WUNIVERSIDAD DE SAN CARLOS DE GUATEMALA
FACULTAD DE INGENIERÍA
ESCUELA DE CIENCIAS Y SISTEMAS
SISTEMAS OPERATIVOS 1
SEGUNDO SEMESTRE 2020
ING. ALLAN MORATAYA
TUTOR ACADÉMICO: SEBASTIAN SANCHEZ

PROYECTO

MANUAL

MAYNOR DAVID SALGUERO GUILLÉN
CARNE: 201504192
GUATEMALA 10 DE OCTUBRE DEL 2020

Programa Cliente

El programa cliente consta de un pequeño programa en consola, en Python el cual tiene métodos para la lectura, impresión y envió de datos.

Variables globales:

```
datos = []
objetos = []
usuarios = ['usuario1', 'usuario2', 'usuario3', 'usuario4', 'usuario5', 'usuario6', 'usuario7', 'usuario8', 'usuario09', 'usuario10']
```

Métodos:

```
def leerArchivo():
    global datos, objetos
    print('Ingrese ruta del archivo: ')
    ruta = input()
    archivo = open(ruta, "r")
    datos = re.findall('[^.?!]+[.?!]+', archivo.read())
    archivo.close()
    for oracion in datos:
        objetos.append(json.dumps({"autor": random.choice(usuarios), "nota": oracion}))

def enviarDatos(direccion):
    for oracion in datos:
        data = {
            "database": "baseSopes",
            "collection": "oracion",
            "Document": {
            "autor": random.choice(usuarios),
            "nota": oracion
          }
        }
        print(data)
        requests.post(direccion, json = data)
```

```python
def menu():
    print('_______MENU_______ \n1. Leer archivo \n2. Enviar datos \n3. Salir \n')
    opcion = input()

    if opcion == '1':
        print('_______LEER ARCHIVO_______')
        leerArchivo()
        print('\n')
        menu()
    elif opcion == '2':
        print('_______ENVIAR DATOS_______\n')
        print('Ingrese direccion para realizar las peticiones:')
        direccion = input()
        enviarDatos(direccion)
        input()
        menu()
    elif opcion == '3':
        exit()
    else:
        print('Opcion invalida\n')
        input()
        menu()
```

Servidores A y B

Los servidores A y B son idénticos, ambos cuentan con los siguientes elementos:

RAM MODULE:

Donde la función se realiza en el my_proc_show

```c
static int my_proc_show(struct seq_file *m, void *v){
    struct sysinfo i;
    si_meminfo(&i);
    long ramLibre = i.freeram;
    long ramTotal = i.totalram;
    long porcentajeLibre = (ramLibre * 100) / ramTotal;
    seq_printf(m, "%ld,%ld,%ld", ramTotal, ramLibre, porcentajeLibre);
    return 0;
}
```

Utilizando la estructura de sysinfo i;

CPU MODULE:

Utiliza el contador interno de cpu del modulo stat para llevar el control

```c
static int show_stat(struct seq_file *p, void *v)
{
    int i, j;
    u64 user, nice, system, idle, iowait, irq, softirq, steal;
    u64 guest, guest_nice;
    u64 sum = 0;
    u64 sum_softirq = 0;
    unsigned int per_softirq_sums[NR_SOFTIRQS] = {0};
    struct timespec64 boottime;

    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);
        sum += arch_irq_stat_cpu(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;
        }
    }
    sum += arch_irq_stat();
```

Donde el calculo se realizo con la siguiente información en el api de Python:

APY EN PHYTON:

