

COORDINACIÓN Y SINCRONIZACIÓN DE NODOS EN SISTEMAS DISTRIBUIDOS, asegurando consistencia y disponibilidad.

Zookeeper

Juan Miguel Sarria Orozco

1. Construcción básica del sistema

a) Construcción de nuestra API REST

Código:

Variables necesarias:

Para construir la API REST, se requieren las siguientes variables clave:

- dato: Variable que almacena el dato generado aleatoriamente.
- url: Dirección a la cual se enviarán los datos.
- id: Identificador único del dispositivo o instancia de la aplicación.
- zk: Cliente de Kazoo para interactuar con Zookeeper.

Funciones utilizadas:

El sistema implementa varias funciones para manejar la lógica de la aplicación:

- enviarDato: Envía un dato medido a la URL especificada.
- interrupt_handler: Permite salir de la aplicación de forma controlada al presionar Ctrl+C.
- crearNodo: Crea un nodo en Zookeeper con un valor asignado.
- existeNodo: Verifica si un nodo existe.
- getListaHijos: Obtiene la lista de nodos hijos.
- getValoresHijos: Devuelve los valores de los nodos hijos.
- getDatosNodo: Recupera los datos de un nodo específico.
- setDatosNodo: Actualiza el valor de un nodo.
- leader_func: Ejecuta la lógica del nodo líder, calculando la media de los valores de los nodos hijos y enviándola al servidor.
- election_func: Gestiona el proceso de elección del líder entre las instancias activas.

El código de la aplicación será el siguiente:

```
# -*- coding: utf-8 -*-
from kazoo.client import KazooClient
from kazoo.recipe.election import Election
import os
import threading
import time
import random
import requests
import signal

#VARIABLES
#=====
dato = 0
url = 'http://127.0.0.1/detectar'
id = os.environ.get('INSTANCE_ID', str(random.randint(10000000,
99999999)))
ZOOKEEPER_HOST = os.getenv("ZOOKEEPER_HOST", "localhost:2181")
zk = KazooClient(hosts=ZOOKEEPER_HOST, timeout=30)
#Comienza el cliente zookeeper
zk.start()

#FUNCIONES
#=====
#Funcion que le envias una url y un valor y envia el dato a esa url
def enviarDato(valor):
    params = {'dato': valor}
    try:
        response = requests.get(url, params=params) # Hacemos la petición
        GET con el parámetro y guardamos la respuesta en una variable
    except:
        print("El dato no se envió al servidor dado que hubo un error al
conectar con el mismo")

# Función que se ejecuta cuando se recibe la señal de interrupción y se
cierra
def interrupt_handler(signal, frame):
    exit(0)

#Funcion que crea un nodo con un valor
def crearNodo(valor):
    try:
        zk.ensure_path("/my/favorite")
        zk.create(f"/my/favorite/{id}", str(valor).encode("utf-8"),
ephemeral=True)
        print("Nodo creado")
    except:
        print('Error al crear nodo.')
```

```

#Funcion que devuelve si existe o no el nodo
def existeNodo():
    return zk.exists("/my/favorite") is not None

#Funcion que devuelve el nodo
def getNodo():
    try:
        (data, _) = zk.get(f"/my/favorite/{id}")
        return data.decode("utf-8")
    except Exception as e:
        print(f"Error al obtener el nodo: {e}")
        return None

#Funcion que devuelve la lista de hijos (que no el valor)
def getListaHijos():
    try:
        children = zk.get_children("/my/favorite")
        return children
    except Exception as e:
        print(f"Error al obtener los hijos: {e}")
        return []

#Funcion para devolver el valor de los hijos
def getValoresHijos():
    # Obtener todos los hijos de /my/favorite
    if existeNodo():
        children = zk.get_children("/my/favorite")

        # Lista para almacenar los valores de los hijos
        valores = []

        # Obtener el valor de cada hijo
        for hijo in children:
            try:
                data, _ = zk.get(f"/my/favorite/{hijo}")
                valores.append(float(data))
            except Exception as e:
                print(f"Error al obtener el valor del hijo {hijo}: {e}")
        return valores
    else:
        print("El nodo /my/favorite no existe.")
        return []

