GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

APLICACIÓN DE ZENOH A LA AUTOMATIZACIÓN INDUSTRIAL

Autor: José Manuel Guinea Sotillo

Director: José Antonio Rodríguez Mondéjar

Madrid

Julio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Aplicación de Zenoh a la Automatización Industrial

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2024/25 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: José Manuel Guinea Sotillo Fecha: 20/ Julio/ 2025

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Antonio Rodríguez Mondéjar Fecha: ……/ ……/ ……

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

APLICACIÓN DE ZENOH A LA AUTOMATIZACIÓN INDUSTRIAL

Autor: José Manuel Guinea Sotillo

Director: José Antonio Rodríguez Mondéjar

Madrid

APLICACIÓN DE ZENOH A LA AUTOMATIZACIÓN INDUSTRIAL

Autor: Guinea Sotillo, José Manuel.
Director: Rodríguez Mondéjar, José Antonio.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

En este proyecto se ha puesto en marcha un sistema de climatización para un modelo de vivienda virtual mediante el uso del protocolo de comunicación Zenoh. El proyecto ha consistido en el desarrollo de dicho modelo de vivienda, la implementación de Zenoh y el análisis de los resultados obtenidos en cuanto a estado de las conexiones y velocidad de respuesta.

Palabras clave: Zenoh, Protocolo de Comunicación

1. Introducción

Actualmente, existen en el mercado un gran número de protocolos de comunicación. De entre ellos, Zenoh, un protocolo de tipo pub/sub/query, ofrece diversas ventajas, como pueden ser eficiencia a cualquier escala (de microcontroladores a centrales de datos), su baja latencia, y por su capacidad para gestionar datos tanto en movimiento como en reposo [1]. En los últimos años ha experimentado un gran crecimiento, principalmente en los campos de la automatización y la robótica, y, por tanto, se muestra como una opción a tener en cuenta frente a otros protocolos ya establecidos en el mercado como Profinet u OPC UA.

2. Definición del proyecto

El proyecto tiene como objetivo principal explorar, estudiar y analizar las capacidades y posibilidades del protocolo Zenoh. Se pretende poner a prueba sus propiedades, su simplicidad, su escalabilidad, su eficiencia y las posibilidades en cuanto a seguridad y cifrado. Para ello, se ha ideado un sistema de climatización de un modelo de vivienda virtual. En este sistema, todos los elementos miembros se encontrarán conectados mediante el protocolo Zenoh, para posteriormente poder analizar las conexiones y comunicaciones entre ellos.

3. Descripción del sistema

Se ha desarrollado en Python un sistema de climatización para una vivienda de 5 habitaciones. Dicho sistema cuenta con un sensor y un actuador por habitación, un proceso que simulará la temperatura en la vivienda, una consola desde la que el usuario podrá interactuar con el sistema, y un sniffer cuyo objetivo será registrar los datos que se intercambien todos los elementos del sistema. Para montar dicho sistema, se hará uso de la herramienta de software Docker [2]. A continuación, en la Ilustración 1 se puede observar un esquema del sistema desarrollado.

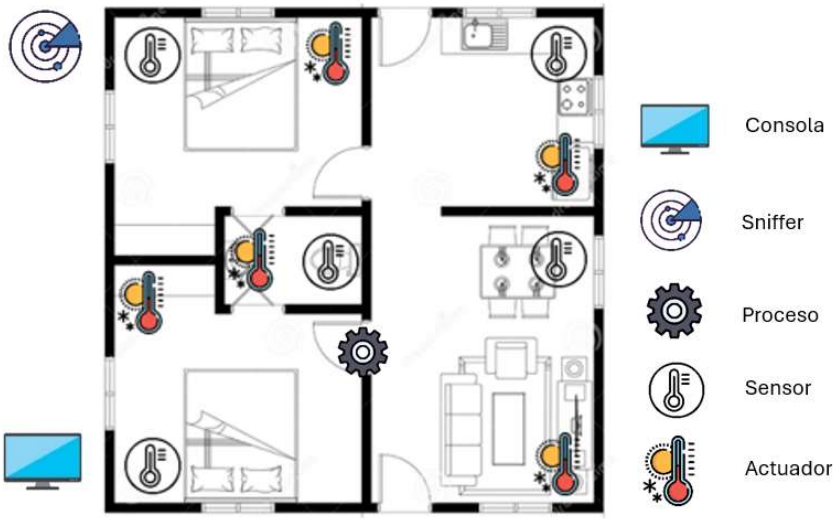


Ilustración 1. Esquema de la vivienda con los elementos del sistema.

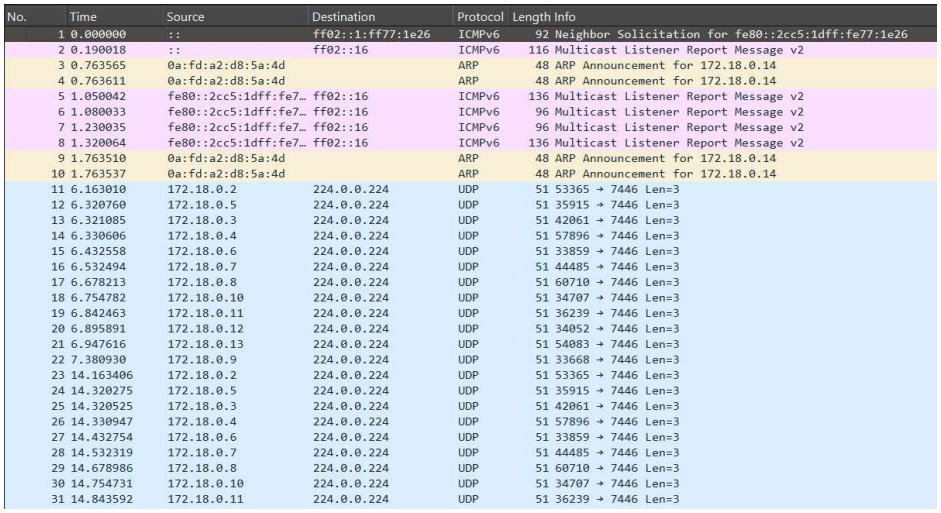
Para el correcto desarrollo del proyecto, este ha sido dividido en tres fases, en donde se abarcarán aspectos concretos del sistema.

- Fase 1: Sistema de Control.
  Se desarrollará el sistema básico sobre el que se construirá el resto del proyecto. En esta fase se creará la lógica de los sensores, actuadores, consola y proceso. Se montará la red de Docker con todos los componentes necesarios, y se fundamentarán los cimientos para poder seguir desarrollando el sistema. Se explorará la funcionalidad multicast de Zenoh (peer – to – peer).
- Fase 2: Seguridad.
  Sobre el modelo de lógica creado en la Fase 1, se añadirá una capa de seguridad y cifrado mediante TLS al sistema. Se explorarán las opciones que Zenoh ofrece en estos campos y se actualizará el sistema para poder implementarlas. Se explorará la funcionalidad unicast de Zenoh (cliente – servidor).
- Fase 3: Escalado.
  Se escalarán ambos modelos (de 5 a 50 habitaciones, progresivamente), con la intención de comprobar el estado del sistema al experimentar cambios en tamaño. Se aumentará el número de habitaciones para los dos modelos (multicast vs unicast), tratando de llevar al fallo al sistema para comprobar si un aumento de elementos provoca un bajón en la eficiencia y la velocidad del mismo.

4. Resultados

A la hora de analizar los resultados, se ha hecho uso de los programas Wireshark [3] y Tcpdump [4], además de la información recabada por el sniffer. Los resultados se agrupan en tres bloques distintos, en función de la fase analizada.

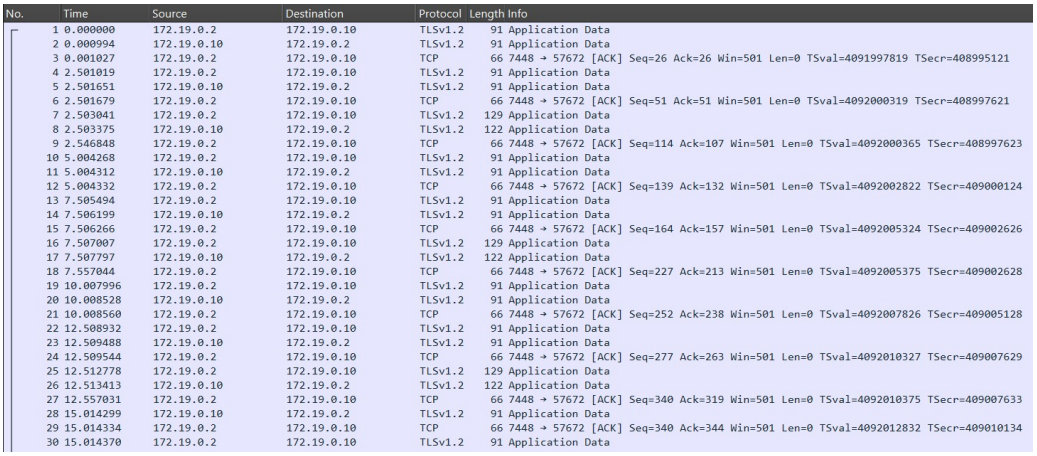
- Fase 1:
  Se analizan las comunicaciones entre los elementos del sistema, así como la respuesta del mismo. En cuanto a las comunicaciones, se observa como las mallas peer – to – peer realizan el descubrimiento por multicast, mientras que sus comunicaciones internas son por unicast (TCP), de acuerdo con el modelo (Ilustración 2). En cuanto a la respuesta del sistema, se encuentra en el rango óptimo de trabajo.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::	ff02::1:ff77:1e26	ICMPv6	92	Neighbor Solicitation for fe80::2cc5:1dff:fe77:1e26
2	0.190018	::	ff02::16	ICMPv6	116	Multicast Listener Report Message v2
3	0.763565	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
4	0.763611	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
5	1.050042	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
6	1.080033	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
7	1.230035	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
8	1.320064	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
9	1.763510	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
10	1.763537	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
11	6.163010	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
12	6.320760	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
13	6.321085	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
14	6.330606	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
15	6.432558	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
16	6.532494	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
17	6.678213	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
18	6.754782	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
19	6.842463	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3
20	6.895891	172.18.0.12	224.0.0.224	UDP	51	34052 → 7446 Len=3
21	6.947616	172.18.0.13	224.0.0.224	UDP	51	54083 → 7446 Len=3
22	7.380930	172.18.0.9	224.0.0.224	UDP	51	33668 → 7446 Len=3
23	14.163406	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
24	14.320275	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
25	14.320525	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
26	14.330947	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
27	14.432754	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
28	14.532319	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
29	14.678986	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
30	14.754731	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
31	14.843592	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3

Ilustración 2. Descubrimiento multicast de la Fase 1.

- Fase 2:
  Se analizan las comunicaciones entre elementos del sistema, así como la respuesta del mismo. En cuanto a las comunicaciones, se observa como ya no hay descubrimiento por multicast, y que la red cliente – servidor cuenta con todas sus comunicaciones cifradas vía TLS (Ilustración 3). En cuanto a la respuesta del sistema, esta se encuentra en el rango óptimo de trabajo.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
2	0.000994	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
3	0.001027	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=26 Ack=26 Win=501 Len=0 TSval=4091997819 TSecr=408995121
4	2.501019	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
5	2.501651	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
6	2.501679	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=51 Ack=51 Win=501 Len=0 TSval=4092000319 TSecr=408997621
7	2.503041	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
8	2.503375	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
9	2.546848	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=114 Ack=107 Win=501 Len=0 TSval=4092000365 TSecr=408997623
10	5.004268	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
11	5.004312	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
12	5.004332	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=139 Ack=132 Win=501 Len=0 TSval=4092002822 TSecr=409000124
13	7.505494	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
14	7.506199	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
15	7.506266	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=164 Ack=157 Win=501 Len=0 TSval=4092005324 TSecr=409002626
16	7.507007	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
17	7.507797	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
18	7.557044	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=227 Ack=213 Win=501 Len=0 TSval=4092005375 TSecr=409002628
19	10.007996	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
20	10.008528	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
21	10.008560	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=252 Ack=238 Win=501 Len=0 TSval=4092007826 TSecr=409005128
22	12.508932	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
23	12.509488	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
24	12.509544	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=277 Ack=263 Win=501 Len=0 TSval=4092010327 TSecr=409007629
25	12.512778	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
26	12.513413	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
27	12.557031	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=319 Win=501 Len=0 TSval=4092010375 TSecr=409007633
28	15.014299	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
29	15.014334	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=344 Win=501 Len=0 TSval=4092012832 TSecr=409010134
30	15.014370	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data

Ilustración 3. Comunicaciones de la Fase 2 entre dos entidades, cifradas con TLS.

- Fase 3:
  Se analiza la respuesta del sistema al escalado de ambos modelos. Se observa que el modelo de la fase 1 falla antes de llegar a las 25 habitaciones, puesto que la red Docker deja de funcionar. El modelo de la fase 2, por el contrario, llega a las 25 habitaciones, pero falla antes de llegar a las 50, también por colapso de la red Docker. No se observan bajones de rendimiento en ningún momento que indiquen un fallo de Zenoh.

5. Conclusiones

En cuanto al desarrollo del proyecto, se ha podido observar como el protocolo Zenoh cumple con las expectativas en lo referente a simplicidad y compatibilidad, dado de su implementación ha sido sencilla, y se ha podido usar junto a programas externos como Docker, Wireshark o FastAPI.

En cuanto a la eficiencia, ambos modelos han funcionado correctamente para tamaños pequeños. Sin embargo, al aumentar el tamaño, el fallo ha venido de la mano de Docker y no de Zenoh (puesto que la respuesta seguía dentro del rango óptimo antes del colapso de la red). Por tanto, al excederse las capacidades de la máquina virtual en donde se llevan a cabo las simulaciones, no se ha podido lograr observar el fallo real de Zenoh. Por tanto, hasta donde se sabe, el escalado por parte de Zenoh ha sido un éxito, pues la pérdida de rendimiento no ha llegado antes del colapso de la red.

6. Referencias

[1] Zenoh, «What is Zenoh?,» [En línea]. Available: https://zenoh.io/docs/overview/what-is-zenoh/. [Último acceso: 18 Julio 2025].

[2] Docker, «What is Docker?,» Docker, [En línea]. Available: https://docs.docker.com/get-started/docker-overview/. [Último acceso: 18 Julio 2025].

[3] Wireshark, «Wireshark's User Guide,» Wireshark, [En línea]. Available: https://www.wireshark.org/docs/wsug_html_chunked/. [Último acceso: 18 Julio 2025].

[4] M. Vergara Carmona, «Guía del comando tcpdump,» SYSADMIN, 3 Enero 2024. [En línea]. Available: https://vergaracarmona.es/guia-del-comando-tcpdump/. [Último acceso: 18 Julio 2025].

TÍTULO DEL TFG (EN INGLÉS)

Author: Guinea Sotillo, José Manuel.
Supervisor: Rodríguez Mondéjar, José Antonio.
Collaborating Entity: ICAI – Universidad Pontificia Comillas.

ABSTRACT

In this project, an air conditioning system was implemented for a virtual housing model using the Zenoh communication protocol. The project consists in developing the housing model, implementing Zenoh, and analyzing the results obtained in terms of connection status and response speed.

Keywords: Zenoh, Communication Protocol

1. Introduction

Nowadays, there are a large number of communication protocols available on the market. Among them, Zenoh, a pub/sub/query protocol, offers several advantages, such as efficiency at any scale (from microcontrollers to data centers), low latency, and its ability to manage data both in motion and at rest [1]. It has experienced significant growth in recent years, primarily in the fields of automation and robotics, and therefore it is an interesting option compared to other established protocols on the market, such as Profinet or OPC UA.

2. Project Definition

The project's main objective is to explore, study, and analyze all the capabilities and possibilities of the Zenoh protocol. The goal is to test its simplicity, scalability, efficiency, and security and encryption features. In order to do so, a climate control system for a virtual home model has been designed. In this system, all member elements will be connected using the Zenoh protocol, allowing for subsequent analysis of the connections and communications between them.

3. System Description

For this project, an air conditioning system for a 5-bedroom home has been developed in Python. This system includes one sensor and one actuator per room, one process that simulates the temperature in the home, one console from which the user can interact with the system, and one sniffer to record the data exchanged between all the elements of the system. To assemble this system, the Docker software tool will be used [2]. A diagram of the developed system can be seen in Figure 1 below.

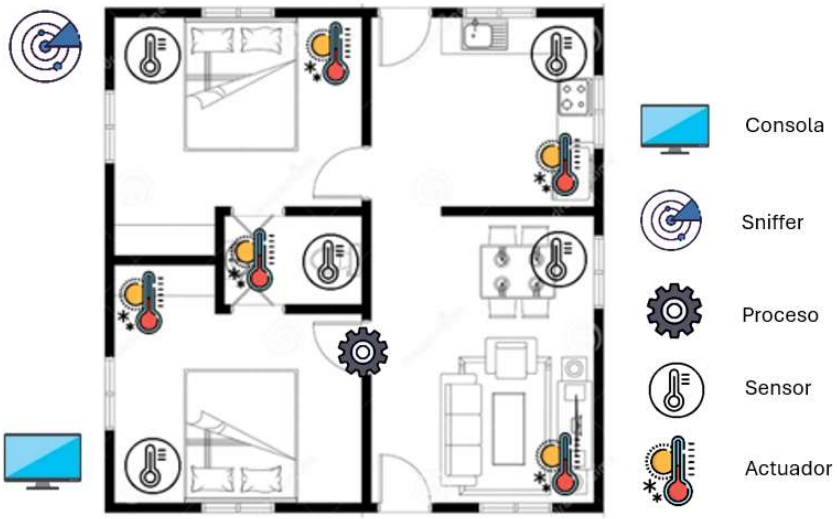


Figure 1. System elements over the house diagram.

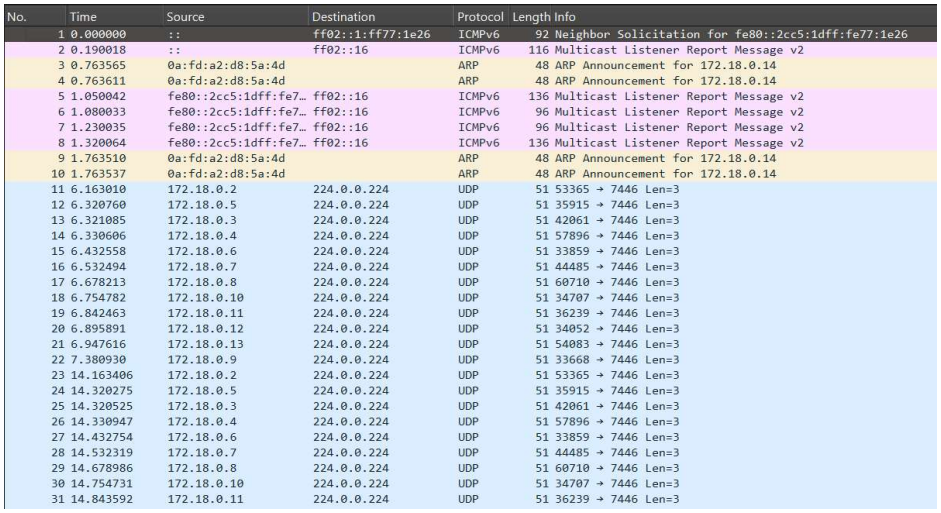
For the proper development of the project, it has been divided into three phases, each of which will cover specific aspects of the system.

- Phase 1: Control System.
  The basic system on which the rest of the project will be built will be developed. In this phase, the logic for the sensors, actuators, the console, and the process will be created. The Docker network will be set up with all the necessary components, and the foundation will be laid for further system development. Zenoh's multicast (peer-to-peer) feature will be explored.
- Phase 2: Security.
  Based on the logic model created in Phase 1, a security layer and TLS encryption will be added to the system. The options Zenoh offers in these areas will be explored, and the system will be updated to implement them. Zenoh's unicast (client-server) feature will be explored.
- Phase 3: Scaling.
  Both models will be scaled (from 5 to 50 rooms, progressively), with the intention of testing the system's health as it undergoes changes in size. The number of rooms will be increased for both models (multicast vs. unicast), attempting to force the system to fail to see if an increase in the number of elements causes a decrease in efficiency and speed.

4. Outcomes

When analyzing the results, the programs Wireshark [3] and Tcpdump [4] were used, in addition to the information collected by the sniffer. The results are grouped into three different blocks, depending on the phase analyzed.

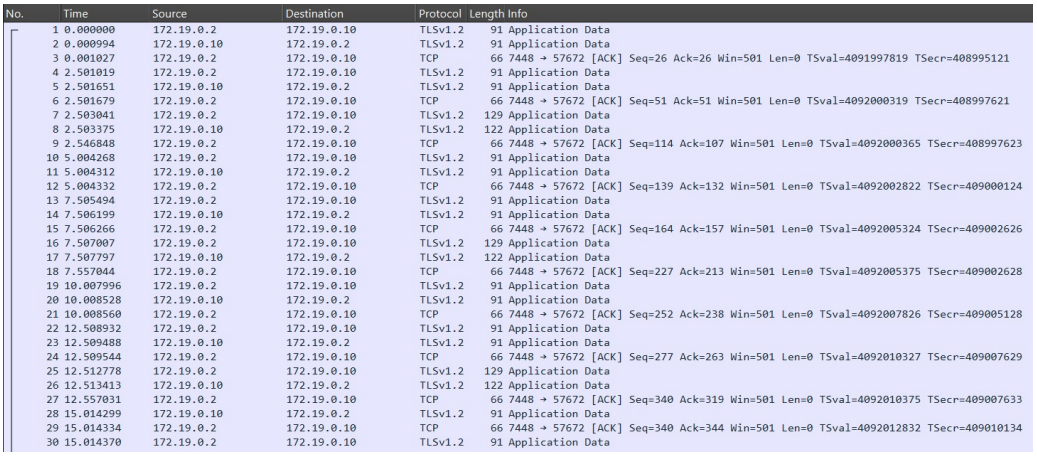
- Phase 1:
  Communications between system elements, as well as the system's response, are analyzed. Regarding communications, peer-to-peer meshes perform multicast discovery, while their internal communications are unicast (TCP), in accordance with the model (Figure 2). The system's response is within the optimal operating range.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::	ff02::1:ff77:1e26	ICMPv6	92	Neighbor Solicitation for fe80::2cc5:1dff:fe77:1e26
2	0.190018	::	ff02::16	ICMPv6	116	Multicast Listener Report Message v2
3	0.763565	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
4	0.763611	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
5	1.050042	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
6	1.080033	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
7	1.230035	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
8	1.320064	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
9	1.763510	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
10	1.763537	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
11	6.163010	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
12	6.320760	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
13	6.321085	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
14	6.330606	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
15	6.432558	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
16	6.532494	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
17	6.678213	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
18	6.754782	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
19	6.842463	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3
20	6.895891	172.18.0.12	224.0.0.224	UDP	51	34052 → 7446 Len=3
21	6.947616	172.18.0.13	224.0.0.224	UDP	51	54083 → 7446 Len=3
22	7.380930	172.18.0.9	224.0.0.224	UDP	51	33668 → 7446 Len=3
23	14.163406	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
24	14.320275	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
25	14.320525	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
26	14.330947	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
27	14.432754	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
28	14.532319	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
29	14.678986	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
30	14.754731	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
31	14.843592	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3

Figure 2. Multicast scouting on Fase 1.

- Phase 2:
  Communications between system elements, as well as the system's response, are analyzed. Regarding communications, it is observed that multicast discovery is no longer present, and that all client-server communications are encrypted via TLS (Figure 3). The system's response is within the optimal operating range.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
2	0.000994	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
3	0.001027	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=26 Ack=26 Win=501 Len=0 TSval=4091997819 TSecr=408995121
4	2.501019	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
5	2.501651	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
6	2.501679	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=51 Ack=51 Win=501 Len=0 TSval=4092000319 TSecr=408997621
7	2.503041	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
8	2.503375	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
9	2.546848	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=114 Ack=107 Win=501 Len=0 TSval=4092000365 TSecr=408997623
10	5.004268	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
11	5.004312	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
12	5.004332	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=139 Ack=132 Win=501 Len=0 TSval=4092002822 TSecr=409000124
13	7.505494	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
14	7.506199	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
15	7.506266	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=164 Ack=157 Win=501 Len=0 TSval=4092005324 TSecr=409002626
16	7.507007	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
17	7.507797	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
18	7.557044	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=227 Ack=213 Win=501 Len=0 TSval=4092005375 TSecr=409002628
19	10.007996	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
20	10.008528	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
21	10.008560	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=252 Ack=238 Win=501 Len=0 TSval=4092007826 TSecr=409005128
22	12.508932	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
23	12.509488	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
24	12.509544	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=277 Ack=263 Win=501 Len=0 TSval=4092010327 TSecr=409007629
25	12.512778	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
26	12.513413	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
27	12.557031	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=319 Win=501 Len=0 TSval=4092010375 TSecr=409007633
28	15.014299	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
29	15.014334	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=344 Win=501 Len=0 TSval=4092012832 TSecr=409010134
30	15.014370	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data

Figure 3. Communications between two elements on Fase 2, secured by TLS.

- Phase 3:
  The system's response to scaling is analyzed for both models. It is observed that the phase 1 model fails before reaching 25 rooms, as the Docker network stops working. The phase 2 model, on the other hand, reaches 25 rooms but fails before reaching 50, also due to a Docker network crash. No performance drops that would indicate a Zenoh failure are observed at any point.

5. Conclusion

Regarding the project development, it has been observed that the Zenoh protocol meets its simplicity and compatibility expectations, given its simple implementation and the ability to be used alongside external programs such as Docker, Wireshark, or FastAPI.

Regarding efficiency, both models worked correctly for small sizes. However, as the size increased, the failure occurred in Docker and not in Zenoh (since the response was still within the optimal range before the network collapse). Therefore, by exceeding the capacities of the virtual machine where the simulations were carried out, it was not possible to observe the actual failure of Zenoh. To conclude, as far as is known, Zenoh's scaling has been successful, as there was no performance loss did before the network collapse.

6. Referencias

[1] Zenoh, «What is Zenoh?,» [En línea]. Available: https://zenoh.io/docs/overview/what-is-zenoh/. [Último acceso: 18 Julio 2025].

[2] Docker, «What is Docker?,» Docker, [En línea]. Available: https://docs.docker.com/get-started/docker-overview/. [Último acceso: 18 Julio 2025].

[3] Wireshark, «Wireshark's User Guide,» Wireshark, [En línea]. Available: https://www.wireshark.org/docs/wsug_html_chunked/. [Último acceso: 18 Julio 2025].

[4] M. Vergara Carmona, «Guía del comando tcpdump,» SYSADMIN, 3 Enero 2024. [En línea]. Available: https://vergaracarmona.es/guia-del-comando-tcpdump/. [Último acceso: 18 Julio 2025].

Índice de la memoria

Índice de figuras

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En el año 2016, el economista Klaus Schwab, fundador del Fondo Económico Mundial, acuñó en su libro "La Cuarta Revolución Industrial" el término Industria 4.0. Este término hace referencia a la "fabricación informatizada, que combina avanzadas técnicas de producción con tecnologías inteligentes que se integrarán en las organizaciones y en la vida de las personas" [5]. Y es que, en los últimos siglos, y a lo largo de tres revoluciones industriales, se ha vivido una revolución en el mundo de la industria que ha llevado al ser humano de las máquinas manuales en pequeños talleres artesanales a las gigantescas fábricas actuales, que permiten la producción a gran escala de bienes y productos. Sin embargo, esta cuarta revolución se plantea desde un nuevo punto de vista: la posibilidad de aplicar las nuevas tecnologías de inteligencia artificial, machine learning y sistemas de comunicación avanzados al mundo industrial. Sobre estos últimos recae una gran responsabilidad, dado que con los avances tecnológicos que llegan prácticamente de forma diaria a las fábricas a lo largo del globo, cada vez es más necesario para las empresas el poseer métodos de control y automatización de todos sus dispositivos. Es aquí donde entran en juego los protocolos de comunicación, los cuales permiten que los distintos dispositivos que componen una cadena de producción (motores, sensores, robots, ordenadores, sistemas de almacenamiento de datos…) se encuentren interconectados y puedan ser todos controlados de forma centralizada, en un intento de aumentar la eficiencia de dicha cadena. Es aquí donde entra en juego el protocolo ZENOH, un protocolo pub/sub/query que unifica los datos en reposo y en movimiento. Zenoh plantea una solución a los problemas que se pueden presentar en algunos procesos industriales que carecen de la capacidad de banda ancha necesaria para poder tener un protocolo de comunicación y automatización capaz de cumplir con todos los requisitos y cubrir todas las necesidades de la industria en cuestión. Su capacidad para ser eficiente a cualquier escala (opera tanto a nivel de microcontroladores como a nivel de centros de datos) es, sin duda, una cualidad interesante y muy a tener en cuenta en los próximos años. Este proyecto tratará de ahondar en esta y otras características del protocolo Zenoh, con el fin de determinar su verdadero potencial y su viabilidad en la industria actual.

Capítulo 2. DESCRIPCIÓN DE LAS TECNOLOGÍAS

Este capítulo se encuentra dividido en dos partes: introducción de conceptos de comunicaciones y explicación del protocolo Zenoh.

2.1 INTRODUCCIÓN A CONCEPTOS DE COMUNICACIONES:

Antes de comenzar con la explicación del protocolo, se ha considerado relevante hacer una breve explicación a modo de introducción de ciertos conceptos de comunicaciones que serán mencionados de forma recurrente en el documento. Dichos conceptos son, en primer lugar, el modelo de referencia de comunicaciones OSI, y, en segundo lugar, los métodos de transmisión de datos unicast y multicast. Sin ser conceptos excesivamente complejos, el hecho de ser mencionados con recurrencia a lo largo de este trabajo requiere que sean explicados de forma adecuada, con el fin de evitar dudas sobre los mismos.

2.1.1 MODELO OSI

El Open Systems Interconnection Model, o modelo OSI, es un modelo conceptual de referencia para el establecimiento de conexiones abiertas establecido por la Organización Internacional de Normalización (ISO) en la década de 1980 [6]. Este modelo divide la conexión de redes en 7 capas distintas, cada una con una función específica, apiladas cada una sobre la anterior. La intención con la creación de este modelo es la simplificación de la red para poder facilitar tanto la conexión entre elementos de la misma, como la identificación de errores en los sistemas (si se puede identificar la capa del error, es más sencillo solventarlo). Las capas (Ilustración 4) son las siguientes [7]:

- Capa 7 de Aplicación: Es la capa de interacción con el usuario, y es responsable de los protocolos y la manipulación de datos. En esta capa se incluyen, por ejemplo, protocolos como HTTP.

- Capa 6 de Presentación: Esta capa está encargada de la preparación de los datos para que puedan ser utilizados por la capa de aplicación. Hace labores de cifrado, decodificación y traducción de los datos.
- Capa 5 de Sesión: Esta capa es la encargada tanto del inicio como del cierre de las comunicaciones, lo que se conoce como "sesión". En dicha capa se sincroniza también la transferencia de datos mediante puntos de control.
- Capa 4 de Transporte: En esta capa es en la que reside la responsabilidad de las comunicaciones entre dispositivos. Los datos son fragmentados y enviados al receptor, que deberá rearmarlos para poder consumirlos. Se encarga también del control tanto de errores como de flujo.
- Capa 3 de Red: El objetivo de esta capa es facilitar la transferencia de datos entre dos redes diferentes, encargándose de encontrar la ruta óptima entre ellas (enrutamiento).
- Capa 2 de Enlace de Datos: Es similar a la capa 3, pero para elementos que se encuentran dentro de la misma red.
- Capa 1 Física: Es la capa que contiene el equipo físico implicado en la transferencia de datos (cables, conmutadores…). En esta capa, los datos se convierten en cadenas de bits.

Ilustración 4. Capas del modelo OSI [8]

Un ejemplo del modelo OSI aplicado a una comunicación, sería, por ejemplo, una conversación entre A y B. A quiere enviarle un mensaje a B, y, por tanto, hace uso de la capa 7 para redactarlo y enviarlo. El mensaje llega a la capa de presentación, donde se determina un protocolo para él y se envía a la capa de sesión para que se inicie la comunicación. En la capa de transporte, el mensaje es segmentado y enviado a la capa 3 o la capa 2, dependiendo de si A y B comparten o no red. Finalmente, este mensaje segmentado acabará llegando al soporte físico de B como una secuencia de bits, para repetir este mismo proceso, pero en sentido inverso, de forma que B pueda, finalmente, leer el mensaje (Ilustración 5).

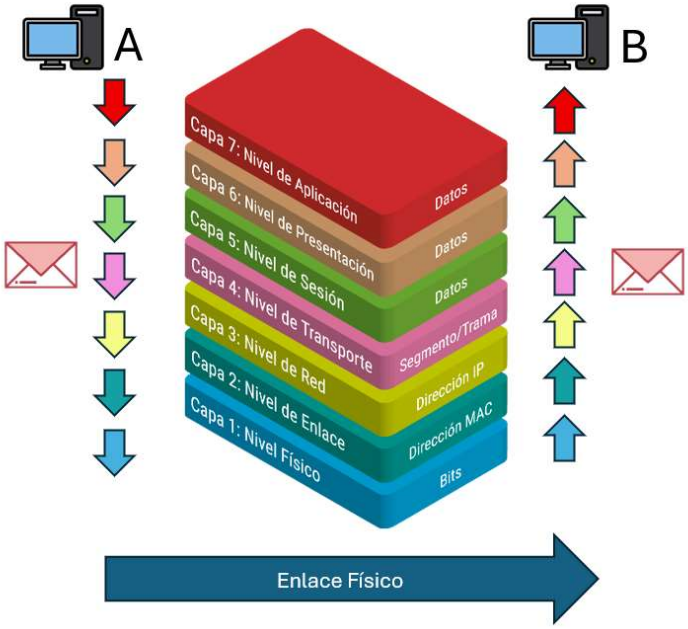


Ilustración 5. Comunicación entre A y B según el modelo OSI.

Dentro de la capa 3 del modelo OSI es donde se encuentran los modelos de transmisión de datos de tráfico IP. Dependiendo de entre cuantos miembros estén enfocadas las comunicaciones, se pueden clasificar en tres grupos, como se muestra en el esquema de la Ilustración 6.

- Unicast: Conexiones uno a uno.
- Multicast: Conexiones uno a muchos.

- Broadcast: Conexiones uno a todos.

