Desarrollo de Software Crítico
Práctica 3

Diseño de una Infraestructura de Mediciones con un Sistema de Coordinación Distribuida y Elección de Líder mediante Zookeeper y Docker Compose

Daniil Nemchenko
3ºA Ingeniería del Software

Parte 1: Desarrollo de la aplicación de dispositivo de medición

En primer lugar, se ha desarrollado el código para el dispositivo de medición. El archivo final que integra los requisitos de los tres primeros apartados es `main_part_1_2_3.py`.

Asimismo, se ha creado una versión adaptada llamada `main_all_docker.py`. La única diferencia entre ambos es que esta última versión recibe el ID del dispositivo mediante un parámetro de ejecución en lugar de utilizar la función `input()`, facilitando así su integración en un contenedor Docker.

A continuación, se detallan las diferencias en la implementación:

En `main_part_1_2_3.py` :

```
# Crear un identificador para la aplicación
id = input("Introduce un identificador: ")
```

En `main_all_docker.py`:

```
if len(sys.argv) > 1:
    id = sys.argv[1]
else:
    print("Error: Se requiere un identificador como argumento.")
    sys.exit(1)
```

Análisis del Código Fuente

En el desarrollo del código fuente, destacan los siguientes aspectos técnicos y de arquitectura:

Configuración mediante Variables de Entorno

Para facilitar la portabilidad y el despliegue en contenedores Docker, se han utilizado variables de entorno para definir parámetros críticos como `SAMPLING_PERIOD`, `ZK_HOST` (host de Zookeeper) y `API_URL`. Esto permite modificar el comportamiento del sistema sin necesidad de alterar el código fuente.

Arquitectura Líder-Seguidor (Leader-Follower)

La lógica del programa se divide en dos roles principales:

Rol de Líder (`leader_func()`): Esta sección gestiona la coordinación global mediante *watchers* específicos (se establecen solamente para el líder):

- **`watch_api_url()`:** Monitoriza cambios en el nodo de configuración.
- **`children_watcher()`:** Supervisa los dispositivos activos en el sistema.

En su bucle principal, el líder recopila las mediciones de todos los seguidores y calcula el promedio. Los datos se envían a la API solo si existe al menos una medición activa. Este control es crítico para gestionar desconexiones temporales o errores de red de los dispositivos. Además, se ha implementado la receta de Barrera de la librería *kazoo* para asegurar la sincronización de los seguidores.

Rol de Seguidor (Follower): Los seguidores ejecutan una lógica enfocada en la obtención de datos:

- `watch_sampling_period()`: Este watcher se declara en la sección de seguidor. Permite detectar cambios en el periodo de muestreo definidos en la configuración.
- **Generación de datos**: En cada iteración, el seguidor genera una medición y la guarda en su nodo correspondiente (`mediciones/id`).

Gestión de Excepciones y Resiliencia

Un aspecto importante es la gestión de la excepción `NoNodeError` al guardar mediciones. Dado que los nodos son efímeros (se eliminan automáticamente si se pierde la conexión), existe una pequeña probabilidad de que el nodo desaparezca momentáneamente durante el arranque debido a micro-cortes de red (Se ha detectado en las pruebas, aunque eran ocasiones puntuales). Para mitigar esto, el sistema detecta la ausencia del nodo y lo recrea automáticamente con el valor medido, garantizando el funcionamiento correcto del servicio.

A continuación, se presenta la implementación de esta solución:

```
try:
    # Modificar el valor en el nodo correspondiente
    zk.set( path: f"/mediciones/{id}", str(value).encode("utf-8"))
    print(f"Node {id}: {value} sent to zk\n")

except NoNodeError:
    # Node was deleted by ZooKeeper - recreate it
    print(f"Node {id}: Nodo fue eliminado por un fallo de conexion, recreando...")
    zk.create( path: f"/mediciones/{id}", str(value).encode("utf-8"), ephemeral=True)
    print(f"Node {id}: {value} sent to zk (recreated)\n")
```

