

Desarrollo de un sistema distribuido con Zookeeper

Carlos García Guzmán
Universidad de Málaga
carlosgarcia552@uma.es



UNIVERSIDAD
DE MÁLAGA

Contents

1.	Funcionamiento básico del sistema	3
1.1.	Lógica de la Aplicación	3
1.2.	Funcionalidad del Líder	3
1.3.	Robustez	3
1.4.	Ejemplo de ejecución inicial	3
2.	Utilización de Watchers	4
2.1.	Monitorización de Dispositivos (ChildrenWatch)	5
2.1.1.	Ejemplo de monitorización de dispositivos	5
2.2.	Configuración Distribuida (DataWatch)	5
2.2.1.	Ejemplo de configuración distribuida	5
3.	Sincronización Avanzada	6
3.1.	Barreras (Barrier)	6
3.2.	Contador Distribuido (Counter)	6
4.	Despliegue con Docker Compose	7
4.1.	Dockerfile	7
4.2.	Docker Compose	7
5.	Despliegue en Clúster de ZooKeeper	8
5.1.	Pruebas de Tolerancia a Fallos	8
5.1.1.	Estado Inicial	8
5.1.2.	Prueba de tolerancia a fallos de infraestructura (zookeeper)	9
6.	Conclusiones	9

1. Funcionamiento básico del sistema

Se ha desarrollado una solución en Python `src/app.py` que orquesta un conjunto de nodos sensores coordinados mediante **Apache ZooKeeper**, concretamente a través de la librería `kazoo` disponible para Python.

El sistema se compone de un conjunto de nodos que se comunican con un servidor **ZooKeeper** para coordinar sus acciones. Cada nodo se identifica con un ID único y realiza dos funciones principales: actuar como sensor y participar en un proceso de elección de líder.

1.1. Lógica de la Aplicación

Tal como se acaba de mencionar, cada nodo/instancia de la aplicación realiza dos funciones concurrentes:

1. **Sensor (Thread secundario)**: Genera mediciones aleatorias cada 5 segundos y las publica en ZooKeeper. Se utilizan `znodes efímeros` en la ruta `/mediciones/app{id}`, asegurando que las mediciones de un nodo solo existan mientras dure la sesión de ese nodo.
2. **Coordinador/Elección (Thread principal)**: Utilizando la receta `Election` de la librería `kazoo`. El nodo escogido como líder es el encargado de recolectar los datos que escriben todos los nodos (incluido él) y enviarlos a la API desarrollada en la práctica anterior.

1.2. Funcionalidad del Líder

El líder ejecuta un bucle infinito donde:

- Recupera la lista de nodos hijos en `/mediciones`.
- Lee el valor de cada nodo, gestionando posibles condiciones de carrera (nodos que se desconectan durante la lectura).
- Calcula la media aritmética de los valores disponibles.
- Envía el resultado a una API externa `http://localhost:8080/nuevo?dato=valor`

1.3. Robustez

El código incluye manejo de señales (`SIGINT`) para un cierre ordenado de la conexión con ZooKeeper y bloques `try-except` para gestionar errores de red al comunicar con la API o inconsistencias temporales en los datos de ZooKeeper.

1.4. Ejemplo de ejecución inicial

Antes de probar la correcta ejecución del sistema nos debemos asegurar de que tenemos un contenedor ejecutando Zookeeper, así como otro que ejecute la API desarrollada en la práctica anterior (por simplicidad, en esta sección usaremos el fichero `compose-inicial.yaml`, el cual se encarga de ejecutar tanto nuestra API como zookeeper):

```
docker compose -f compose-inicial.yaml up
```

Si ahora ejecutamos varias instancias de nuestra app podemos observar que una de ellas es escogida como líder, como se puede observar en Figure 2:

Desarrollo de Software Crítico

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 1
I'm the leader: (id = 1)
Media: 75.0
I'm the leader: (id = 1)
Media: 82.0
I'm the leader: (id = 1)
Media: 75.5
I'm the leader: (id = 1)
Media: 83.5
I'm the leader: (id = 1)
Media: 80.5
I'm the leader: (id = 1)
Media: 83.5
I'm the leader: (id = 1)
Media: 83.5

(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 2
I'm the leader: (id = 2)
Media: 81.5
I'm the leader: (id = 2)
Media: 79.5
```

Figure 2: Ejecución simple de $N = 2$ instancias de nuestra aplicación.

Si ahora interrumpimos al líder, podemos ver que automáticamente la otra instancia es elegida como nuevo líder, como se muestra en Figure 3

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 1
I'm the leader: (id = 1)
Media: 75.0
I'm the leader: (id = 1)
Media: 82.0
I'm the leader: (id = 1)
Media: 75.5
I'm the leader: (id = 1)
Media: 83.5
I'm the leader: (id = 1)
Media: 80.5
I'm the leader: (id = 1)
Media: 83.5
I'm the leader: (id = 1)
Media: 83.5
I'm the leader: (id = 1)
Media: 79.5
I'm the leader: (id = 1)
Media: 78.0
I'm the leader: (id = 1)
Media: 82.0
I'm the leader: (id = 1)
Media: 80.5
I'm the leader: (id = 1)
Media: 77.5
I'm the leader: (id = 1)
Media: 81.0
I'm the leader: (id = 1)
Media: 77.5
I'm the leader: (id = 1)
Media: 82.0
I'm the leader: (id = 1)
Media: 81.5
^C
Interrupt received
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ 
```

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 2
I'm the leader: (id = 2)
Media: 81.5
I'm the leader: (id = 2)
Media: 79.5
```

Figure 3: Interrupción del líder

Para verificar el correcto funcionamiento de nuestra API, podemos usar el navegador para comprobar que efectivamente todas las medias de las medidas están siendo publicadas, tal como se muestra en Figure 4:

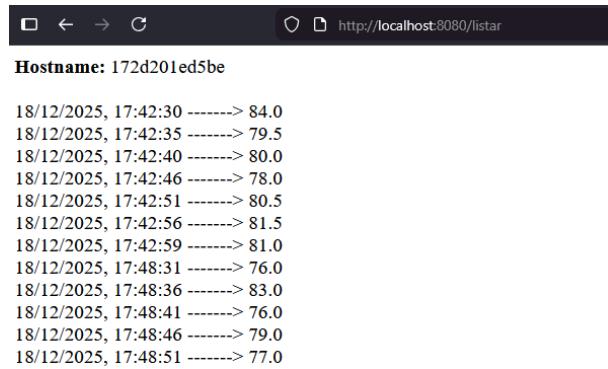


Figure 4: Listado de las medias

2. Utilización de Watchers

En esta fase del desarrollo se han añadido mecanismos de vigilancia (**Watchers**) para dotar al sistema de mayor reactividad y flexibilidad. Se han implementado dos tipos de watchers proporcionados por la librería **kazoo**:

2.1. Monitorización de Dispositivos (ChildrenWatch)

El líder ahora tiene la responsabilidad de monitorizar la conexión y desconexión de los nodos sensores. Para ello, se utiliza un **ChildrenWatch** sobre la ruta **/mediciones**. Cada vez que un nodo se conecta/desconecta, el líder mostrará todos los nodos que se encuentran conectados en el momento (nodos cuya sesión sigue activa).

2.1.1. Ejemplo de monitorización de dispositivos

Al igual que antes, en Figure 5 se muestra una ejecución que permite ver el funcionamiento de estas nuevas funcionalidades. En este caso podemos ver que al iniciar cada instancia inmediatamente se muestran los valores de **/config/sampling_period** y **/config/api_url**, puesto que este es el funcionamiento por defecto del constructor **DataWatch**, el cual se explicará posteriormente. Además, cuando lanzamos el segundo proceso, el líder lo detecta y muestra por pantalla que efectivamente hay 2 dispositivos conectados. Algo similar ocurre cuando interrumpimos al proceso líder, pues el segundo proceso es escogido como nuevo líder y detecta que ahora ya sólo se encuentra él en el sistema.

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 1
[WATCHER] sampling_period value: 5 seconds
[WATCHER] API URL value: http://localhost:8080/nuevo
[WATCHER] Connected devices: ['1']
I'm the leader: (id = 1)
Media: 78.0
I'm the leader: (id = 1)
Media: 84.0
[WATCHER] Connected devices: ['1', '2']
I'm the leader: (id = 1)
Media: 79.5
I'm the leader: (id = 1)
Media: 82.0
I'm the leader: (id = 1)
Media: 83.0
^C
Interrupt received
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ 
```