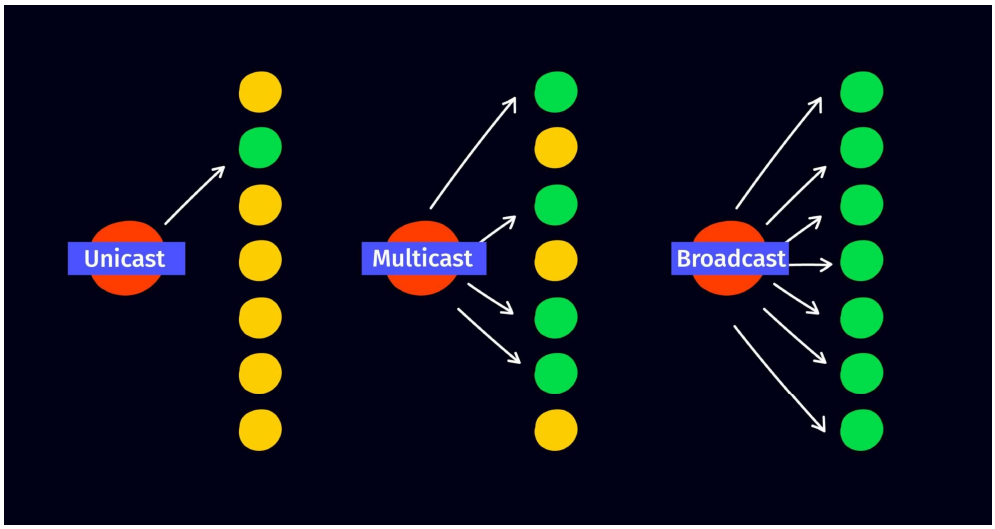


Ilustración 6. Esquema de modelos unicast, multicast y broadcast [9].

2.1.2 MODELO UNICAST

El modelo de tráfico de IP unicast consiste en una transmisión de información en el que el envío del paquete de datos se produce de un único emisor a un único receptor (comunicaciones uno – uno), como se muestra en la Ilustración 7. Es el modelo de transmisión con mayor uso y recorrido en el panorama comunicativo actual, utilizado desde correos electrónicos hasta acceso a páginas web [10].

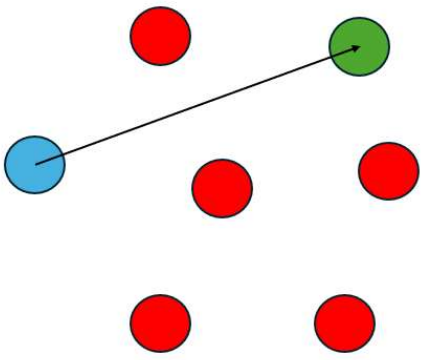


Ilustración 7. Representación del modelo unicast.

Las ventajas que aporta este modelo de transmisión de datos son, en primer lugar, la eficiencia de ancho de banda para conexiones entre dos puertos concretos, ya que solo se envía una transmisión recibida por un único destinatario. Adicionalmente, en términos de seguridad, las transmisiones unicast tienden a ser mucho más seguras que las transmisiones por otro tipo de modelos debido a que los datos no son compartidos con terceros elementos que puedan estar implicados de manera oculta, puesto que la dirección es única. Además, cuenta con un repertorio considerablemente alto de métodos de cifrado y ciberseguridad, como pueden ser SSL o TLS. Por último, la fiabilidad de la transmisión unicast es mayor que la de otros modelos, puesto que los protocolos que emplea tienen métodos de comprobación tanto de entrega como de orden en los datos. Por tanto, son más certeros a la hora de transmitir información [10].

Por otro lado, es un método que plantea desventajas, principalmente a la hora de construir grandes redes en las que se vean implicados un gran número de elementos. Cuando hay muchos elementos receptores, la red puede verse saturada debido a que cada receptor requiere de una conexión propia para poder entablar comunicaciones con el resto de las entidades, lo que, a su vez, implica un mayor gasto de recursos tanto en mantenimiento como en operación de la red [10].

El protocolo de comunicación más usado con métodos unicast es el Transmission Control Protocol (TCP). Este protocolo, que opera en la capa de transporte del modelo OSI (capa 4), tiene como objetivo principal garantizar la entrega del mensaje a través de la red a través del mantenimiento de la conexión entre emisor y receptor [11]. La forma en la que opera es la siguiente (Ilustración 8): en primer lugar, se establece la conexión entre fuente y destino, para posteriormente enviar la información compartimentada en paquetes de datos más pequeños, que son enviados de manera ordenada. El hecho del mantenimiento de la conexión, junto con los métodos de certificación de que se ha entregado el mensaje, hacen de TCP un protocolo muy fiable, pero a su vez costoso, puesto que se han de invertir un mayor número de recursos en el para que funcione de manera adecuada.

Ilustración 8. Funcionamiento del protocolo TCP [12].

2.1.3 MODELO MULTICAST

El tráfico IP multicast, también conocido como multidifusión IP, es un método de comunicación entre dispositivos, el cual permite el envío de forma simultánea de información a un grupo de receptores, los cuales son denominados clientes, que están configurados de manera específica para ser capaces de recibir dicho tráfico de red (Ilustración 9). El punto fundamental del tráfico multicast es que este no requiere de replicar los paquetes de datos que serán enviados a cada receptor, sino que se hace un único envío, el cual es recibido por todos los clientes que se encuentren dentro del grupo receptor. Este método de comunicación permite la transmisión efectiva de datos tanto en sistemas de uno a muchos como muchos a muchos (el emisor puede ser un único elemento o varios) [13].

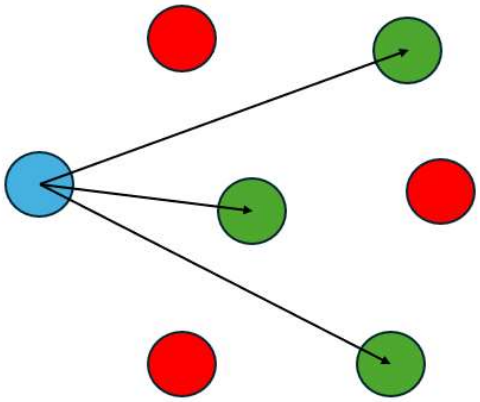


Ilustración 9. Representación del modelo multicast.

Una de las mayores ventajas que tiene este método de transmisión de datos frente a los otros métodos citados es la eficiencia en lo referido al ancho de banda. Al enviarse una única transmisión de datos, en lugar de una copia de los mismos a cada receptor, la carga de la red se ve considerablemente reducida. Adicionalmente a la eficiencia del ancho de banda, también aumenta la eficiencia de la administración de la red, debido a que una única transmisión simplifica la gestión y mantenimiento de esta. Y de la misma forma que la eficiencia es relevante, lo es también la velocidad, pues el hecho de enviar los paquetes de información de manera simultánea, el tiempo entre comunicación emisor – receptor se ve notablemente disminuido. Asimismo, el tráfico multicast presenta una gran facilidad de cara a la escalabilidad de los sistemas, debido a que el coste y complejidad de la transmisión de la información no aumenta de manera proporcional con el aumento del número de receptores [13].

Sin embargo, también presenta también ciertas desventajas, como pueden ser la falta de mecanismos para la corrección de errores o la falta de factores de autenticación de peso. Sobre la primera, en redes inestables o cuya tasa de pérdida de paquetes de datos sea excesivamente alta, la transmisión multicast puede perder fiabilidad, ya que dicha pérdida puede afectar de forma simultánea a varios clientes. Sobre la segunda, la naturaleza del método hace que los sistemas de autenticación y seguridad sean más complicados de

implementar. Adicionalmente, la realidad del panorama actual de comunicaciones es que muchos elementos de software y redes antiguos tienen problemas para soportar el tráfico multicast. Por tanto, no suele ser el principal método de comunicación empleado [13].

Los ejemplos de uso de mayor relevancia para el tráfico multicast son, por ejemplo, las transmisiones en directo. En España, este método de comunicación es ampliamente utilizado por compañías de televisión de pago y operadores de fibra óptica, como puede ser el caso de Movistar. En este caso, los clientes receptores interesados serían aquellos que contasen con su decodificador instalado. También cobra peso a la hora de desarrollar videojuegos con modos de jugabilidad "en línea", en los cuales diversos jugadores (clientes) requieren de recibir datos sobre el propio juego de forma simultánea y en directo.

El protocolo de comunicación más usado con del tráfico multicast, que es a su vez el protocolo que aplica Zenoh, es el User Datagram Protocol, más conocido por sus siglas UDP. UDP (Ilustración 10) es uno de los protocolos principales dentro de la cuarta capa del modelo OSI la capa de transporte, y su característica principal es que no está orientado a establecer o mantener una conexión entre el emisor y el receptor a la hora de realizar una transmisión de información [14]. Esta información, agrupada en paquetes denominados "datagramas" es enviada de forma directa al destino, sin establecer una primera conexión, sin establecer el orden de los paquetes (esto puede hacer que lleguen desordenados) y sin comprobar que los paquetes han sido recibidos en el destino (esto puede hacer que los paquetes se pierdan y no se puedan recuperar).

Ilustración 10. Funcionamiento del protocolo UDP [14].

La ausencia del establecimiento previo de la conexión permite que el envío de los datagramas sea más rápido, y es por ello por lo que es el principal protocolo de comunicación a escoger en proyectos en los cuales la velocidad de transmisión tenga mayor peso que la exactitud en la entrega de paquetes. Dicho de otra forma, cuando en un sistema la tolerancia a los errores es alta y es necesario que la información sea intercambiada de la forma más veloz posible, UDP suele ser el protocolo escogido.

2.2 ZENOH

Un protocolo de comunicación es, de acuerdo con la empresa de telecomunicaciones Lowi, un sistema de reglas que permite la comunicación entre diferentes dispositivos que se conectan entre sí para intercambiar datos [15]. Por tanto, una descripción clara, concisa y certera de Zenoh, de acuerdo con su página web, es que es un protocolo de comunicación pub/sub/query que unifica datos en movimiento, en reposo y computación distribuida, con el enfoque principalmente en la eficiencia y la baja latencia [1]. Dentro del modelo OSI, Zenoh actuaría entre las capas 5 y 7 (véase Ilustración 11).

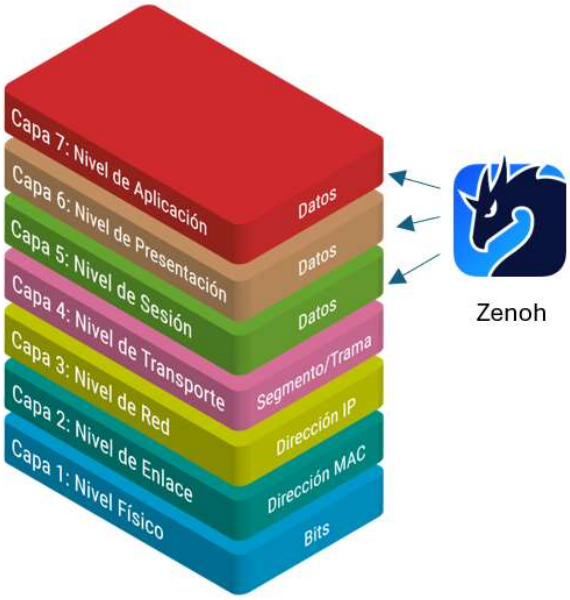


Ilustración 11. Zenoh en el modelo OSI.

Está diseñado para trabajar tanto a grandes niveles como centros de datos, como a escala reducida en microcontroladores, y proporciona transparencia de localización en los datos, siendo la primera tecnología capaz de hacer esto para datos en reposo. De esta manera, el usuario no tiene que preocuparse por la ubicación física de los datos para poder operar con ellos. El protocolo está pensado para ser fácil de usar, y a su vez tener un alto rendimiento a lo largo de todo amplio rango de aplicación pese a ser energéticamente eficiente.

2.2.1 PROTOCOLO PUB/SUB/QUERY

Un protocolo pub/sub, cuyo nombre viene de los términos en inglés “Publisher” (publicador) y “Subscriber” (suscriptor), es un servicio de mensajería asíncrona que separa los servicios entre aquellos que emiten mensajes (pubs) y aquellos que reciben mensajes (subs) [16]. Es asíncrono porque el momento en el que se publica la información y en la que se recibe es independiente: un publicador puede enviar la información cuando la tenga lista, y, el suscriptor, recibirla en el momento en que esté preparado (Ilustración 12). Para ello, se emplean expresiones de clave o tópicos, en los cuales las entidades publican y reciben la información. Por otro lado, query (del inglés “consulta”) está pensado para soportar las

consultas sobre los datos en reposo y el registro de funciones. Esta parte del protocolo es la que permite a las aplicaciones Zenoh la utilización de funciones como "GET" o "POST", que serán empleadas en el proyecto (véase el apartado 5.1.4).

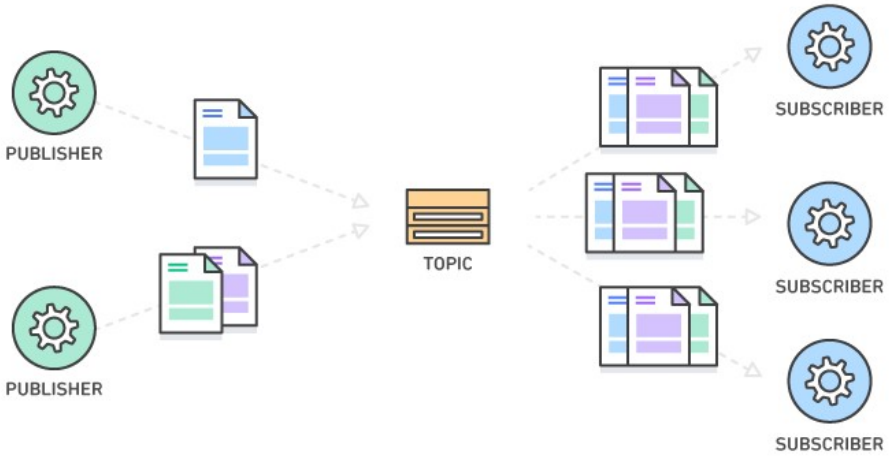


Ilustración 12. Representación de protocolos pub/sub [17].

2.2.2 MODELOS DE COMUNICACIÓN

Zenoh proporciona dos modelos distintos de comunicación: comunicación cliente – router y comunicación peer – to – peer [18].

2.2.2.1 Cliente – router

El modelo cliente – router, o cliente – servidor, es un modelo de comunicaciones que permite la distribución de tareas dentro de una red de comunicación [19]. Esta distribución es fundamental y en lo que se basa, principalmente, el modelo. En este caso, hay dos agentes con papeles distintos: un cliente encargado de realizar peticiones, y un servidor encargado de procesarlas y responderlas (Ilustración 13). Cabe destacar que las redes cliente – servidor suelen contar con un servidor para varios clientes, a los que tiene la labor de administrar sus peticiones.

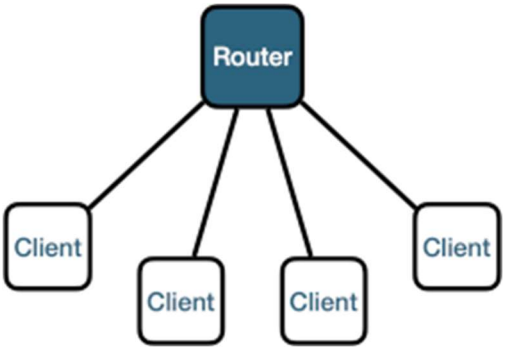


Ilustración 13. Conexión cliente - servidor [18].

Las ventajas de este modelo son las siguientes [19]:

- Centralización: al contar con un servidor central, todos los recursos pasan por ahí, y, por tanto, la administración y el mantenimiento de la red resulta más sencillo.
- Gestión de derechos de acceso: el hecho de contar con un almacenamiento central facilita la identificación de quién tiene acceso a qué cosas.
- Recursos independientes: un servidor puede dar soporte a un gran número de clientes, y estos, mientras estén conectados, no tienen por qué estar localizados cerca. Por tanto, los recursos no tienen por qué estar todos destinados en el mismo sitio.

Por otro lado, este modelo también plantea desventajas, como [19]:

- Gasto en recursos: si bien todos los recursos van principalmente al servidor, este puede ser caro de mantener, y si ve sus recursos limitados, puede tener repercusiones sobre los clientes.
- Dependencia: debido a la centralización del sistema, en caso de caída del servidor central, cae con el todo el sistema. Por tanto, es vital el mantenimiento concienzudo del mismo.
- Tiempo: la configuración de este tipo de sistemas suele requerir una gran cantidad de tiempo hasta que terminan estando operativos.

En el caso de Zenoh, es el router el que hace de servidor, permitiendo a los clientes conectarse.

2.2.2.2 Peer – to – peer

Una red P2P (peer – to – peer), es una red en la que todos los elementos tienen los mismos privilegios y las mismas funciones [20]. A diferencia de la comunicación cliente – servidor, las entidades pueden realizar peticiones como un cliente, y responder a ellas como un servidor. Además, otra diferencia respecto a las redes cliente – servidor es que se encuentran descentralizadas, puesto que no necesitan un servidor principal (Ilustración 14). Cuando se realiza una petición en P2P, no se comunica con un servidor central, si no con el resto de los miembros de la malla (grupo de peers), los cuales proporcionaran los datos solicitados en tanto que los posean.

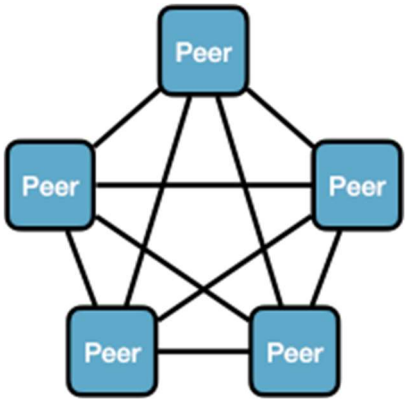


Ilustración 14. Conexión peer - to - peer [18].

Las ventajas de dicho modelo son las siguientes [20]:

- Escalabilidad: la capacidad de una red de este estilo es, sobre la teoría, ilimitada. Adicionalmente, con cada nuevo miembro de la red, esta aumenta su rendimiento, puesto que dicho miembro está aportando a la misma sus capacidades y recursos.
- Seguridad: si no hay un servidor central, este no puede ser atacado, y, por tanto, los ataques maliciosos quedan reducidos a entidades concretas, que pueden ser fácilmente eliminadas de la red.
- Flexibilidad: dado que todas las entidades tienen la capacidad de actuar como cliente o como servidor, las tareas pueden ser divididas en función de las capacidades de cada una de las entidades.

Sin embargo, también presenta inconvenientes, como pueden ser [20]:

- Gestión complicada: al no existir una entidad central, todos los cambios y mantenimiento deben llevarse a cabo miembro a miembro, dificultando la tarea e incrementando el tiempo de la misma.
- Dependencia: si bien es un modelo menos dependiente que el cliente – servidor, realizar cambios en uno de los miembros del sistema puede conllevar cambios en todos los miembros de este, puesto que están todos conectados.
- Legalidad: dentro de una red P2P, donde no hay un servidor central desde el que se gestione todo, las persecuciones judiciales se vuelven mucho más complejas.

Este es el modo de configuración predeterminado de Zenoh.

2.2.2.3 Modelo mixto

Adicionalmente, Zenoh permite la combinación de ambos modelos de comunicación con el fin de construir grandes redes de comunicaciones [18]. Si bien de forma predeterminada no se conectarán, esta conexión puede ser forzada mediante código, de forma que cada peer actúe como un peer dentro de su malla, y como un cliente para el router, como se puede observar en la Ilustración 15.

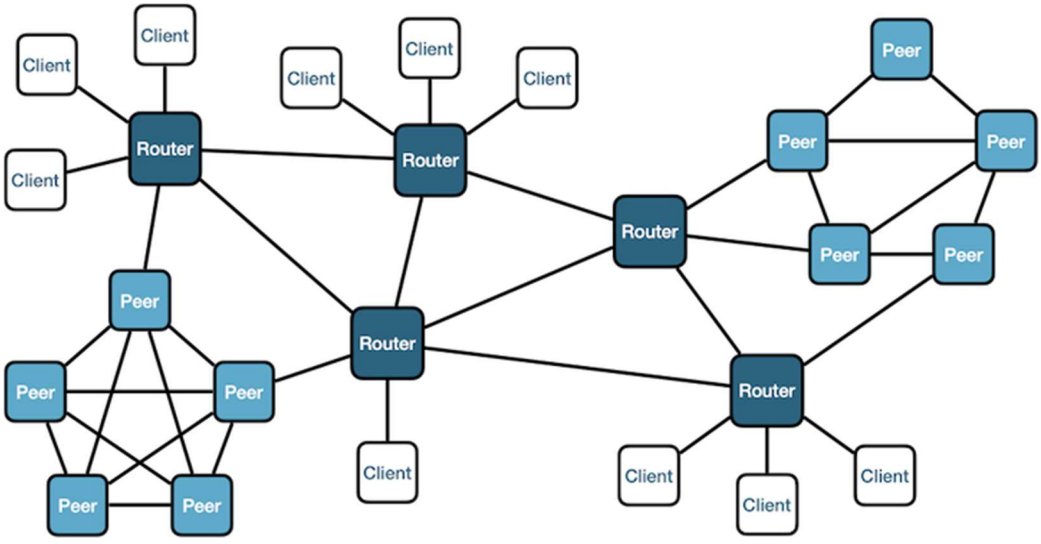


Ilustración 15. Modelo mixto P2P y cliente - servidor [18].

2.2.3 SCOUTING

El scouting en Zenoh es la forma en la que, tanto en modo peer – to – peer, como en modo cliente – router, las entidades del sistema reconocen la presencia de otras entidades dispuestas a establecer una comunicación. Para ello, se emplea el método de descubrimiento por multicast. De forma predeterminada, este descubrimiento se realiza en la dirección 224.0.0.224, y, mediante el protocolo UDP, se envían mensajes de rastreo ("scout"), para descubrir a otras entidades. Las entidades que se revelan en esta dirección entonces pueden decidir si establecer una conexión o no, y, en función de lo que se determine, comenzará la transferencia de datos (Ilustración 16).

Ilustración 16. Proceso de scouting simplificado.

Una vez se establecen estas conexiones, el resto de las comunicaciones se producen de forma unicast. Es por ello que se puede determinar que Zenoh cuenta con un modelo de comunicación multicast, que está soportado por unicast. Y si bien esto es un añadido al modelo peer – to – peer, es una exigencia del modelo cliente – servidor: las conexiones deben ser únicamente unicast. Por tanto, en el modelo cliente – servidor, el scouting se produce a nivel de código, indicándose los endpoints en donde se deben establecer las comunicaciones, como se puede observar en el siguiente fragmento:

```
{
  mode: client,
  connect: {
    endpoints: ["tcp/192.168.1.1:7447", "tcp/192.168.1.2:7447"],
  },
}
```

2.2.4 SEGURIDAD

Uno de los factores a tener en cuenta a la hora de aplicar Zenoh es la cantidad de capas de seguridad que se le puede aplicar. Al igual que en el resto de sus características, la forma en la que está enfocada la seguridad en Zenoh es modular: la seguridad se puede añadir en forma de capas diferentes y apilables (Ilustración 17). El modelo de seguridad de Zenoh cubre autenticación, autorización y cifrado.

- Autenticación: Zenoh permite la autenticación de los elementos mediante certificados, tokens o identidades firmadas. Admite también la identificación por usuario y contraseña (véase sección 5.2.1).
- Autorización: La autorización en Zenoh tiene relación con los datos. Se puede aplicar un control todos los aspectos de los mismos, como control sobre publicaciones y suscripciones, control sobre las consultas, o control sobre el router y quién puede actuar como tal.
- Cifrado: Zenoh puede implementar métodos de cifrado de extremo a extremo, como pueden ser QUIC o TLS (véase sección 5.2.2). Este cifrado es aplicable a todos los datos, tanto en reposo como en movimiento.

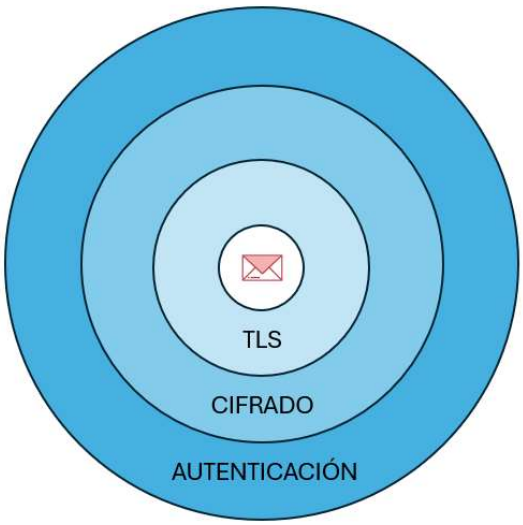


Ilustración 17. Diagrama de capas de seguridad.

Por otro lado, aunque no es una característica propia del cifrado conviene comentar cómo son transmitidos los datos mediante Zenoh. Zenoh tiene su propia estructura binaria denominada "zbytes", los cuales tienen como objetivo encapsular los datos que se transmiten. Es una capa adicional y complementaria a la capa de cifrado, actuando por debajo (véase sección 5.1.3.1).

2.2.5 PLUGINS

Un plugin es un programa complementario que tiene como finalidad ampliar las funciones de un programa o página web [21]. Estos son utilizables gracias a las API (Application Programming Interfaces), las cuales se encargan de unificar la transmisión de datos entre las diferentes partes de un programa. Es importante destacar que un plugin no puede funcionar por sí solo, sino que debe estar combinado con el programa principal. Existen muchos tipos de plugins [22], pero Zenoh los categoriza en 5 grupos distintos, dependiendo de la función que desempeñen. Estos grupos son:

- Almacenamiento: permiten tanto el almacenamiento como la recuperación de datos persistentes publicados en la red. Sirven para soportar consultas sobre datos históricos (no actuales) y permiten construir sistemas de tipo "event sourcing". Algunos ejemplos son SQLite o RocksDB.
- Bridging: permiten a Zenoh conectarse con otros protocolos, de forma que pueda ser implementado en sistemas heterogéneos donde haya uno o más protocolos de comunicación distintos y diferentes de Zenoh. Algunos ejemplos son DDS, MQTT ROS 2 o REST (véase sección 5.1.4).
- Descubrimiento: permiten descubrir otros nodos en la red mediante broadcast, multicast, DNS o cualquier otra configuración personalizada. Esto facilita la configuración de las comunicaciones y la automatización de los despliegues. Plugins como mDNS entran en esta categoría.
- Seguridad: otorgan al protocolo funcionalidades extras relativas a la autenticación, autorización o cifrado de los datos, permitiendo así protegerlos en entornos no seguros. Ejemplos son los plugins crypto o token.

- Análisis: se centran, principalmente, en aportar capacidades de monitorización y gestión de información y datos, facilitando la supervisión de la red para detectar anomalías o cuellos de botella. Un ejemplo sería Prometheus.

Los plugins son vitales para los protocolos de comunicación, puesto que permiten que, sobre la base, se puedan añadir funcionalidades de acuerdo con las necesidades personalizadas del sistema en que se trabaja.

Capítulo 3. ESTADO DE LA CUESTIÓN

Actualmente existen diversos protocolos de comunicación que buscan solucionar los problemas que pueda acarrear la automatización industrial. De entre todos los protocolos existentes (ETHERNET/IP, MODBUS, DEVICENET, HART…) se detallarán las características principales de dos de ellos: PROFINET y OPC UA.

3.1 PROFINET

PROFINET (PROcess Field NETwork) es, actualmente, el estándar en comunicación de automatización de procesos industriales [23]. Se trata de un protocolo de comunicación que permite conectar dispositivos industriales con equipamiento productivo, facilitando la comunicación entre el mencionado equipo industrial (como pueden ser motores o sensores) y los sistemas de control (Ilustración 18). PROFINET posee diversas características que le hacen erigirse como el protocolo escogido por las grandes empresas del sector industrial, como pueden ser Siemens, ABB o Logitek [24]. En primer lugar, la alta velocidad de comunicación que posee dicho protocolo (de alrededor de 100Mb/s) permite que se puedan realizar diversos procesos de manera simultánea, lo que deriva en un aumento total de la eficiencia del sistema. En segundo lugar, la arquitectura del protocolo cobra un valor superlativo, pues al ser escalable, se permite añadir dispositivos dentro de cualquier nivel siempre que sea necesario sin encontrar mayor problema [23]. De esta forma, es mucho más sencillo el aplicar soluciones específicas a los problemas encontrados, aumentando, de nuevo, la eficiencia del sistema. Adicionalmente, esta arquitectura, basada en una red abierta (Ethernet, IEEE 802.3), permite el control total de los dispositivos conectados, lo cual deriva en que la prevención de errores y fallos sea aún más fácil [24]. Además, PROFINET permite a sus usuarios realizar un seguimiento en tiempo real de cada uno de los dispositivos conectados, de forma que se puedan realizar tanto diagnósticos individuales o del sistema, como predicciones respecto a las posibilidades de error de los dispositivos o del propio sistema, sin necesidad de detener la producción. Esto genera que el mantenimiento sea

mucho más sencillo y eficiente. En términos de ciberseguridad, el protocolo en sí mismo posee mecanismos de seguridad como autentificación o cifrado de datos, para asegurar la protección de los mismos frente a virus o malwares [23]. Sin embargo, la seguridad no es fuerte del protocolo. Por otro lado, PROFINET no deja de ser una evolución de la tecnología Fieldbus desarrollada por Siemens a finales de los 90, por lo que presenta una gran compatibilidad con PROFIBUS [24]. Al igual que esta última, PROFINET es un protocolo que necesita de un bus físico, pudiendo transmitir datos tanto de tipo binario como de tipo analógico o digital vía cable con una elevada eficiencia debido a los requerimientos bajos de banda ancha. Finalmente, uno de los mayores puntos fuertes de PROFINET se encuentra en su elevadísima compatibilidad tanto con otros protocolos ya existentes (como pueden ser SNMP o LLDP), como con tecnologías inalámbricas.

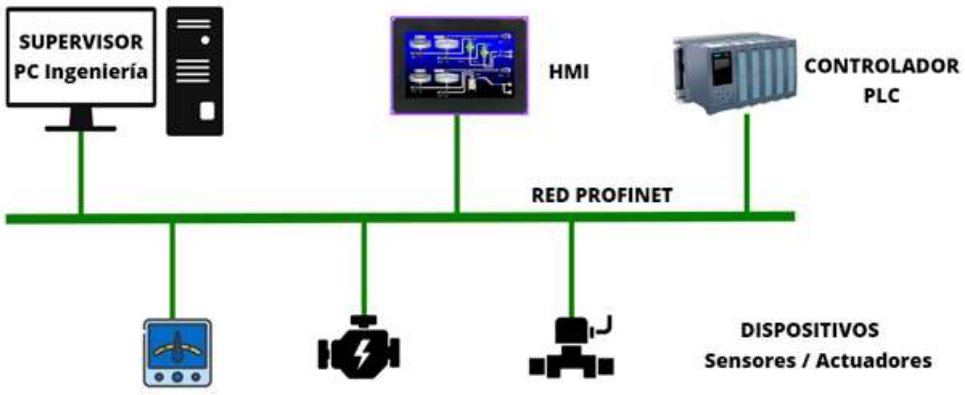


Ilustración 18. Esquema de aplicación del protocolo PROFINET [25].

3.2 OPC UA

Por otro lado, además del ya mencionado PROFINET, hay otros protocolos como puede ser el previamente citado OPC UA (Ilustración 19). Para comenzar, OPC UA son las siglas en inglés de Open Platform Communications Unified Arquitecture, y es un protocolo de comunicación independiente y orientado a servicios, cuyo diseño trata de integrar las

funcionalidades de su predecesor (OPC Classic) [26]. Este protocolo, escalable, interoperable y orientado a servicios (al igual OPC Classic), sin embargo, el principal punto a favor que tiene este protocolo con respecto a su predecesor es que su independencia con respecto a la plataforma y/o sistema operativo permite que pueda ser implementado en cualquier situación, cosa que su predecesor no podía al verse forzado a depender de tecnología ya sea COM o Windows. Ofrece también una gran seguridad debido a sus mecanismos de encriptación de datos [27], superando en estos términos al previamente mencionado PROFINET. Estos mecanismos también son exclusivos de la versión OPC UA, ya que OPC Classic no cuenta con ellos. En cuanto a operabilidad, OPC UA funciona mediante el uso de nodos que representan información (como atributos o valores), a los cuales se puede acceder desde modelos de datos. Para soportar estos datos, se hace uso de protocolos como HTTP o TCP. Tiene su rango de aplicación, principalmente, en el sector de la energía, la agricultura y el sector farmacéutico, para tareas como monitoreo, diagnóstico o mantenimiento de sistemas. Sin embargo, su configuración puede llegar a resultar compleja, en función de los datos que se manejen.

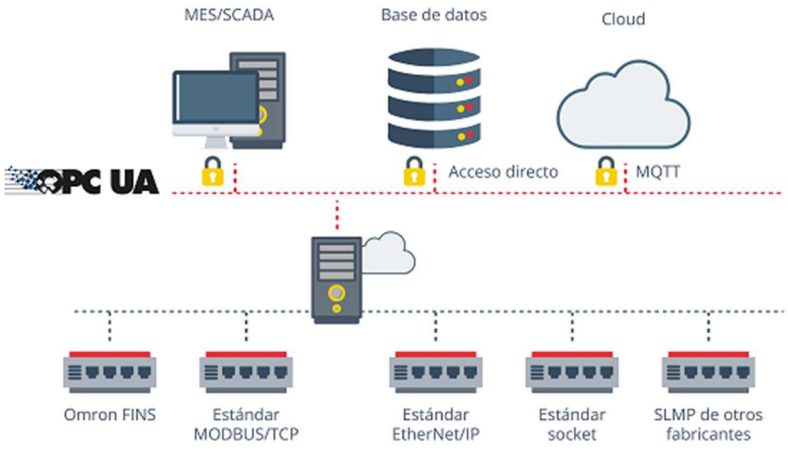


Ilustración 19. Esquema de aplicación del protocolo OPC UA [27].

Capítulo 4. DEFINICIÓN DEL TRABAJO

4.1 JUSTIFICACIÓN

Como se ha podido observar, existen actualmente en el mercado bastantes protocolos de comunicación (PROFINET, OPC UA…) que cumplen con las funciones y las características necesarias para tener un rol importante dentro de esta nueva revolución industrial. Por tanto, ¿por qué habría de plantearse la implementación del protocolo Zenoh, habiendo tantas otras opciones contrastadas en el mercado? ¿Merece realmente la pena tratar de desarrollar esta tecnología, y aplicarla a los procesos industriales actuales, ya no solo como complemento, si no como actor principal en el plano de la automatización? ¿Qué cosas puede aportar este nuevo protocolo que difieran del resto de proveedores del actual mercado? Todas estas preguntas son perfectamente razonables, puesto que es sabiduría popular que no es necesario arreglar algo que no está roto. Sin embargo, el protocolo Zenoh es el único en el mundo que puede trabajar tanto con hardware de nivel de servidor y redes avanzadas, como con redes altamente inestables y con conectividades débiles o mínimas. En ese aspecto, a priori, Zenoh presenta una mejora considerable con respecto a los demás productos, puesto que dicho protocolo puede adaptarse a cualquier nivel de complejidad de un sistema sin perder eficiencia. Esta versatilidad es la que permite a Zenoh ser tenido en cuenta, dado que una situación perfectamente factible en una fábrica o cadena de producción de menor tamaño es que los medios hábiles no permitan disponer de la capacidad de conexión o banda ancha necesaria para poder aplicar el resto de los protocolos, en uno o en todos sus componentes. Muchas veces es necesaria un aumento de la inversión con tal de mejorar el equipo para que pueda verse incluido dentro del sistema de automatización. Zenoh, a priori, plantearía una solución efectiva a ese problema, facilitando la inclusión de todo tipo de dispositivos a los sistemas de control.