```

```

#Funcion que devuelve el dato de un nodo
def getDatosNodo(id):
    try:
        (data, _) = zk.get(f"/my/favorite/{id}")
        return float(data)
    except Exception as e:
        print(f"Error al obtener los datos del nodo {id}: {e}")
        return None

#Funcion que setea el valor de un nodo
def setDatosNodo(valor):
    try:
        if zk.exists(f"/my/favorite/{id}"):
            zk.set(f"/my/favorite/{id}", str(valor).encode("utf-8"))
        else:
            zk.create(f"/my/favorite/{id}", str(valor).encode("utf-8"),
ephemeral=True)
    except Exception as e:
        print(f"Error al setear el valor del nodo: {e}")

# Funcion que ejecuta el lider para hacer la media de todos los hijos y
enviarla al servidor
def leader_func():
    print("Soy el nuevo lider")
    while True:
        valores = getValoresHijos()

        if valores: # Evitar dividir por cero
            print(valores)
            media = sum(valores) / len(valores)
            print(f"La media es: {media}")
            enviarDato(media)
        else:
            print("No hay valores disponibles para calcular la media.")

        # Enviar la media usando requests
        time.sleep(5)

#Funcion que decide el lider
def election_func():
    election.run(leader_func)

```

```

#PROGRAMA
#=====

# Registrar la función como el manejador de la señal de interrupción
signal.signal(signal.SIGINT, interrupt_handler)

# Crear una elección entre las aplicaciones y elegir un líder
election = Election(zk, "/election",id)

# Crear un hilo para ejecutar la función election_func
election_thread = threading.Thread(target=election_func, daemon=True)

# Iniciar el hilo
election_thread.start()
# Enviar periódicamente un valor a una subruta de /mediciones con el
identificador de la aplicación
while True:
    # Generar una nueva medición aleatoria
    dato = random.randint(75, 85)

    # Crea un nodo si no existe o setea un nuevo valor del nodo
    if existeNodo():
        setDatosNodo(dato)
    else:
        crearNodo(dato)
    time.sleep(5)

```

Ejecución en local:

Para ejecutar esta aplicación en un entorno local, necesitamos que la aplicación de la Práctica 2, Zookeeper y Redis estén en funcionamiento. Esto se logra ejecutando instancias de RedisTimeSeries y Zookeeper en contenedores Docker.










Para iniciarlos, ejecutamos los siguientes comandos en una terminal de Windows (CMD):Redis:

docker run --name some-redis -p 6379:6379 -d redislabs/redis-timeseries

Zookeeper:

docker run --name some-zookeeper --restart always -d -p 2181:2181 zookeeper

Estos comandos descargarán las imágenes necesarias de Docker Hub y ejecutarán los contenedores correspondientes. Puedes verificar que están activos abriendo Docker Desktop y comprobando el estado de los servicios.

	Name	Container ID	Image	Port(s)	CPU (%)	Last started	Actions
	 some-redis	b04235524354	redislabs/redis-timeseries	6379:6379 	0.1%	21 minutes ago	  
	 some-zookeepe	bac5e74769c6	zookeeper	2181:2181 	0.07%	2 hours a	  

[Acceso a la aplicación:](#)

Con los contenedores de Redis y Zookeeper en funcionamiento, iniciamos nuestra aplicación Flask (Práctica 2) ejecutando el archivo Python correspondiente. Luego, abrimos un navegador web y accedemos a la URL: <http://127.0.0.1>

Mediciones de Juan Miguel Sarria Orozco

Servidor: DESKTOP-EGFG9TS

[Ver datos de anomalías](#)

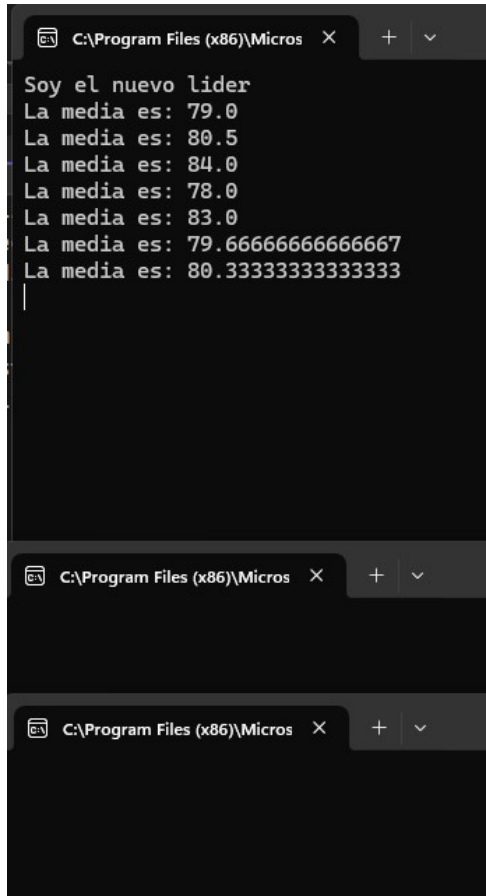
Lista de mediciones guardadas:

- Medición 0: Error al conectar con Redis: TSDB: the key does not exist

Al ingresar esta dirección, deberíamos ver la aplicación funcionando correctamente en el navegador.

Añadir Mediciones desde Zookeeper:

Una vez con nuestra web activa en la URL 127.0.0.1 y los servicios de Redis y Zookeeper funcionando ejecutamos nuestro nuevo código (Practica3) 3 veces para ver su funcionamiento:



```
Soy el nuevo lider
La media es: 79.0
La media es: 80.5
La media es: 84.0
La media es: 78.0
La media es: 83.0
La media es: 79.66666666666667
La media es: 80.33333333333333
```

Podemos ver como el líder va calculando medias y enviándolas a nuestro servidor:

Mediciones de Juan Miguel Sarria Orozco

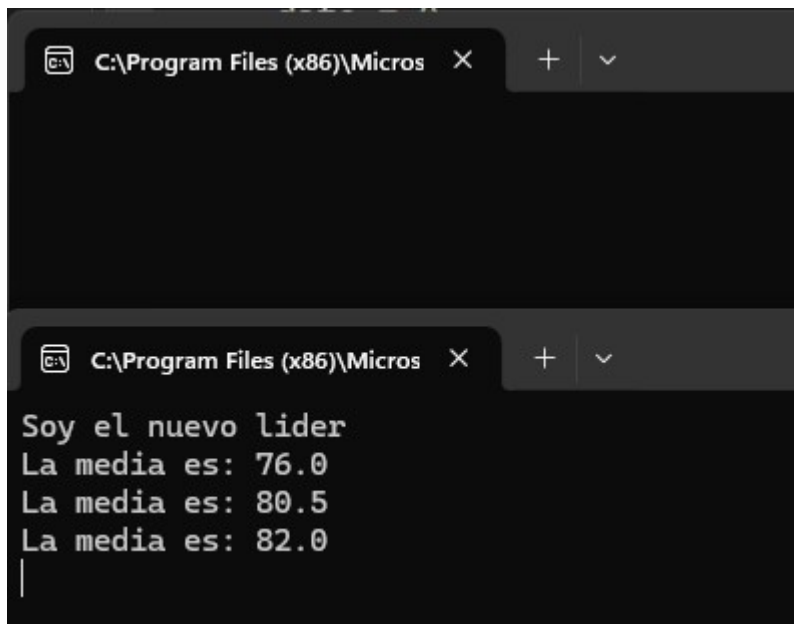
Servidor: DESKTOP-EGFG9TS

[Ver datos de anomalías](#)

Lista de mediciones guardadas:

- Medición 1: Timestamp: 1734089895723, Valor: 79.0
- Medición 2: Timestamp: 1734089900823, Valor: 80.5
- Medición 3: Timestamp: 1734089905922, Valor: 84.0
- Medición 4: Timestamp: 1734089911016, Valor: 78.0
- Medición 5: Timestamp: 1734089916359, Valor: 83.0
- Medición 6: Timestamp: 1734089921713, Valor: 79.66666666666667
- Medición 7: Timestamp: 1734089927079, Valor: 80.33333333333333

Además si cerramos el servidor líder, vemos como otro coge el rol de líder en su lugar y sigue enviando peticiones:



```
C:\Program Files (x86)\Micros >
Soy el nuevo lider
La media es: 76.0
La media es: 80.5
La media es: 82.0
|
```

Mediciones de Juan Miguel Sarria Orozco

Servidor: DESKTOP-EGFG9TS

[Ver datos de anomalías](#)

Lista de mediciones guardadas:

- Medición 1: Timestamp: 1734089895723, Valor: 79.0
- Medición 2: Timestamp: 1734089900823, Valor: 80.5
- Medición 3: Timestamp: 1734089905922, Valor: 84.0
- Medición 4: Timestamp: 1734089911016, Valor: 78.0
- Medición 5: Timestamp: 1734089916359, Valor: 83.0
- Medición 6: Timestamp: 1734089921713, Valor: 79.66666666666667
- Medición 7: Timestamp: 1734089927079, Valor: 80.33333333333333
- Medición 8: Timestamp: 1734089932594, Valor: 82.33333333333333
- Medición 9: Timestamp: 1734089945999, Valor: 76.0
- Medición 10: Timestamp: 1734089951331, Valor: 80.5
- Medición 11: Timestamp: 1734089956658, Valor: 82.0

2. Desarrollo de stack Docker para la solución propuesta

Codigo:

Hemos realizado algunos ajustes en el código para que funcione correctamente con Docker y Zookeeper. Estos cambios incluyen:

Establecer la URL como:

```
url = 'http://web:80/detectar'
```

Añadir variables de entorno para Zookeeper y el identificador de la instancia:

```
id = os.environ.get('INSTANCE_ID', str(random.randint(10000000, 99999999)))
ZOOKEEPER_HOST = os.getenv("ZOOKEEPER_HOST", "localhost:2181")
```

DockerFile:

Dado que nuestra aplicación se llama app.py, está escrita en Python y se expondrá en el puerto 80 (HTTP), el archivo Dockerfile es el siguiente::

```
# Use an official Python runtime as a parent image
FROM python:3.11-slim

# Set the working directory to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
ADD . /app

# Install any needed packages specified in requirements.txt
RUN pip install --trusted-host pypi.python.org -r requirements.txt

# Make port 80 available to the world outside this container
EXPOSE 80

# Run app.py when the container launches
CMD ["python", "practica3.py"]
```









Este archivo Dockerfile es bastante similar al ejemplo de la práctica. A continuación, para crear nuestro contenedor, ejecutaremos el siguiente comando en el CMD de Windows:

docker build -t practica3 .

```
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker build -t practica3 .
[+] Building 5.2s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile               0.0s
=> => transferring dockerfile: 529B                               0.0s
=> [internal] load metadata for docker.io/library/python:3.11-slim 1.4s
=> [auth] library/python:pull token for registry-1.docker.io      0.0s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                       0.0s
=> [1/4] FROM docker.io/library/python:3.11-slim@sha256:370c586a6ffc8c619e6d652f81c094b34b14b8f2fb9251f092de23f1 0.9s
=> => resolve docker.io/library/python:3.11-slim@sha256:370c586a6ffc8c619e6d652f81c094b34b14b8f2fb9251f092de23f1 0.0s
=> => sha256:5f20101c4b63a3742b4155dcffc296c8ee0e2c7b83dbe6c7687441d991f358d4 250B / 250B 0.2s
=> => sha256:51e2bd9c4b085ec5483ea37c7aff2b4fa3e49a780ff4db8f971fa68c9539db9c 16.20MB / 16.20MB 0.7s
=> => sha256:3a75594a45d1452af40f08d7e4d4fa96572c3a09fb46ae142cf45e98973b3c69 3.32MB / 3.32MB 0.5s
=> => extracting sha256:3a75594a45d1452af40f08d7e4d4fa96572c3a09fb46ae142cf45e98973b3c69 0.1s
=> => extracting sha256:51e2bd9c4b085ec5483ea37c7aff2b4fa3e49a780ff4db8f971fa68c9539db9c 0.2s
=> => extracting sha256:5f20101c4b63a3742b4155dcffc296c8ee0e2c7b83dbe6c7687441d991f358d4 0.0s
=> [internal] load build context                                  0.1s
=> => transferring context: 1.41MB                                   0.0s
=> [2/4] WORKDIR /app                                           0.0s
=> [3/4] ADD . /app                                              0.0s
=> [4/4] RUN pip install --trusted-host pypi.python.org -r requirements.txt 2.1s
=> exporting to image                                           0.6s
=> => exporting layers                                              0.4s
=> => exporting manifest sha256:574877c2a83ce3f39b4ac00697d672bc495771ee3a39f5a572fb95b256c85b13 0.0s
=> => exporting config sha256:b4b81a2c338cf232f6fb38203bd3794b2155a68c90d19b050f4e14f647c5f137 0.0s
=> => exporting attestation manifest sha256:b5d84126b5a2af76d5d63fcf2f485967d9e8318ad49c474b61b4cd37cf5242a3 0.0s
=> => exporting manifest list sha256:8c28a245ebe6cdd60c22300101c73616a328a8529a0215d876c03e8ab610ab3e 0.0s
=> => naming to docker.io/library/practica3:latest                 0.0s
=> => unpacking to docker.io/library/practica3:latest              0.1s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ldw96gre99tu8mpp2k639illf
```

Una vez que el contenedor haya sido creado, podremos verlo en la sección de Imágenes de Docker Desktop, junto al contenedor de Redis que creamos anteriormente.