Conexión a la base de datos en mongo por el puerto 27017:

```python
class MongoAPI:
    def __init__(self, data):
        log.basicConfig(level=log.DEBUG, format='%(asctime)s %(levelname)s:\n%(message)s\n')
        #self.client = MongoClient("mongodb://localhost:27017/")  # When only Mongo DB is running on Docker.
        self.client = MongoClient("mongodb://mymongo_1:27017/")     # When both Mongo and This application is running on
                                                                    # Docker and we are using Docker Compose

        database = data['database']
        collection = data['collection']
        cursor = self.client[database]
        self.collection = cursor[collection]
        self.data = data

    def read(self):
        log.info('Reading All Data')
        documents = self.collection.find()
        output = [{item: data[item] for item in data if item != '_id'} for data in documents]
        return output

    def write(self, data):
        log.info('Writing Data')
        new_document = data['Document']
        response = self.collection.insert_one(new_document)
        output = {'Status': 'Successfully Inserted',
                  'Document_ID': str(response.inserted_id)}
        return output
```

Método Post para la RAM

```python
@app.route('/getRam')
def get_ram():
    file = open('/elements/procs/ram.txt','r')
    data = file.readline()
    file.close()
    datos = data.split(',')
    return Response(response=json.dumps({"ramLibre":datos[2]}),
                    status=200,
                    mimetype='application/json')
```

Método Post para el CPU

```python
@app.route('/getCPU')
def get_cpu():
    file = open('/elements/procs/stat.txt','r')
    data = file.readline()
    file.close()
    datos = data.split(' ')
    t_total = int(datos[2]) + int(datos[3]) + int(datos[4]) + int(datos[5]) + int(datos[6]) + int(datos[7]) + int(datos[8]) + int(datos[9])
    t_idle = int(datos[5]) + int(datos[6])
    t_usage = int(t_total) - int(t_idle)
    cpu_porcentaje = 100 - ((t_usage * 100) / t_total)
    return Response(response=json.dumps({"cpuLibre":cpu_porcentaje}),
                status=200,
                mimetype='application/json')
```

Método GET para la lectura de datos a la DB en mongo:

```python
@app.route('/mongodb', methods=['GET'])
def mongo_read():
    data = {
        "database": "baseSopes",
        "collection": "oracion"
    }
    if data is None or data == {}:
        return Response(response=json.dumps({"Error": "Please provide connection information"}),
                        status=400,
                        mimetype='application/json')
    obj1 = MongoAPI(data)
    response = obj1.read()
    return Response(response=json.dumps(response),
                    status=200,
                    mimetype='application/json')
```

Método POST para la inserción de datos a la DB en mongo:

```python
@app.route('/mongodb', methods=['POST'])
def mongo_write():
    data = request.json
    if data is None or data == {} or 'Document' not in data:
        return Response(response=json.dumps({"Error": "Please provide connection information"}),
                        status=400,
                        mimetype='application/json')
    obj1 = MongoAPI(data)
    response = obj1.write(data)
    return Response(response=json.dumps(response),
                    status=200,
                    mimetype='application/json')

if __name__ == '__main__':
    app.run(debug=True, port=80, host='0.0.0.0')
```

Contenido del DockerFile Para levantar la instancia de python:

```
# Step 1 select default OS image
FROM alpine


# # Step 2 tell what you want to do

RUN apk add --no-cache python3-dev
RUN apk add py3-pip

# # Step 3 Configure a software
# # Defining working directory
WORKDIR /app

# # Copy everything which is present in my docker directory to working (/app)
COPY /requirements.txt /app

RUN pip3 install -r requirements.txt

COPY ["Mongo_API.py", "/app"]

# Exposing an internal port
EXPOSE 80
EXPOSE 5000

RUN mkdir -p /elements/procs
RUN touch /elements/procs/ram.txt
RUN touch /elements/procs/stat.txt

# Step 4 set default commands
# These are permanent commands i.e even if user will provide come commands those will be considered as argunemts of this command
ENTRYPOINT [ "python3" ]

# These commands will be replaced if user provides any command by himself
CMD ["Mongo_API.py"]
```

Docker compose para levantar las instancias de Docker simultáneamente:

```yaml
version: "3"
services:
    mymongo_1:
        image: "mongo"
        ports:
          - 5000:27017

    myreader:
        build: .
        depends_on:
            - mymongo_1
        ports:
            - "80:80"
        volumes:
            - /proc/ram-module:/elements/procs/ram.txt
            - /proc/stat:/elements/procs/stat.txt
```