4.2 OBJETIVOS

Los objetivos fundamentales del proyecto serán entender el funcionamiento de Zenoh, explorando las distintas opciones de escalado, cifrado y seguridad que puede ofrecer, para posteriormente analizar sus ventajas e inconvenientes.

En primer lugar, se llevará a cabo un proceso de investigación sobre el funcionamiento del protocolo Zenoh. Se partirá desde el proceso de instalación hasta la puesta en marcha, mediante la creación de una aplicación de Zenoh en Python. Se explorarán las distintas opciones que presenta Zenoh, en sus formas multicast y unicast, y se buscará hacer uso de todas las funcionalidades que se consideren convenientes.

En segundo lugar, se evaluarán los conocimientos aprendidos en el primer punto y se determinarán las ventajas y desventajas del protocolo Zenoh. Se valorarán distintos aspectos como la eficiencia, la velocidad, la seguridad, la adaptabilidad o la escalabilidad, con la intención de determinar su viabilidad como protocolo independiente y no como parte de un grupo.

4.3 ALINEACIÓN CON LOS OBJETIVOS DE DESARROLLO SOSTENIBLE

En el año 2015, la Organización de las Naciones Unidas puso en marcha un plan de desarrollo sostenible, constituido por diecisiete objetivos que se buscan cumplir para el año 2030. El noveno objetivo pretende “construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación” [28], y en dicho punto es sobre donde recae la relación entre el protocolo Zenoh y los Objetivos de Desarrollo Sostenible (ODS). Actualmente, en plena cuarta revolución industrial, todo proyecto innovador, capaz de aportar, mejorar y complementar a los elementos ya existentes en la industria debe ser bien recibido. Zenoh plantea una revolución a nivel de innovación al ser capaz de funcionar en condiciones de inestabilidad en las que otros protocolos no son capaces. Adicionalmente, un protocolo de comunicación capaz de trabajar en las precarias condiciones mencionadas

anteriormente presenta, sin duda, una gran solución asumible para países en vías de desarrollo, en los cuales no se pueda asegurar que se cumplen en todo momento los requerimientos mínimos necesarios para que otros protocolos de comunicación puedan ser empleados. De esta forma, se facilita el acceso a sistemas de control y automatización de forma global a toda la industria, favoreciendo a la creación del ecosistema necesario para que el resto de los ODS puedan ser cumplidos con éxito.

4.4 METODOLOGÍA

El proyecto quedará dividido en dos etapas claramente diferenciadas, cada una de ellas orientada a uno de los objetivos que se pretenden tratar. En primer lugar, se procederá al desarrollo de la aplicación Zenoh, sobre la cual se trabajará y en donde se pondrán a prueba las capacidades del protocolo. Para ello, se simulará un sistema de calefacción de una vivienda, y se trabajarán en él las distintas opciones que el protocolo Zenoh puede ofrecer. Dentro de esta etapa, además, se diferenciará entre tres fases distintas: una primera fase de construcción de la aplicación, una segunda fase de seguridad y cifrado, y una última fase de escalado (Ilustración 20).

Ilustración 20. Diagrama con las fases del proyecto.

Posteriormente, se procederá al análisis de resultados que se hayan obtenido en las recurrentes simulaciones, mediante el cual se tratará de discernir y justificar las ventajas y desventajas que se hayan ido obteniendo.

Capítulo 5. SISTEMA DESARROLLADO

Con la finalidad de alcanzar el objetivo fijado para este trabajo, se pondrá en marcha un sistema cuyo protocolo fundamental de comunicación sea ZENOH, de forma que se pueda estudiar las conexiones, desarrollo y comportamiento del mismo. Para ello, se procederá a la simulación de las conexiones del sistema de climatización de una sencilla vivienda unifamiliar. Dicha vivienda contará con distintas habitaciones, y, a su vez, cada una de las habitaciones contará con su propio sensor de temperatura y su propio actuador. El actuador hará las veces de aire acondicionado y calefacción. Adicionalmente, el sistema contará con una consola, desde la cual el administrador podrá tener un control de la temperatura de cada estancia, tanto a nivel informativo (comprobar la temperatura, en grados, de la sala requerida) como a nivel operativo (determinar si una de las salas debe ser calentada o enfriada).

Como ha sido mencionado previamente en este documento, el proyecto ha sido dividido en tres fases. En este capítulo, se entrará en detalle en cada una de las fases, explicando todos los elementos que las componen. Las tres fases citadas (Ilustración 21) son:

- Fase 1: Sistema de Control

  En esta primera fase se construirá la vivienda, con sus diferentes partes y la estructura básica de la misma, se estudiarán las conexiones entre los elementos y se desarrollará el método de conexión multicast.
- Fase 2: Seguridad

  En esta segunda fase se explorarán las opciones para autenticación y cifrado que ofrece el protocolo ZENOH. Se aplicará cifrado por autenticación y contraseña y TLS. Asimismo, se analizarán las conexiones por método unicast y se modificará la estructura del proyecto para soportarlas.
- Fase 3: Escalado

  En esta última fase se aumentará la cantidad de elementos en el sistema, tanto para la vivienda construida en la fase 1 como para la vivienda construida en la fase 2. Se

planteará la escalabilidad desde el punto de vista de un sistema multicast y de un sistema unicast, para que posteriormente puedan ser comparados y analizados.

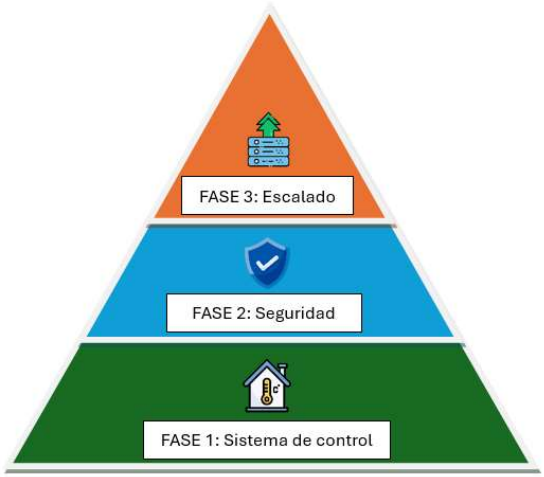


Ilustración 21. Fases del proyecto.

5.1 FASE 1

Esta primera fase podría considerarse la fase con mayor importancia dentro del proyecto, puesto sobre esta se fundamentan tanto la Fase 2 como la Fase 3. El objetivo de la fase será crear unos cimientos sólidos sobre los cuales se pueda soportar el resto del proyecto. Se procederá a la construcción desde cero de la vivienda virtual, realizándose así la base de los códigos sobre los que luego se añadirá la capa de seguridad, y se pondrá a prueba una de las dos modalidades que ofrece el protocolo ZENOH: la comunicación por tráfico IP multicast. Como ya ha sido comentado previamente (véase la sección 2.1.3), el tráfico IP multicast permite el envío de información de forma simultánea a un grupo de clientes, configurados de manera específica para poder recibir el mencionado tráfico de red. Las principales ventajas del tráfico multicast son la eficiencia, la velocidad y la escalabilidad, y estas dos últimas son especialmente relevantes para el sistema de climatización que se planea construir. El punto principal de este sistema sería la capacidad para el administrador de observar las temperaturas de cada una de las estancias en tiempo real, y poder modificar al alza o a la baja dichas temperaturas. El tráfico IP multicast suele ser el método de comunicación escogido en sistemas que requieran de una transmisión de información en

directo, como pueden ser transmisiones de televisión o videojuegos en línea, por tanto, presenta una gran utilidad para el sistema de climatización de la vivienda, puesto que va acorde con el punto principal del sistema y la intención con el que se construye.

El esqueleto de dicho sistema de comunicación está formado por tres elementos principales: sensores, actuadores, y la consola.

En primer lugar, es relevante entender como es la distribución del domicilio y lo que se pretende conseguir. Se ha determinado dicho domicilio como una vivienda de cinco habitaciones bien diferenciadas:

- Salón
- Cocina
- Baño
- Primer Dormitorio
- Segundo Dormitorio

A continuación, en la Ilustración 22 se muestra un plano de la vivienda a modo de imagen esquemática de la distribución de las estancias de la vivienda simulada:

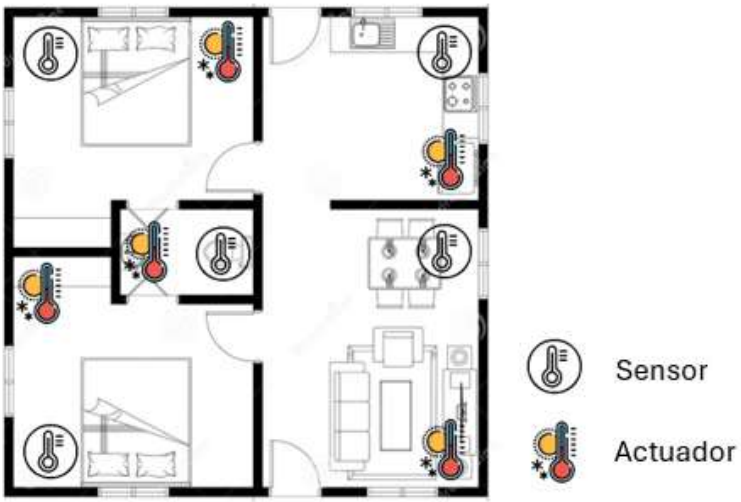


Ilustración 22. Plano de la vivienda.

Como se puede observar, cada una de las ya mencionadas habitaciones cuenta con un sensor de temperatura y con un actuador propios. La temperatura en cada una de las estancias se

considera independiente, al igual que la capacidad de los actuadores de influir en la misma. Esto significa qué, si la temperatura de la cocina es de 21 ºC, la temperatura del salón no tiene por qué ser necesariamente de 21 ºC. De la misma manera, si se da la consigna de calentar el salón, solo su temperatura se verá alterada, y no así la temperatura de los dormitorios o del baño. La temperatura de cada sala, incluyendo los cambios en la misma será simulada a través de componente auxiliar, el cual será denominado "proceso". Con el fin de simplificar el sistema, los datos de temperatura se modelarán como números decimales, y no como números en coma flotante.

Finalmente, el último componente auxiliar será el sniffer. Este componente, que se explicará en profundidad en el Capítulo 6. tiene como función principal escuchar las distintas conexiones, traducirlas y registrarlas para su posterior análisis.

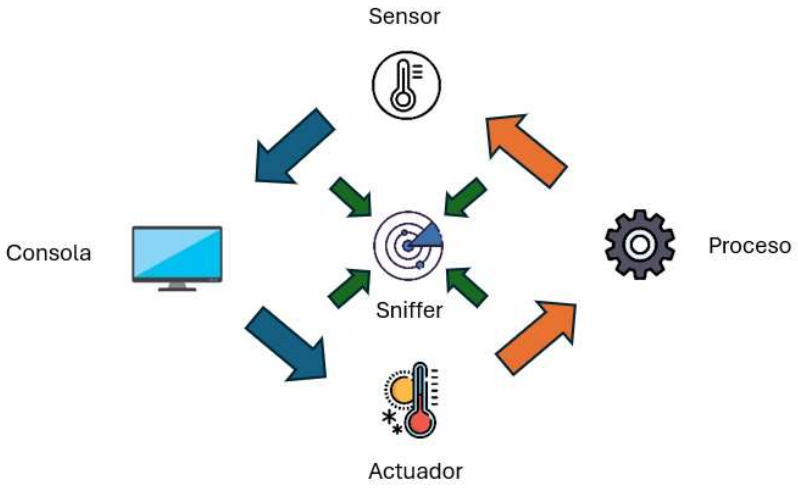


Ilustración 23. Conexiones del sistema.

En cuanto a las suscripciones y publicaciones, todos los elementos del sistema, a excepción del sniffer, actúan a las veces como publicador y como suscriptor. El sniffer únicamente actúa como suscriptor. En la Ilustración 23 se puede observar los roles de suscriptor y publicación, siendo las flechas las que indican qué elemento publica para qué suscriptor. Por ejemplo, el sensor actúa como publicador para la consola, pero como suscriptor para el proceso. Sin embargo, entre consola y proceso, o entre sensor y actuador no se comparte información directamente.

Tiene también relevancia la comprensión de cómo está estructurado dicho proyecto. Como punto de partida se encuentra la carpeta "Vivienda" (Ilustración 24), sobre la cual está construido el proyecto. En dicha carpeta encontramos una carpeta ZENOH, en la cual se encuentran los archivos de la instalación tanto del protocolo como del router. Dicha carpeta contiene, además, los pluggins y el archivo ejecutable. A su vez, se puede encontrar una carpeta denominada zenoh-config, en la cual se encuentra el archivo json5 con la configuración del router. De la misma manera, está el archivo docker-compose.yml, que servirá para configurar las imágenes de los contenedores docker. Por último, se encuentran las carpetas con los distintos elementos. Cada uno de los elementos del sistema ha sido construido como un script de Python, en el cual se han utilizado las funciones y elementos que proporciona el soporte de ZENOH en este lenguaje de programación. Estos códigos pueden ser consultados en su totalidad en el ANEXO I: Código de la Fase I. A su vez, se ha hecho uso de Docker (apartado 5.1.6) para generar un contenedor para cada uno de los elementos de cada habitación. Esto quiere decir que no hay, por ejemplo, cinco scripts diferentes para cada sensor (uno por habitación), sino que es un único script para cada elemento, a partir del cual se crean las estancias para cada sala. Por tanto, la composición de carpetas básica para cada uno de los elementos es la siguiente:

- Un archivo de Python (por ejemplo, actuador.py).
- Un Dockerfile (necesario para la inicialización de dicho código en Docker).
- Un archivo de texto requirements.txt (necesario también para la inicialización en Docker).
- Una copia de la carpeta ZENOH.

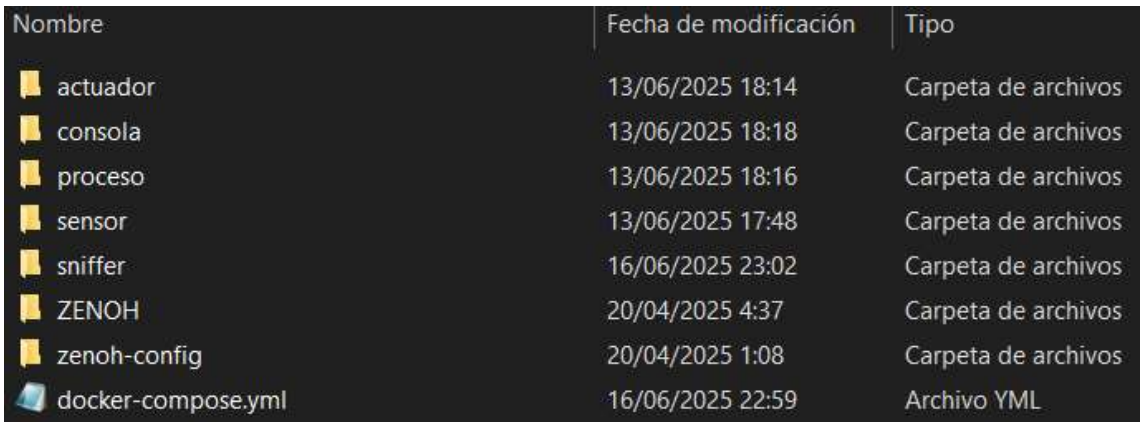


Ilustración 24. Carpeta "Vivienda" y contenidos.

En los siguientes puntos se procederá a la descripción de cada uno de los elementos principales y auxiliares del sistema, analizando para cada componente su configuración, las funciones empleadas, la lógica del código y su papel en el sistema.

5.1.1 PROCESO

En primer lugar, se explicará en detalle el proceso. El proceso surge tras la idea de conseguir una forma de simular el comportamiento real de las temperaturas dentro de una vivienda. Era necesario conseguir modelar una forma de que las temperaturas pudiesen ser modificadas por el actuador, y, a su vez, entendidas por el sensor. Se planteó, en primer lugar, la inclusión de dicho proceso dentro de los sensores, pero esa idea fue descartada, puesto que incumplía uno de los principios básicos de la programación, el Principio de Responsabilidad Única, que establece que cada objeto debe realizar una única función. Si bien es cierto que se suele aplicar a funciones dentro de la programación, se ha considerado importante cumplirlo también a la hora de desarrollar los elementos. Que tanto los actuadores como los sensores realicen solamente una única función permite que el sistema sea más sencillo de manejar, y los errores más fáciles de aislar y depurar. Es por ello por lo que el proceso se plantea como un elemento que también poseerá características de suscriptor/publicador ZENOH, y que se conectará tanto a los sensores como a los actuadores (Ilustración 25).

Ilustración 25. Esquema de comunicaciones del proceso.

Dentro de la carpeta del proceso (Ilustración 6), podemos encontrar los siguientes elementos:

- Proceso.py
- Dockerfile
- Requirements.txt
- Carpeta ZENOH

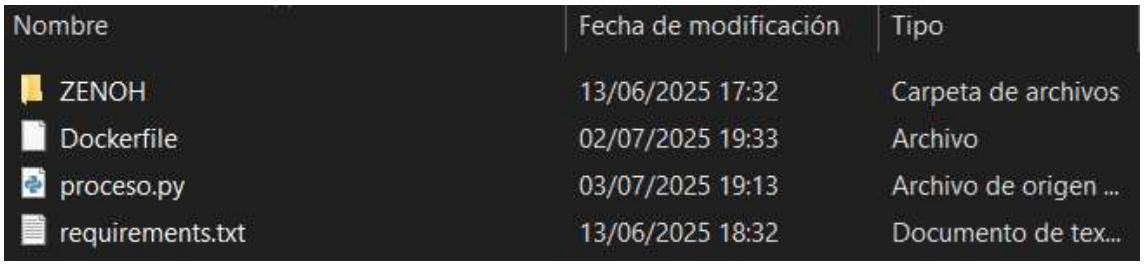


Ilustración 26. Contenidos de la carpeta "proceso".

En esta sección se centrará el tiro en el archivo “proceso.py”, el cual contiene el código en Python del proceso. El resto de los elementos de la carpeta serán explicados más delante de forma conjunta, debido a que la composición de los Dockerfile y requirements.txt es prácticamente idéntica para todos los elementos del sistema, con varianzas mínimas que serán resaltadas y explicadas.

```
import time
import threading
import zenoh
import os
```

En primer lugar, y ya en referencia al código dentro del archivo “proceso.py” es importante destacar las librerías que han sido importadas y los motivos por las cuales son necesarias:

- Zenoh: es la librería fundamental, que contiene todas las funciones necesarias para que el protocolo funcione dentro del archivo.
- Os: permite acceder a variables de entorno.
- Time: permite realizar pausas temporales.
- Threading: permite ejecutar diversas tareas en paralelo.

```
habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]

estado_habitaciones = {
    hab: {"temperatura": 21.0, "modo": "ninguno"} for hab in habitaciones
}

estado_lock = threading.Lock()
```

Posteriormente, se pasa a la definición de las variables que serán relevantes en el código:

- Habitaciones: almacena las habitaciones de la vivienda, por nombre.
- Estado_habitaciones: almacena la temperatura de la habitación, y el estado en el que se encuentra (calentar, enfriar o, de forma predeterminada, ninguno). De forma predeterminada, cada habitación comienza a 21 ºC.
- Estado_lock: crea un lock que permita el acceso a los hilos de forma segura.

```
def consigna_listener(room):
    def callback(sample):
        try:
            payload = bytes(sample.payload).decode("utf-8").strip().lower()
            print(f"[{room}] Consigna recibida: {payload}")

            with estado_lock:
                if payload in ["calentar", "enfriar"]:
                    estado_habitaciones[room]["modo"] = payload
                else:
                    estado_habitaciones[room]["modo"] = "ninguno"
        except Exception as e:
            print(f"[{room}] Error al procesar consigna: {e}")
    return callback
```

Se define la función “consigna_listener”, la cual crea una función callback personalizada para cada una de las habitaciones. En caso de que se reciba alguna consigna, de coincidir con “calentar” o “enfriar”, se actualizará el modo de la habitación para la cual se haya solicitado dicha consigna. En caso de que la muestra recibida no se corresponda con alguno

de los dos modos definidos, la ignorará. Cuenta con dos funciones printf para mostrar en pantalla si se ha recibido la consigna, o, si, por lo contrario, ha ocurrido un error, en cuyo caso lo mostrará.

```
def simulador_clima(session):
    while True:
        with estado_lock:
            for room, datos in estado_habitaciones.items():
                modo = datos["modo"]
                temp = datos["temperatura"]

                if modo == "calentar" and temp < 25:
                    datos["temperatura"] += 1
                elif modo == "enfriar" and temp > 18:
                    datos["temperatura"] -= 1

                temp_key = f"myhome/{room}/temp"
                session.put(temp_key, str(round(datos["temperatura"], 2)))
                print(f"[{room}] Temp publicada: {datos['temperatura']}°C")
        time.sleep(5)
```

A continuación, se define la segunda función, simulador_clima, sobre la cual actúa toda la lógica de aumento y disminución de temperatura. Se ha tomado como referencia para la simulación de los cambios de temperatura los fundamentos de un control P. En caso de que se reciba la consigna de “calentar”, la función actualizará la temperatura de dicha sala a razón de un aumento de 1 grado cada 5 segundos, hasta un límite, el cual se ha establecido en 25 ºC. De la misma manera, disminuirá la temperatura a razón de 1 grado cada 5 segundos en caso de que la consigna sea “enfriar”, hasta un límite inferior de 18 ºC. La Ilustración 27 y la Tabla 1 muestran los valores de la temperatura en una habitación a lo largo del tiempo, de acuerdo con lo simulado en el proceso y las distintas consignas recibidas.

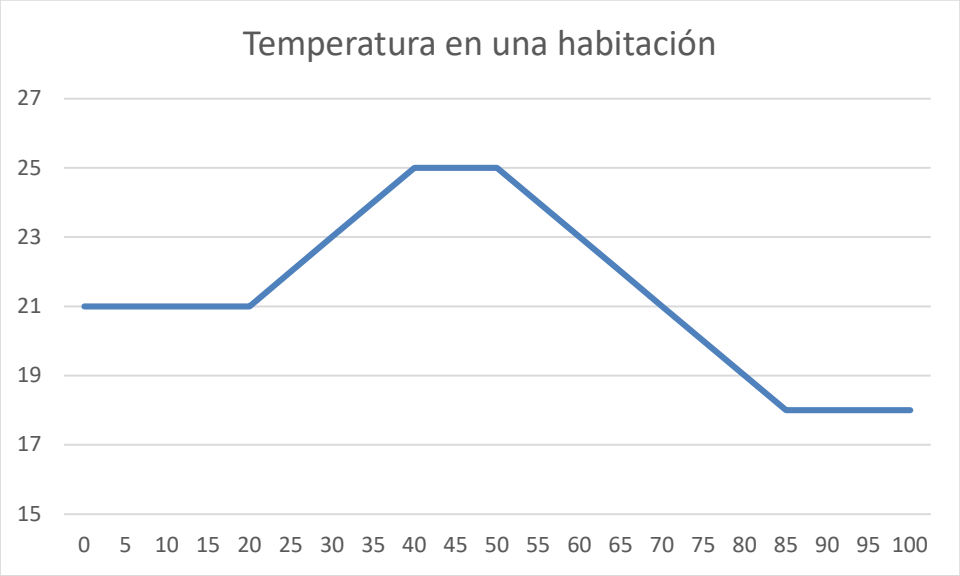


Ilustración 27. Gráfico de temperaturas de cada habitación, a lo largo del tiempo.

Tiempo (s)	**Consigna**	**Temperatura (ºC)**
0	Ninguna	21
5	Ninguna	21
10	Ninguna	21
15	Ninguna	21
20	Ninguna	21
23	De ninguna a calentar	21
25	Calentar	22
30	Calentar	23
35	Calentar	24
40	Calentar	25

45	Calentar	25
50	Calentar	25
51	De calentar a enfriar	25
55	Enfriar	24
60	Enfriar	23
65	Enfriar	22
70	Enfriar	21
75	Enfriar	20
80	Enfriar	19
85	Enfriar	18
90	Enfriar	18
95	Enfriar	18
100	Enfriar	18

Tabla 1. Valores de consigna y temperatura a lo largo del tiempo.

Posteriormente, publica dichas temperaturas en el tópico "temp" de cada habitación. Esta función también cuenta con un printf, que indica la temperatura que está siendo publicada en cada momento.

```
if __name__ == "__main__":

    config = zenoh.Config()
    with zenoh.open(config) as session:
        for room in habitaciones:
            key = f"myhome/{room}/consigna"
            session.declare_subscriber(key, consigna_listener(room))
```

```
            print(f" Subscrito a: {key}")

        threading.Thread(target=simulador_clima, args=(session,),
daemon=True).start()

        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(" Proceso detenido.")
```

Finalmente, dentro de la ejecución principal se lanza la sesión de ZENOH con la configuración predeterminada, y se declara un suscriptor del tópico "consigna" según lo definido en la función "consigna_listener", del cual recibirá las comandas para calentar o enfriar las habitaciones. Cuenta con una función threading.Thread para que la simulación de las temperaturas corra en un hilo separado, de forma que pueda estar en ejecución continua sin afectar al funcionamiento del sistema. Cuenta también con dos printf informativos, uno de ellos para informar de la suscripción al tópico deseado, y otro dentro del bucle que mantiene el proceso activo, para informar de que ha sido interrumpido debido a una petición del usuario mediante teclado.

A modo de resumen, el código del proceso cuenta con dos funciones principales. La primera simula la temperatura de las habitaciones, predeterminada en 21ºC, y la aumenta o disminuye a razón de 1 grado cada 5 segundos en función de la consigna. Además, publica dicha temperatura para que el sensor pueda obtenerla. La segunda simplemente escucha las comandas del actuador y actualiza en función de ellas el modo de las habitaciones.

5.1.2 ACTUADOR

El actuador tiene como objetivo simular un sistema de aire acondicionado o de calefacción. Su lógica es relativamente sencilla: desde la consola se podrán enviar dos consignas diferentes una para calentar la sala y otra para enfriar la sala. El actuador recibirá dicho comando, y se pondrá en marcha para aumentar o disminuir la temperatura de dicha estancia. El actuador entonces mandará un mensaje al proceso, para que este modifique su temperatura (Ilustración 28).

Ilustración 28. Esquema de las conexiones del actuador.

Dentro de la carpeta del actuador (Ilustración 29), se pueden encontrar los mismos elementos que se encontraban en la carpeta del proceso:

- Actuador.py
- Dockerfile
- Requirements.txt
- Carpeta ZENOH

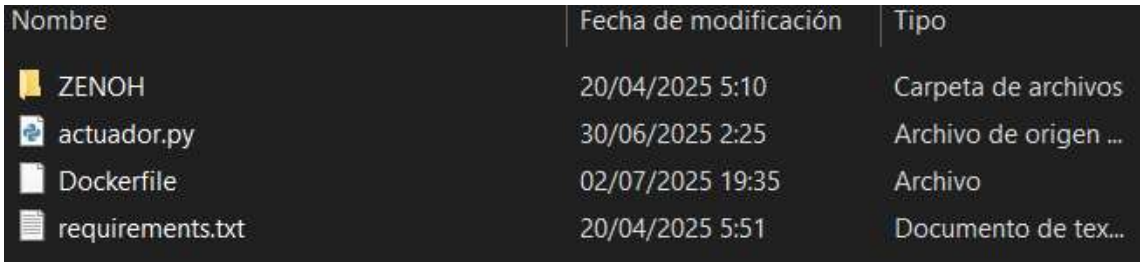


Ilustración 29. Contenidos de la carpeta "actuador".

Al igual que se hizo en la sección del proceso, se procederá a explicar únicamente y en detalle los contenidos del archivo “actuador.py”.

```
import zenoh
import os
import time
import threading
```

Como se puede observar, no existen cambios en relación con las librerías importadas en el código del proceso. Estas librerías han sido seleccionadas por los mismos motivos que en el caso del proceso.

A continuación, en el código se define la función principal (main), la cual ejecutará la lógica del programa. Dicha función se dividirá en diferentes partes, y, sobre estas, se procederá a la explicación detallada del código.

```
    room = os.getenv("ROOM", "unknown_room")

    accion_topic = f"myhome/{room}/accion"
    consigna_topic = f"myhome/{room}/consigna"

    print(f" El actuador de '{room}' escuchando acciones en: {accion_topic} y
reenviando a: {consigna_topic}")

    modo_actual = None
    modo_lock = threading.Lock()
```

En primer lugar, observamos la definición de tres variables:

- Room: almacena la variable de entorno, la cual le indica la estancia en la que se encuentra. En caso de que no exista, toma el valor de unknown_room.
- Accion_topic: es el tópico donde el actuador escuchará las acciones.
- Consigna_topic: es el tópico donde el actuador reenviará las acciones.

La función printf se trata de un checkpoint para comprobar que tanto la habitación como los tópicos a los que se conecta el actuador son correctos.

Por otro lado, tenemos otras dos variables relacionadas con el modo:

- Modo_actual: almacena la última acción que se recibe. Dicha acción podrá ser, en función de la decisión del usuario, “enfriar” o “calentar”. Será inicializada en “none” (aún no se le ha enviado ninguna consigna).
- Modo_lock: esta variable tiene su utilidad en evitar conflictos cuando se intente acceder a la variable modo_actual.

```
with zenoh.open(zenoh.Config()) as session:
```

Esta línea de código contiene una función que abre una sesión ZENOH aplicando la configuración por defecto. Dicha función, además, tiene definidas dos funciones: accion_callback y reenviar_loop.

```
def accion_callback(sample):
            nonlocal modo_actual
            try:
                modo = bytes(sample.payload).decode("utf-8").strip().lower()
                print(f"[{room}] Acción recibida: {modo}")
                with modo_lock:
                    modo_actual = modo
            except Exception as e:
                print(f"[{room}] Error en accion_callback: {e}")
```

Esta función se ejecutará cada vez que se escuche un mensaje en accion_topic. Dentro, en la variable "modo" almacena el mensaje en forma de payload recibido, previamente decodificándolo de los zbytes propios de ZENOH a un formato de tipo string. Posteriormente, almacena el contenido de la variable "modo" dentro de la variable "modo_actual". El código además cuenta con dos comandos de printf, uno para informar qué acción ha sido recibida y en qué habitación, y otro para, en caso de error, indicar que ha ocurrido un error en la función.

```
session.declare_subscriber(accion_topic, accion_callback)
```

Este comando se permite al actuador suscribirse al tópico accion_topic según el callback que ha sido definido previamente.

```
def reenviar_loop():
            while True:
                with modo_lock:
                    modo = modo_actual
                if modo:
                    session.put(consigna_topic, modo.encode("utf-8"))
                    print(f"[{room}] Reenviando comando: {modo}")
                time.sleep(5)
```

La función reenviar_loop es una función que recorre un hilo separado, de forma que se encarga de enviar de forma periódica (cada 5 segundos, ajustable según la función time.sleep) el modo en el que se encuentra el actuador. Si existe un modo definido, entonces el actuador lo publica en el tópico "consigna_topic". Mientras el modo no cambie, se seguirá publicando. La función cuenta también con un printf que informa sobre el reenvio del comando.

```
hilo_reenvio = threading.Thread(target=reenviar_loop, daemon=True)
        hilo_reenvio.start()
```

Estas dos líneas de comando se encargan de crear y ejecutar el hilo en segundo plano, de acuerdo con lo definido en la función reenviar_loop.

```
try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(f" El actuador de '{room}' detenido.")

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

Finalmente, el código termina con una función que mantiene el programa encendido y en escucha activa, guardando una excepción para una detención manual del programa con el comando Ctrl + C. El comando if final ejecuta directamente el main.

Como se puede observar, es un código sencillo pero eficiente, el cual tiene dos ramas distintas: una funciona como suscriptor (y recibe la información de la consola sobre si es necesario enfriar o calentar la habitación, la cual se publica en el tópico "accion_topic"), y la segunda actúa en segundo plano como publicador, manteniendo constantemente actualizado el tópico "consigna_topic".

5.1.3 SENSOR

El sensor tiene como objetivo simular un sensor de temperatura real, capaz de leer la temperatura del medio y mostrarla en tiempo real como valor numérico. La lógica del sensor, al igual que la del actuador, es también sencilla. El sensor se conectará al proceso, del cual leerá la temperatura que este le indique, para posteriormente publicarla en un tópico accesible para la consola (Ilustración 30). Se puede entender a dicho sensor como un intermediario directo entre el proceso y la consola. A efectos únicamente de la simulación, el sensor no deja de ser un paso adicional del cual se puede prescindir, puesto que su función podría ser cubierta por el proceso directamente. Sin embargo, debido a que el propósito de este trabajo de investigación no deja de ser el estudio entre las conexiones de los distintos elementos y la forma en que el protocolo ZENOH interactúa con ellos, se ha decidido mantener al sensor como miembro del sistema, debido a que el proceso no deja de ser una forma de simular un elemento real como puede ser la temperatura de una sala. En el caso de

que no se tratase de un modelo de vivienda simulado, sí sería imprescindible el papel del sensor, y, como el objetivo también es acercarse lo máximo posible a la realidad, se ha considerado oportuno mantenerlo.

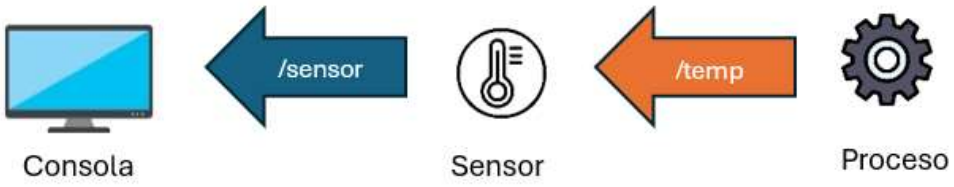


Ilustración 30. Esquema de las comunicaciones del sensor.

Dentro de la carpeta del sensor (Ilustración 31), al igual que en la carpeta del actuador, podemos encontrar los siguientes elementos:

- Sensor.py
- Dockerfile
- Requirements.txt
- Carpeta ZENOH

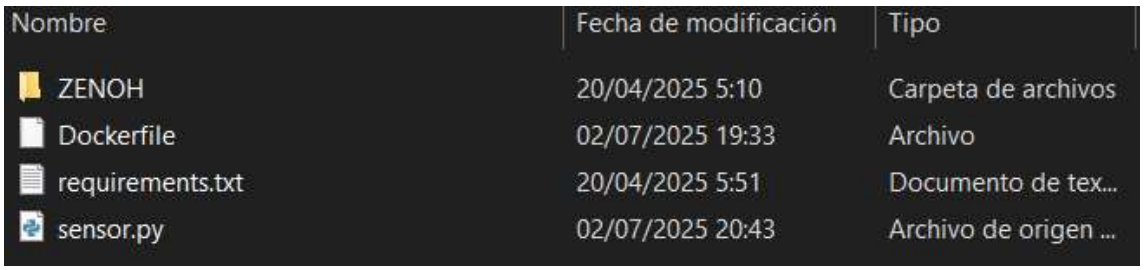


Ilustración 31. Contenidos de la carpeta "sensor".

Al igual que en la descripción que se ha hecho previamente del resto de elementos del sistema, se procederá al análisis en profundidad del código del sensor.

En primer lugar, se encuentran las librerías importadas:

```
import time
import zenoh
import os
```

A diferencia de los casos previos, en esta ocasión no ha sido necesaria la importación de la librería threading, ya que no se tiene pensado realizar ninguna acción simultánea a la lectura de temperaturas en segundo plano.

A continuación, en el código se define la función principal o main, encargada de ejecutar la lógica del programa. En este caso, dicha función main solo tiene definida una única función, debido a la sencillez del programa.