<input type="checkbox"/>	Name	Tag	Image ID	Created	Size	Actions
<input type="checkbox"/>	 redislabs/redis-timeseries	latest	5a15401e18be	2 years ago	226.58 MB	  
<input type="checkbox"/>	 practica2	latest	b2e94ff0473b	29 seconds ago	229.91 MB	  

Subir el contenedor a DockerHub:

Procedemos a realizar el push del contenedor a DockerHub para tenerlo disponible en nuestro repositorio. El proceso será el siguiente:

- Inicia sesión en DockerHub con el comando docker login.
- Etiqueta la imagen para subirla a DockerHub:
- Realizamos el push a DockerHub:

```
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker login
Authenticating with existing credentials...
Login Succeeded

C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker tag practica3 kazukigd/practica3:part1

C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker push kazukigd/practica3:part1
The push refers to repository [docker.io/kazukigd/practica3]
92512649bd31: Pushed
bc0965b23a04: Mounted from kazukigd/practica2
3a75594a45d1: Pushed
5f20101c4b63: Pushed
629b7d16f57a: Pushed
6e1d882e3ca5: Pushed
a1133a183996: Pushed
51e2bd9c4b08: Pushed
part1: digest: sha256:8c28a245ebe6cdd60c22300101c73616a328a8529a0215d876c03e8ab610ab3e size: 856

C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>
```

Una vez subido, podremos ver la imagen en nuestro repositorio de DockerHub.

LocalHub

kazukigdSearchView Scout dashboard

	Tags	OS	Vulnerabilities	Last pushed	Size
kazukigd/practica3	part1		Inactive	43 seconds ago	55.41 MB
kazukigd/practica2	part2		Inactive	9 days ago	1.45 GB
	part1		Inactive	9 days ago	58.58 MB

[Crear el archivo docker-compose.yml](#)

El siguiente paso es crear un archivo **docker-compose.yml** para gestionar todos nuestros servicios. El contenido de este archivo será el siguiente:

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: kazukigd/practica2:part2
    deploy:
      replicas: 5
      restart_policy:
        condition: on-failure
    ports:
      - "4000:80"
    environment:
      - REDIS_HOST=redis
    volumes:
      - shared-data:/app # Volumen compartido entre las replicas
    networks:
      - webnet

# Servicio de ZooKeeper
zookeeper:
  image: zookeeper:latest # Imagen oficial de ZooKeeper
  ports:
    - "2181:2181" # Puerto estándar de comunicación de ZooKeeper
  networks:
    - webnet # Conectar al mismo red que los otros servicios
# Servicios para las tres instancias de Practica3.py
practica3_instance_1:
  image: kazukigd/practica3:part1
  environment:
    - ZOOKEEPER_HOST=zookeeper
    - INSTANCE_ID=instance_1 # Asignar ID única a esta instancia
  networks:
    - webnet
  depends_on:
    - zookeeper # Depende del servicio de ZooKeeper

practica3_instance_2:
  image: kazukigd/practica3:part1
  environment:
    - ZOOKEEPER_HOST=zookeeper
    - INSTANCE_ID=instance_2 # Asignar ID única a esta instancia
  networks:
    - webnet
  depends_on:
    - zookeeper # Depende del servicio de ZooKeeper
```

```
practica3_instance_3:
  image: kazukigd/practica3:part1
  environment:
    - ZOOKEEPER_HOST=zookeeper
    - INSTANCE_ID=instance_3 # Asignar ID única a esta instancia
  networks:
    - webnet
  depends_on:
    - zookeeper # Depende del servicio de ZooKeeper

visualizer:
  image: dockersamples/visualizer:stable
  ports:
    - "8080:8080"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
  deploy:
    placement:
      constraints: [node.role == manager]
  networks:
    - webnet

redis:
  image: redislabs/redis-timeseries:latest
  ports:
    - "6379:6379"
  deploy:
    placement:
      constraints: [node.role == manager]
  networks:
    - webnet