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 2
[WATCHER] sampling_period value: 5 seconds
[WATCHER] API URL value: http://localhost:8080/nuevo
[WATCHER] Connected devices: ['2']
I'm the leader: (id = 2)
Media: 84.0
I'm the leader: (id = 2)
Media: 78.0
I'm the leader: (id = 2)
Media: 79.0
^C
Interrupt received
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ 
```

Figure 5: Funcionamiento de ChildrenWatch

2.2. Configuración Distribuida (DataWatch)

Para evitar tener valores de configuración estáticos o depender únicamente de variables de entorno que requieren reinicios para cambiarse, se ha añadido un sistema de configuración distribuida utilizando **DataWatch**.

Se han definido dos rutas en ZooKeeper para almacenar la configuración:

- **/config/sampling_period**: Controla el tiempo de espera entre mediciones.
- **/config/api_url**: Define la dirección del servidor al que se envían los datos.

Cada nodo instala un **DataWatch** en estas rutas. Cuando un script externo modifica el valor de estos znodes, la función de callback asociada se ejecuta en todos los nodos, actualizando sus variables globales **SAMPLING_PERIOD** o **API_URL** en tiempo real.

2.2.1. Ejemplo de configuración distribuida

En este ejemplo se utiliza un único proceso y el fichero **src/init_config.py** para cambiar la configuración del sistema. Mientras el proceso se encuentra en ejecución podemos ejecutar el anterior fichero de la siguiente forma:

```
python3 src/init_config.py 6 http://localhost:8081/nuevo
```

Esta nueva configuración lleva al sistema a un estado de error, puesto que la API escucha peticiones en el puerto 8080. Esta interacción se puede ver en la siguiente Figure 6:

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 ./src/app.py 1
[WATCHER] sampling_period value: 5 seconds
[WATCHER] API_URL value: http://localhost:8080/nuevo
[WATCHER] Connected devices: ['1']
I'm the leader: (id = 1)
Media: 79.0
I'm the leader: (id = 1)
Media: 85.0
I'm the leader: (id = 1)
Media: 75.0
I'm the leader: (id = 1)
Media: 77.0
I'm the leader: (id = 1)
Media: 76.0
[WATCHER] sampling_period value: 6 seconds
[WATCHER] API_URL value: http://localhost:8081/nuevo
I'm the leader: (id = 1)
Media: 75.0
ERROR: Failed to contact API: HTTPConnectionPool(host='localhost', port=8081): Max retries exceeded with url: /nuevo?dato=75.0 (Caused by NewConnectionError("HTTPConnectionPool(host='localhost', port=8081): Failed to establish a new connection: [Errno 111] Connection refused"))
I'm the leader: (id = 1)
Media: 75.0
ERROR: Failed to contact API: HTTPConnectionPool(host='localhost', port=8081): Max retries exceeded with url: /nuevo?dato=75.0 (Caused by NewConnectionError("HTTPConnectionPool(host='localhost', port=8081): Failed to establish a new connection: [Errno 111] Connection refused"))
^C
```

Figure 6: Funcionamiento de ChildrenWatch

3. Sincronización Avanzada

En esta iteración sobre el diseño del sistema, se ha modificado la arquitectura para utilizar primitivas de sincronización más avanzadas, sustituyendo los temporizadores locales independientes por una coordinación centralizada mediante barreras.