```
room = os.getenv("ROOM", "unknown_room")

    input_topic = f"myhome/{room}/temp"
    output_topic = f"myhome/{room}/sensor"

    print(f" El sensor de '{room}' escuchando en: {input_topic}, y reenviando a:
{output_topic}")
```

La función comienza con la definición de tres variables, al igual que en el caso del actuador:

- Room: almacena la variable de entorno, la cual le indica la estancia en la que se encuentra. En caso de que no exista, toma el valor de unknown_room.
- Input_topic: se trata del tópico al cual se suscribirá el sensor con el objetivo de escuchar las temperaturas publicadas por el proces.
- Output_topic: nombre del tópico en el que el sensor publicará las temperaturas obtenidas del proceso.

A continuación, se muestra una función printf cuyo objetivo es la comprobación de que los tópicos de publicación y suscripción son los correctos, y que el sensor está ubicado correctamente, de acuerdo con la variable de entorno que le ha sido asignada.

```
 with zenoh.open(zenoh.Config()) as session:
```

Se abre la sesión en ZENOH aplicando la configuración por defecto, de la misma manera que se ha abierto en el resto de los elementos.

```
def callback(sample):
            try:
```

```
            temp = bytes(sample.payload).decode("utf-8")
            print(f"[{room}] Temperatura recibida: {temp}°C")
            session.put(output_topic, temp)

        except Exception as e:
            print(f"[{room}] Error en reenvío: {e}")
```

Se define la función callback, la cual se ejecutará automáticamente cada vez que se reciba algún dato en el tópico "input_topic". Esta función tiene como objetivo traducir el paquete de datos recibido del proceso y convertirlo de Zbytes a string.

5.1.3.1 Problemática del binario zenoh

Esta línea de código ha resultado conflictiva a lo largo del desarrollo del proyecto, y ha tenido que ser modificada en diversas ocasiones debido a la incapacidad de Python de cambiar el tipo de variable de una a otra. La intención detrás de cambiar la forma del contenido a string es que resulte más accesible a la hora de recibir las temperaturas en la terminal de salida, y estas puedan ser publicadas en formato numérico (acompañado del correspondiente símbolo de unidades), en lugar de como un archivo codificado en el método de bytes de ZENOH. Sin embargo, la forma en la que está configurado Python generó que esta tarea resultase complicada, debido a la imposibilidad de modificar dentro del propio Python la naturaleza de la variable.

Para facilitar la comprensión de dicho conflicto, se plantea el siguiente ejemplo: póngase qué el sensor del baño recibe una información de que el baño registra una temperatura de 21 ºC. Esta información llega al sensor en forma de Zbytes, y el sensor entonces trata de transformarlo en un string para poder mostrarlo de forma que el usuario sea capaz de entenderlo. Sin embargo, en el momento de decodificar dicha variable ejecutando la línea de código, mientras el sensor considera que ya se puede manejar como una variable de tipo string, para Python la variable sigue manteniendo la forma de Zbytes. Por tanto, cualquier operación en la que se trate dicha variable como algo distinto a los Zbytes será interpretado por Python como un error, provocando el fallo del código.

La solución que se encontró fue mediante prueba y error de diferentes líneas de comando, hasta finalmente se optó por hacer una doble conversión: en primer lugar, se decodificaría

de zbytes (en binario Zenoh) a bytes (en binario), y, posteriormente, dichos bytes se transformarían en un string (Ilustración 32).

Ilustración 32. Esquema de la transformación de datos.

Continuando con el código, se puede observar cómo hay una función printf, que se encarga de mostrar la temperatura leída, para posteriormente publicarla en el tópico de salida "output_topic". Además, se añade otro printf para indicar si ha ocurrido un error, y la naturaleza del mismo.

```
session.declare_subscriber(input_topic, callback)
```

Esta línea de código tiene como objetivo suscribir al sensor al tópico "input_topic", de acuerdo con la lógica que ha sido definida en la función callback.

```
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(f" Sensor '{room}' detenido.")
```

Finalmente, se añade el bucle principal, el cual mantendrá el proceso activo hasta el final del mismo, o la interrupción por parte del usuario.

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

Esta última línea de código ejecuta el main.

Como se ha podido comprobar, el sensor posee una lógica sencilla que le hace ser un intermediario entre el proceso y la consola, obteniendo la temperatura de uno y publicándola en un tópico al que la consola pueda acceder. El principal conflicto, como se ha visto, ha ocurrido con la decodificación de los paquetes de información que se envían de un lugar a otro, pero finalmente fue solventado con éxito. Pese a poder ser prescindible, se considera que su mantenimiento en el sistema tiene un aporte valioso de cara al desarrollo del proyecto.

5.1.4 CONSOLA

La consola es la interfaz principal, desde la cual el usuario tendrá la potestad de determinar la consigna para la temperatura de cada habitación, y, adicionalmente comprobar dicha temperatura. La consola se encontrará conectada tanto a los actuadores como a los sensores: funcionará como un suscriptor a la hora de identificar la temperatura, y como un publicador a la hora mandar una consigna (Ilustración 33).

Ilustración 33. Esquema de comunicaciones de la consola.

Con la finalidad de que la interfaz cuente con una mayor accesibilidad para el usuario, y sea más fácil operar a partir de ella, se ha desarrollado con ayuda de FastAPI. FastAPI es un framework de alto rendimiento para poder construir APIs web con Python, principalmente centrado en crear servicios RESTful. FastAPI genera documentación interactiva a través de Swagger UI o ReDoc, ambas sumamente intuitivas, lo cual hace que sea muy cómodo tanto para el usuario final como para el desarrollador su uso y testeo.

Adicionalmente, se ha usado el motor de plantilla de Python Jinja2, pensado para generar contenido HTML. Este motor se combina con FastAPI para poder renderizar una página

HTML dinámica (Ilustración 34), que también facilite el uso de la consola por parte tanto del usuario como del desarrollador.

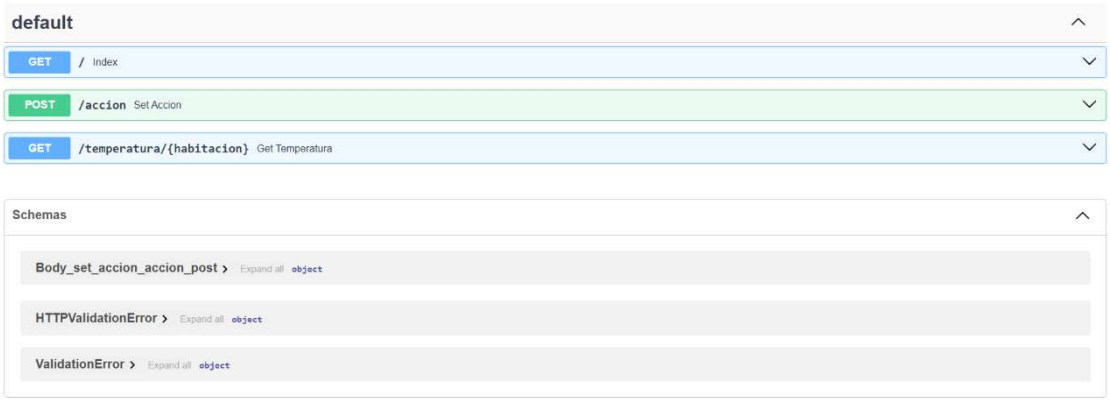


Ilustración 34. Interfaz creada con FastAPI para la consola.

Dentro de la carpeta de la consola (Ilustración 35) se pueden observar diferencias con el resto de las carpetas, debido a la implementación de la API. Los contenidos son los siguientes:

- Consola.py
- Dockerfile
- Requirements.txt
- Carpeta ZENOH
- Carpeta templates

El contenido de la carpeta templates consiste en un archivo “index.html”, el cual construye la plantilla HTML generada por el motor Jinja2.

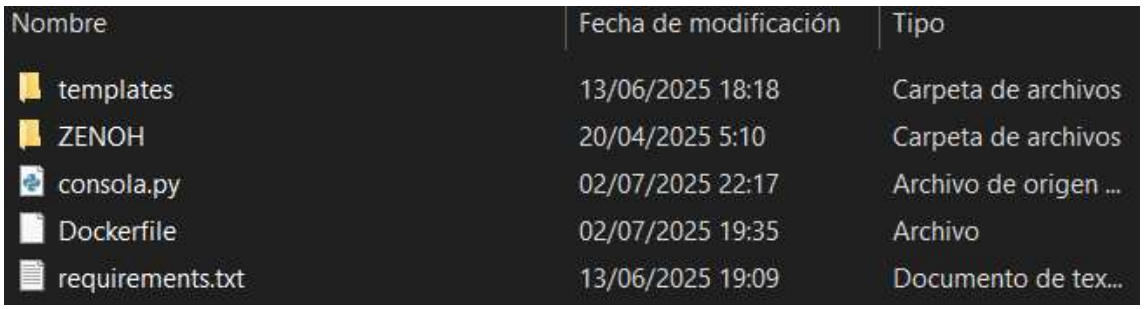


Ilustración 35. Contenidos de la carpeta "consola".

De la misma forma que se ha hecho con el resto de los elementos, se procederá al análisis detallado del código de la consola. Nótese que, debido a la naturaleza de esta, este código se plantea como el de mayor complejidad. Si bien no se han realizado tareas excesivamente complejas, el punto de mayor conflicto en el desarrollo del código ha sido la implementación simultánea tanto de la lógica de ZENOH como de la lógica propia de FastAPI.

Para comenzar, se puede observar que las importaciones son considerablemente distintas a las del resto de códigos:

```
from fastapi import FastAPI, Request, Form, HTTPException, status
from fastapi.responses import HTMLResponse, PlainTextResponse
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
import zenoh
import threading
```

A los ya conocidos zenoh y threading, se suman diversas importaciones (todas de la propia FastAPI), para poder llevar a cabo las distintas dependencias que el código requiere. A continuación, la explicación de cada una de las importaciones:

- FastAPI: importa el framework necesario para poder crear la API web.
- Request: presenta la solicitud HTTP que llega al servidor, con información como método, headers, cookies…
- Form: permite recibir datos de formularios.
- HTTPException: permite lanzar errores en los endpoints.
- Status: Contiene las constantes que representan cada código de estado HTTP.

- HTMLResponse: importa tipos de respuesta HTTP, en este caso respuesta HTML.
- PlainTextResponse: mismo motivo que HTMLResponse, importa tipos de respuesta HTTP, en este caso texto plano.
- Jinja2Templates: importa el motor de plantillas HTML dinámico Jinja2.
- CORSMiddleware: permite realizar peticiones desde otros dominios.

A continuación, se procede a levantar FastAPI:

```
app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

templates = Jinja2Templates(directory="templates")
```

En primer lugar, se crea la instancia "app", y, a continuación, se permite, mediante CORSMiddleware, que cualquier origen pueda realizar peticiones a dicho backend. Adicionalmente, se indica que los archivos necesarios para la template HTML se encuentran en el directorio "templates" dentro de la carpeta (como se ha comentado previamente, al definir el contenido de la misma).

```
habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]
temperaturas = {hab: "—" for hab in habitaciones}
temperaturas_lock = threading.Lock()
```

Posteriormente, se define un listado con las habitaciones de la vivienda. Como se ha comentado, la vivienda constará de cinco habitaciones, siendo estas: salón, cocina, baño (Para evitar conflictos de caracteres, quedará escrita como "bano", sin ñ), primer dormitorio y segundo dormitorio. Por otro lado, se crea un diccionario llamado "temperaturas", en el que se guarda la última temperatura recibida por la consola. A su vez, "temperaturas_lock" permite que los distintos hilos no accedan simultáneamente al diccionario, provocando errores.

```
zsession = zenoh.open(zenoh.Config())
```

```
def listener(room):
    def callback(sample):
        try:
            payload_bytes = bytes(sample.payload)
            payload_str = payload_bytes.decode("utf-8")

            with temperaturas_lock:
                temperaturas[room] = payload_str

            print(f"[{room}] Temperatura actualizada: {payload_str}°C")
        except Exception as e:
            print(f"Error al procesar muestra para {room}: {e}")
    return callback
```

Con la línea zsession, se abre la sesión ZENOH con la configuración estándar para poder comunicarse con los sensores y actuadores. Es entonces cuando se define la función "listener", que recibe como parámetro una de las habitaciones y devuelve la función "callback" específica. Dicha función tiene un comportamiento muy similar a la que ha sido comentada para el sensor. El mismo problema con la naturaleza de la variable se plantea aquí, y la solución es idéntica: en primer lugar, se recoge la temperatura recibida en zbytes y se almacena en una variable "payload_bytes" transformada en bytes. Posteriormente, el valor de dicha variable es transformado en un string y almacenado en "payload_str", temperatura la cual se inserta dentro del diccionario de temperaturas, en la habitación correspondiente. La función contiene, además, dos printf para mostrar la temperatura que ha sido actualizada, y en caso de haber un error, el motivo de este y la habitación en el que se ha dado.

```
def suscribir_temperaturas():
    for hab in habitaciones:
        topic = f"myhome/{hab}/temp"
        zsession.declare_subscriber(topic, listener(hab))
        print(f"Suscrito a: {topic}")

threading.Thread(target=suscribir_temperaturas, daemon=True).start()
```

A continuación, se define la función para suscribirse a los tópicos de ZENOH de temperatura de cada una de las habitaciones. La lógica es sencilla: un bucle for que recorre las habitaciones, se declara suscriptor de dicha habitación según la lógica de la función "listener", y devuelve un printf con la habitación en la que se ha suscrito, a modo de

confirmación. Esta función se lanza en un hilo a parte mediante la función "threading.Thread" con el objetivo de no saturar o bloquear el servidor de FastAPI.

Posteriormente, se comienzan a definir los endpoints de la API.

```
@app.get("/", response_class=HTMLResponse)
async def index(request: Request):
    with temperaturas_lock:
        temp_copy = temperaturas.copy()

    return templates.TemplateResponse("index.html", {
        "request": request,
        "temperaturas": temp_copy,
        "habitaciones": habitaciones,
    })
```

Este primer endpoint (Ilustración 36) muestra la página principal del sistema a través de la plantilla HTML, basándose en el archivo "index_html". Realiza una copia del diccionario de temperaturas, y renderiza dicho contenido en una plantilla HTML que contiene los datos de las temperaturas y las habitaciones.

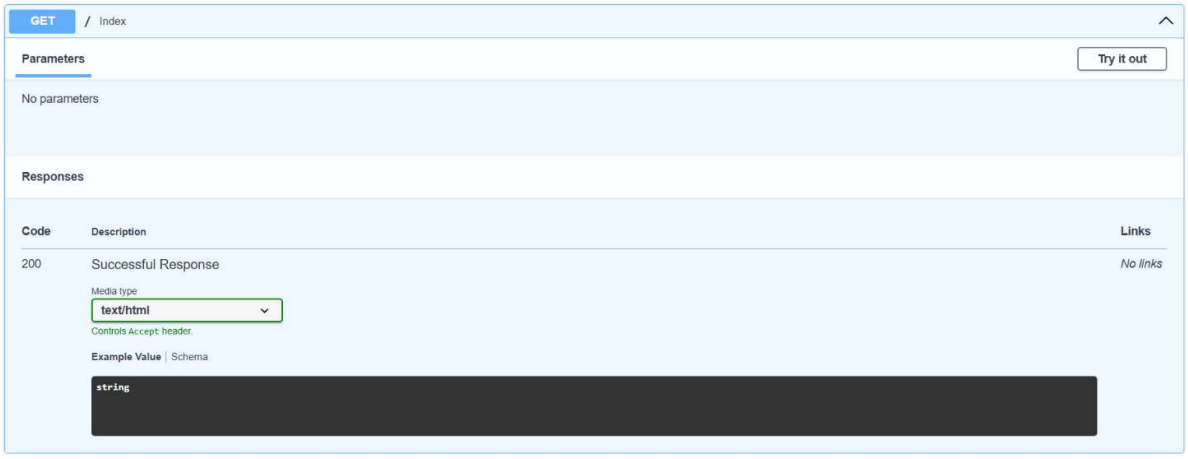


Ilustración 36. Endpoint del primer GET.

```
@app.post("/accion", response_class=HTMLResponse)
async def set_accion(request: Request, habitacion: str = Form(...), accion: str =
Form(...)):
    try:
        if habitacion not in habitaciones:
            mensaje = f" Habitación no reconocida: {habitacion}"
        else:
            key = f"myhome/{habitacion}/accion"
```

```
            zsession.put(key, accion.encode("utf-8"))
            mensaje = f"Acción '{accion}' enviada a {habitacion}"
            print(f"[consola] Enviada acción: {accion} → {key}")

        with temperaturas_lock:
            temp_copy = temperaturas.copy()

        return templates.TemplateResponse("index.html", {
            "request": request,
            "temperaturas": temp_copy,
            "habitaciones": habitaciones,
            "mensaje": mensaje,
        })

    except Exception as e:
        print(f"Error en set_accion: {e}")
        return PlainTextResponse("Internal Server Error",
status_code=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

En este segundo endpoint (Ilustración 37) es desde donde se realizan las consignas para la climatización de las salas. Según la lógica codificada, recibe un formulario de tipo HTML con dos parámetros distintos: la consigna (que puede ser calentar o enfriar), y la habitación en la que se requiere. Es entonces cuando la consola publica en el tópico "acción" de cada habitación, la consigna que se ha determinado. Además, cuenta con diversos printf, preparados para informar en caso de que no se haya determinado la habitación (dicha habitación no existe o no está recogida en la lista), en caso de que haya ocurrido un error (informando de dicho error) y para informar de que todo ha funcionado de manera correcta. Asimismo, la función return devuelve una plantilla similar a la del índice, pero con la información actualizada.

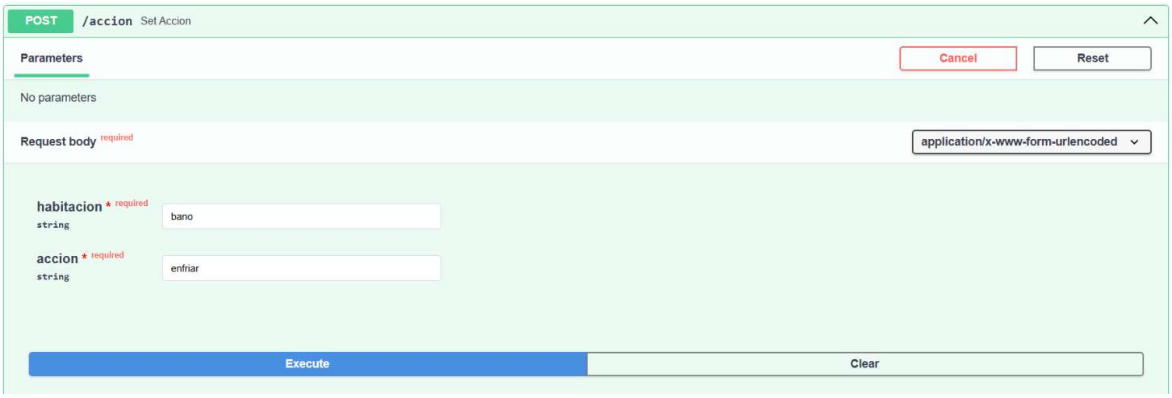


Ilustración 37. Endpoint para postear la acción.

```
@app.get("/temperatura/{habitacion}")
async def get_temperatura(habitacion: str):
    habitacion = habitacion.lower()
    if habitacion not in habitaciones:
        raise HTTPException(status_code=404, detail="Habitación no encontrada")
    with temperaturas_lock:
        temp = temperaturas.get(habitacion, "—")
    return {"habitacion": habitacion, "temperatura": temp}
```

Por último, este endpoint es el encargado de devolver la temperatura de la habitación en la cual se quiere realizar una consulta. Este endpoint (Ilustración 38) es muy sencillo, pues simplemente toma la habitación introducida, estandarizándola a minúsculas (lo cual evita errores por datos de entrada que contengan mayúsculas), verifica que la habitación exista y sea válida (comunicando un error en caso de no serlo) y accede a la temperatura dentro del diccionario, para devolverlo en formato "habitación: -, temperatura:" (Ilustración 39).

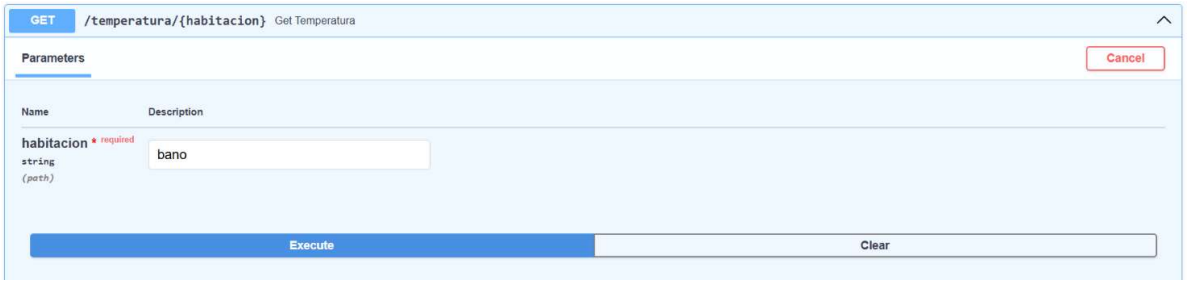


Ilustración 38. Endpoint del segundo GET para obtener la temperatura.

Ilustración 39. Respuesta a la petición de temperatura.

Sin ser la lógica excesivamente compleja, consola.py es, de los cuatro códigos analizados en esta sección, el más complicado y completo, debido a la implementación de la API. Sin embargo, los resultados avalan el uso de la misma, y sin duda presentan una interfaz sencilla, intuitiva y manejable que facilita las labores tanto a usuario como desarrollador a la hora de interactuar con el sistema.

Con el archivo de la consola se finaliza, por el momento, el análisis de código de los elementos que conforman el sistema. Sin embargo, antes de continuar con el desglose de la parte correspondiente a Docker, cabe realizar un último análisis con relación a la configuración del archivo de configuración del router.

5.1.5 CONFIGURACIÓN DE ZENOH

Como se ha mencionado previamente a la hora de enumerar los archivos que componen la carpeta "Vivienda" en donde se ha construido el modelo de vivienda virtual, todas las carpetas de cada uno de los elementos cuentan con una copia de la carpeta "ZENOH". Esta, además de contener el archivo ejecutable y las dependencias necesarias para la propia ejecución, cuenta con un archivo de configuración que ha sido nombrado zenoh-myhome.json5. Este archivo de tipo json5 (que, a diferencia de los archivos json normales, admite la escritura de comentarios en sus líneas de código) tiene la configuración básica que será necesaria para lanzar el programa de ZENOH. El contenido de dicho archivo es el siguiente:

```
{
  plugins: {
    rest: {
      http_port: 8000
```

```
    },
    storage_manager: {
      storages: {
        myhome: {
          key_expr: "myhome/**",
          volume: {
            id: "memory"
          }
        }
      }
    }
  }
}
```

En primer lugar, se puede observar la definición de los plugins. En este caso, el único plugin utilizado es REST. Este plugin es una extensión que se añade a ZENOH para permitir que pueda usar protocolos HTTP para poder interactuar con los distintos datos (los servicios RESTful mencionados previamente en el apartado sobre FastAPI).

A continuación, el apartado "storage_manager" se encarga de habilitar la gestión del almacenamiento de los datos en Zenoh. El nombre que le da al almacenamiento es "myhome" y, por tanto, todos los datos que lleven la clave myhome/ serán almacenados automáticamente. Finalmente, se establece que el tipo de almacenamiento sea en la memoria RAM y no en el disco, por tanto, en caso de reinicio, los datos se perderán. El motivo de la elección de uso de la RAM es para no saturar la memoria local con los datos del proyecto.

5.1.6 COMPOSICIÓN EN DOCKER

Un punto fundamental dentro de la composición del proyecto es el uso de Docker. Docker es una plataforma abierta de software que permite desarrollar, lanzar y correr aplicaciones mediante el uso de contenedores [2]. Estos contenedores son entornos portables, ligeros y autosuficientes, en los cuales están contenidas todas las dependencias necesarias para para poder ejecutar dicha aplicación. Dentro de los componentes de Docker, se pueden destacar los tres con mayor relevancia para el proyecto. Estos son:

- Imágenes Docker: las imágenes Docker son plantillas inmutables en las cuales están contenidos tanto las instrucciones de montaje de los contenedores como el sistema de archivos necesario para la construcción del mismo.
- Contenedores Docker: son cada una de las instancias de ejecución creadas en base a las imágenes.
- Docker Engine: es el motor principal que construye, prepara y ejecuta los contenedores.

El uso de Docker presenta una serie de ventajas para el proyecto que son de gran relevancia. En primer lugar, la disponibilidad para poder ejecutar Docker Desktop desde cualquier sistema operativo. El proyecto se ha desarrollado con sistema operativo Windows, y, al contrario que otros programas, presenta las mismas facilidades para operar con él que con otros sistemas más centrados en la programación, como puede ser Linux. El hecho de ejecutar los contenedores de forma aislada, además, permite que no existan interferencias entre los programas, y, por tanto, los errores sean más fácilmente localizables, interpretables y solucionables. Respecto a las máquinas virtuales, los contenedores son más ligeros, debido a que comparten el kernel (el núcleo, parte fundamental del sistema operativo que se encarga de conceder el acceso al hardware de forma segura para todo el software que lo solicita [29]) del sistema operativo. Asimismo, es muy sencillo integrarlo con herramientas del tipo CI/CD, como puede ser GitHub (lugar en donde se encuentra el repositorio de ZENOH).

Sin embargo, el factor fundamental para el uso de Docker es la escalabilidad que plantea. Al poder construirse los contenedores de forma independiente, solamente ha sido necesario programar el código base del actuador y del sensor, y, en lugar de tener cinco archivos separados para los sensores de cada habitación, y otros cinco archivos para los actuadores, se tiene un único código madre a partir del cual, en cada contenedor, se construye el elemento de la habitación correspondiente (Ilustración 40). La escalabilidad que plantea, además de la modularidad (cada contenedor no deja de ser un módulo independiente) son vitales para el desarrollo del proyecto.

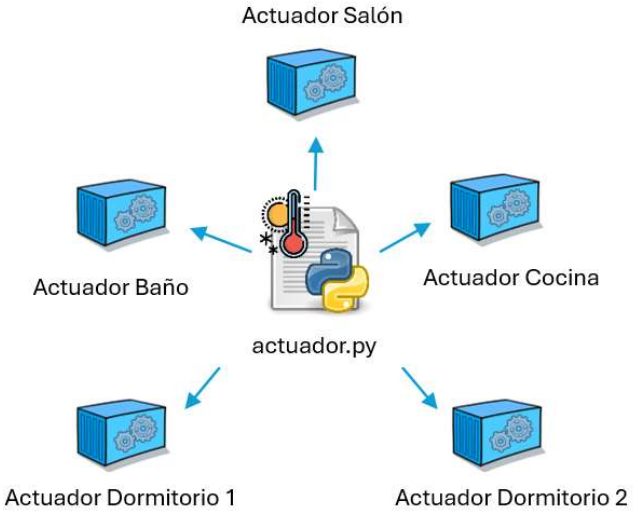


Ilustración 40. Desglose del script del actuador en 5 contenedores.

Además de todas estas ventajas que han sido citadas, ZENOH presenta una muy buena sinergia con Docker. Cada nodo de ZENOH puede construirse y desplegarse como un contenedor Docker, lo cual permite potenciar la escalabilidad que ya de por sí plantea ZENOH como protocolo. Además, la maleabilidad de ZENOH le permite coexistir dentro de los contenedores con otros servicios, como las APIs REST que han sido previamente mencionadas.

En resumen, el uso de Docker aporta simplicidad, compartimentación y escalabilidad al proyecto, y la sinergia que mantiene con ZENOH permite que sea fácilmente puesto en marcha.

5.1.6.1 Dockerfile

El Dockerfile, como se ha mencionado en cada uno de los apartados de los distintos elementos que componen el sistema, es el archivo básico que contiene las instrucciones a partir de las cuales se ha de construir una imagen personalizada (Ilustración 41). En este caso, es necesaria su redacción debido a que las imágenes para cada uno de los miembros del sistema (actuador, sensor, consola…) son personalizadas. Un Dockerfile contiene

información como la imagen base que se usará, los archivos que se deben copiar, los comandos a ejecutar, los puertos a exponer y las variables de entorno a definir.

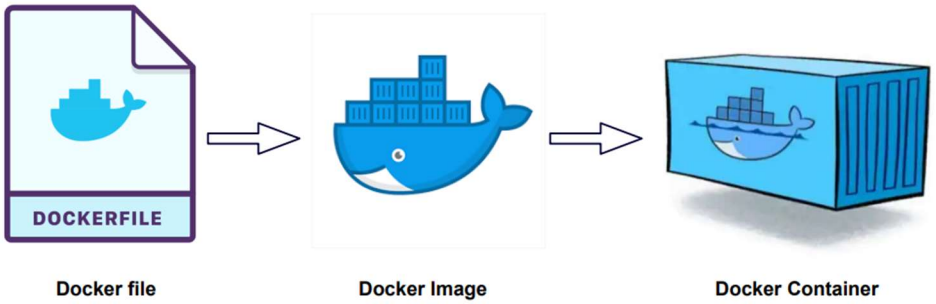


Ilustración 41. Proceso de creación de un contenedor [30].

A continuación, se muestra el Dockerfile básico que se ha diseñado para poder construir las imágenes de cada uno de los distintos contenedores que forman el sistema:

```
# Dockerfile para actuador

FROM python:3.10-slim

WORKDIR /app
```

Este Dockerfile se corresponde con el presente en la carpeta del actuador. El comando "FROM" es el que toma la imagen base de Python, en este caso, la versión 3.10-slim. A partir de esta imagen se construirá el resto de la imagen. "WORKDIR" crea el directorio de trabajo dentro del contenedor, cuyo nombre será /app.

```
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*
```

A continuación, se instalan las dependencias necesarias. Estas son:

- Build-essential: herramientas de compilació (como gcc) necesarias para compilar determinadas líneas de código en Python.

- Curl: necesario para descargar archivos a partir de URLs.
- Git: necesario para clonar repositorios de GitHub.
- Tcpdump: herramienta de análisis de tráfico de red y comunicaciones.

Finalmente, la última línea del comando limpia cachés y aligera la imagen.

```
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

ENV PATH="/root/.cargo/bin:$PATH"
```

Posteriormente, se descargan e instalan los paquetes de Rust y Cargo. Rust es un lenguaje de programación, distinto a Python, que es utilizado principalmente en el desarrollo de software y programación de alto nivel. Será necesario para compilar ciertas partes de la biblioteca zenoh-python, puesto que Python no es capaz. Por otro lado, Cargo es la herramienta encargada de gestionar los proyectos en Rust, como la compilación o la gestión de dependencias. Después de instalar ambos paquetes, se añade Cargo al PATH para que esté disponible de forma global dentro del contenedor, y pueda ser accesible durante la construcción y la ejecución del mismo.

```
COPY requirements.txt .
```

Ahora, se copia el archivo requierments.txt desde el sistema anfitrión. Este archivo, que se trata de un archivo básico de texto plano, contiene las librerías necesarias para el correcto montaje del contenedor. Los contenidos para los archivos requirements.txt tanto del actuador, el sensor como del proceso, son idénticos y son los siguientes:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
paho-mqtt
requests
```

Donde:

- Git+(…): es la dirección al repositorio de zenoh-python, repositorio el cual debe ser clonado.

- Paho-mqtt: es una biblioteca de Python para la implementación del protocolo MQTT, que puede ser útil junto con ZENOH.
- Request: es una biblioteca de Python que permite la comunicación con APIs de tipo REST.

Por otro lado, el archivo requirements.txt de la consola tiene los siguientes contenidos:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
python-multipart
fastapi
uvicorn[standard]
requests
```

Donde se vuelve a mencionar la dirección del repositorio de GitHub, y la librería request, y además se añaden:

- Python-multipart: es una biblioteca de Python que permite la gestión y manejo de formularios dentro de APIs web, como la que se usa con FastAPI.
- FastAPI: como se mencionó en la consola, se hace uso de FastAPI para construir la API web.
- Uvicorn: es un servidor ASGI (Asynchronous Server Gateway Interface), el cual será necesario para ejecutar correctamente las aplicaciones de FastAPI.

El cambio entre ambos archivos se debe a las diferentes necesidades que presenta cada uno.

```
RUN pip install --upgrade pip

RUN pip install -r requirements.txt
```

Continuando con el código, se instala y actualiza la herramienta PIP, que, al igual que Cargo en Rust, es el gestor de proyectos de Python, para posteriormente instalar todas las librerías contenidas dentro del archivo requierements.txt.

```
COPY ZENOH/zenohd.exe /app/zenohd

COPY actuador.py .

EXPOSE 8080
```

Más adelante, se copia el archivo ejecutable zenohd de la carpeta ZENOH, clonada en cada una de las carpetas. También se copia el archivo principal del programa, en este caso el archivo de código del actuador (en función del elemento, esta línea de código cambia). Para terminar, se expone el puerto en el que dicho contenedor va a trabajar. En este caso, actuador, sensor y proceso usarán el puerto 8080, mientras que la consola usará el puerto 8000.

```
CMD ["python", "actuador.py"]
```

Por último, se indica que, en cada contenedor basado en la imagen generada, se correrá el comando "python actuador.py" (o "sensor.py", "proceso.py" …), que pondrá en marcha la lógica del elemento. En el caso de la consola, esta línea de código cambia, y pasa a ser la siguiente:

```
CMD ["uvicorn", "consola:app", "--host", "0.0.0.0", "--port", "8000"]
```

A través de esta línea de código, se procede a la ejecución del servidor de Uvicorn en donde está montado FastAPI.