Prueba de ejecución

Se han lanzado tres instancias para realizar la prueba.
La primera instancia se convierte en líder:

```
----------WATCHER----------
Dispositivos conectados: ['1']
-----------------------------

Node 1: 82 sent to zk

Valor actual del contador: 25
Node 1: Esperando en la barrera...

----------WATCHER----------
Dispositivos conectados: ['1', '2']
-----------------------------


LEADER :
Hay 2 dispositivos activos: ['1', '2']
Valor del dispositivo 1 a considerar :  82
Valor del dispositivo 2 a considerar :  82
La media es: 82.0
Enviando request...
El nuevo valor fue enviado a la API correctamente!

Abriendo la barrera...
Node 1: 83 sent to zk

Valor actual del contador: 28
Node 1: Esperando en la barrera...

----------WATCHER----------
Dispositivos conectados: ['1', '2', '3']
-----------------------------
```

Observamos que al ser líder, el watcher correspondiente detecta cuando se conectan o desconectan dispositivos y muestra el mensaje correspondiente.

También a diferencia de seguidores el nodo líder muestra `LEADER`, cita los seguidores activos en ese momento: `Hay 2 dispositivos activos: ['1', '2']` así como el valor medido de cada uno de ellos: `Valor del dispositivo 2 a considerar :  82` . Al final se muestra la media

calculada y el resultado de request a la API: `El nuevo valor fue enviado a la API correctamente!` A continuación, se abre la barrera.

Por otro lado tanto el líder como los seguidores muestran su medición junto con el valor del contador global de mediciones (se debe resetear manualmente en el Zookeeper). Hay que considerar que el líder está ejecutando dos hilos (el de líder y de seguidor) por lo que el output puede solaparse por concurrencia.

```
Valor actual del contador: 29
Node 3: Esperando en la barrera...
Node 3: 79 sent to zk

Valor actual del contador: 32
Node 3: Esperando en la barrera...
Node 3: 85 sent to zk

Valor actual del contador: 35
Node 3: Esperando en la barrera...
Node 3: 80 sent to zk

Valor actual del contador: 37
Node 3: Esperando en la barrera...
Node 3: 85 sent to zk
```

Del valor de contador se puede deducir que hay tres dispositivos de medición ya que aumenta en tres unidades cada ronda.

Si eliminamos cualquiera de los nodos (ya sea líder o seguidor), el líder lo detecta y muestra el mensaje correspondiente. Lo mismo pasa si modificamos los valores de configuración en el Zookeeper.

```
----------WATCHER----------
Dispositivos conectados: ['1', '3']
---------------------------
```

(Se ha eliminado el nodo 2)

Funcionamiento del líder con tres dispositivos (incluido lider):

```
----------WATCHER----------
Dispositivos conectados: ['1', '2', '3']
---------------------------


LEADER :
Hay 3 dispositivos activos: ['1', '2', '3']
Valor del dispositivo 1 a considerar :  83
Valor del dispositivo 2 a considerar :  83
Valor del dispositivo 3 a considerar :  84
La media es: 83.33333333333333
Enviando request...
El nuevo valor fue enviado a la API correctamente!

Abriendo la barrera...
Node 1: 77 sent to zk

Valor actual del contador: 32
Node 1: Esperando en la barrera...
```

Si se cambia la configuración se detecta con el watcher y se muestra el mensaje correspondiente:

```
Valor actual del contador: 46
Node 1: Esperando en la barrera...

----------WATCHER----------
Nuevo valor de SAMPLING_PERIOD: 8
---------------------------


----------WATCHER----------
Nuevo valor de API_URL: http://127.0.0.1:4001/
---------------------------


LEADER :
```

2. Contenerización de la aplicación de dispositivo de medición

Una vez terminado y probado el código `main_all_docker.py` en local, se ha procedido a su contenerización. A continuación, se muestra el Dockerfile diseñado para construir la imagen y subirla a HubDocker.