3.1. Barreras (Barrier)

Se ha implementado una barrera simple `/barrier` para sincronizar el ciclo de medición de todos los dispositivos:

- **Dispositivos:** Tras enviar su medición, se quedan bloqueados esperando en la barrera `barrier.wait()` en lugar de esperar el tiempo `SAMPLING_PERIOD`.
- **Líder:** Es el encargado de controlar el ritmo. Crea la barrera al inicio del ciclo, espera el tiempo de muestreo `SAMPLING_PERIOD`, procesa los datos y finalmente elimina la barrera `barrier.remove()`, liberando a todos los dispositivos simultáneamente para la siguiente iteración.

Esto garantiza que el procesamiento del líder ocurra mientras los dispositivos están en espera, y que todos comiencen el siguiente ciclo a la vez.

3.2. Contador Distribuido (Counter)

Adicionalmente, se ha incorporado un contador distribuido `/counter` para llevar un registro global del número de mediciones realizadas por el clúster. Cada dispositivo incrementa este contador atómicamente `counter += 1` antes de esperar en la barrera.

Cabe destacar que, aunque el incremento en ZooKeeper es atómico y consistente, la impresión por pantalla del valor puede mostrar condiciones de carrera visuales (varios nodos imprimiendo el mismo número). Este problema se podría solucionar fácilmente adquiriendo un `lock` y no soltarlo hasta que se imprima el valor.

En la siguiente imagen Figure 7, se muestra un ejemplo en el que se puede apreciar cómo ambos procesos incrementan el contador. El sistema sigue siendo tolerante a fallos de los nodos pues la elección automática de líder sigue funcionando correctamente.

Desarrollo de Software Crítico

```
(.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 1
I'm the leader: (id = 1)
Counter: 4
Media: 85.0
I'm the leader: (id = 1)
Counter: 5
Media: 75.0
I'm the leader: (id = 1)
Counter: 6
[WATCHER] Connected devices: ['1', '2']
Media: 81.0
I'm the leader: (id = 1)
Counter: 9
Media: 82.0
I'm the leader: (id = 1)
Counter: 10
Media: 81.5
I'm the leader: (id = 1)
Counter: 13
Media: 81.5
I'm the leader: (id = 1)
Counter: 15
Media: 81.5
I'm the leader: (id = 1)
Counter: 17
Media: 79.5
I'm the leader: (id = 1)
Counter: 18
Media: 77.5
I'm the leader: (id = 1)
Counter: 21
Media: 77.5
I'm the leader: (id = 1)
Counter: 23
Media: 82.5
I'm the leader: (id = 1)
Counter: 24
Media: 82.0
I'm the leader: (id = 1)
Counter: 27
^C
Interrupt received

● (.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$ python3 src/app.py 2
Counter: 7
Counter: 8
Counter: 11
Counter: 12
Counter: 14
Counter: 16
Counter: 19
Counter: 20
Counter: 23
Counter: 25
Counter: 27
[WATCHER] Connected devices: ['2']
I'm the leader: (id = 2)
^C
Interrupt received
○ (.venv) carlos@carlos-pc:~/3sw/DSC/practica3-dsc$
```

Figure 7: Funcionamiento del contador

4. Despliegue con Docker Compose

Para facilitar el despliegue y la orquestación de los distintos componentes del sistema (ZooKeeper, API y múltiples instancias de la aplicación), se ha creado un fichero `compose-final.yaml` y un `Dockerfile` para la aplicación.