Por tanto, y a modo de resumen, el Dockerfile se encarga de instalar, copiar y generar todo aquello que sea necesario para la construcción de las imágenes de los contenedores.

5.1.6.2 Docker-compose.yml

El archive Docker-compose es la última herramienta de Docker que queda por analizar. Este archivo tiene como objetivo determinar cómo debe ser la construcción, la comunicación y la ejecución de todos los contenedores que se van a plantear. El proyecto cuenta con un total de 13 contenedores, que serán listados a continuación en la Tabla 2.

Tipo de contenedor	**Nombre del contenedor**
	actuador_salon
	actuador_cocina

Actuador	actuador_bano
	actuador_dormitorio1
	actuaor_dormitorio2
Sensor	sensor_salon
	sensor_cocina
	sensor_bano
	sensor_dormitorio1
	sensor_dormitorio2
Consola	consola
Proceso	proceso
Sniffer	sniffer

Tabla 2. Contenedores del sistema.

El contenido del archivo docker-compose.yml consta de una primera línea para determinar la versión a utilizar, en este caso, la versión 3.8, y posteriormente la definición de la red y los servicios.

En primer lugar, queda definida la red virtual que utilizará Docker para comunicar entre sí los distintos contenedores. Esta red se trata de una red interna que simula la conexión física entre los distintos elementos del sistema.

```
networks:
  zenoh-net:
    driver: bridge
```

Como se puede observar, la red construida es de tipo bridge. Este, que es el modo por defecto

de Docker, crea una interfaz de red virtual y asigna a cada uno de los contenedores una dirección IP única y privada en el interior de la misma. Cada contenedor entonces puede acceder al nombre de servicio del resto de contenedores para establecer una comunicación. Adicionalmente, la red solo permite a los contenedores que formen parte de ella el acceso a dicha información, y por tanto, crea un entorno aislado y libre de riesgos.

El sistema actual está basado en las conexiones multicast soportadas por unicast. En la red virtual que se creará para las conexiones entre elementos, se podrán encontrar dos niveles de comunicación (Ilustración 42):

- Primer nivel superior de multicast (host), donde los elementos realizarán el descubrimiento. El descubrimiento consiste en el envío de paquetes de datos multicast a un puerto concreto, "descubriéndose" así a otros nodos que puedan estar interesados en la conexión. Si un nodo está interesado en dicha conexión, entonces se establece el proceso de "handshake", y se inicia la conexión unicast.
- Segundo nivel inferior de unicast, donde las conexiones que se han determinado gracias al descubrimiento tienen lugar. Esto evita saturar la red, y filtrar a cada nodo lo que necesita.

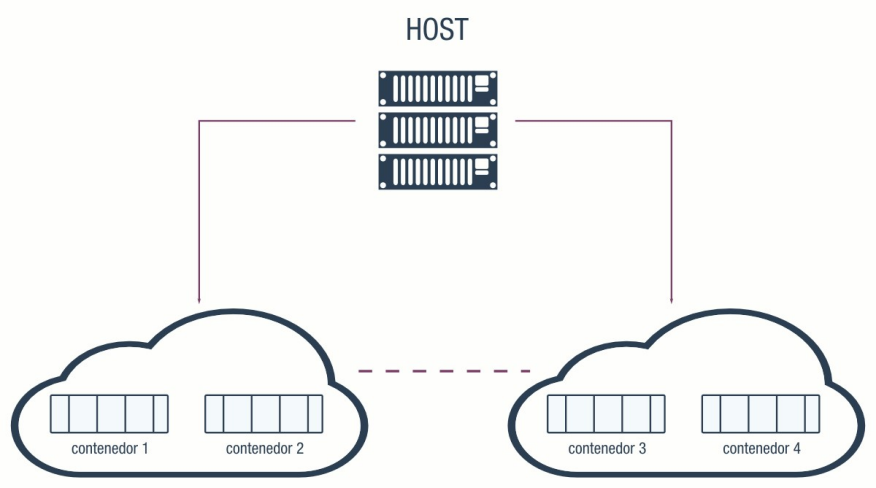


Ilustración 42. Esquema de la red Bridge creada [31].

Por otro lado, en cuanto a los servicios, cada uno de los contenedores tiene una instancia declarada que sigue un patrón idéntico en los casos del actuador y sensor, y muy similar en el caso del proceso y de la consola, con ligeras variaciones. Así pues, cada uno de los servicios presenta una instancia como la que será presentada a continuación:

```
sensor_salon:
    build:
      context: ./sensor
      dockerfile: Dockerfile
```

En primer lugar, se determina el nombre del contenedor. En este caso, el ejemplo es el del sensor del salón "sensor_salon". De la misma manera, se comienza con "actuador_salon" para el actuador del salón, "proceso" para el proceso o "consola" para la consola. Posteriormente, se inicia la construcción del contenedor. Para ello, se determinan dos puntos principales:

- Context: indica dónde se deben buscar los archivos necesarios para la construcción del contenedor (en este caso, al ser el sensor del salón, los archivos se encuentran en la carpeta del sensor).
- Dockerfile: se indica el nombre del archivo Dockerfile (en este caso, todos los archivos se llaman así)

```
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
```

Se procede a montar los volúmenes necesarios para permitir compartir los archivos entre el host local y los contenedores. En este caso, se comparte la carpeta zenoh-config (en donde se encuentra el archivo zenoh-myhome.json5) y la carpeta ZENOH (en donde se encuentra el archivo ejecutable zenohd).

```
    environment:
      - ROOM=salon
```

El comando "environment" permite definir las variables de entorno dentro del contenedor, para poder determinar la estancia en la que se encuentra. En este caso, al ser el sensor del

salón, ROOM = salon. En el caso del servicio del proceso, este bloque no está presente (ya que el proceso no es de una habitación, sino que está presente en todas).

```
    networks:
      - zenoh-net
    restart: always
```

Finalmente, estas dos últimas líneas de comando indican que el sensor estará conectado la red virtual (zenoh-net), a través de la cual todos los elementos podrán comunicarse entre sí, y que, en caso de reinicio del sistema o colapso del contenedor, se debe reiniciar siempre.

En el caso de la consola, adicionalmente, se añade este bloque de código:

```
ports:
      - "8000:8000"
```

Se expone el puerto 8000 del contenedor, lo cual permite el acceso de la API REST diseñada.

El docker-compose es, en resumen, el archivo que indica como deben construirse, conectarse y ejecutarse cada uno de los contenedores cuyas imágenes han sido definidas previamente.

Como se ha podido observar, esta primera fase ha consistido en un desarrollo en profundidad de cada uno de los diferentes elementos que van a formar parte del sistema de climatización del modelo de vivienda virtual. Sobre estas bases se edificarán el resto de las fases, y servirán de cimientos sólidos para el correcto desarrollo del proyecto. A partir de este punto, la lógica no será modificada, puesto que el funcionamiento es correcto, sino que se irán agregando las distintas capas que le faltan al código para su finalización.

5.2 FASE 2

Esta fase consiste, fundamentalmente, en la aplicación de métodos de seguridad y cifrado a las comunicaciones entre los distintos elementos. Para ello, se va a hacer uso de las posibilidades que otorga el propio protocolo ZENOH, y se añadirán dos capas de seguridad: autenticación por usuario y contraseña, y cifrado TLS. A continuación, se procederá a

detallar en qué consiste cada uno de los dos métodos empleados, analizando el qué son en sí mismos, qué aportan al sistema y cómo se han implementado.

5.2.1 AUTENTICACIÓN POR USUARIO Y CONTRASEÑA

Los factores de autenticación son los métodos que se utilizan para verificar que la identidad de un usuario a la hora de acceder a una cuenta o a un sistema es la correcta, y se tiene autorización para ello [32]. Existen cinco tipos de factores de autenticación [33], y según el tipo de factor, se pueden agrupar de la siguiente manera (Ilustración 43):

- Algo que se sabe: Es el factor de autenticación más común, y está basado en el conocimiento que debe tener el usuario y únicamente el usuario. Este tipo incluye contraseñas, preguntas de seguridad y números de identificación personal. Son los más fáciles de implementar, y también los más sencillos de corromper, por tanto, suelen complementarse con otro tipo de factores.
- Algo que se tiene: Son factores basados en la posesión de elementos físicos a los que solo debería tener acceso un usuario legítimo. Son mucho más fuertes que los factores basados en el conocimiento, pero siguen siendo vulnerables a robos o extravíos.
- Algo que se es: Basados en la herencia, los factores biométricos derivan de los rasgos fisiológicos del usuario, como pueden ser las huellas dactilares, el reconocimiento facial o los escaneos de iris. Son sencillos de utilizar y presentan un nivel de seguridad altamente elevado.
- Algo que se hace: Son los más modernos y novedosos, y se basan en el comportamiento. Acciones como la dinámica de pulsación de las teclas del teclado o los patrones en el movimiento del ratón pueden ser relevantes en los modelos de autenticación multifactorial, junto con otros factores de autenticación más tradicionales
- El lugar en el que se está: este tipo de factores está basado en la ubicación del usuario que trata de acceder al sistema. Pueden otorgar acceso solo en caso de estar en una

localización geográfica específica, o filtrar por direcciones IP válidas. Suelen actuar como factores de autenticación complementarios.

Ilustración 43. Esquema de los factores de autenticación.

Como se ha mencionado, los factores de autenticación más comunes son los pertenecientes al grupo de "algo que se sabe", y, dentro de ese grupo, la autenticación por usuario y contraseña es, sin lugar a dudas, el método de seguridad más utilizado en el panorama actual. La forma más común de implementar este factor de autenticación es mediante la identificación del usuario por el nombre que se haya establecido para él, sumado a unas credenciales que han sido previamente acordadas entre el usuario y el sistema. Estas credenciales componen la contraseña. En otras palabras, se trata de comprobar que se es quién se dice ser, y, para ello, se exige la demostración de que se conocen las credenciales que se deberían conocer.

Para poder implementar este factor de autenticación correctamente, previamente se debe haber realizado un registro del usuario, en donde dicho usuario escogerá sus credenciales: su nombre de usuario, y la contraseña que deberá emplear. Posteriormente, se establece un inicio de sesión, donde el usuario debe utilizar el nombre y contraseñas escogidos para que

se complete su identificación, y pueda acceder a los servicios ofrecidos por el sistema. La mayoría de redes sociales (Facebook, Instagram, Snapchat…) y directorios de correo electrónico (Google, Outlook…) emplean este método de autenticación (Ilustración 44).

Ilustración 44. Inicio de sesión por usuario y contraseña en Google.

Aplicado a ZENOH, el usuario pasaría a ser cualquiera de las entidades que componen el sistema (router, peer o cliente). Dicha entidad deberá demostrar su identidad ante cualquiera de las otras entidades con las que desee establecer una comunicación. Por lo tanto, la comunicación entre las dos entidades se da de la siguiente manera (Ilustración 45):

- Comienzo de la conexión: el cliente, por ejemplo, trata de establecer una conexión con otra entidad, como el router. Se inicia entonces el handshake.
- Handshake: del término en inglés para "apretón de manos", el handshake en un protocolo es una práctica que permite a los elementos de un sistema establecer los parámetros de la comunicación que se va a dar entre ellos [34]. En este caso, el router, que cuenta con la función de autenticación habilitada, preguntará al cliente por sus credenciales.
- Validación: el router entonces compara si la dupla usuario/contraseña entregados se corresponde con alguna de las que tenga archivadas como válidas en el diccionario. Si todo es correcto, se acepta la autenticación.

- Creación de la sesión: en caso de que haya sido correcta la autenticación, se inicia la sesión para el intercambio de información entre ambas entidades. En caso contrario, se rechaza la conexión

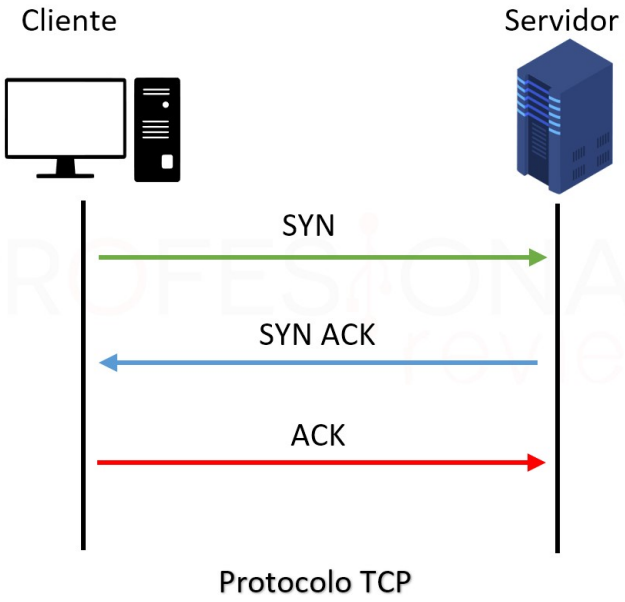


Ilustración 45. Esquema de comunicaciones TCP [35].

La contraseña es transmitida como parte del handshake propio del protocolo ZENOH. Sin embargo, si no se ha especificado de otra forma, este contenido no está cifrado: esto quiere decir que, en caso de una interferencia maliciosa desde el exterior, se podría tener acceso a la información de dicho handshake, y obtener las credenciales de usuario y contraseña de las entidades implicadas. Es por ello por lo que ZENOH ofrece medidas para el cifrado de dichos canales de comunicación. En el desarrollo de esta fase, se aplicará el cifrado TLS como medida de refuerzo de seguridad.

5.2.2 CIFRADO TLS

Transport Layer Security, conocido por sus siglas TLS, es un protocolo de seguridad y cifrado diseñado para facilitar la privacidad de los datos en comunicaciones [36]. TLS garantiza encriptación de los datos, autenticación de las partes e integridad en el contenido. Mediante un método de cifrado simétrico, dicho protocolo garantiza la seguridad en el

contenido de los mensajes frente a posibles interferencias externas, además de asegurar que el contenido no ha sido modificado durante su trayecto [37]. Para ello, TLS emplea certificados digitales. Los certificados TLS son documentos emitidos por una autoridad de certificación de confianza (CA), y contiene la clave pública del servidor e información relevante sobre el dominio.

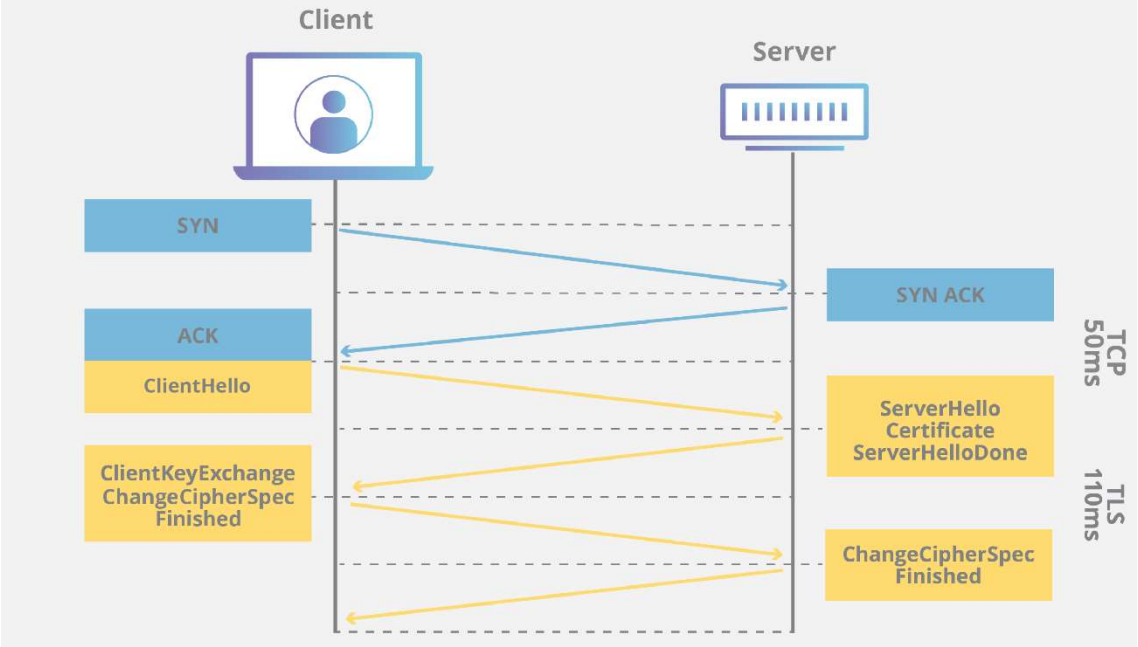


Ilustración 46. Proceso de handshake con TLS [37].

Las conexiones funcionan de la siguiente manera (Ilustración 46):

- En primer lugar, se inicia el enlace entre el servidor y el cliente. Durante este enlace, se especifican la versión de TLS a utilizar (para el caso que acontece, será la versión 1.2) y las suites de cifrado, se autentica la identidad del servidor mediante el certificado TLS del mismo, y se generan las claves de sesión para la encriptación de los mensajes.
- Se establece un conjunto de algoritmos de cifrado para generar tanto claves de encriptación como claves de sesión. Las claves de encriptación son cadenas de caracteres que tienen la intención de alterar los datos de forma que acaben pareciendo aleatorios [38]. Se sigue la siguiente fórmula:

$$texto\ plano + clave = texto\ cifrado$$

Un ejemplo clásico de claves de encriptación es el Cifrado César (Ilustración 47). Denominado así por el dictador romano Julio César, que lo utilizaba para encriptar sus mensajes militares, consiste en el desplazamiento en una dirección de las letras del abecedario un número determinado de puestos, denominado "rotación" [39]. Por ejemplo: la palabra "hola" con un cifrado César de rotación 1 quedaría como "ipmb". Posteriormente, para su desencriptado, se realiza la tarea inversa: se resta la clave al texto cifrado, obteniendo el texto original. Por otro lado, las claves de sesión tienen como fin encriptar la sesión de comunicación, como su propio nombre indica [40]. Estas claves son temporales y de un único uso y se usará para encriptar y desencriptar los datos enviados.

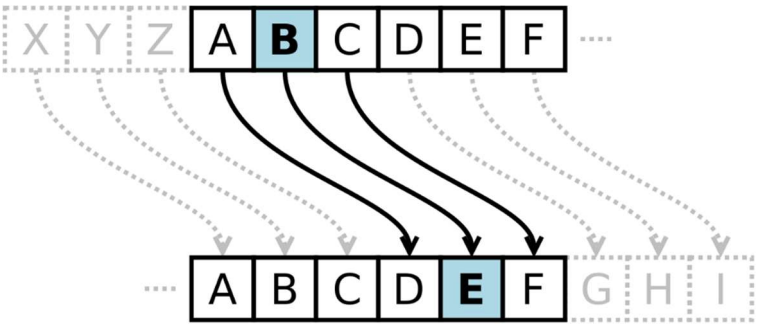


Ilustración 47. Representación del Cifrado César [41].

- Se lleva a cabo el proceso de autenticación (Ilustración 28) usando las claves públicas del servidor, las cuales son claves unidireccionales con las que decodificar los datos encriptados por la clave privada. Una vez los datos son encriptados y autenticados, se firma un MAC (Message Autentication Code), que actúa como el sello de lacre en una carta: el cliente sabe que la carta no se ha abierto, porque el sello no está roto.

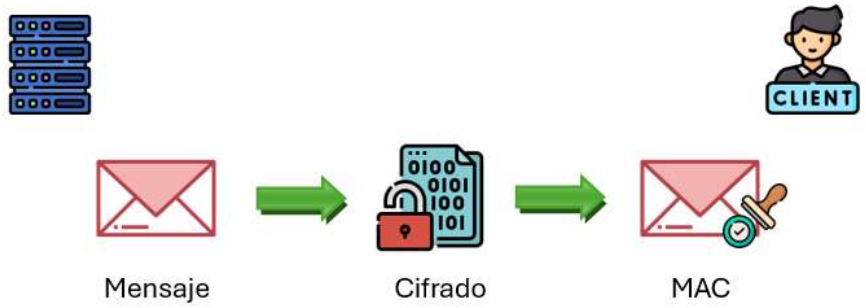


Ilustración 48. Esquema del proceso de cifrado TLS.

Para poder llevar a cabo tanto la autenticación como el cifrado, se deben realizar dos procesos distintos. En primer lugar, se deberán generar todas las claves y credenciales necesarias para el correcto funcionamiento de los factores de autenticación y TLS. En segundo lugar, se deberán actualizar los scripts de todos los elementos para que tengan en cuenta esta nueva configuración, así como la modificación tanto de los archivos docker-compose y los archivos json de configuración.

5.2.3 GENERACIÓN DE CLAVES

Como se ha mencionado, a la hora de realizar cualquier tipo de autenticación, cifrado, encriptado, o, en general, cualquier método de ciberseguridad, es necesario tener credenciales. Estas credenciales deben ser, por un lado, fiables, seguros y de confianza, mientras que por otro deben tener la capacidad de estar soportados dentro del sistema.

En este proyecto, se necesitarán los siguientes archivos de certificados:

- Certificado de Autoridad (CA).
- Certificado TLS para el servidor (en este caso, el router ZENOH).
- Certificado TLS para el cliente (sensores, actuadores…).
- Archivo de credenciales (nombre de usuario y contraseña).

Se ha hecho uso de la biblioteca de software de código abierto OpenSSL [42], que ayuda a aplicar protocolos de seguridad, como puede ser TLS. Permite la generación de claves públicas y privadas (como las que serán utilizadas en este proyecto) y la firma y verificación

de datos. Para generar estos certificados, se ha creado un script gen_certs.sh, programado en Bash. Adicionalmente, se ha creado un archivo test_certs.sh para comprobar la validez de dichos certificados (puede ser consultado en el ANEXO II: Código de la Fase II, junto con el resto de códigos de la Fase 2). A continuación, se procederá a la explicación del contenido del script de generación de credenciales.

```
export MSYS_NO_PATHCONV=1
set -e
mkdir -p certs
cd certs
rm -f *.pem *.csr *.txt *.srl

echo " Generando certificados TLS para Zenoh"
```

En primer lugar, se prepara un entorno seguro para la generación de estos certificados. MSYS_NO_PATHCONV evitará problemas en la traducción de rutas (Bash suele ser utilizado en Linux, y este proyecto está construido en Windows), mientras que set -e detendrá el script en caso de errores. A continuación, se crea el directorio "certs" (en caso de no estar creado) y se avanza dentro de dicho directorio, para, con el comando rm, eliminar los archivos que pueda contener dicha carpeta. Si todo ha sido correcto y el script se ha iniciado, se indica mediante la función echo (el equivalente en Bash al printf de Python) que el programa está corriendo.

```
echo " Generando clave privada de la CA"
openssl genrsa -out ca-key.pem 4096

echo " Generando certificado de la CA"
openssl req -new -x509 -days 3650 -key ca-key.pem -sha256 -out ca-cert.pem \
  -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=MyHomeCA"
```

El primer certificado generado será el CA. En primer lugar, se genera una clave RSA de 4096 bits con la cual se firmarán otros certificados. Posteriormente, el certificado se autofirma, actuando así como raíz de confianza. Será válido por 3650 días (10 años) y tendrá por nombre MyHomeCA.

```
echo " Generando clave privada del servidor"
openssl genrsa -out server-key.pem 4096
```

```
echo " Generando solicitud de certificado del servidor"
openssl req -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=zenoh-router" \
  -sha256 -new -key server-key.pem -out server.csr

cat > server-extensions.txt << 'EOF'
[v3_req]
subjectAltName = @alt_names
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[alt_names]
DNS.1 = zenoh-router
DNS.2 = zenoh
DNS.3 = localhost
DNS.4 = router
IP.1 = 127.0.0.1
IP.2 = 0.0.0.0
EOF

echo " Generando certificado del servidor"
openssl x509 -req -days 3650 -in server.csr -CA ca-cert.pem -CAkey ca-key.pem \
  -out server-cert.pem -extensions v3_req -extfile server-extensions.txt -
CAcreateserial
```

Para el router, el procedimiento es similar. En primer lugar, se genera la clave privada del servidor y se realiza una solicitud de firma para el certificado. A continuación, se crea el archivo de extensiones. Este archivo tiene como objetivo configurar los campos avanzados que no se incluyen de forma predeterminada al generarse el certificado, pero son igualmente esenciales. Estos definen la forma en la que puede ser utilizado el certificado, y así permitir que TLS pueda realizar verificaciones de identidad. Sin estas extensiones, podría no establecerse una conexión segura, pudiendo esta ser rechazada. En este archivo vienen recogidos los nombres que serán aceptados para el router (zenoh, localhost, zenoh-router y router) así como las IP válidas (0.0.0.0 y 127.0.0.1). También se indica que dicho certificado no es CA (no podrá usarse para firmar otros certificados), se determina su uso (comando keyUsage) y se determina que será utilizado en servidores TLS. Finalmente, se firma con el CA.

```
echo " Generando clave privada del cliente"
openssl genrsa -out client-key.pem 4096

echo " Generando solicitud de certificado del cliente"
openssl req -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=zenoh-client" \
```

```
  -new -key client-key.pem -out client.csr

cat > client-extensions.txt << 'EOF'
[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth
EOF

echo " Generando certificado del cliente"
openssl x509 -req -days 3650 -in client.csr -CA ca-cert.pem -CAkey ca-key.pem \
  -out client-cert.pem -extensions v3_req -extfile client-extensions.txt -
CAcreateserial
```

El proceso para los certificados de cliente es el mismo que para el del router, con un ligero cambio. En este caso, no se listan los diferentes nombres e IP de los clientes, puesto que es opcional, y, en caso de escalarse, se tendría que estar modificando constantemente el archivo generador de credenciales.

```
# 11. Crear archivo de credenciales
echo " Creando archivo de credenciales"
cat > zenoh_credentials.txt << 'EOF'
router:routerpass
sensor:sensorpass
actuador:actpass
sniffer:sniffpass
proceso:procpass
consola:conspass
EOF
```

Se crea ahora el archivo de credenciales con los usuarios y contraseñas del sistema. Como se puede comprobar, hay un nombre de usuario y una contraseña que se asociará a cada uno de los elementos del sistema.

```
rm -f server.csr client.csr server-extensions.txt client-extensions.txt

chmod 600 *-key.pem
chmod 644 *-cert.pem ca-cert.pem zenoh_credentials.txt

echo ""
echo "🔍 Verificando certificados"
echo "Verificando certificado del servidor:"
if openssl verify -CAfile ca-cert.pem server-cert.pem; then
    echo " Certificado del servidor: VÁLIDO"
else
    echo " Certificado del servidor: INVÁLIDO"
```

```
    exit 1
fi

echo "Verificando certificado del cliente:"
if openssl verify -CAfile ca-cert.pem client-cert.pem; then
    echo " Certificado del cliente: VÁLIDO"
else
    echo " Certificado del cliente: INVÁLIDO"
    exit 1
fi
```

Finalmente, se limpian los archivos temporales (las extensiones, por ejemplo, o las solicitudes) y se protegen las claves privadas, de forma que solo estén disponibles en modo lectura para le propietario, y permite a los servicios leer los certificados. Después, se procede a la verificación de ambos certificados, para comprobar que funcionan correctamente.

Por tanto, la carpeta de certificados (Ilustración 49) queda compuesta por los siguientes elementos:

- Ca-cert.pem
- Ca-cert.srl
- Ca-key.pem
- Client-cert.pem
- Client-key.pem
- Server-cert.pem
- Server-key.pem
- Zenoh-credentials.txt

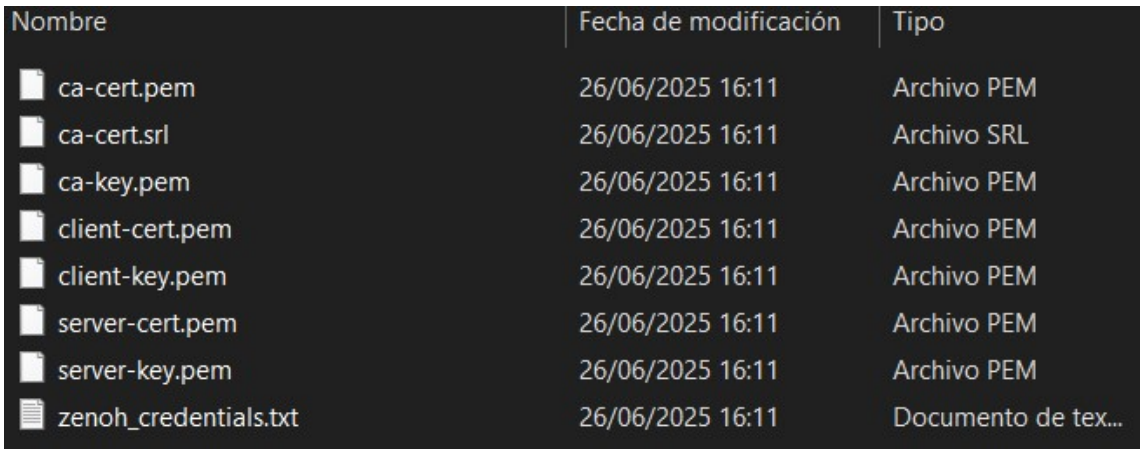


Ilustración 49. Contenidos de la carpeta "certs".

5.2.4 ACTUALIZACIÓN DE CÓDIGOS

Debido a la introducción de los certificados, se han realizado modificaciones imprescindibles en los scripts de todas las entidades que pertenecen al sistema. De la misma manera, se ha cambiado la configuración del router y de Docker, para poder hacer frente al cambio de paradigma.

5.2.4.1 Zenoh-myhome.json

Y es que, a la hora de aplicar el cifrado TLS a ZENOH, hay que realizar también un cambio importante en cuanto a la estructura de comunicaciones del proyecto. TLS en ZENOH está soportado para protocolos TCP. Esto quiere decir que se debe cambiar la modalidad de multicast soportada por unicast de la Fase 1 a unicast puro. Es por ello por lo que se realizarán varias modificaciones en el proyecto, las cuales tendrán que ver, principalmente, con la configuración del router y el archivo zenoh-myhome.json5.

El primer cambio es sencillo y no requiere demasiado detalle: se ha cambiado el tipo de archivo de json5 a json. El motivo es el mejor funcionamiento del mismo, y el hecho de dar menos errores a la hora de poner el sistema en marcha. En cuanto a los cambios dentro del propio archivo, son los siguientes:

```
"mode": "router",
  "plugins": {
    "rest": {
```

```
      "http_port": 8010
    },
```

En primer lugar, se indica que esta estancia de Zenoh tendrá el comportamiento de un router, y se definen los plugins que serán utilizados. Como en la fase 1, se seguirá usando el plugin REST, con la diferencia de que ahora el puerto de salida será el 8010.

```
"listen": {
    "endpoints": [
      "tcp/0.0.0.0:7447",
      "tls/0.0.0.0:7448"
    ]
  },
```

El bloque “listen” indica los dos puertos de entrada utilizados:

- 7447: para conexiones TCP no seguras.
- 7448: para conexiones TLS seguras.

```
"transport": {
    "unicast": {
      "accept_timeout": 10000,
      "accept_pending": 100,
      "max_sessions": 1000,
      "max_links": 1
    },
```

El siguiente bloque, “transport”, define el manejo de las conexiones de red a través de los apartados “unicast”, “link” y “auth”. En primer, como se ha mencionado, pasará a ser unicast puro. Contiene el tiempo máximo en ms para aceptar una conexión, el número máximo de conexiones simultáneas y pendientes, y el número máximo de conexiones por enlace.

```
"link": {
      "tls": {
        "root_ca_certificate": "/certs/ca-cert.pem",
        "listen_certificate": "/certs/server-cert.pem",
        "listen_private_key": "/certs/server-key.pem",
        "enable_mtls": true,
        "verify_name_on_connect": true
      }
```

Dentro del apartado "link" se encuentra la configuración TLS para las comunicaciones. En este bloque, se determina:

- Root_ca_certificate: la ruta que contiene el CA de confianza.
- Listen_certificate: la ruta que contiene el certificado público del router.
- Listen_private_key: la ruta que contiene la clave privada del router.
- Enable_mtls: activa el mTLS. El Mutual TLS es una extensión dentro de TLS en la que ambos, cliente y router, se autentican mutuamente con sus certificados digitales. De lo contrario, es solamente el cliente el que verifica la identidad del servidor.
- Verify_name_on_connect: se verifica que el nombre del cliente coincida con el nombre esperado.

```
"auth": {
      "usrpwd": {
        "user": "router",
        "password": "routerpass",
        "dictionary_file": "/certs/zenoh_credentials.txt"
      }
```

Finalmente, en el apartado "auth" se determinan tanto el usuario como la contraseña del router, y se implementa el directorio en donde se encuentran el resto de las credenciales.

Como se puede observar, los cambios en el archivo json han sido direccionados al soporte de TLS y la migración al modelo unicast.

5.2.4.2 Zenoh-client.json

Mientras que en la primera fase no era necesario, en esta segunda fase se va a implementar un archivo de json de configuración de clientes, al igual que se ha hecho con el router. Este archivo es más sencillo y compacto, y presentará la información necesaria para que los clientes puedan adoptar las nuevas condiciones del entorno de trabajo.

```
"mode": "client",
  "connect": {
    "endpoints": ["tls/zenoh:7448"]
  },
```

En primer lugar, se determina que el modo será del tipo cliente, y que se deberán conectar al puerto con endpoint 7448 (el puerto con TLS activo).

```
"transport": {
    "link": {
      "tls": {
        "root_ca_certificate": "/certs/ca-cert.pem",
        "connect_certificate": "/certs/client-cert.pem",
        "connect_private_key": "/certs/client-key.pem",
        "enable_mtls": true,
        "verify_name_on_connect": false
      }
    }
  }
```

El otro bloque presente dentro del archivo es el bloque “transport”, el cual, al contrario que en el caso del router, solo presenta el apartado “link”. Dentro están determinados los mismos parámetros que en su predecesor, con dos únicas variaciones: connect_certificate y connect_private_key tiene como objetivo indicar la ruta donde se encuentran los certificados para la correcta autenticación del cliente. Además, se omite la verificación del nombre del servidor. Este cambio se debe a que, como el entorno es cerrado, no se corre ningún riesgo de intrusión, mientras que facilita la búsqueda y depuración de errores.

El archivo zenoh-client.json, aunque sencillo, contiene la configuración necesaria para ser implementada en los elementos clientes del sistema.

5.2.4.3 Scripts

La implementación de autenticación y TLS requiere una adición de código a los scripts base planteados para la Fase 1. Dado que las adiciones serán las mismas para todas, solo se analizarán los cambios en uno de ellos.

```
import logging

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)
```

Primeramente, se importa la librería “logging” para poder crear un logger que muestre los

mensajes del código en la consola. Será muy útil a la hora de comprobar que los certificados funcionan.

```
def create_console_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"consola"')
    config.insert_json5("transport/auth/usrpwd/password", '"conspass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"
```

Ahora, se determina la configuración de la sesión, en este caso de la consola. Este es el cambio más significativo con respecto a la Fase 1, puesto que ahora la configuración dejará de ser la predeterminada para ser personalizada. Se crea un objeto de configuración al que se le indica que se trabajará en modo cliente, y posteriormente se indican las credenciales de la entidad. A continuación, se identifica si el TLS se encuentra activo. En caso contrario, utilizará conexiones TCP. El motivo de la adición de esta línea de comando es marcar un checkpoint para la correcta identificación y depuración de errores. Durante el desarrollo de esta fase, como es habitual, se dieron numerosos errores. Con la intención de saber si se debían al TLS o a otros motivos (conexiones, fallos en el router o los clientes, etc..) se integró esta línea para que el sistema funcionase aún sin TLS. De esta forma, si los errores seguían dándose, no tendrían que ver con dicho protocolo.

```
if tls_enabled:
        logger.info("TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
        logger.info("TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)
```

En caso de que tls_enabled tenga valor “true”, se procederá a la habilitación del TLS. Se cargan los certificados necesarios, como se ha visto en el archivo zenoh-client.json. En caso de que el valor sea “false”, se informará mediante un mensaje en pantalla, y se cambiará el endpoint al 7447 (el que ha sido habilitado para las conexiones TCP planas).

```
def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error(" Faltan certificados TLS:")
        for m in missing:
            logger.error(f" - {m}")
        return False
```

Ahora, se procede a la verificación de los certificados. Como se ha comentado antes, uno de los principales errores que se han dado en esta fase han tenido relación con el TLS. Una de las partes fundamentales de TLS es que los certificados tengan validez, y por ello se ha hecho especial hincapié en la fiabilidad de los mismos. Es por ello que cada uno de los distintos elementos cuenta con una función que verifique que los certificados estén presentes en la carpeta correspondiente, y que no estén vacíos. Cualquiera de estos dos casos daría lugar a un fallo en las comunicaciones, y de cara a rastrear los errores en el TLS, es especialmente útil comprobar el estado de los certificados.

En primer lugar, se define la lista de certificados que deberían existir. A continuación, la función “missing” identifica que no falte ninguno de los certificados de la lista.

```
for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f"El archivo {cert_file} está vacío")
                    return False
```

```
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f"El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
            logger.error(f"Error leyendo {cert_file}: {e}")
            return False
```

A continuación, para cada uno de los certificados, se verifica que estos no se encuentren vacíos, y, adicionalmente, que tengan el formato válido. Se informa del certificado corrupto en caso de ocurrir un error.

Esto finaliza las actualizaciones realizadas a los scripts en general, para poder aplicar TLS. Sin embargo, en la consola se han realizado modificaciones adicionales que tienen como objetivo la correcta identificación, aislamiento y depuración de los errores que se han ido produciendo durante el desarrollo de la Fase 2. Se procederá entonces al resumen detallado de las mencionadas modificaciones:

```
connection_status = {
    "connected": False,
    "tls_enabled": os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true",
    "last_error": None
}
```

Se crea un diccionario que informa en caso de que la consola se haya conectado correctamente a ZENOH (originalmente está establecido como “false”), si el TLS ha sido activado, y el último error en caso de haberlo (predeterminado como “none”). El objetivo de este diccionario vuelve a ser la identificación de los errores y determinar el fallo de los mismos.

```
try:
    zsession = create_console_session()
    connection_status["connected"] = True
    logger.info("Consola MyHome iniciada correctamente")
except Exception as e:
    logger.error(f" Error crítico al inicializar: {e}")
    connection_status["last_error"] = str(e)
    zsession = None
```

Ahora, se tratará de crear una sesión de ZENOH denominada "zsession", de acuerdo con la configuración previamente mencionada. En caso de que todo funcione correctamente, se creará la sesión. En caso de que algo falle, se informará del error y no se creará la sesión.

```
@app.get("/status")
async def get_status():
    """Endpoint para verificar el estado de la conexión y configuración"""
    return {
        "status": "online" if connection_status["connected"] else "offline",
        "tls_enabled": connection_status["tls_enabled"],
        "last_error": connection_status["last_error"],
        "habitaciones": habitaciones,
        "zenoh_session": zsession is not None
    }
```

Posteriormente, se crearán distintos endpoints adicionales para verificar el estado del sistema. El primero de ellos (Ilustración 50) tiene como objetivo devolver un informe sobre el ya mencionado estado general del sistema. Muestra la siguiente información:

- Status: online y offline en caso de que se haya o no conectado correctamente.
- Tls_enabled: true o false en función de si TLS está o no habilitado.
- Last_error: informa del último error, en caso de haberlo.
- Habitaciones: lista las habitaciones que están levantadas.
- Zenoh_session: true o false en función de si se ha creado la sesión o no.

Estos datos de diagnóstico se muestran en la interfaz web (Ilustración 51).

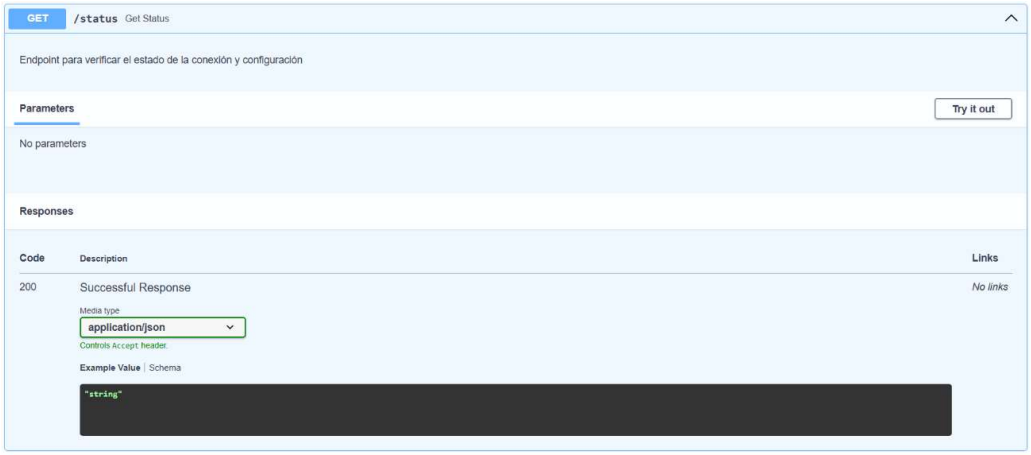


Ilustración 50. Endpoint GET de verificación de Status.

Ilustración 51. Respuesta al GET del Status.

```python
@app.get("/health")
async def health_check():
    """Endpoint de verificación de salud del servicio"""
    return {
        "status": "healthy" if zsession else "unhealthy",
        "tls": connection_status["tls_enabled"],
        "zenoh_connected": connection_status["connected"]
    }
```

Este segundo endpoint (Ilustración 52) se encarga de la verificación del estado de salud del sistema. Devuelve lo siguiente (Ilustración 53):

- Status: healthy o unhealthy en función de si hay o no una sesión activa.
- Tls: true o falte en función de si está o no habilitado TLS.
- Zenoh_connected: true o false en función de si se pudo o no conectar al router.

Este endpoint también se mostrará en la interfaz web. Y, al igual que el endpoint "status" y el resto de las funciones de verificación presentes en el código, tiene como objetivo facilitar la identificación y solución de los posibles errores del sistema.

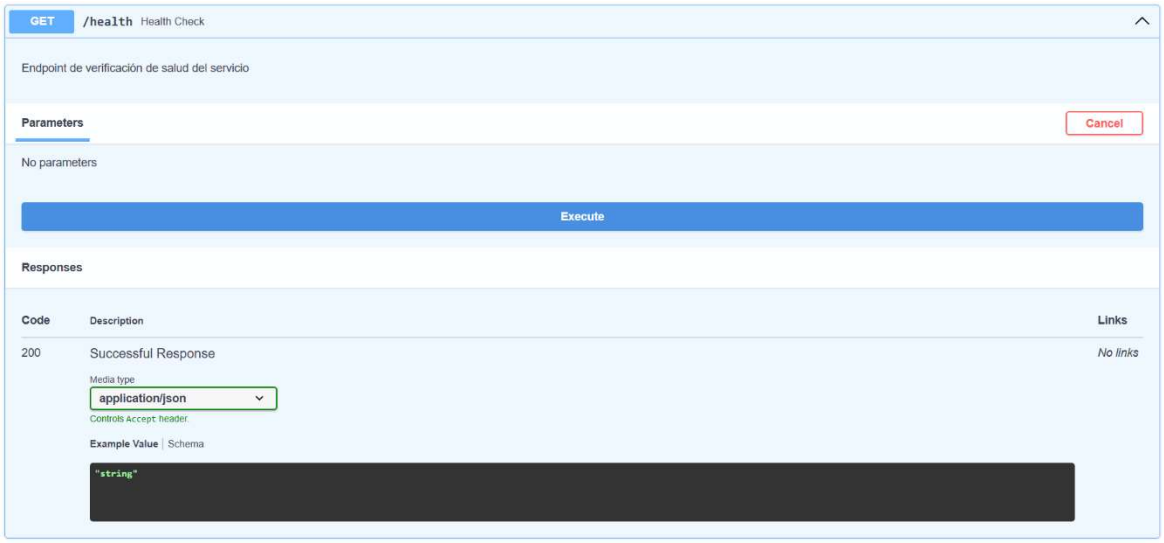


Ilustración 52. Endpoint GET para el estado de salud.

Ilustración 53. Respuesta del GET con la salud del sistema.

La nueva interfaz de la consola queda, por tanto, de la manera indicada en la:

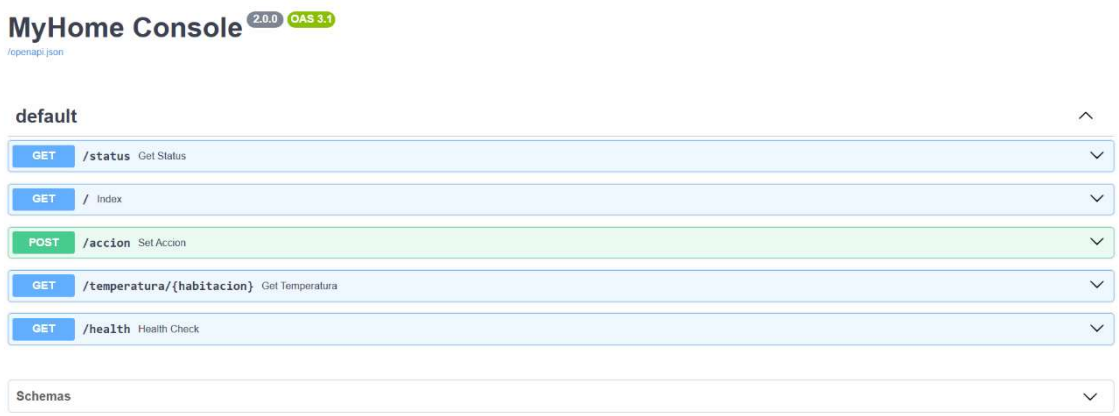


Ilustración 54. Interfaz de consola generada en la Fase 2.

5.2.4.4 Docker-compose

Finalmente, el último archivo que se ha debido modificar es el docker-compose.yml. Los cambios en los contenedores son sencillos, y alteran muy poco el archivo, pero es relevante informar de ellos. Asimismo, también se ha añadido un nuevo contenedor para el router ZENOH. Se comenzará analizando los cambios en los contendores ya existentes. Tomando como referencia el contenedor de la consola, observamos que los contenidos son los siguientes:

```
consola:
    build:
      context: ./consola
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=consola
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    ports:
      - "8010:8010"
    restart: always
    depends_on:
      - zenoh
    ulimits:
```

```
    nofile:
      soft: 4096
      hard: 4096
```

En primer lugar, en el apartado de volúmenes, se añade el directorio para el acceso a los certificados y credenciales. Posteriormente, en el entorno se añaden las opciones tanto de TLS habilitado, como el directorio del archivo correspondiente a la configuración del cliente (zenoh-client.json). El puerto cambia a ser el 8010, para diferenciarlo del puerto de la Fase 1, y finalmente, se añade un nuevo apartado, ulimits, que permite modificar el límite máximo de archivos abiertos por el sistema, evitando así cuellos de botella. El motivo de añadir este apartado ha sido la gestión de los certificados. Como se ha visto a lo largo de toda esta sección que engloba la Fase 2, se han añadido numerosos métodos de identificación y comprobación de errores, para facilitar la depuración de los mismos. El principal problema que se obtenía, con respecto al estado de los certificados, era la invalidez de los mismos. Tras comprobarse que estos eran válidos según los métodos de verificación TLS, se cambió el foco de mira a la posibilidad de que dichos archivos no fuesen válidos debido a que no podían ser accedidos por el sistema. Al incrementar la cantidad de archivos que este puede mantener abiertos, los errores cesaron. Por tanto, para cada uno de los contenedores, se determinó incrementar ese límite. Para el resto de los contenedores, los cambios en volúmenes y entorno son iguales.

La otra modificación se ha dado con la adición de un nuevo contenedor para el router. Previamente, en la Fase 1, las conexiones eran en multicast, soportado por unicast. Esto quiere decir que, aunque la comunicación entre los contenedores era unicast, todos los contenedores se encontraban unidos unos a otros, y se establecía la conexión de acuerdo con lo publicado en la red multicast. Sin embargo, el hecho de que ahora el método de comunicación sea unicast puro, con conexiones entre cliente y router, obliga a la creación de un contenedor específico para dicho router. En este modelo de red, todos los contenedores se conectan al router, y es el router el que reenvía la información en función de las necesidades del sistema (Ilustración 55).

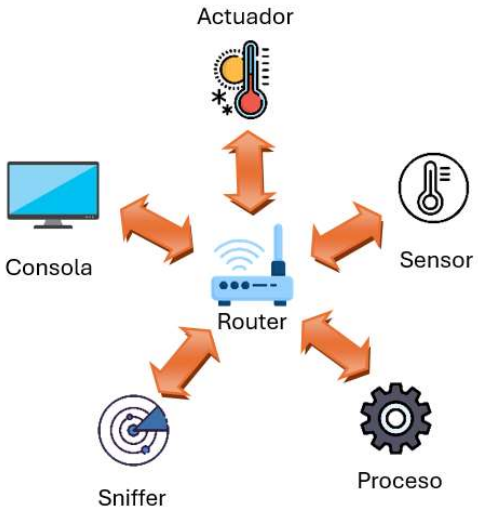


Ilustración 55. Esquema de conexiones del router.

El contenedor añadido al archivo Docker Compose tiene la siguiente forma:

```
services:

  zenoh:
    image: eclipse/zenoh:latest
    container_name: zenoh-router
    restart: unless-stopped
    ports:
      - "7447:7447"
      - "7448:7448"
      - "8011:8010"
```

En primer lugar, el router no tiene Dockerfile, por tanto, su imagen es directamente la imagen oficial eclipse/zenoh. La extensión “latest” indica que se debe usar la versión más reciente. Se define también el nombre del contenedor (zenoh-router) y se consigna a reiniciarse siempre, a menos que sea detenido. En cuanto a los puertos, observamos tres puertos distintos. Como se ha mencionado previamente, el puerto 7447 estará encargado de soportar las conexiones TCP planas, mientras que el puerto 7448 soportará las conexiones TLS. El puerto 8010 será el que se utilice internamente para correr la API REST, sin embargo, este es expuesto como 8011 al host para evitar posibles confusiones.

```
    volumes:
      - ./seguridad/certs:/certs:ro
```

```
      - ./zenoh-config:/root/.zenoh:ro
    environment:
      - RUST_LOG=debug
      - ZENOH_LOG=debug
    command: ["-c", "/root/.zenoh/zenoh-myhome.json"]
```

En cuanto a los volúmenes, se monta el directorio de los certificados, habilitado exclusivamente en modo lectura para el host, y el directorio donde se encuentra la configuración del router (zenoh-myhome.json). La línea "command" indica el archivo que se debe cargar como configuración, mientras que en el entorno se activa el logging detallado, para poder notificar y depurar errores correctamente.

```
networks:
      zenoh-net:
        aliases:
          - zenoh-router
          - zenoh
          - router
    ulimits:
      nofile:
        soft: 65536
        hard: 65536
    sysctls:
      - net.core.somaxconn=1024
      - net.ipv4.tcp_max_syn_backlog=1024
```

Por último, se crean diversos alias para la red, para facilitar las conexiones entre los distintos elementos del sistema, se aumenta la cantidad de archivos abiertos permitidos, y se añaden un par de parámetros del kernel con intención de mejorar el manejo de las conexiones entrantes.

En resumen, en la Fase 2 se ha desarrollado todo el entramado de capas de cifrado y conexiones seguras. Para ello, se ha implementado la autenticación por usuario y contraseña y el protocolo de seguridad TLS. Adicionalmente, se ha experimentado con el modo de conexión unicast, y se ha planteado el asunto de la depuración de errores desde un punto de vista exhaustivo en la búsqueda del error.

5.3 FASE 3

La Fase 3 es la última de las fases de desarrollo, y su principal objetivo es escalar el proyecto. El objetivo es claro: se debe aumentar el tamaño del modelo de vivienda virtual de las cinco habitaciones básicas a números más elevados, para poder comprobar el funcionamiento y los límites del sistema (Ilustración 56). En otras palabras, se pondrá a prueba la escalabilidad de ZENOH.

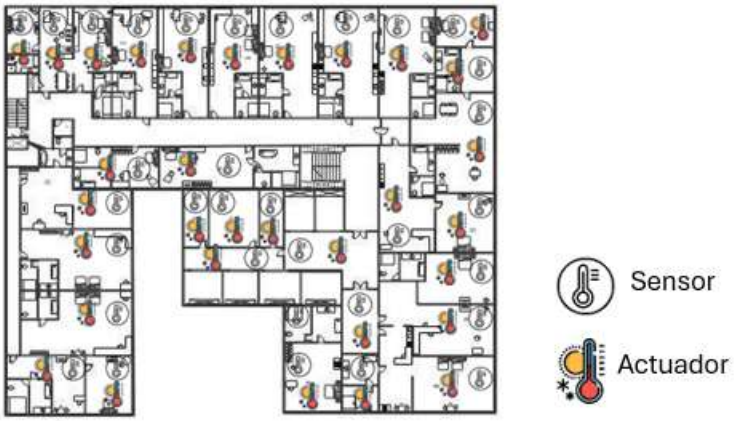


Ilustración 56. Imagen de la vivienda escalada.

Para realizar este proceso de escalado, se elevará progresivamente el número de habitaciones de la vivienda: se pasará de las 5 habitaciones iniciales a 50 habitaciones, incrementando en 5 habitaciones cada instancia hasta el fallo de alguno de los modelos. Con el fin de facilitar la tarea de análisis de resultados, se establecerán tres puntos relevantes durante el camino del escalado: 10 habitaciones, 25 habitaciones y 50 habitaciones. En estos puntos se detallará el análisis de la respuesta del sistema (Ilustración 57). Además, se realizará este tipo de escalado tanto para el modelo de vivienda creado en la Fase 1 (multicast soportado por unicast) como para el modelo de la Fase 2 (unicast con cifrado y autenticación).

Ilustración 57. Esquema del proceso de escalado del sistema.

Pese a que esta fase consiste únicamente en el aumento de tamaño de los modelos creados en las fases previas, ha sido necesario realizar ciertas modificaciones para asegurar el correcto funcionamiento del sistema, principalmente en cuanto a temas de eficiencia. En primer lugar, tanto en la consola como en el proceso se ha modificado el código del script de Python para cambiar el método de determinación de habitaciones. De esta línea de código:

```
habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]
```

Se ha pasado a la siguiente:

```
import json

with open("habitaciones.json", "r") as f:
    habitaciones = json.load(f)
```

Primero, se importa la librería “json” para poder leer archivos de dicho tipo. A continuación, se crea un bucle que carga en la variable “habitaciones” todas las habitaciones que se encuentren dentro de dicho archivo json. De esta forma, se evita tener que modificar manualmente el código del proceso o de la consola para añadir las nuevas habitaciones a la lista.

Este archivo habitaciones.json ha sido generado de forma automática junto con un archivo docker-compose.yml gracias a un script de python que se ha programado para, dado un

número N de habitaciones, generar el listado de las mismas (habitacion1, habitacion2, … habitacionN), y generar el archivo docker-compose de acuerdo a como se ha explicado en cada una de las fases. De esta forma, los contenedores para los sensores y actuadores de las nuevas habitaciones son añadidos de forma automática, sin necesidad de añadirlos manualmente. No se entrará en detalle sobre dicho archivo, puesto que su lógica consta de la impresión de los modelos que se han seguido para cada uno de los servicios, y, en los actuadores y sensores, un bucle que recorre todas las habitaciones para generar los correspondientes contenedores. Sin embargo, el código puede consultarse en el ANEXO III: Código de la Fase III.

A modo de resumen del capítulo, el proyecto tiene como objetivo la construcción de un modelo de vivienda virtual sobre el que poder aplicar el protocolo ZENOH, con el fin de observar su funcionamiento y puesta en marcha. Si bien las fases están diferenciadas, cada una de ellas depende de la anterior. En la primera fase, se experimentó con el modo multicast de ZENOH, construyendo un modelo de vivienda con cinco habitaciones, que cuenta con sensores, actuadores, consola y un proceso que simula la temperatura interna de la vivienda. Con la ayuda del software de Docker, se lanzó dicha vivienda, separando cada elemento del sistema en un contenedor para poder estudiar posteriormente sus conexiones. En la segunda fase, se añadió una capa de seguridad basada en autenticación por usuario y contraseña, y cifrado TLS en las conexiones. Se evolucionó en la depuración de errores y se migró el sistema de conexiones multicast a unicast puro. Por último, en la tercera parte se procedió al aumento del número de habitaciones de ambos modelos de vivienda (multicast y unicast), para poder realizar una comparativa sobre la capacidad de escalado de cada modelo.

Capítulo 6. ANÁLISIS DE RESULTADOS

En este capítulo se procederá al análisis detallado de las conexiones obtenidas entre todos los elementos miembros del sistema. Se explicarán los métodos con los que se han obtenido los datos, el software utilizado para analizarlos, y el análisis por fases de las comunicaciones. Durante el análisis de la Fase 3, se realizará una comparativa entre los resultados de los dos distintos modelos de vivienda para cada uno de los niveles de escalado. Adicionalmente, se añadirá una explicación teórico-técnica sobre el funcionamiento de las conexiones internas en Docker, y sobre la naturaleza de la propia red virtual.

6.1 HERRAMIENTAS UTILIZADAS

6.1.1 WIRESHARK

En primer lugar, cabe resaltar el software utilizado para realizar el análisis de las conexiones entre los distintos contenedores. Nacido en 1998, Wireshark es un analizador de protocolos de red de código abierto que permite la captura en tiempo real de los paquetes de datos que circulan por una red, para su posterior análisis y examen.

Wireshark permite la administración de redes y el diagnóstico de su estado (siendo capaz de encontrar problemas como pérdidas de latencia o cuellos de botella), la identificación de actividad sospechosa o elementos que traten de introducirse dentro de la red, y, como el motivo principal de utilización dentro de este proyecto, permite el desarrollo de protocolos y aplicaciones.

Para lograr el análisis de la red, Wireshark se apoya en librerías de tipo pcap las cuales registran todos los datos que captura dentro de la interfaz web (Ethernet, Wi-fi, puerto USB, Bluetooth…). Esta captura (Ilustración 58) se realiza en modo “promiscuo”, lo cual quiere decir que examina todas las comunicaciones que se producen dentro de la red. Para obtener este permiso, y al ser este un proyecto en Windows, Wireshark debe ejecutarse como

administrador. A continuación, se transfiere esa información sobre los paquetes capturados al espacio de usuario, desde donde son posteriormente decodificados, para poder mostrar no solamente el contenido de los paquetes, sino también otros datos como la línea de tiempo en que se ha registrado, el origen y destino de dicho paquete, o el protocolo desde el que ha sido enviado.

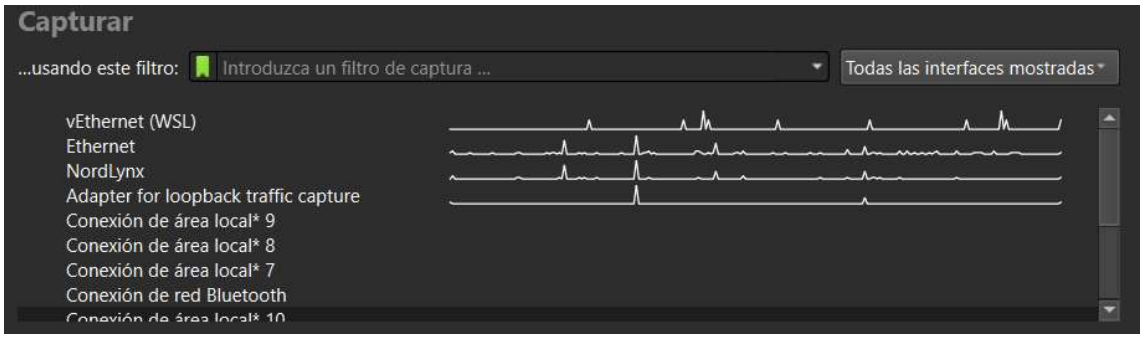


Ilustración 58. Ejemplo de captura de redes con Wireshark.

Sin embargo, existen dos problemas principales para utilizar Wireshark en este proyecto: el tipo de red virtual montado por Docker, y el uso de los zbytes en ZENOH para la transmisión de datos.

Respecto al primer problema, es importante explicar cómo funciona la red virtual de Docker en el proyecto. Como se ha visto en el Capítulo 5. se ha creado una red de tipo bridge denominada “zenoh-net”, mediante la cual se comunican todos los contenedores. Dentro de esta red (Ilustración 59), podemos encontrar dos niveles: a nivel de kernel, se crea la interfaz bridge virtual denominada docker0, que enruta los paquetes que son enviados entre contenedores y host; mientras que, por otro lado, se crean dos interfaces virtuales veth, en donde un extremo (eth0) se conecta al contenedor, mientras que el otro se conecta al bridge (docker0). Como se mencionó previamente, cada contenedor cuenta con una dirección IP privada dentro de la propia red, distintas entre sí.

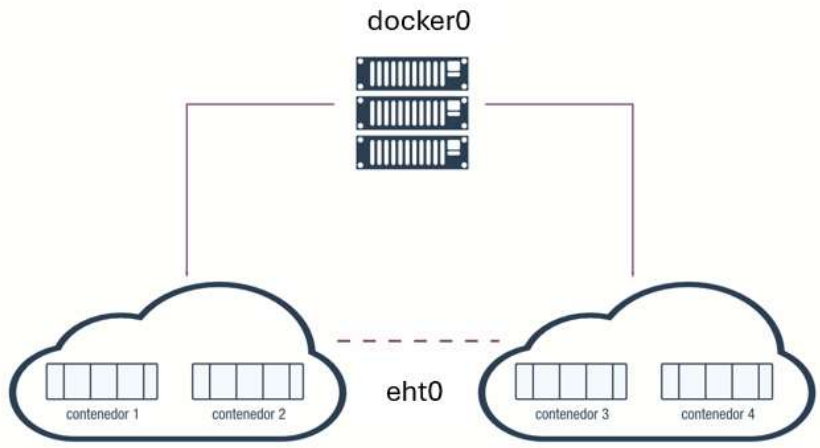


Ilustración 59. Red bridge con docker0 y eth0.

Sin embargo, esta red virtual no está sostenida realmente sobre Windows: para generarla, Docker hace uso de una máquina virtual (WLS en este caso) basada en Linux. Por ello, la red docker0 figura como "no existente" para Windows, puesto que está generada a nivel de kernel de la máquina virtual de Linux. Y, al no ser una interfaz de red generada desde el sistema operativo de Windows, la librería Npcap (la librería de estilo pcap que se utiliza en Wireshark de Windows), no puede acceder a ella para capturar datos de red. Todo el tráfico entre contenedores se encuentra entonces encapsulado, y es imposible para Wireshark acceder a él. Para solucionar este problema, se ha hecho uso de tcpdump.

6.1.2 TCPDUMP

Tcpdump es una herramienta de comandos para la captura y análisis de paquetes de red, diseñada para Linux. Su ligereza y velocidad la hacen muy útil para implementarse en el entorno del proyecto. Su funcionamiento es muy similar a Wireshark: apoyado en librerías de tipo pcap, activa el modo promiscuo y captura todos los paquetes de datos que circulen por la interfaz designada. A partir de esta captura, es capaz de generar un archivo de tipo "captura.pcap" que es legible por otros analizadores de red, como el propio Wireshark. Como se mostró en el Capítulo 5. cada Dockerfile cuenta con una consigna de instalación de la herramienta tcpdump, de forma que se tenga una instancia en cada contenedor. Por tanto, se pretende capturar el tráfico en la red en dos niveles: en primer lugar, a nivel de

kernel en la red docker0, y en segundo lugar, a nivel de eth0 (la salida de los contenedores), tal y como se muestra en la Ilustración 60.

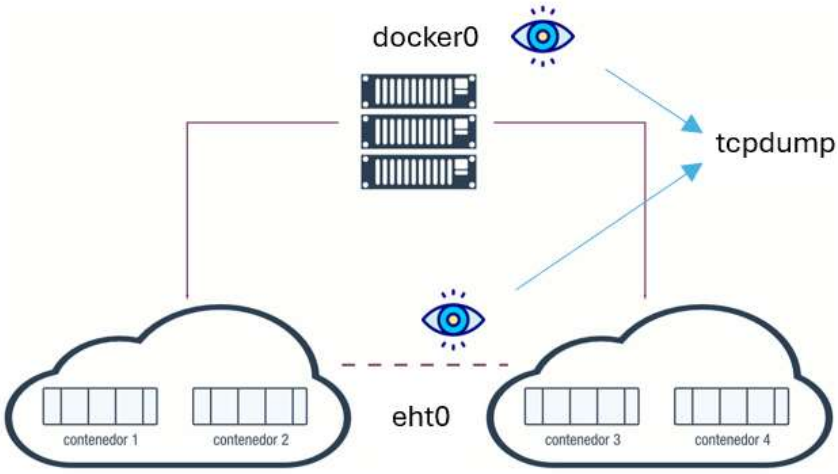


Ilustración 60. Esquema del espionaje de la red Bridge.

En relación con el segundo problema, el hecho de que el protocolo ZENOH sea tan novedoso, implica que Wireshark no posee aún la capacidad de decodificar los zbytes. Wireshark emplea archivos de tipo lua (programados en ese lenguaje) dentro de los disectores para, entre otras cosas, poder interpretar protocolos nuevos o personalizados. Para el caso de ZENOH, este tipo de archivos no existe, ni dentro del propio Wireshark, ni en repositorios externos. Por ello, se tomó la determinación de añadir un elemento nuevo al sistema que fuese capaz de traducir dichos zbytes y hacer las veces de traductor del sistema: el sniffer.

6.1.3 SNIFFER

En informática, un sniffer se refiere a un analizador de protocolos red. El archivo sniffer diseñado tiene como objetivo escuchar todos los logs que se producen en los distintos tópicos del sistema, y registrarlos en un archivo de texto plano legible, junto con la hora de captura. De esta forma, se puede comprobar que la comunicación entre todos los elementos funciona correctamente, puesto que el contenido que está siendo enviado es el correcto. La captura de

tiempo, además, permitirá la medición de la velocidad de respuesta del sistema a los cambios que se realizan. La carpeta del sniffer (Ilustración 61) contiene los siguientes elementos:

- Sniffer.py
- Dockerfile
- Requirements.txt
- Carpeta ZENOH
- Carpeta Logs.

La única variación con respecto a las carpetas del actuador o del sensor es la implementación de una carpeta "logs" en donde se almacenará el archivo de texto plano generado por el sniffer, con la información recabada.

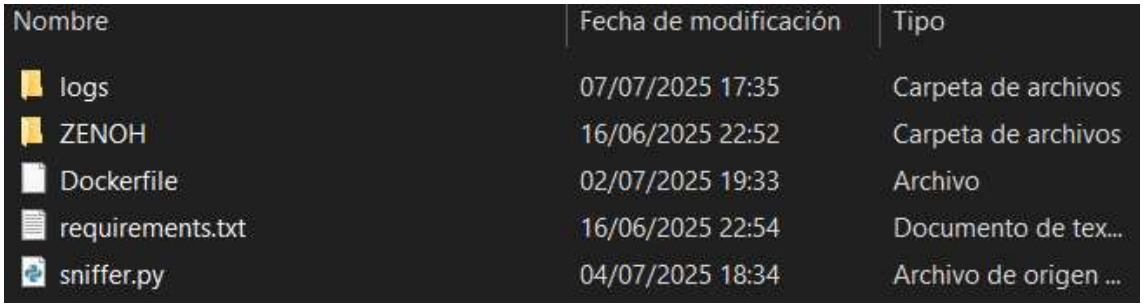


Ilustración 61. Contenidos de la carpeta "sniffer".

Los contenidos del Dockerfile y requirements.txt son idénticos a los contenidos del resto de contenedores, con los correspondientes cambios de nombre. Es por ello que, al igual que en el Capítulo 5, se procederá a explicar los contenidos del script sniffer.py

```
import zenoh
import time

LOG_FILE = "/app/logs/capturas_zenoh.txt"
```

En primer lugar, se importan las librerías tanto zenoh como time. La librería time permitirá el registro de la fecha y hora de cada una de las capturas realizadas. Después, se define la ruta en donde se generará el archivo de texto con los mensajes recibidos.

```
def main():
    with zenoh.open(zenoh.Config()) as session:
        with open(LOG_FILE, "a", encoding="utf-8") as f:
```

Se genera una sesión utilizando la configuración Zenoh por defecto, y se abre el archivo de texto en donde se escribirán los logs escuchados. Se utilizará codificación UTF-8, la forma estándar de representación de texto en Python, que se ha ido utilizando a lo largo del proyecto.

```
def callback(sample):

                try:
                    texto = bytes(sample.payload).decode("utf-8",
errors="replace")
                    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
                    linea = f"[{timestamp}] [{sample.key_expr}] → {texto}"
                    print(linea)
                    f.write(linea + "\n")
                    f.flush()
                except Exception as e:
                    print(f"Error al procesar muestra: {e}")
```

A continuación, se crea la función callback, que será llamada cada vez que se reciba un mensaje en cualquiera de los tópicos myhome. Los tópicos myhome incluyen los tópicos de acción y consigna usados en el actuador, y los tópicos de temperatura usados por el sensor. Se convierten en texto los bytes capturados, con el método que se aplicó en el Capítulo 5 para el sensor y el proceso, se añade la fecha (año, mes y día) y hora (hora, minuto y segundo) del momento de la captura, y se construye la línea que será impresa: primero la fecha, luego la hora, después el tópico en donde se ha recibido el mensaje, y finalmente el mensaje en sí. Dicha línea se imprime en el archivo de texto y se genera un salto de línea, para poder imprimir la siguiente. Se añade también un mensaje de error, en caso de haberlo.

```
session.declare_subscriber("myhome/**", callback)

            try:
                while True:
                    time.sleep(1)
            except KeyboardInterrupt:
                print("\nSniffer detenido por el usuario.")
```

Finalmente, se produce la suscripción a todos los tópicos que comiencen con /myhome, en

base a lo configurado en la función callback. Se mantiene el proceso vivo en un bucle infinito hasta que el sistema o el usuario los detengan.

Por tanto, el sniffer aportará información clave sobre si los cambios en las consignas y temperaturas se están realizando de manera correcta o no, y sobre la velocidad de respuesta del sistema. Cabe destacar que, para la Fase 2, el script incluye también los métodos de implementación de TLS que se han desarrollado en el Capítulo 5.

De ahora en adelante, se analizarán los resultados obtenidos. El proceso de análisis constará de tres puntos para cada una de las fases:

- Análisis del tráfico en docker0.
- Análisis del tráfico en eth0.
- Análisis de la información del sniffer.

En la Fase 3 se centrará el tiro fundamentalmente en el último punto, puesto que se plantea la comparativa entre los dos distintos modelos construidos.

6.2 FASE

Como se ha explicado en el apartado Capítulo 5. , la forma en la que está configurada la red consta de dos capas o niveles: el nivel multicast (en este caso, docker0) y el nivel unicast (eth0). Por lo tanto, se espera que en el archivo pcap relativo al tráfico de docker0 se encuentren únicamente peticiones por multicast (protocolo UDP), mientras que las conexiones TCP se encuentren en el archivo pcap relativo al tráfico de eth0, tal y como se muestra en la Tabla 3.

Red:	**Protocolo**	**Conexiones**
Docker0	UDP	Unidireccionales
Eth0	TCP	Bidireccionales

Tabla 3. Resultados de red esperados para la Fase 1.

En primer lugar, debe conocerse la identificación IP de los elementos del sistema, para poder comprender entre qué miembros del mismo se está intercambiando información. A continuación, se muestra la Tabla 4, en la que se recogen las direcciones IP de cada elemento del sistema.

	Actuador	**Sensor**
Salón	172.18.0.9	172.18.0.3
Cocina	172.18.0.6	172.18.0.5
Baño	172.18.0.14	172.18.0.13
Dormitorio 1	172.18.0.7	172.18.0.8
Dormitorio 2	172.18.0.12	172.18.0.11
Consola	**Proceso**	**Sniffer**
172.18.0.2	172.18.0.10	172.18.0.4

Tabla 4. Direcciones IP de la Fase 1.

Como se puede observar, no existe la dirección IP 172.18.0.1, puesto que queda reservada para uso de Docker.

6.2.1 ANÁLISIS DEL TRÁFICO EN DOCKER0

Durante esta escucha, se han recogido un total de 239 paquetes de envío durante 150 segundos, de los cuales se puede realizar la siguiente división, recogidos en la Ilustración 62:

- 229 paquetes con protocolo UDP.

- 6 paquetes con protocolo ICMPv6.
- 4 paquetes con protocolo ARP.

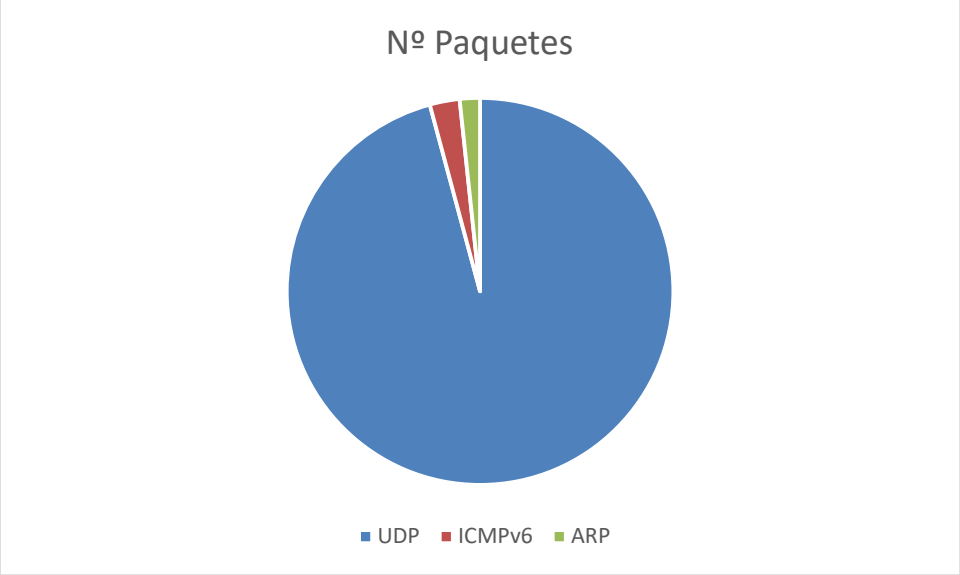


Ilustración 62. Gráfico de sectores sobre la aparición de protocolos.

A continuación, se muestra en la Ilustración 63 una parte de los paquetes capturador por el archivo pcap:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::	ff02::1:ff77:1e26	ICMPv6	92	Neighbor Solicitation for fe80::2cc5:1dff:fe77:1e26
2	0.190018	::	ff02::16	ICMPv6	116	Multicast Listener Report Message v2
3	0.763565	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
4	0.763611	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
5	1.050042	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
6	1.080033	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
7	1.230035	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
8	1.320064	fe80::2cc5:1dff:fe7…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
9	1.763510	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
10	1.763537	0a:fd:a2:d8:5a:4d		ARP	48	ARP Announcement for 172.18.0.14
11	6.163010	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
12	6.320760	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
13	6.321085	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
14	6.330606	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
15	6.432558	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
16	6.532494	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
17	6.678213	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
18	6.754782	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
19	6.842463	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3
20	6.895891	172.18.0.12	224.0.0.224	UDP	51	34052 → 7446 Len=3
21	6.947616	172.18.0.13	224.0.0.224	UDP	51	54083 → 7446 Len=3
22	7.380930	172.18.0.9	224.0.0.224	UDP	51	33668 → 7446 Len=3
23	14.163406	172.18.0.2	224.0.0.224	UDP	51	53365 → 7446 Len=3
24	14.320275	172.18.0.5	224.0.0.224	UDP	51	35915 → 7446 Len=3
25	14.320525	172.18.0.3	224.0.0.224	UDP	51	42061 → 7446 Len=3
26	14.330947	172.18.0.4	224.0.0.224	UDP	51	57896 → 7446 Len=3
27	14.432754	172.18.0.6	224.0.0.224	UDP	51	33859 → 7446 Len=3
28	14.532319	172.18.0.7	224.0.0.224	UDP	51	44485 → 7446 Len=3
29	14.678986	172.18.0.8	224.0.0.224	UDP	51	60710 → 7446 Len=3
30	14.754731	172.18.0.10	224.0.0.224	UDP	51	34707 → 7446 Len=3
31	14.843592	172.18.0.11	224.0.0.224	UDP	51	36239 → 7446 Len=3

Ilustración 63. Archivo pcap de capturas de la red docker0.

En primer lugar, se hará una explicación de los distintos apartados presentes en el archivo pcap que se ha abierto desde Wireshark, los cuales servirán de guía para el resto de las imágenes del capítulo. De izquierda a derecha, se encuentran 7 apartados distintos, cuyos contenidos son:

- No: indica el número del paquete. En este caso, va de 1 a 239.
- Time: indica el tiempo en el que se ha capturado, partiendo del inicio de la captura (tiempo 0)
- Source: indica el punto de salida del paquete.
- Destination: indica el punto de llegada del paquete.
- Protocol: indica el protocolo empleado para la transmisión del paquete
- Length: indica la longitud del paquete.
- Info: añade información sobre el paquete (data, protocolo de Internet, protocolo de transmisión, etc)

Los primeros paquetes capturados se corresponden con todos los paquetes capturados por protocolos distintos a UDP, y son correspondientes al inicio de sesión de tcpdump. En cuanto a los paquetes capturados con protocolo UDP, se puede observar que, aunque el emisor va cambiando, el destino es siempre el mismo: la dirección 224.0.0.224. Esto se debe a que en docker0 se producen los descubrimientos multicast, y dicha dirección IP es la dirección que ZENOH tiene reservada para ellos. Por tanto, todos los nodos del sistema envían mensajes a dicha dirección, pero no reciben ninguna respuesta, puesto que solo la están utilizando para darse a conocer a los demás miembros del sistema.

6.2.2 ANÁLISIS DEL TRÁFICO EN ETH0

La escucha se ha realizado en el contenedor correspondiente al actuador del baño (IP 172.18.0.14). Por tanto, se espera que todas las conexiones tengan relación con esta dirección IP. La escucha en dicho contenedor se realizó durante 30 segundos, capturándose un total de 969 paquetes, de los cuales se realiza la siguiente división (Ilustración 64):

- 817 paquetes con protocolo TCP.

- 152 paquetes con protocolo UDP.

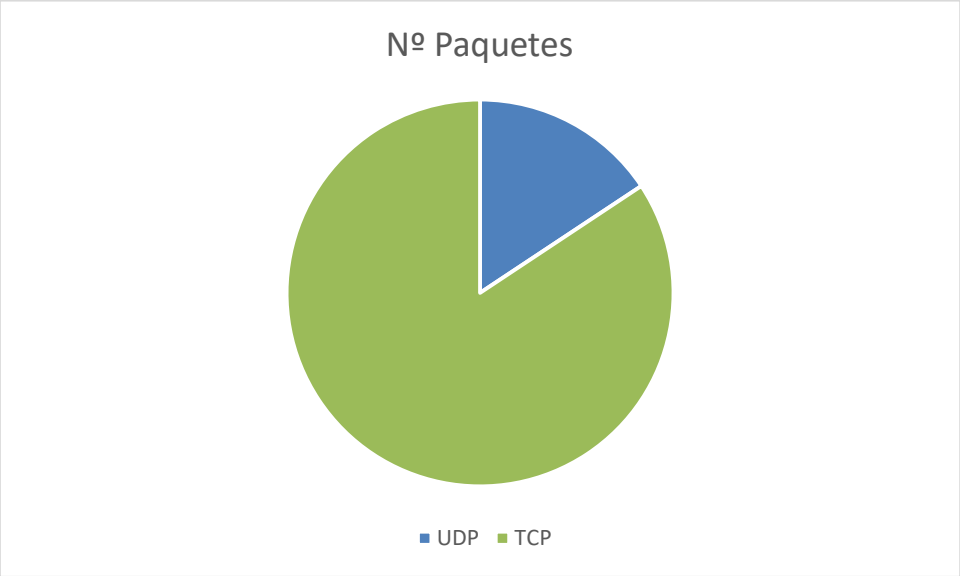


Ilustración 64. Gráfico de sectores sobre la aparición de protocolos.

Los resultados de esta captura pueden resultar chocantes, puesto que, a priori, debería obtenerse únicamente tráfico en protocolo TCP para los contenedores. Además, si se analiza en detalle el tráfico de los paquetes, de observan discrepancias aún mayores. Se centrará el ojo, en primer lugar, en el tráfico TCP. En concreto, entre los paquetes 34 y 38 (Ilustración 65):

No.	Time	Source	Destination	Protocol	Length	Info
34	0.012308	172.18.0.14	172.18.0.13	TCP	69	46602 → 40777 [PSH, ACK] Seq=1 Ack=1 Win=501 Len=3 TSval=2891223574 TSecr=2934529734
35	0.012362	172.18.0.14	172.18.0.9	TCP	69	43652 → 39735 [PSH, ACK] Seq=1 Ack=4 Win=501 Len=3 TSval=1898322980 TSecr=2036954575
36	0.012389	172.18.0.9	172.18.0.14	TCP	66	39735 → 43652 [ACK] Seq=4 Ack=4 Win=509 Len=0 TSval=2036954587 TSecr=1898322980
37	0.012529	172.18.0.13	172.18.0.14	TCP	69	40777 → 46602 [PSH, ACK] Seq=1 Ack=4 Win=509 Len=3 TSval=2934532235 TSecr=2891223574
38	0.012532	172.18.0.14	172.18.0.13	TCP	66	46602 → 40777 [ACK] Seq=4 Ack=4 Win=501 Len=0 TSval=2891223575 TSecr=2934532235
39	0.018612	172.18.0.14	172.18.0.8	TCP	69	57402 → 35441 [PSH, ACK] Seq=1 Ack=4 Win=501 Len=3 TSval=658589574 TSecr=1984718849
40	0.018641	172.18.0.8	172.18.0.14	TCP	66	35441 → 57402 [ACK] Seq=4 Ack=4 Win=509 Len=0 TSval=1984718863 TSecr=658589574
41	0.021854	172.18.0.14	172.18.0.5	TCP	69	47054 → 40499 [PSH, ACK] Seq=1 Ack=4 Win=501 Len=3 TSval=3787486740 TSecr=1801520388
42	0.021885	172.18.0.5	172.18.0.14	TCP	66	40499 → 47054 [ACK] Seq=4 Ack=4 Win=506 Len=0 TSval=1801520403 TSecr=3787486740
43	0.021899	172.18.0.14	172.18.0.2	TCP	69	34203 → 39956 [PSH, ACK] Seq=1 Ack=4 Win=509 Len=3 TSval=520468818 TSecr=1730483011
44	0.021945	172.18.0.2	172.18.0.14	TCP	66	39956 → 34203 [ACK] Seq=4 Ack=4 Win=501 Len=0 TSval=1730483029 TSecr=520468818
45	0.024033	172.18.0.14	172.18.0.3	TCP	69	34203 → 36318 [PSH, ACK] Seq=1 Ack=4 Win=504 Len=3 TSval=2235277216 TSecr=756827027
46	0.024061	172.18.0.3	172.18.0.14	TCP	66	36318 → 34203 [ACK] Seq=4 Ack=4 Win=501 Len=0 TSval=756827045 TSecr=2235277216
47	0.024074	172.18.0.14	172.18.0.10	TCP	69	35088 → 37225 [PSH, ACK] Seq=1 Ack=4 Win=502 Len=3 TSval=1108496074 TSecr=813530963
48	0.024098	172.18.0.10	172.18.0.14	TCP	66	37225 → 35088 [ACK] Seq=4 Ack=4 Win=509 Len=0 TSval=813530986 TSecr=1108496074
49	0.024110	172.18.0.14	172.18.0.11	TCP	69	34203 → 36066 [PSH, ACK] Seq=1 Ack=4 Win=509 Len=3 TSval=820104527 TSecr=2553524058
50	0.024156	172.18.0.11	172.18.0.14	TCP	66	36066 → 34203 [ACK] Seq=4 Ack=4 Win=502 Len=0 TSval=2553524079 TSecr=820104527
51	0.024168	172.18.0.14	172.18.0.7	TCP	69	34203 → 49966 [PSH, ACK] Seq=1 Ack=4 Win=509 Len=3 TSval=71024970 TSecr=1550492581
52	0.024183	172.18.0.7	172.18.0.14	TCP	66	49966 → 34203 [ACK] Seq=4 Ack=4 Win=501 Len=0 TSval=1550492605 TSecr=71024970
53	0.025331	172.18.0.14	172.18.0.12	TCP	69	34203 → 35202 [PSH, ACK] Seq=1 Ack=4 Win=509 Len=3 TSval=2650305590 TSecr=401814918
54	0.025360	172.18.0.12	172.18.0.14	TCP	66	35202 → 34203 [ACK] Seq=4 Ack=4 Win=501 Len=0 TSval=401814936 TSecr=2650305590
55	0.025381	172.18.0.14	172.18.0.12	TCP	69	41206 → 45721 [PSH, ACK] Seq=1 Ack=4 Win=501 Len=3 TSval=2650305591 TSecr=401814913
56	0.025426	172.18.0.12	172.18.0.14	TCP	66	45721 → 41206 [ACK] Seq=4 Ack=4 Win=509 Len=0 TSval=401814937 TSecr=2650305591
57	0.025438	172.18.0.14	172.18.0.7	TCP	69	59632 → 44315 [PSH, ACK] Seq=1 Ack=4 Win=501 Len=3 TSval=71024972 TSecr=1550492582
58	0.025461	172.18.0.7	172.18.0.14	TCP	66	44315 → 59632 [ACK] Seq=4 Ack=4 Win=509 Len=0 TSval=1550492607 TSecr=71024972

Ilustración 65. Capturas TCP del actuador del baño.

Como se puede observar, todos los paquetes tienen como origen o destino la dirección IP del baño (172.18.0.14). Sin embargo, observamos conexiones que no tienen sentido, puesto que no debería transmitirse ninguna información entre dichos contenedores, por ejemplo, con el sensor de la cocina (172.18.0.5), o el actuador del dormitorio 1 (172.18.0.7). Si se centra la mira en los paquetes con protocolos UDP, por ejemplo, entre los paquetes 67 y 90 (Ilustración 66):

No.	Time	Source	Destination	Protocol	Length	Info
67	0.607246	172.18.0.3	224.0.0.224	UDP	45	48883 → 7446 Len=3
68	0.607344	172.18.0.14	172.18.0.3	UDP	84	43950 → 48883 Len=42
69	0.720222	172.18.0.4	224.0.0.224	UDP	45	35050 → 7446 Len=3
70	0.720439	172.18.0.14	172.18.0.4	UDP	84	43950 → 35050 Len=42
71	0.720767	172.18.0.5	224.0.0.224	UDP	45	48407 → 7446 Len=3
72	0.720953	172.18.0.14	172.18.0.5	UDP	84	43950 → 48407 Len=42
73	0.818690	172.18.0.6	224.0.0.224	UDP	45	38605 → 7446 Len=3
74	0.818854	172.18.0.14	172.18.0.6	UDP	84	43950 → 38605 Len=42
75	0.889611	172.18.0.7	224.0.0.224	UDP	45	50723 → 7446 Len=3
76	0.889764	172.18.0.14	172.18.0.7	UDP	84	43950 → 50723 Len=42
77	1.038512	172.18.0.8	224.0.0.224	UDP	45	39446 → 7446 Len=3
78	1.038653	172.18.0.14	172.18.0.8	UDP	84	43950 → 39446 Len=42
79	1.092342	172.18.0.9	224.0.0.224	UDP	45	60601 → 7446 Len=3
80	1.092494	172.18.0.14	172.18.0.9	UDP	84	43950 → 60601 Len=42
81	1.098919	172.18.0.10	224.0.0.224	UDP	45	48987 → 7446 Len=3
82	1.099077	172.18.0.14	172.18.0.10	UDP	84	43950 → 48987 Len=42
83	1.172729	172.18.0.11	224.0.0.224	UDP	45	38120 → 7446 Len=3
84	1.172939	172.18.0.14	172.18.0.11	UDP	84	43950 → 38120 Len=42
85	1.253824	172.18.0.2	224.0.0.224	UDP	45	48336 → 7446 Len=3
86	1.253980	172.18.0.14	172.18.0.2	UDP	84	43950 → 48336 Len=42
87	1.353499	172.18.0.13	224.0.0.224	UDP	45	59141 → 7446 Len=3
88	1.353621	172.18.0.14	172.18.0.13	UDP	84	43950 → 59141 Len=42
89	1.368888	172.18.0.12	224.0.0.224	UDP	45	44795 → 7446 Len=3
90	1.369070	172.18.0.14	172.18.0.12	UDP	84	43950 → 44795 Len=42

Ilustración 66. Capturas UDP del actuador del baño.

Se puede observar cómo no solamente se recoge el tráfico propio de dicho contenedor, sino que se recoge también el tráfico de otros contenedores, como el actuador de la cocina (172.18.0.6) o el proceso (172.18.0.4) con la dirección 224.0.0.224. Además, se establecen conexiones UDP también entre contenedores.

El motivo de estos dos sucesos es la forma en la que ZENOH se configura en modo multicast. En la capa unicast, se queda montada una malla de conexiones entre todos los nodos (los contenedores) que forman el sistema. Aunque no se esté compartiendo información sobre el propio proyecto entre ellos (por ejemplo, el actuador del baño y el de la cocina no deberían compartir ninguna información), las conexiones TCP están montadas igualmente para

compartir otro tipo de datos, como enrutamiento o suscripciones. Estas rutas están creadas de forma preventiva, y pueden servir para enrutar de forma más eficiente la transmisión de paquetes de datos. Puede entenderse como una red de carreteras. Póngase el ejemplo entre tres ciudades españolas: Madrid, León y Oviedo. Pese a que en un momento del tiempo no haya tráfico entre Madrid y León, no quiere decir que en un futuro no pueda haberlo. Y, si por algún motivo, el tráfico entre Madrid y Oviedo se ve comprometido, se puede enrutar dicho tráfico como Madrid – León – Oviedo, debido a que existen las carreteras para ello.

Por otro lado, el descubrimiento en multicast funciona de tal modo que todos los nodos conectados a la red reciben el mismo paquete informando de la presencia del nodo. Después, cualquier nodo puede mostrar interés en contactar con otro y, mediante tráfico UDP, comunicar su IP para establecer la conexión TCP. Como ejemplo, póngase el caso de tres nodos: nodo 1, nodo 2 y nodo 3. Los tres nodos se descubren a la misma dirección, y cada nodo recibe el paquete de descubrimiento de los otros dos. Si el nodo 1 considera que el nodo 2 está suscrito a claves relevantes, o contiene información importante, establecerá con él una conexión UDP para comunicar su dirección IP y poder pasar a comunicarse por una vía más "privada" como TCP.

6.2.3 ANÁLISIS DE LA INFORMACIÓN DEL SNIFFER

El sniffer recogerá toda la información que esté publicada en cualquiera de los cuatro tópicos del sistema. Calculará la temperatura que publica tanto el proceso como los sensores, así como los cambios en el tópico acción y consigna. De forma predeterminada, estos figuran como "none", por tanto, mientras no se actualicen, no cambiarán. En el tópico "acción" solo se tendrá un registro por cambio de estado, de forma que, si se consigna a la habitación X a calentar, mientras que en el proceso este se mantendrá como "calentar" (y publicado en el tópico consignas como tal), en la acción no.

```
[2025-07-07 16:30:20] [myhome/salon/temp] → 21.0
[2025-07-07 16:30:20] [myhome/cocina/temp] → 21.0
[2025-07-07 16:30:20] [myhome/bano/temp] → 21.0
[2025-07-07 16:30:20] [myhome/dormitorio1/temp] → 21.0
[2025-07-07 16:30:20] [myhome/dormitorio2/temp] → 21.0
[2025-07-07 16:30:20] [myhome/dormitorio2/sensor] → 21.0
[2025-07-07 16:30:20] [myhome/dormitorio1/sensor] → 21.0
[2025-07-07 16:30:20] [myhome/cocina/sensor] → 21.0
[2025-07-07 16:30:20] [myhome/bano/sensor] → 21.0
[2025-07-07 16:30:20] [myhome/salon/sensor] → 21.0
[2025-07-07 16:30:25] [myhome/salon/temp] → 21.0
[2025-07-07 16:30:25] [myhome/cocina/temp] → 21.0
[2025-07-07 16:30:25] [myhome/bano/temp] → 21.0
[2025-07-07 16:30:25] [myhome/dormitorio1/temp] → 21.0
[2025-07-07 16:30:25] [myhome/dormitorio2/temp] → 21.0
[2025-07-07 16:30:25] [myhome/salon/sensor] → 21.0
[2025-07-07 16:30:25] [myhome/cocina/sensor] → 21.0
[2025-07-07 16:30:25] [myhome/bano/sensor] → 21.0
[2025-07-07 16:30:25] [myhome/dormitorio1/sensor] → 21.0
[2025-07-07 16:30:25] [myhome/dormitorio2/sensor] → 21.0
[2025-07-07 16:30:30] [myhome/salon/temp] → 21.0
[2025-07-07 16:30:30] [myhome/cocina/temp] → 21.0
```

Ilustración 67. Ejemplo de logs del sniffer.

Nótese como en este ejemplo de logs del sniffer (Ilustración 67) no se observan publicaciones en ninguno de los tópicos acción o consigna (aún no se ha determinado ninguna) y todas las temperaturas figuran como 21 ºC (estado inicial del sistema). Adicionalmente, se puede observar que los logs son actualizados cada 5 segundos (es el periodo de tiempo en el que el proceso aumenta o disminuye un grado).

La información relevante, en este caso, es la que permita comprobar la eficiencia del sistema, esto siendo la velocidad que se tarda desde que se da la consigna hasta que se recibe y se aplica. El tiempo máximo que debe durar este proceso es 5 segundos, ya que es el periodo de tiempo entre cada grupo de actualizaciones de temperatura por parte de sensores y proceso, y que recoge el sniffer en el documento de texto. Por lo tanto, se considerará un funcionamiento correcto que se ponga en marcha la consigna determinada en esa ventana de tiempo.

Se han realizado las siguientes consignas (Tabla 5):

Habitación	**Consigna**	**Hora**

Baño	Enfriar	16:31:29
Baño	Calentar	16:40:25
Dormitorio 1	Enfriar	16:40:46

Tabla 5. Consignas por habitación y fecha de la Fase 1.

Obsérvense pues los logs para cada una de esas fechas:

```
[2025-07-07 16:31:25] [myhome/dormitorio1/sensor] → 21.0
[2025-07-07 16:31:29] [myhome/bano/accion] → enfriar
[2025-07-07 16:31:30] [myhome/salon/temp] → 21.0
[2025-07-07 16:31:30] [myhome/salon/sensor] → 21.0
[2025-07-07 16:31:30] [myhome/cocina/temp] → 21.0
[2025-07-07 16:31:30] [myhome/bano/temp] → 21.0
[2025-07-07 16:31:30] [myhome/dormitorio1/temp] → 21.0
[2025-07-07 16:31:30] [myhome/dormitorio2/temp] → 21.0
[2025-07-07 16:31:30] [myhome/bano/consigna] → enfriar
[2025-07-07 16:31:30] [myhome/dormitorio2/sensor] → 21.0
[2025-07-07 16:31:30] [myhome/cocina/sensor] → 21.0
[2025-07-07 16:31:30] [myhome/bano/sensor] → 21.0
[2025-07-07 16:31:30] [myhome/dormitorio1/sensor] → 21.0
[2025-07-07 16:31:35] [myhome/bano/consigna] → enfriar
[2025-07-07 16:31:35] [myhome/salon/temp] → 21.0
[2025-07-07 16:31:35] [myhome/cocina/temp] → 21.0
[2025-07-07 16:31:35] [myhome/bano/temp] → 20.0
[2025-07-07 16:31:35] [myhome/dormitorio1/temp] → 21.0
[2025-07-07 16:31:35] [myhome/dormitorio2/temp] → 21.0
[2025-07-07 16:31:35] [myhome/cocina/sensor] → 21.0
[2025-07-07 16:31:35] [myhome/salon/sensor] → 21.0
[2025-07-07 16:31:35] [myhome/dormitorio1/sensor] → 21.0
[2025-07-07 16:31:35] [myhome/dormitorio2/sensor] → 21.0
[2025-07-07 16:31:35] [myhome/bano/sensor] → 20.0
```

Ilustración 68. Logs de la consigna "enfriar" del baño.

En primer lugar, como se puede observar en los logs referentes al baño (Ilustración 68, señaladas en verde, las líneas importantes), la actualización del estado de la acción se produce en un tiempo inferior al segundo. En la siguiente remesa de información recogida por sensores y proceso, la consigna del baño pasa a figurar como “enfriar”, sin embargo, la temperatura del baño se mantiene en 21 ºC. Esto se debe a que, pese a que se ha recibido la consigna de enfriar el baño, dicha consigna ha quedado registrada después de actualizarse la temperatura del baño, puede apreciarse en los logs. En los próximos casos se observará como, cuando la consigna queda registrada antes de actualizarse la temperatura, el cambio de dicha temperatura es instantáneo.

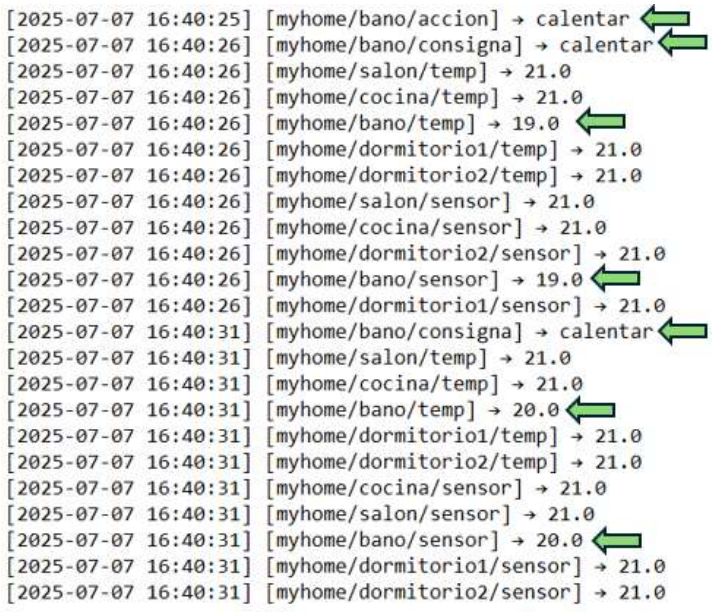


Ilustración 69. Logs de la consigna "calentar" del baño.

A continuación, se procede a calentar el baño (Ilustración 69). Cabe destacar un cambio en el intervalo de publicación de los sensores: ahora los dígitos finales de la cifra de los segundos que intercala es 6 y 1, en lugar de 0 y 5 como en el primer caso. Esto se debe a que, entre la publicación, la captura y la impresión hay un pequeño desfase que hace que no sean 5 segundos exactos, y, por tanto, se vaya acumulando. Se puede observar cómo, otra vez, el cambio de consigna se hace fuera de la ventana de publicación, y que, tras un segundo, la consigna del baño pasa a tener el valor “calentar”. Además, se observa que esta vez el cambio de temperatura ha sido instantáneo, en lugar de tardar 5 segundos en realizarse. Como se observará más adelante, este suceso únicamente ocurre en esta simulación Se observa, además, como en la última remesa la temperatura ha pasado a tener el valor de 20 ºC.

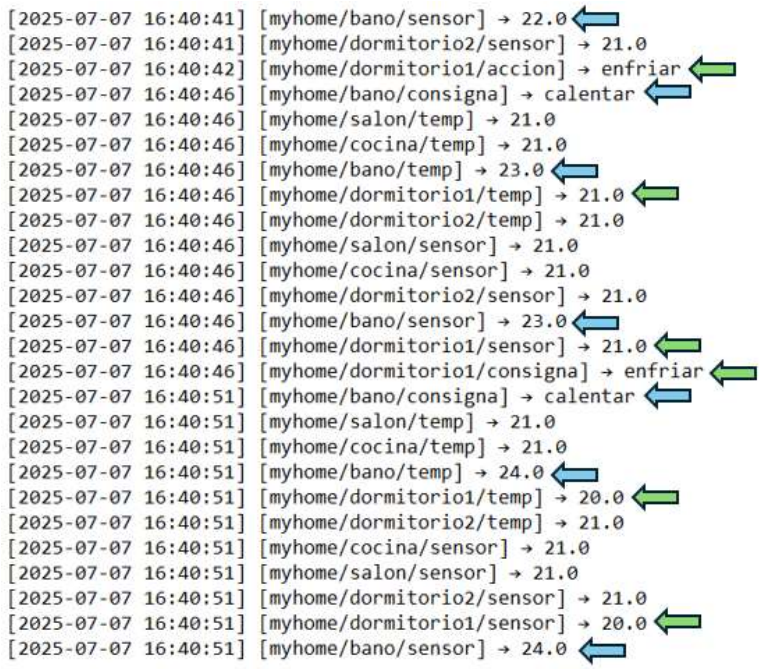


Ilustración 70. Logs de la consigna "enfriar" el dormitorio 1.

Finalmente, analizando los logs con el cambio de consigna en el dormitorio 1 (Ilustración 70, señalados con flechas verdes), observamos un comportamiento idéntico a la consigna “enfriar” en el baño: se recoge el cambio en acción en el momento en el que se produce, y, al pasar menos de 5 segundos, si bien la consigna cambia, las temperaturas no lo hacen hasta la siguiente remesa (9 segundos tras consigna). En azul, además, se han destacado los logs referenctes al baño, de forma que se pueda observar el proceso subyacente de calentamiento progresivo del baño.

6.2.4 CONCLUSIÓN

Como conclusión, se debe destacar que los resultados obtenidos concuerdan con los resultados esperados:

- En la red docker0, el tráfico era fundamentalmente UDP unidireccional, anunciándose todos los contenedores en la dirección 224.0.0.224.
- En la red eth0, el tráfico era tanto TCP (en mayor medida), como UDP (en menor medida). Si bien esto puede resultar chocante y parecer que no cuadra con el

resultado esperado, se ha explicado como el funcionamiento del descubrimiento multicast genera tráfico UDP a nivel de contenedor.

- Los logs del sniffer muestran que la respuesta del sistema a los cambios en las consignas es la correcta, puesto que se encuentra dentro del marco de los 5 segundos en los que sensores y proceso publican las temperaturas.

6.3 FASE 2

En este caso, el modelo de vivienda que ha sido diseñado para la Fase 2 basa sus comunicaciones en métodos puramente unicast, entre los distintos clientes y el router. Por tanto, se espera que, a nivel de docker0, no haya realmente ninguna clase de conexión, mientras que estas sí que existan en la red eth0, las cuales deberían estar cifradas con TLS. En la Tabla 6 se resumen los resultados esperados.

Red	**Protocolo**	**Conexiones**
Docker0	-	-
Eth0	TCP con TLS	Bidireccionales

Tabla 6. Resultados de red esperados para la Fase 2.

Al igual que en la fase anterior, se procederá en primer lugar al listado de las direcciones IP de cada uno de los elementos del sistema, recogidos en la Tabla 7.

	Actuador	**Sensor**
Salón	172.19.0.14	172.19.0.12
Cocina	172.19.0.13	172.19.0.7
Baño	172.19.0.5	172.19.0.10

Dormitorio 1	172.19.0.3	172.19.0.4
Dormitorio 2	172.19.0.6	172.19.0.15
	Router	**Consola**
	172.19.0.2	172.19.0.8
	Proceso	**Sniffer**
	172.19.0.9	172.19.0.11

Tabla 7. Direcciones IP de la Fase 2.

Al igual que en el caso anterior, no hay dirección 172.19.0.1, por los mismos motivos.