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

# Para asegurar que los logs se emitan en tiempo real se vean en el contentenedor
ENV PYTHONUNBUFFERED=1

# Set the working directory to /app
WORKDIR /app

# Install any needed packages
RUN pip install kazoo requests

# Copy the source code
COPY src/main_all_docker.py .

# Configurar el punto de entrada para permitir el paso de argumentos por línea de comandos
# Permite que Docker Compose le pase el ID único a cada instancia
ENTRYPOINT ["python", "main_all_docker.py"]

# Define environment variable
ENV ZK_HOST=zookeeper:2181
ENV API_URL=http://api-measurements:4000/
ENV SAMPLING_PERIOD=5
```

Creación de la imagen y subida el HubDocker:

```
PS C:\Software\Python\Critical-Software\swc-lab-3> docker build -t measuring-device:v1 .
[+] Building 1.6s (10/10) FINISHED                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                0.0s
 => => transferring dockerfile: 782B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.11-slim                                                 1.1s
 => [auth] library/python:pull token for registry-1.docker.io                                                       0.0s
 => [internal] load .dockerignore                                                                                   0.0s
 => => transferring context: 189B                                                                                   0.0s
 => [1/4] FROM docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c  0.0s
 => => resolve docker.io/library/python:3.11-slim@sha256:158caf0e080e2cd74ef2879ed3c4e697792ee65251c8208b7afb56683c  0.0s
 => [internal] load build context                                                                                   0.0s
 => => transferring context: 67B                                                                                    0.0s
 => CACHED [2/4] WORKDIR /app                                                                                       0.0s
 => CACHED [3/4] RUN pip install kazoo requests                                                                     0.0s
 => CACHED [4/4] COPY src/main_part_1_2_3.py .                                                                      0.0s
 => exporting to image                                                                                              0.5s
 => => exporting layers                                                                                             0.0s
 => => exporting manifest sha256:a6ff9e56b827f283d8e9b553731c3126e2b395cc2e1d057f84219c28eb5a9822                   0.0s
 => => exporting config sha256:3bc076c583447561e9879fda9540a9aeda294bdfd6c168787188e4854af195ff                     0.0s
 => => exporting attestation manifest sha256:da56d2d2b6f21af2abae3ff6b9b2fa967106688ddfbc37373b83b14a48ddae21       0.0s
 => => exporting manifest list sha256:53293014f94863a45660c6bc12c1bd7e82e92254ef50256647ca53b3258755c9              0.0s
 => => naming to docker.io/library/measuring-device:v1                                                              0.0s
 => => unpacking to docker.io/library/measuring-device:v1                                                           0.4s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/jxixkh0qr6vxflyynoq23a2km
PS C:\Software\Python\Critical-Software\swc-lab-3> docker tag measuring-device:v1 danielnem/measuring-device:v1
PS C:\Software\Python\Critical-Software\swc-lab-3> docker push danielnem/measuring-device:v1
The push refers to repository [docker.io/danielnem/measuring-device]
3f0cdbca744e: Pushed
72cf4c3b8301: Pushed
4d55cfecf366: Pushed
805e840fa157: Pushed
bb2e68b14220: Pushed
1733a4cd5954: Pushed
757263c6c2d1: Pushed
2d726ced8158: Pushed
v1: digest: sha256:53293014f94863a45660c6bc12c1bd7e82e92254ef50256647ca53b3258755c9 size: 856
PS C:\Software\Python\Critical-Software\swc-lab-3>
```

Última versión de la imagen: `danielnem/measuring-device:v2`

Pruebas del contenedor en local

Comprobamos el correcto funcionamiento del contenedor en local: Creamos dos contenedores idénticos con id 1 y 2 respectivamente y comprobamos que funcionan correctamente y que interactúan bien con la API y Zookeeper que se han lanzado manualmente en contenedores correspondientes.