grafana:
  image: grafana/grafana
  ports:
    - 3000:3000
  volumes:
    - grafana_data:/var/lib/grafana
  depends_on:
    - redis
  environment:
    GF_INSTALL_PLUGINS: redis-datasource
  networks:
    - webnet

volumes:
  grafana_data:
  shared-data: # Definimos el volumen compartido

networks:
  webnet:
```

En este archivo docker-compose.yml, además de todo lo necesario de la practica2 hemos añadido 3 copias de nuestro nuevo programa, además de un zookeeper para que todos se conecten a este zookeeper.

Desplegar el stack

Para desplegar nuestro stack de Docker, primero iniciaremos con el comando `docker swarm init` y después utilizaremos el comando `docker stack deploy -c docker-compose.yml practica3Stack`

```
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker stack deploy -c docker-compose.yml practica3Stack
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating network practica3Stack_webnet
Creating service practica3Stack_practica3_instance_1
Creating service practica3Stack_practica3_instance_2
Creating service practica3Stack_practica3_instance_3
Creating service practica3Stack_visualizer
Creating service practica3Stack_redis
Creating service practica3Stack_grafana
Creating service practica3Stack_web
Creating service practica3Stack_zookeeper
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>
```

Esto iniciará todos los servicios definidos en el archivo docker-compose.yml. Podemos comprobar que el stack se ha desplegado correctamente ejecutando el siguiente comando: `docker stack ls`

```
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker stack ls
NAME                SERVICES
practica3Stack      8

C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>docker service ls
ID                NAME                                MODE                REPLICAS    IMAGE                                  PORTS
iui6ldywwz4rc    practica3Stack_grafana             replicated          1/1          grafana/grafana:latest              *:3000->3000/tcp
wiz5g8skb6yy     practica3Stack_practica3_instance_1 replicated          1/1          kazukigd/practica3:part1
9yboiqwuoyai     practica3Stack_practica3_instance_2 replicated          1/1          kazukigd/practica3:part1
l6qm2230bprq     practica3Stack_practica3_instance_3 replicated          1/1          kazukigd/practica3:part1
hdx6oLrspfzd     practica3Stack_redis               replicated          1/1          redislabs/redis-timeseries:latest   *:6379->6379/tcp
vd7w5w7i8x29     practica3Stack_visualizer          replicated          1/1          dockersamples/visualizer:stable     *:8080->8080/tcp
owk983dc548u     practica3Stack_web                 replicated          5/5          kazukigd/practica2:part2            *:4000->80/tcp
uldvkrhbj51v     practica3Stack_zookeeper           replicated          1/1          zookeeper:latest                    *:2181->2181/tcp
C:\Uni 2024\Repositorios\SoftwareCritico\Practica3>
```

Esto mostrará el estado de nuestra stack, y podremos verificar si todos los servicios están funcionando correctamente.

[Acceder a los servicios](#)

Una vez que el stack esté corriendo, podemos acceder a los diferentes servicios desde el navegador:

Aplicación web: Disponible en **<http://localhost:4000>**, donde podemos observar cómo se añaden los datos automáticamente.

Grafana: Accesible en **<http://localhost:3000>**, donde podremos ver cómo cada 5 segundos se actualizan las medias de los datos almacenados.

De esta forma se vería nuestra aplicación web y podemos comprobar cómo se añaden los datos automáticamente

Mediciones de Juan Miguel Sarria Orozco

Servidor: 8e5859462698

Insertar

Limpiar

[Ver datos de anomalías](#)

Lista de mediciones guardadas:

- Medición 1: Timestamp: 1734103892669, Valor: 79.66666666666667
- Medición 2: Timestamp: 1734103897753, Valor: 79.33333333333333
- Medición 3: Timestamp: 1734103902835, Valor: 78.0
- Medición 4: Timestamp: 1734103907916, Valor: 78.66666666666667
- Medición 5: Timestamp: 1734103913131, Valor: 80.66666666666667
- Medición 6: Timestamp: 1734103918294, Valor: 77.33333333333333
- Medición 7: Timestamp: 1734103923448, Valor: 80.66666666666667
- Medición 8: Timestamp: 1734103928601, Valor: 77.66666666666667
- Medición 9: Timestamp: 1734103933757, Valor: 81.0

Además también podemos ver en Grafana como se añaden automáticamente cada 5 segundos la media de los datos:

