime;</pre>
RAM
<pre>si_meminfo(&info); available = si_mem_available(); ram_usage = ((info.totalram - available - info.bufferram) * 100) / (info.totalram);</pre>

CPU

```
user = nice = system = idle = iowait =
    irq = softirq = steal = 0;
guest = guest_nice = 0;
getboottime64(&boottime);

for_each_possible_cpu(i)
{
    struct kernel_cpustat *kcs = &kcpustat_cpu(i);

    user += kcs->cpustat[CPUTIME_USER];
    nice += kcs->cpustat[CPUTIME_NICE];
    system += kcs->cpustat[CPUTIME_SYSTEM];
    idle += get_idle_time(kcs, i);
    iowait += get_iowait_time(kcs, i);
    irq += kcs->cpustat[CPUTIME_IRQ];
    softirq += kcs->cpustat[CPUTIME_SOFTIRQ];
    steal += kcs->cpustat[CPUTIME_STEAL];
    guest += kcs->cpustat[CPUTIME_GUEST];
    guest_nice += kcs->cpustat[CPUTIME_GUEST_NICE];
    sum += kstat_cpu_irqs_sum(i);

    for (j = 0; j < NR_SOFTIRQS; j++)
    {
        unsigned int softirq_stat = kstat_softirqs_cpu(j, i);

        per_softirq_sums[j] += softirq_stat;
        sum_softirq += softirq_stat;
    }
}

t_total = (user + nice + system + idle + iowait + irq + softirq +
steal) - t_total0;
t_total0 = user + nice + system + idle + iowait + irq + softirq +
steal;
t_idle = idle + iowait;
t_usage = (t_total - t_idle) - t_usage0;
t_usage0 = t_total - t_idle;
cpu_usage = ((t_usage * 100) / t_total) / 10000000000;
```

Escritura

```
seq_printf(m, "%ld;%lld", ram_usage, cpu_usage);
```

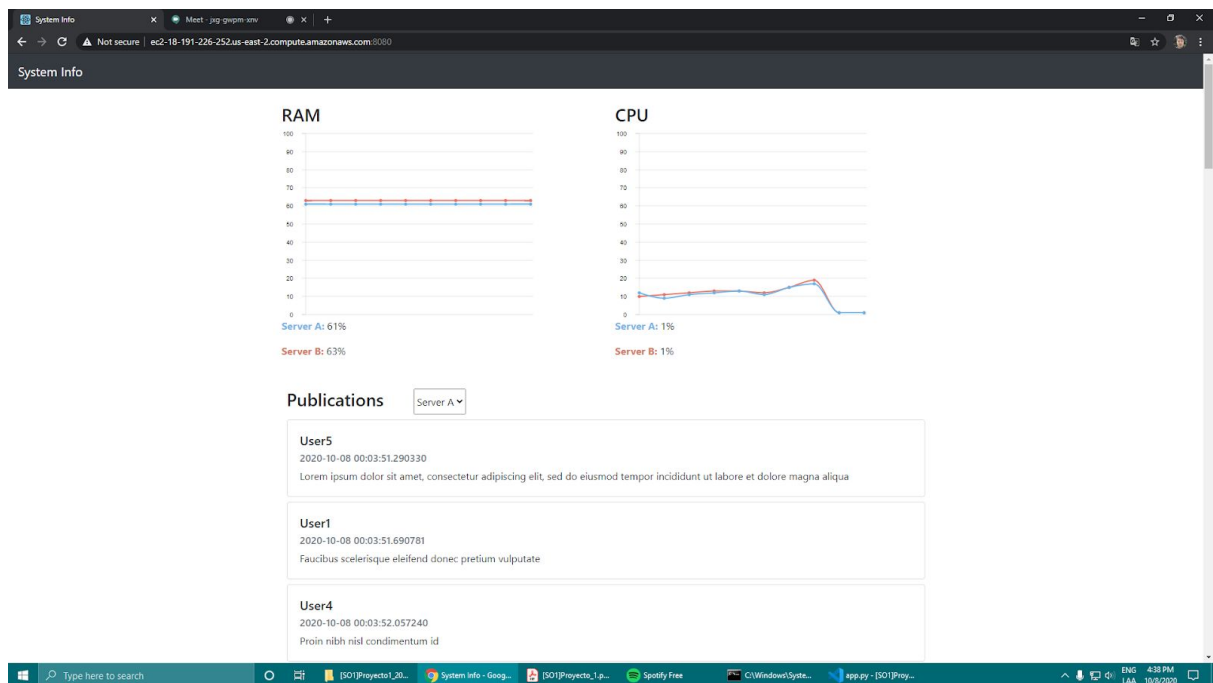

Makefile

```
obj-m += so_info.o
all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD)
modulesclean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Despliegue

Comando	Descripción
<code>sudo su</code>	Comando utilizado para entrar en modo superusuario.
<code>cd /proc</code>	Comando utilizado para entrar al directorio principal donde se encuentran todos los procesos.
<code>rmmod so_info</code>	Removiendo algún módulo que tenga el mismo nombre de nuestro nuevo módulo.
<code>cd /home/ubuntu/src/so_info/</code>	Entrando a la carpeta principal de nuestro módulo.
<code>rm so_info.c</code>	Removiendo archivos por si existen.
<code>nano so_info.c</code>	Creación del archivo de código fuente para el nuevo módulo.
<code>rm Makefile</code>	Removiendo archivos por si existen.
<code>nano Makefile</code>	Creación del archivo de código fuente para el nuevo módulo.
<code>make</code>	Compilando el código de nuestro módulo.
<code>insmod so_info.ko</code>	Insertando el nuevo módulo al sistema.

Server 2



Este servidor contiene una página web realizada con React JS, la aplicación web está desplegada en un servidor Nginx.

Dockerfile

Código	Descripción
<pre>FROM nginx:latest COPY . /usr/share/nginx/html/</pre>	Crea una imagen usando como base la última versión de Nginx.

Comandos

Código	Descripción
<pre>docker build -t webserver .</pre>	Comando utilizado para crear la imagen del servidor.
<pre>docker run -it --rm -d -p 8080:80 --name web webserver</pre>	Comando utilizado para crear un contenedor usando como base la imagen creada.

Repositorio

https://github.com/gpeitzner/SO1_Proyecto1

Balanceador de carga

<http://ec2-18-191-226-252.us-east-2.compute.amazonaws.com:8080/>