Servidor 1A y 1B

Metodo POST que calcula de acuerdo a las peticiones a los servidores A y B, si debe hacer un post a la base de datos en servidor A o en Servidor B:

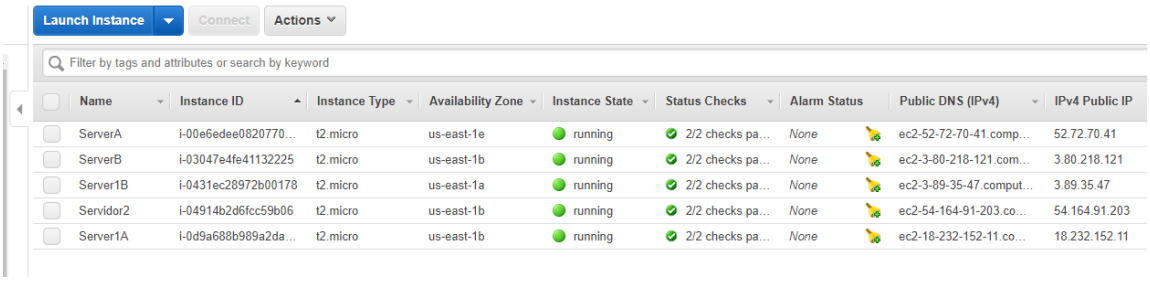
```python
@app.route('/mongodb', methods=['POST'])
def mongo_write():
    data = request.json
    direccion = ''
    resp1 = requests.get('http://52.72.70.41:80/mongodb')
    datos_a = len(resp1.json())
    resp2 = requests.get('http://3.80.218.121:80/mongodb')
    datos_b = len(resp2.json())
    if datos_a < datos_b:
        direccion = 'http://52.72.70.41:80/mongodb'
    elif datos_a > datos_b:
        direccion = 'http://3.80.218.121:80/mongodb'
    else:
        resp1 = requests.get('http://52.72.70.41:80/getRam')
        datos_a = json.loads(json.dumps(resp1.json()))['ramLibre']
        resp2 = requests.get('http://3.80.218.121:80/getRam')
        datos_b = json.loads(json.dumps(resp2.json()))['ramLibre']
        if datos_a < datos_b:
            direccion = 'http://3.80.218.121:80/mongodb'
        elif datos_a > datos_b:
            direccion = 'http://52.72.70.41:80/mongodb'
        else:
            resp1 = requests.get('http://52.72.70.41:80/getCPU')
            datos_a = json.loads(json.dumps(resp1.json()))['cpuLibre']
            resp2 = requests.get('http://3.80.218.121:80/getCPU')
            datos_b = json.loads(json.dumps(resp2.json()))['cpuLibre']
            if datos_a < datos_b:
                direccion = 'http://3.80.218.121:80/mongodb'
            elif datos_a > datos_b:
                direccion = 'http://52.72.70.41:80/mongodb'
            else:
                direccion = 'http://52.72.70.41:80/mongodb'
    response = requests.post(direccion, json = data)
    return Response(response=json.dumps(response.json()),
                    status=200,
                    mimetype='application/json')
```

Docker File para esta API en Python:

```
# Step 1 select default OS image
FROM alpine


# # Step 2 tell what you want to do

RUN apk add --no-cache python3-dev
RUN apk add py3-pip

# # Step 3 Configure a software
# # Defining working directory
WORKDIR /app

# # Copy everything which is present in my docker directory to working (/app)
COPY /requirements.txt /app

RUN pip3 install -r requirements.txt
RUN pip install requests

COPY ["Python_API.py", "/app"]

# Exposing an internal port
EXPOSE 80




# Step 4 set default commands
# These are permanent commands i.e even if user will provide come commands those will be considered as argunemts of this command
ENTRYPOINT [ "python3" ]

# These commands will be replaced if user provides any command by himself
CMD ["Python_API"]
```

Servidor 2

Instancias EC2 Activas:



Balanceador de cargas

Instancia en AWS del balanceador de cargas