```
PS C:\Software\Python\Critical-Software\swc-lab-3> docker run -d -e ZK_HOST=host.docker.internal:2181 -e API_URL=http://ho
st.docker.internal:4001/ --name measuring-device danielnem/measuring-device:v2 1
Unable to find image 'danielnem/measuring-device:v2' locally
v2: Pulling from danielnem/measuring-device
0bd218119fd5: Pull complete
bb2e68b14220: Pull complete
805e840fa157: Pull complete
Digest: sha256:e4d013681f8fb2f3a1d42711b281587b662d2ebefce9b55ce9e6536b3dee133f
Status: Downloaded newer image for danielnem/measuring-device:v2
b5ed3f726db9f3db737361bc20022b37ae8db294b91f7991c261c0e60448acc3
PS C:\Software\Python\Critical-Software\swc-lab-3> docker run -d -e ZK_HOST=host.docker.internal:2181 -e API_URL=http://ho
st.docker.internal:4001/ --name other-measuring-device danielnem/measuring-device:v2 2
20502187e70d123dfa721dc23986b49526f446f13976e932560b9be9c66abfa1
```

3. Despliegue con el Docker Compose

A continuación, se ha desarrollado el fichero `docker-compose.yml` para desplegar tres instancias del dispositivo de medición (descrito en el primer apartado) junto con la API, Zookeeper y todos los servicios relacionados con la API (Grafana y Redis).

Aspectos destacados del `docker-compose.yml` :

```yaml
zookeeper:
  image: zookeeper:3.9
  container_name: zookeeper
  restart: always
  ports:
    - "2181:2181"
  environment:
    ZOO_MY_ID: 1
    ZOO_STANDALONE_ENABLED: "true"
  healthcheck:
    test: [ "CMD", "nc", "-z", "localhost", "2181" ]
    interval: 5s
    timeout: 3s
    retries: 5
    start_period: 10s
  networks:
    - app-network
```

Todos los servicios citados anteriormente utilizan la misma red `app-network`

```yaml
measuring-device-1:
  image: danielnem/measuring-device:v2
  container_name: measuring-device-1
  restart: always
  depends_on:
    zookeeper:
      condition: service_healthy
  command: [ "1" ]
  environment:
    ZK_HOST: zookeeper:2181
    API_URL: http://web-api:80/
    SAMPLING_PERIOD: 5
  networks:
    - app-network
```