4.1. Dockerfile

Se ha creado una imagen de Docker basada en `python:3.11-slim`. Caben destacar los siguientes aspectos:

- Se establece `ENV PYTHONUNBUFFERED=1` para asegurar que los logs de Python aparezcan inmediatamente en la consola de Docker.
- Se define `ENTRYPOINT ["python", "src/app.py"]` para que el contenedor actúe como un ejecutable de nuestra aplicación, aceptando el ID como argumento.

En la imagen Figure 8 se muestran los contenidos del dockerfile a partir del cual se ha construido la imagen `carlosgg0/practica3`, la cual se encuentra disponible en **Docker Hub**. Cabe destacar que se han subido dos versiones para cubrir los distintos escenarios de la práctica:

- `carlosgg0/practica3:v1`: Soporta correctamente ZooKeeper en modo **standalone** (nodo único).
- `carlosgg0/practica3:v2`: Soporta ZooKeeper en modo **cluster** (múltiples hosts) y reconexión automática.

4.2. Docker Compose

El fichero de composición define los siguientes servicios:

- **zookeeper**: Incluye un `healthcheck` utilizando `nc` para asegurar que el puerto 2181 está disponible antes de iniciar las aplicaciones dependientes.
- **api**: La API REST desarrollada en la práctica anterior.

```
↳ Dockerfile > ...
1  FROM python:3.11-slim
2
3  # Python logs are printed in docker console
4  ENV PYTHONUNBUFFERED=1
5
6  WORKDIR /app
7
8  COPY requirements.txt .
9  RUN pip install -r requirements.txt
10
11 COPY src/ src/
12
13 ENTRYPOINT ["python", "src/app.py"]
14
```

Figure 8: Contenido de `Dockerfile`

- **app1, app2, app3:** Instancias de la aplicación. Se configuran mediante variables de entorno: `ZOOKEEPER_HOST`, `API_URL`, etc. y se les pasa su ID único como argumento del comando que la ejecutan, `command: ["1"]`.

Gracias a la red interna de Docker, las aplicaciones pueden comunicarse con la API utilizando el nombre del servicio `http://api:80/nuevo` en lugar de `localhost`.

5. Despliegue en Clúster de ZooKeeper

Para simular un entorno de producción real y garantizar alta disponibilidad, se ha desplegado ZooKeeper en modo clúster (ensemble) utilizando 3 nodos (`zookeeper1`, `zookeeper2`, `zookeeper3`).

Se ha adaptado el código de la aplicación para aceptar una cadena de conexión con múltiples hosts (variable de entorno `ZOOKEEPER_HOSTS`), permitiendo a la librería `kazoo` gestionar automáticamente la conexión y reconexión a cualquiera de los nodos disponibles del clúster.

5.1. Pruebas de Tolerancia a Fallos

Para verificar la robustez del sistema, se han realizado pruebas deteniendo nodos del clúster de ZooKeeper mientras la aplicación estaba en funcionamiento.

5.1.1. Estado Inicial

El sistema arranca correctamente con los 3 nodos de ZooKeeper y las 3 instancias de la aplicación. Se elige un líder y se reportan mediciones. Tal como se puede observar en Figure 9.

Desarrollo de Software Crítico

```
practica3-2 | Counter: 186
practica3-2 | [WATCHER] Connected devices: ['1', '2']
practica3-1 | Counter: 187
practica3-2 | [WATCHER] Connected devices: ['1', '2', '3']
practica3-3 | Counter: 188
practica3-2 | Media: 88.66666666666667
api-1 | 172.19.0.7 - - [19/Dec/2025 22:57:55] "GET /nuevo?dato=88.66666666666667 HTTP/1.1" 200 -
practica3-2 | I'm the leader: (id = 2)
practica3-3 | Counter: 189
practica3-3 | Counter: 190
practica3-1 | Counter: 191
practica3-2 | Counter: 192
practica3-2 | Media: 79.33333333333333
api-1 | 172.19.0.7 - - [19/Dec/2025 22:58:00] "GET /nuevo?dato=79.33333333333333 HTTP/1.1" 200 -
practica3-2 | I'm the leader: (id = 2)
practica3-1 | Counter: 193
practica3-3 | Counter: 194
practica3-2 | Counter: 195
practica3-3 | Counter: 196
practica3-2 | Media: 88.0

practica3-2 | I'm the leader: (id = 2)
practica3-3 | Counter: 197
practica3-3 | Counter: 198
practica3-1 | Counter: 199
practica3-2 | Counter: 200
```