```yaml
measuring-device-2:
  image: danielnem/measuring-device:v2
  container_name: measuring-device-2
  restart: always
  depends_on:
    zookeeper:
      condition: service_healthy
  command: [ "2" ]
  environment:
    ZK_HOST: zookeeper:2181
    API_URL: http://web-api:80/
    SAMPLING_PERIOD: 5
  networks:
    - app-network
```

En este apartado se lanza solamente una instancia de Zookeeper por lo que se ha especificado: `ZOO_STANDALONE_ENABLED: "true"`
También se ha establecido un `healthcheck`. Dado que los dispositivos de medición dependen del servicio Zookeeper es muy importante que se compruebe su correcto funcionamiento (mediante healthcheck) antes de lanzar los contenedores dependientes.

A cada una de las instancias de dispositivos de medición se le pasa el **id** correspondiente (`command: [ "1" ]`) y los valores de variables de configuración (`SAMPLING_PERIOD`, `ZK_HOST`, `API_URL`).

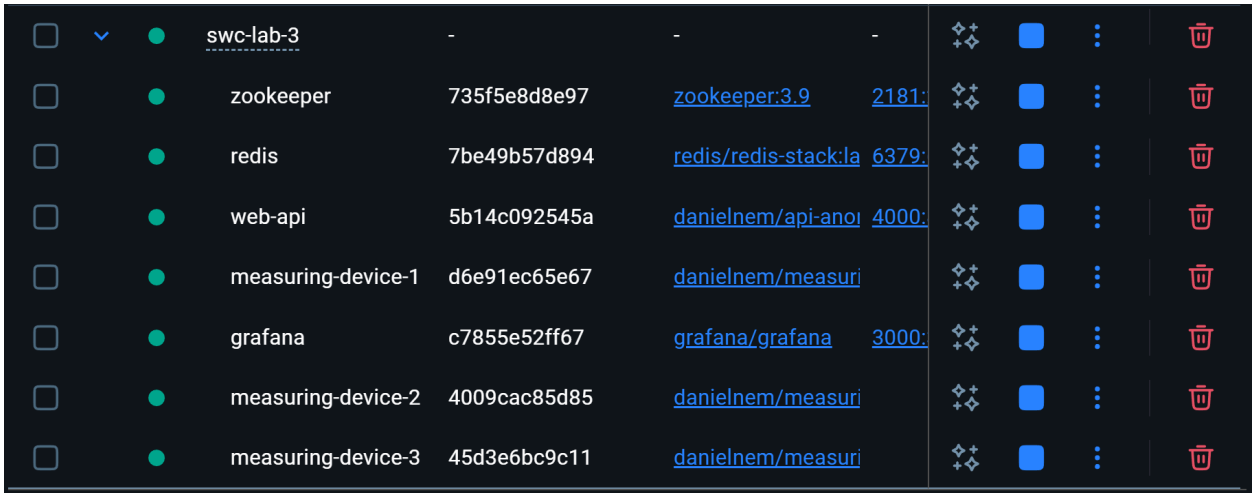
El resto de servicios se han configurado análogamente a la práctica anterior.

Pruebas del sistema desarrollado con docker compose:

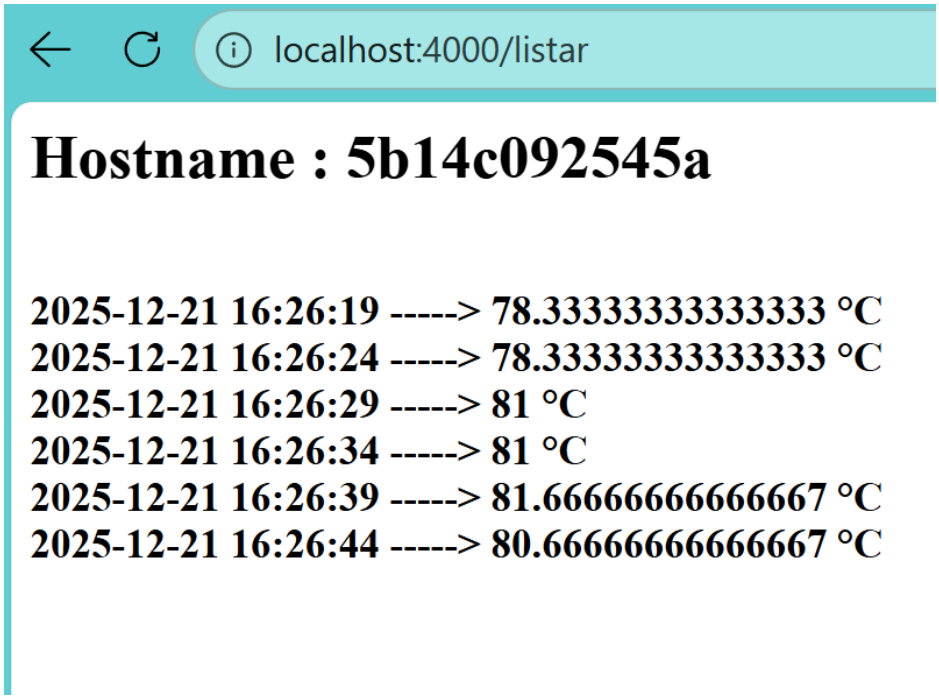
Arrancamos todos los contenedores:

```
PS C:\Software\Python\Critical-Software\swc-lab-3> docker compose up
[+] Running 8/8
 ✔Network swc-lab-3_app-network  Created
 ✔Container redis                Created
 ✔Container zookeeper            Created
 ✔Container web-api              Created
 ✔Container grafana              Created
 ✔Container measuring-device-3   Created
 ✔Container measuring-device-1   Created
 ✔Container measuring-device-2   Created
```

Comprobamos los contenedores lanzados en Docker Desktop:



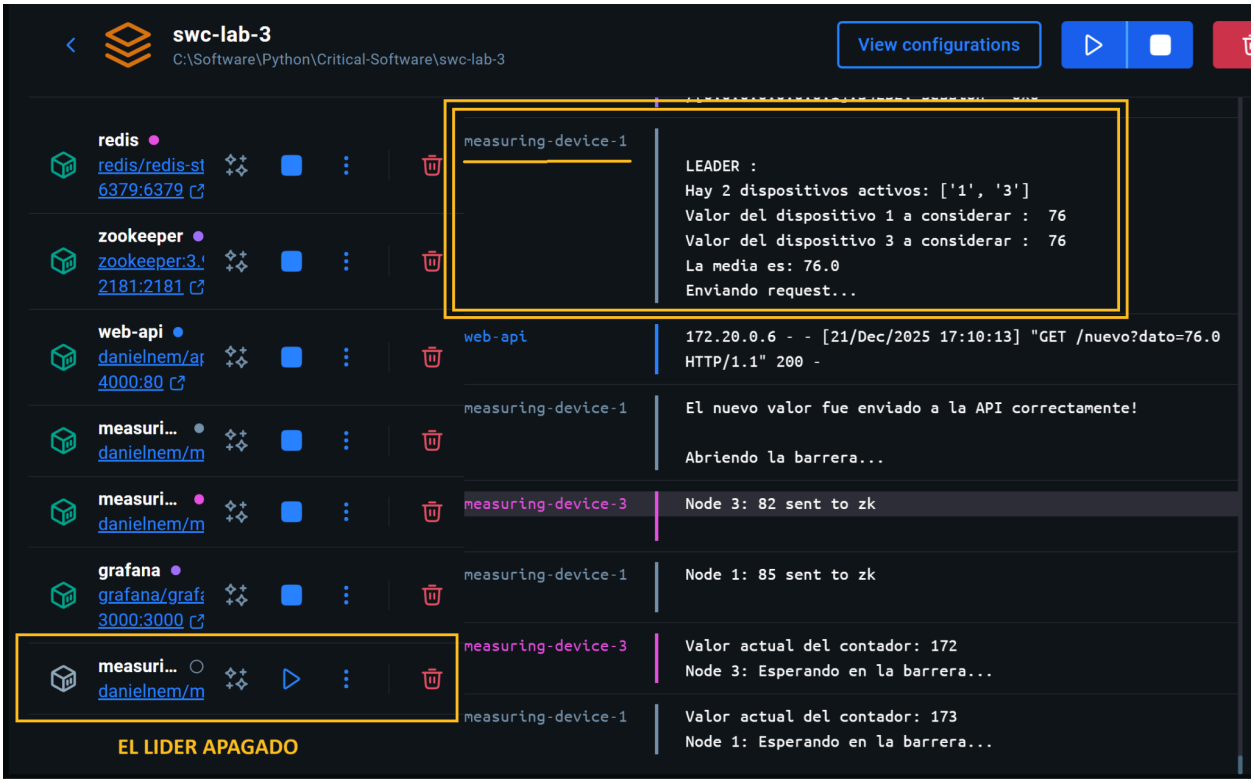
Comprobamos el funcionamiento del sistema accediendo a la API:



Comprobamos el funcionamiento del sistema accediendo a los logs de Docker:

```
measuring-device-2  |
measuring-device-2  | LEADER :
measuring-device-2  | Hay 3 dispositivos activos: ['1', '2', '3']
web-api             | 172.20.0.7 - - [21/Dec/2025 17:05:20] "GET /nuevo?dato=83.0 HTTP/1.1" 200 -
measuring-device-3  | Node 3: 78 sent to zk

measuring-device-1  | Node 1: 75 sent to zk
measuring-device-3  |
measuring-device-1  |



measuring-device-2  | Valor del dispositivo 2 a considerar :  85
measuring-device-1  | Node 1: Esperando en la barrera...

measuring-device-2  | Valor del dispositivo 3 a considerar :  79
measuring-device-2  | La media es: 83.0
measuring-device-2  | Enviando request...
measuring-device-2  | El nuevo valor fue enviado a la API correctamente!
measuring-device-2  |
measuring-device-2  | Abriendo la barrera...
measuring-device-2  | Node 2: 76 sent to zk

measuring-device-3  | Valor actual del contador: 5
measuring-device-3  | Node 3: Esperando en la barrera...
```

Probamos eliminar `measuring-device-2` que es el líder en ese momento:



Observamos que al apagar el líder (`measuring-device-2`), el `measuring-device-1` se convierte en nuevo líder y el sistema sigue funcionando sin interrupciones lo que confirma el funcionamiento correcto de todo el sistema.