Figure 9: Inicio normal del sistema

5.1.2. Prueba de tolerancia a fallos de infraestructura (zookeeper)

Se detiene el contenedor que actúa como líder del ensamble de ZooKeeper (verificable con `zkServer.sh status`), en este caso el líder es el servicio `zookeeper2`. Por tanto, una vez ejecutado el comando `docker stop zookeeper2` podemos observar cómo los servicios de nuestra aplicación lo detectan y se conectan de nuevo recuperandose exitosamente del fallo del sistema, tal como se puede observar en la siguiente Figure 10.

```
zookeeper1 | 2025-12-19 22:58:47,956 [myid:] - INFO [NIOWorkerThread-18:o.a.z.s.q.Learner@165] - Revalidating client: 0x30000e79c8a0000
practica3-2 | --- CONEXIÓN ESTABLECIDA --- Enable Watcher
api-1 | 172.19.0.7 - - [19/Dec/2025 22:58:47] "GET /nuevo?dato=83.33333333333333 HTTP/1.1" 200 -
zookeeper1 | 2025-12-19 22:58:47,972 [myid:] - WARN [QuorumPeer[myid=1]{plain=[0:0:0:0:0:0]:2181}(secure=disabled):o.a.z.s.q.Follower@167] - Got zxid 0x800000001 expected 0x1
practica3-2 | I'm the leader: (id = 2)
practica3-2 | Counter: 225
practica3-2 | Media: 83.33333333333333

api-1 | 172.19.0.7 - - [19/Dec/2025 22:58:52] "GET /nuevo?dato=83.33333333333333 HTTP/1.1" 200 -
practica3-2 | I'm the leader: (id = 2)
practica3-2 | Counter: 226

practica3-3 | Connection dropped: socket connection error: None
zookeeper1 | 2025-12-19 22:58:53,959 [myid:] - INFO [NIOWorkerThread-18:o.a.z.s.q.Learner@165] - Revalidating client: 0x30000e79c8a0001
practica3-3 | Counter: 227

zookeeper1 | 2025-12-19 22:58:54,024 [myid:] - INFO [NIOWorkerThread-2:o.a.z.s.q.Learner@165] - Revalidating client: 0x10000e79ca10000
practica3-1 | Connection dropped: socket connection error: None
practica3-1 | --- CONEXIÓN ESTABLECIDA ---
practica3-1 | Counter: 228
practica3-2 | Media: 81.0
api-1 | 172.19.0.7 - - [19/Dec/2025 22:58:58] "GET /nuevo?dato=81.0 HTTP/1.1" 200 -

practica3-2 | I'm the leader: (id = 2)
practica3-1 | Counter: 229
practica3-1 | Counter: 230
practica3-3 | Counter: 231
practica3-2 | Counter: 232
```

Figure 10: Inicio normal del sistema

6. Conclusiones

El desarrollo de esta práctica ha demostrado los desafíos que presenta la coordinación de sistemas distribuidos y cómo herramientas como Apache ZooKeeper facilitan esta tarea.

El sistema final es capaz de:

- Sincronizar procesos mediante barreras distribuidas.
- Gestionar configuraciones dinámicas en tiempo real sin reinicios (Watchers).
- Mantener la consistencia de datos compartidos (contadores).
- Tolerar fallos de infraestructura gracias al despliegue en clúster y al uso de librerías cliente (`kazoo`) que gestionan la reconexión transparente.