6.3.1 ANÁLISIS DEL TRÁFICO EN DOCKER0

Se han capturado únicamente 10 paquetes durante una captura de 30 segundos (Ilustración 71 e Ilustración 72):

```
tcpdump: listening on any, link-type LINUX_SLL2 (Linux cooked v2), snapshot length 262144 bytes
^C10 packets captured
10 packets received by filter
0 packets dropped by kernel
```

Ilustración 71. Información sobre la captura de tcpdump.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	::	ff02::1:ffb6:2e93	ICMPv6	92	Neighbor Solicitation for fe80::401f:91ff:feb6:2e93
2	0.340056	::	ff02::16	ICMPv6	116	Multicast Listener Report Message v2
3	0.414008	5e:97:bd:dc:04:bf		ARP	48	ARP Announcement for 172.18.0.16
4	0.414039	5e:97:bd:dc:04:bf		ARP	48	ARP Announcement for 172.18.0.16
5	1.060117	fe80::401f:91ff:feb…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
6	1.089979	fe80::401f:91ff:feb…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2
7	1.160100	fe80::401f:91ff:feb…	ff02::16	ICMPv6	136	Multicast Listener Report Message v2
8	1.414800	5e:97:bd:dc:04:bf		ARP	48	ARP Announcement for 172.18.0.16
9	1.414831	5e:97:bd:dc:04:bf		ARP	48	ARP Announcement for 172.18.0.16
10	1.860035	fe80::401f:91ff:feb…	ff02::16	ICMPv6	96	Multicast Listener Report Message v2

Ilustración 72. Captura de tráfico de la red docker0.

Como se puede observar, los paquetes capturados se corresponden con los protocolos ARP y ICMPv6 vistos en la red docker0 de la Fase 1, y se dan debido al inicio de sesión en la escucha multicast de la red docker. Sin embargo, como no se produce ningún descubrimiento por multicast, no se registra ningún tráfico de esa categoría, como sí que se registraba en la Fase 1.

6.3.2 ANÁLISIS DEL TRÁFICO EN ETH0

Se han realizado escuchas en cuatro contenedores distintos: el sensor del baño (IP 172.19.0.10), el actuador del baño (IP 172.19.0.5), la consola (IP 172.19.0.8) y el router (172.19.0.2). Mientras que en las capturas relativas a los contenedores del sensor, actuador o consola solo se deberían poder observar dos direcciones IP, en el router se espera encontrar la totalidad de los contenedores estableciendo comunicación. Los registros se han llevado a cabo durante un periodo de 30 segundos, en los cuales el sensor del baño ha registrado 57 paquetes (Ilustración 73), el actuador del baño 55 paquetes (Ilustración 74), y la consola 72 (Ilustración 75). Mientras tanto, en el router se ha registrado el tráfico de 2087 paquetes de datos (Ilustración 76), durante los 63 segundos que ha durado la captura.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
2	0.000994	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
3	0.001027	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=26 Ack=26 Win=501 Len=0 TSval=4091997819 TSecr=408995121
4	2.501019	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
5	2.501651	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
6	2.501679	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=51 Ack=51 Win=501 Len=0 TSval=4092000319 TSecr=408997621
7	2.503041	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
8	2.503375	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
9	2.546848	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=114 Ack=107 Win=501 Len=0 TSval=4092000365 TSecr=408997623
10	5.004268	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
11	5.004312	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
12	5.004332	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=139 Ack=132 Win=501 Len=0 TSval=4092002822 TSecr=409000124
13	7.505494	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
14	7.506199	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
15	7.506266	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=164 Ack=157 Win=501 Len=0 TSval=4092005324 TSecr=409002626
16	7.507007	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
17	7.507797	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
18	7.557044	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=227 Ack=213 Win=501 Len=0 TSval=4092005375 TSecr=409002628
19	10.007996	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
20	10.008528	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
21	10.008560	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=252 Ack=238 Win=501 Len=0 TSval=4092007826 TSecr=409005128
22	12.508932	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data
23	12.509488	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
24	12.509544	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=277 Ack=263 Win=501 Len=0 TSval=4092010327 TSecr=409007629
25	12.512778	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
26	12.513413	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
27	12.557031	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=319 Win=501 Len=0 TSval=4092010375 TSecr=409007633
28	15.014299	172.19.0.10	172.19.0.2	TLSv1.2	91	Application Data
29	15.014334	172.19.0.2	172.19.0.10	TCP	66	7448 → 57672 [ACK] Seq=340 Ack=344 Win=501 Len=0 TSval=4092012832 TSecr=409010134
30	15.014370	172.19.0.2	172.19.0.10	TLSv1.2	91	Application Data

Ilustración 73. Archivo pcap con los datos de tráfico del sensor del baño.

Como se puede observar, en la captura del sensor del baño solo se producen comunicaciones entre los contenedores de router y sensor, confirmando así el funcionamiento de la red: el

router, actuando a modo de centro de mensajería, se encarga de controlar y manejar todas las comunicaciones del sistema, de forma que para el sensor comunicarse con la consola, debe pasar primer por el router.

No.	Time	Source	Destination	Protocol	Length	Info
22	10.146619	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=168 Ack=188 Win=501 Len=0 TSval=4077768959 TSecr=2497740331
23	10.147967	172.19.0.5	172.19.0.2	TLSv1.2	128	Application Data
24	10.148030	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=168 Ack=250 Win=501 Len=0 TSval=4077768960 TSecr=2497740332
25	12.283445	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
26	12.283469	172.19.0.5	172.19.0.2	TCP	66	57990 → 7448 [ACK] Seq=250 Ack=193 Win=501 Len=0 TSval=2497742468 TSecr=4077771096
27	12.648792	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
28	12.648823	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=193 Ack=275 Win=501 Len=0 TSval=4077771461 TSecr=2497742833
29	14.784457	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
30	14.784494	172.19.0.5	172.19.0.2	TCP	66	57990 → 7448 [ACK] Seq=275 Ack=218 Win=501 Len=0 TSval=2497744969 TSecr=4077773597
31	15.123521	172.19.0.2	172.19.0.5	TLSv1.2	132	Application Data
32	15.123526	172.19.0.5	172.19.0.2	TCP	66	57990 → 7448 [ACK] Seq=275 Ack=284 Win=501 Len=0 TSval=2497745308 TSecr=4077773936
33	15.150161	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
34	15.150204	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=284 Ack=300 Win=501 Len=0 TSval=4077773963 TSecr=2497745335
35	15.151982	172.19.0.5	172.19.0.2	TLSv1.2	127	Application Data
36	15.152025	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=284 Ack=361 Win=501 Len=0 TSval=4077773964 TSecr=2497745336
37	17.624846	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
38	17.653513	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
39	17.653549	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=309 Ack=386 Win=501 Len=0 TSval=4077776466 TSecr=2497747838
40	20.126741	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
41	20.154338	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
42	20.154374	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=334 Ack=411 Win=501 Len=0 TSval=4077778967 TSecr=2497750339
43	20.157296	172.19.0.5	172.19.0.2	TLSv1.2	127	Application Data
44	20.157339	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=334 Ack=472 Win=501 Len=0 TSval=4077778970 TSecr=2497750342
45	22.627471	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
46	22.658290	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
47	22.658325	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=359 Ack=497 Win=501 Len=0 TSval=4077781471 TSecr=2497752843
48	25.128041	172.19.0.2	172.19.0.5	TLSv1.2	91	Application Data
49	25.159469	172.19.0.5	172.19.0.2	TLSv1.2	91	Application Data
50	25.159501	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=384 Ack=522 Win=501 Len=0 TSval=4077783972 TSecr=2497755344
51	25.162531	172.19.0.5	172.19.0.2	TLSv1.2	127	Application Data
52	25.162560	172.19.0.2	172.19.0.5	TCP	66	7448 → 57990 [ACK] Seq=384 Ack=583 Win=501 Len=0 TSval=4077783975 TSecr=2497755347

Ilustración 74. Archivo pcap con los datos de tráfico del actuador del baño.

No.	Time	Source	Destination	Protocol	Length	Info
39	15.236081	172.19.0.2	172.19.0.8	TLSv1.2	91	Application Data
40	15.236151	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=176 Ack=868 Win=501 Len=0 TSval=1152490189 TSecr=300944158
41	17.510958	172.19.0.8	172.19.0.2	TLSv1.2	91	Application Data
42	17.511016	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=868 Ack=201 Win=501 Len=0 TSval=300946433 TSecr=1152492464
43	17.737406	172.19.0.2	172.19.0.8	TLSv1.2	91	Application Data
44	17.737431	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=201 Ack=893 Win=501 Len=0 TSval=1152492690 TSecr=300946659
45	17.737715	172.19.0.2	172.19.0.8	TLSv1.2	129	Application Data
46	17.737718	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=201 Ack=956 Win=501 Len=0 TSval=1152492690 TSecr=300946659
47	17.738057	172.19.0.2	172.19.0.8	TLSv1.2	197	Application Data
48	17.738061	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=201 Ack=1087 Win=501 Len=0 TSval=1152492691 TSecr=300946660
49	17.738196	172.19.0.2	172.19.0.8	TLSv1.2	129	Application Data
50	17.738199	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=201 Ack=1150 Win=501 Len=0 TSval=1152492691 TSecr=300946660
51	20.010822	172.19.0.8	172.19.0.2	TLSv1.2	91	Application Data
52	20.010865	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=1150 Ack=226 Win=501 Len=0 TSval=300948934 TSecr=1152494965
53	20.238774	172.19.0.2	172.19.0.8	TLSv1.2	91	Application Data
54	20.238836	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=226 Ack=1175 Win=501 Len=0 TSval=1152495193 TSecr=300949162
55	21.720696	172.19.0.8	172.19.0.2	TLSv1.2	127	Application Data
56	21.720734	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=1175 Ack=287 Win=501 Len=0 TSval=300950644 TSecr=1152496675
57	22.739418	172.19.0.2	172.19.0.8	TLSv1.2	91	Application Data
58	22.739442	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=287 Ack=1200 Win=501 Len=0 TSval=1152497693 TSecr=300951662
59	22.741019	172.19.0.2	172.19.0.8	TLSv1.2	163	Application Data
60	22.741023	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=287 Ack=1297 Win=501 Len=0 TSval=1152497695 TSecr=300951664
61	22.741406	172.19.0.2	172.19.0.8	TLSv1.2	197	Application Data
62	22.741410	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=287 Ack=1428 Win=501 Len=0 TSval=1152497695 TSecr=300951664
63	24.221648	172.19.0.8	172.19.0.2	TLSv1.2	91	Application Data
64	24.221702	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=1428 Ack=312 Win=501 Len=0 TSval=300953145 TSecr=1152499176
65	25.243378	172.19.0.2	172.19.0.8	TLSv1.2	91	Application Data
66	25.243400	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=312 Ack=1453 Win=501 Len=0 TSval=1152500197 TSecr=300954166
67	26.722781	172.19.0.8	172.19.0.2	TLSv1.2	91	Application Data
68	26.722851	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=1453 Ack=337 Win=501 Len=0 TSval=300955646 TSecr=1152501677
69	27.549740	172.19.0.8	172.19.0.2	TLSv1.2	127	Application Data
70	27.549796	172.19.0.2	172.19.0.8	TCP	66	7448 → 41182 [ACK] Seq=1453 Ack=398 Win=501 Len=0 TSval=300956473 TSecr=1152502504

Ilustración 75. Archivo pcap con los datos de tráfico de la consola.

Al igual que en el caso del sensor, se puede observar que, tanto en la consola como en el actuador, las comunicaciones solo son establecidas con el router. Adicionalente, en el apartado “protocolo”, se puede diferenciar dos tipos de protocolos de transporte: TCP y TLS

v1.2. Esto se debe a que todo el tráfico TLS debe estar soportado por TCP, y al realizarse el comando tcpdump, este registra también la parte de las conexiones TCP de inicio del handshake. Los datos importantes, sin embargo, se contienen dentro de los paquetes cuyo protoclo se muestra como TLSv1.2. Por otro lado, en el apartado de información de los paquetes capturados con TLS, se puede observar que no aparecen datos, sino un rótulo de "Application Data". Esto se debe a que no se ha tenido acceso al contenido del paquete, y, por tanto, no se conoce su información. De esta manera, se demuestra que el cifrado TLS funciona y ha sido correctamente implementado dentro del sistema.

No.	Time	Source	Destination	Protocol	Length	Info
1042	33.629286	172.19.0.2	172.19.0.9	TCP	66	7448 → 50554 [ACK] Seq=1771 Ack=2020 Win=501 Len=0 TSval=2601435411 TSecr=1771422900
1043	33.629586	172.19.0.9	172.19.0.2	TLSv1.2	122	Application Data
1044	33.629590	172.19.0.2	172.19.0.9	TCP	66	7448 → 50554 [ACK] Seq=1771 Ack=2076 Win=501 Len=0 TSval=2601435411 TSecr=1771422900
1045	33.629683	172.19.0.9	172.19.0.2	TLSv1.2	120	Application Data
1046	33.629686	172.19.0.2	172.19.0.9	TCP	66	7448 → 50554 [ACK] Seq=1771 Ack=2130 Win=501 Len=0 TSval=2601435411 TSecr=1771422900
1047	33.629757	172.19.0.2	172.19.0.11	TLSv1.2	141	Application Data
1048	33.629803	172.19.0.9	172.19.0.2	TLSv1.2	127	Application Data
1049	33.629804	172.19.0.11	172.19.0.2	TCP	66	59778 → 7448 [ACK] Seq=351 Ack=6861 Win=2198 Len=0 TSval=2179602313 TSecr=1956441061
1050	33.629807	172.19.0.2	172.19.0.9	TCP	66	7448 → 50554 [ACK] Seq=1771 Ack=2191 Win=501 Len=0 TSval=2601435411 TSecr=1771422900
1051	33.629820	172.19.0.2	172.19.0.11	TLSv1.2	187	Application Data
1052	33.629848	172.19.0.11	172.19.0.2	TCP	66	59778 → 7448 [ACK] Seq=351 Ack=6982 Win=2198 Len=0 TSval=2179602313 TSecr=1956441061
1053	33.629880	172.19.0.2	172.19.0.8	TLSv1.2	197	Application Data
1054	33.629911	172.19.0.9	172.19.0.2	TLSv1.2	127	Application Data
1055	33.629913	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=579 Ack=1916 Win=501 Len=0 TSval=1152762916 TSecr=301216885
1056	33.629917	172.19.0.2	172.19.0.9	TCP	66	7448 → 50554 [ACK] Seq=1771 Ack=2252 Win=501 Len=0 TSval=2601435411 TSecr=1771422900
1057	33.629998	172.19.0.2	172.19.0.8	TLSv1.2	129	Application Data
1058	33.630018	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=579 Ack=1979 Win=501 Len=0 TSval=1152762917 TSecr=301216886
1059	33.630045	172.19.0.2	172.19.0.12	TLSv1.2	129	Application Data
1060	33.630118	172.19.0.2	172.19.0.7	TLSv1.2	129	Application Data
1061	33.630184	172.19.0.2	172.19.0.10	TLSv1.2	129	Application Data
1062	33.630224	172.19.0.10	172.19.0.2	TCP	66	57672 → 7448 [ACK] Seq=637 Ack=742 Win=501 Len=0 TSval=409342909 TSecr=4092345607
1063	33.630269	172.19.0.7	172.19.0.2	TCP	66	55226 → 7448 [ACK] Seq=649 Ack=742 Win=501 Len=0 TSval=3083910835 TSecr=3316983991
1064	33.630427	172.19.0.2	172.19.0.4	TLSv1.2	129	Application Data
1065	33.630508	172.19.0.2	172.19.0.15	TLSv1.2	129	Application Data
1066	33.630560	172.19.0.2	172.19.0.11	TLSv1.2	199	Application Data
1067	33.630584	172.19.0.11	172.19.0.2	TCP	66	59778 → 7448 [ACK] Seq=351 Ack=7115 Win=2221 Len=0 TSval=2179602314 TSecr=1956441062
1068	33.630615	172.19.0.2	172.19.0.8	TLSv1.2	129	Application Data
1069	33.630661	172.19.0.8	172.19.0.2	TCP	66	41182 → 7448 [ACK] Seq=579 Ack=2042 Win=501 Len=0 TSval=1152762917 TSecr=301216886
1070	33.630860	172.19.0.10	172.19.0.2	TLSv1.2	122	Application Data
1071	33.630863	172.19.0.7	172.19.0.2	TLSv1.2	124	Application Data
1072	33.630959	172.19.0.12	172.19.0.2	TLSv1.2	123	Application Data

Ilustración 76. Archivo pcap con los datos de tráfico del router.

En los datos de tráfico del router, por otro lado, se puede observar como aparecen diferentes direcciones IP de los diferentes contenedores (172.19.0.9 proceso, o 172.19.0.12 sensor salón), dado que todos se encuentran en comunicación directa con el router. Si se filtra por direcciones IP, se obtiene el siguiente gráfico (Ilustración 77):

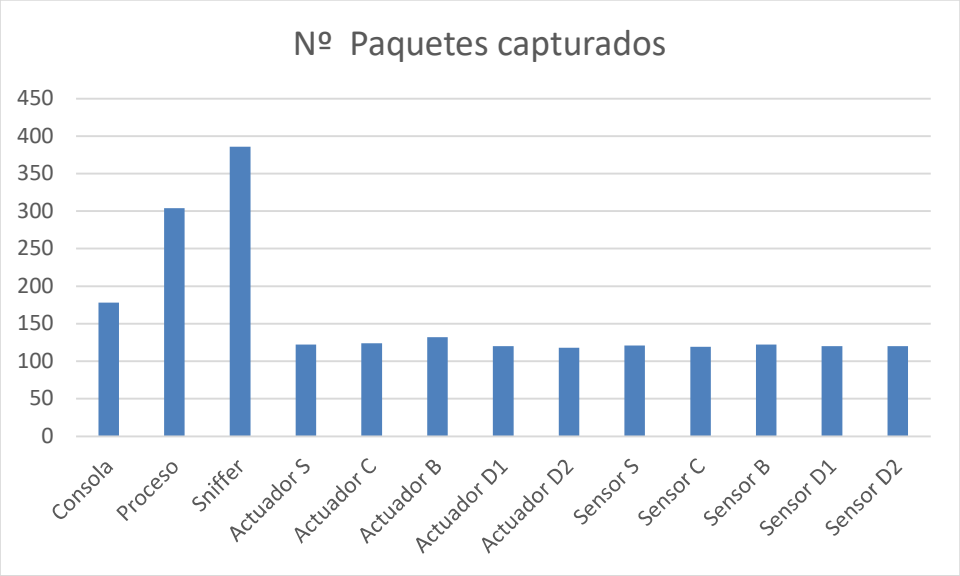


Ilustración 77. Gráfico de barras con el número de capturas por contenedor.

Como se puede observar, tanto el Sniffer como el Proceso son los contenedores que un mayor tráfico registran, lo cual tiene sentido puesto que el sniffer está continuamente actualizando sus registros con aquello servido por proceso y sensores, y el proceso debe estar actualizando continuamente las temperaturas de todas las habitaciones. Por otro lado, la consola está en tercera posición en cuanto a tráfico, puesto que desde ahí se envían las consignas para la petición de temperatura o de acciones. Por otro lado, se observa cómo tanto los actuadores como los sensores tienen un rango de tráfico bastante similar.

6.3.3 ANÁLISIS DE LA INFORMACIÓN DEL SNIFFER

Se han realizado las siguientes consignas, recogidas en la Tabla 8. Consignas por habitación y fecha de la Fase 2.:

Habitación	**Consigna**	**Hora**
Salón	Calentar	12:19:53
Cocina	Enfriar	12:20:13

Baño	Enfriar	12:21:25
Baño	Calentar	12:21:37

Tabla 8. Consignas por habitación y fecha de la Fase 2.

Obsérvense los logs para cada una de esas fechas:

```
[2025-07-04 12:19:49] [myhome/cocina/sensor] → 21.0
[2025-07-04 12:19:49] [myhome/dormitorio2/sensor] → 21.0
[2025-07-04 12:19:53] [myhome/salon/accion] → calentar
[2025-07-04 12:19:54] [myhome/salon/consigna] → calentar
[2025-07-04 12:19:55] [myhome/salon/temp] → 22.0
[2025-07-04 12:19:55] [myhome/cocina/temp] → 21.0
[2025-07-04 12:19:55] [myhome/bano/temp] → 21.0
[2025-07-04 12:19:55] [myhome/dormitorio1/temp] → 21.0
[2025-07-04 12:19:55] [myhome/dormitorio2/temp] → 21.0
[2025-07-04 12:19:55] [myhome/salon/sensor] → 22.0
[2025-07-04 12:19:55] [myhome/cocina/sensor] → 21.0
[2025-07-04 12:19:55] [myhome/bano/sensor] → 21.0
[2025-07-04 12:19:55] [myhome/dormitorio1/sensor] → 21.0
[2025-07-04 12:19:55] [myhome/dormitorio2/sensor] → 21.0
[2025-07-04 12:19:59] [myhome/salon/consigna] → calentar
[2025-07-04 12:20:00] [myhome/salon/temp] → 23.0
[2025-07-04 12:20:00] [myhome/cocina/temp] → 21.0
[2025-07-04 12:20:00] [myhome/bano/temp] → 21.0
[2025-07-04 12:20:00] [myhome/dormitorio1/temp] → 21.0
[2025-07-04 12:20:00] [myhome/dormitorio2/temp] → 21.0
[2025-07-04 12:20:00] [myhome/cocina/sensor] → 21.0
[2025-07-04 12:20:00] [myhome/salon/sensor] → 23.0
[2025-07-04 12:20:00] [myhome/dormitorio1/sensor] → 21.0
```

Ilustración 78. Logs del sniffer para la consigna "calentar" del salón.

Como se puede observar en la Ilustración 78, nada más realizarse la petición de calentar el salón a través de la consola, se registra en el tópico “acción”. Posteriormente, en el próximo ciclo del proceso, la temperatura ya ha aumentado un grado (como ha sido discutido previamente en los resultados de la fase 1) y la consigna sigue manteniéndose como “calentar”). En el ciclo después, la temperatura del salón es ya de 23 ºC.

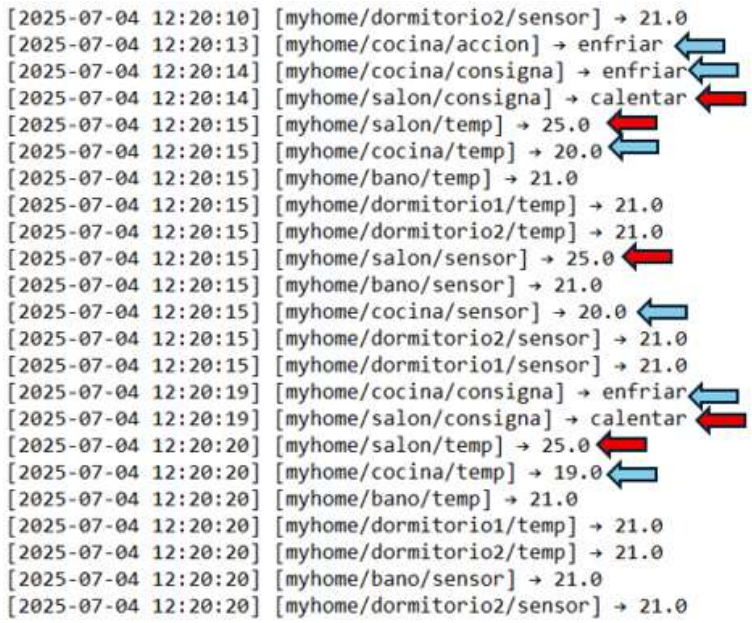


Ilustración 79. Logs del sniffer para la consigna "enfriar" de la cocina.

A continuación, se consigna a la cocina a enfriar, y, como se aprecia en la Ilustración 80Ilustración 79, la respuesta es instantánea para la próxima remesa de consignas, y se observa como la temperatura de la cocina disminuye un grado, hasta los 20 ºC. En el siguiente ciclo, desciende otro grado. Además, señalado en rojo, se sigue manteniendo el proceso subyacente de calentado del salón, que ha alcanzado la cota de los 25 ºC de máxima.

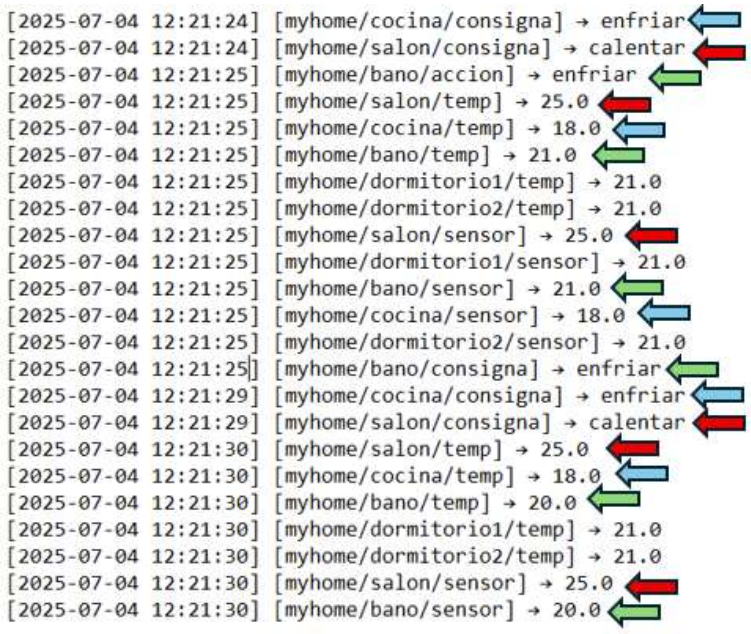


Ilustración 80. Logs del sniffer para la consigna "enfriar" del baño.

Ahora, en verde en la Ilustración 80, se muestra la variación de temperaturas del baño tras establecerse la consigna “calentar”. La acción entra en el mismo momento que el registro, y por tanto, el cambio en el proceso queda registrado como posterior al registro de la temperatura del baño, y, por ello, no se actualiza instantáneamente, como ocurrió también en el enfriado del baño en la fase 1. Por otro lado, en azul y rojo se observan los procesos subyacentes de enfriado de la cocina y calentado del salón, respectivamente, que han llegado a sus temperaturas máxima y mínima en ambos casos.

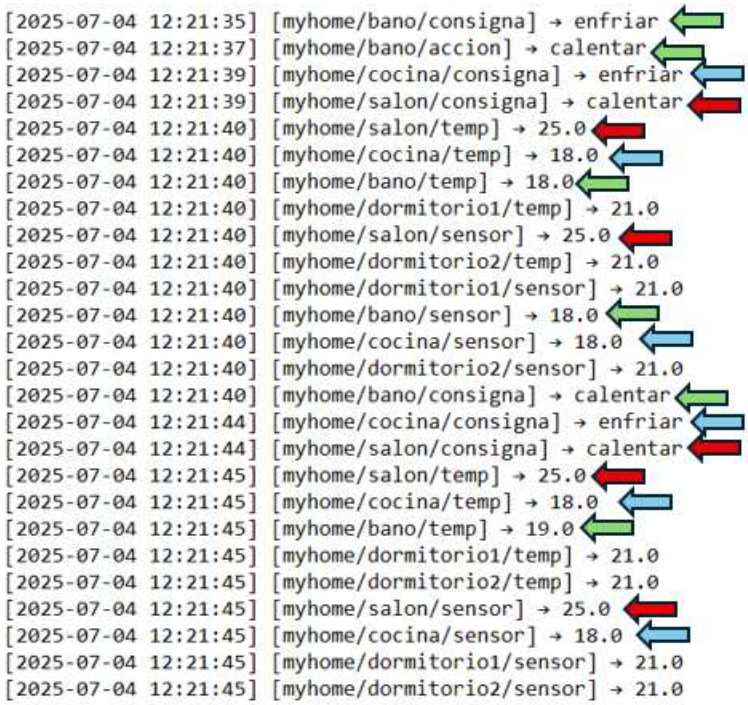


Ilustración 81. Logs del sniffer para la consigna "calentar" del baño.

Al calentarse el baño (Ilustración 81), ocurre lo mismo: la acción entra en el momento en que se introduce por la consola, pero el proceso lo actualiza de forma posterior a la publicación de la temperatura. Sin embargo, para el siguiente ciclo, se observa como ya ha comenzado el proceso de calentamiento del baño. En azul y rojo se siguen mostrando los procesos subyacentes que tienen lugar tanto en cocina como en salón.

6.3.4 CONCLUSIÓN

Como conclusión, se debe destacar que los resultados obtenidos concuerdan con los esperados:

- No existe tráfico en la capa docker0, debido a que esta capa es únicamente para descubrimiento multicast y esta funcionalidad no ha sido tenida en cuenta para el modelo de la fase 2.
- Las comunicaciones se dan entre los clientes y el router, en formato TCP y cifrados con TLS (versión 1.2). El cifrado es además efectivo, porque desde software espía como Wireshark no se puede interpretar el contenido de los mismos.

- Los logs del sniffer vuelven a mostrar que la respuesta del sistema a los cambios en las consignas es la correcta, pese al cambio de modelo, puesto que se encuentra dentro del marco de los 5 segundos en los que sensores y proceso publican las temperaturas.

6.4 FASE 3

En la Fase 3, el objetivo pasará a comparar la respuesta entre ambos modelos escalados para 10, 25 y 50 habitaciones, respectivamente. Si bien el escalado se hará de 5 en 5 habitaciones para poder generar datos suficientes, se marcarán esos tres puntos para realizar las consideraciones necesarias, y actualizar sobre el estado del escalado de ambos modelos. Para analizar dicha respuesta, se atenderá únicamente a la información del sniffer, puesto que las informaciones de las redes docker0 y eth0 es esencialmente la misma que en sus modelos de inferior tamaño. El interés del análisis de resultados en esta fase reside en poder analizar si existe una diferencia significativa entre los modelos multicast soportado por unicast (a partir de ahora, denominado modelo F1), y unicast puro (a partir de ahora, denominado F2).

6.4.1 ESCALADO 0 (5 HABITACIONES)

Pese a que estos resultados ya se han discutido en los puntos anteriores, conviene recordar desde el punto en que se parte para ambos modelos. En ambos casos, la respuesta temporal fue inferior a la ventana de 5 segundos establecida (el tiempo entre actualizaciones de temperatura), y por tanto fue una respuesta óptima. Se buscará que, en el escalado, las respuestas sigan siendo óptimas.

6.4.2 ESCALADO 1 (10 HABITACIONES)

En primer lugar, para el modelo F1, se han determinado las siguientes consignas (Tabla 9):

Habitación	**Consigna**	**Hora**
Habitación 10	Calentar	16:39:03

Habitación 1	Enfriar	16:39:12

Tabla 9. Consignas por habitación y fecha del modelo F1 para el escalado 1.

A continuación, se mostrarán en los logs del sniffer, marcadas con distintos colores (rojo para “calentar”, azul para “enfriar”), ambas consignas (Ilustración 82 e Ilustración 83):

```
[2025-07-04 16:39:01] [myhome/habitacion7/sensor] → 21.0
[2025-07-04 16:39:03] [myhome/habitacion10/accion] → calentar
[2025-07-04 16:39:06] [myhome/habitacion10/consigna] → calentar
[2025-07-04 16:39:06] [myhome/habitacion1/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion2/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion3/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion4/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion5/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion6/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion7/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion8/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion9/temp] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion10/temp] → 22.0
[2025-07-04 16:39:06] [myhome/habitacion1/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion5/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion4/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion9/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion8/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion6/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion2/sensor] → 21.0
[2025-07-04 16:39:06] [myhome/habitacion10/sensor] → 22.0
[2025-07-04 16:39:06] [myhome/habitacion3/sensor] → 21.0
```

Ilustración 82. Logs de la consigna "calentar" de la habitación 10.

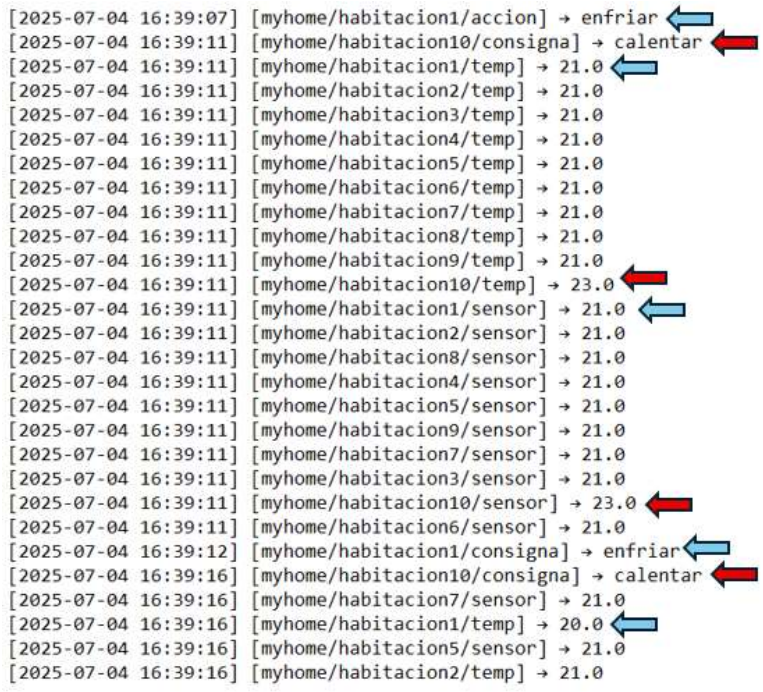


Ilustración 83. Logs de la consigna "enfriar" de la habitación 1.

Se puede apreciar que la actualización del tópico "acción" en ambos casos sigue siendo inmediata. Sin embargo, al igual que le ocurría al modelo F1 sin escalar, la consigna "enfriar" parece entrar tarde (después de las actualizaciones de temperatura), lo que impide que la temperatura se vea alterada en el primer ciclo, si bien disminuye correctamente en el segundo. En cuanto al modo "calentar", la temperatura de la habitación 10 se ve incrementada adecuadamente en la primera remesa de datos proporcionada por el sniffer. Por tanto, se puede confirmar que, aunque tarde más en actualizar un modo que otro, ambos entran dentro de la ventana óptima (no tardan más de un ciclo en actualizarse).

Por otro lado, para el modelo F2 de vivienda, se han determinado las siguientes consignas, recogidas en la Tabla 10. Consignas por habitación y fecha del modelo F2 para el escalado 1:

Habitación	**Consigna**	**Hora**
Habitación 10	Calentar	22:20:51

Habitación 1	Enfriar	22:21:00

Tabla 10. Consignas por habitación y fecha del modelo F2 para el escalado 1.

Al igual que para el modelo F1, en la Ilustración 84 y la Ilustración 85 se mostrarán los logs del sniffer, con las consignas marcadas en distintos colores (azul para “enfriar”, rojo para “calentar”):

```
[2025-07-09 22:20:51] [myhome/habitacion10/accion] → calentar
[2025-07-09 22:20:53] [myhome/habitacion10/consigna] → calentar
[2025-07-09 22:20:55] [myhome/habitacion1/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion2/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion3/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion4/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion5/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion6/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion7/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion8/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion9/temp] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion6/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion10/temp] → 22.0
[2025-07-09 22:20:55] [myhome/habitacion4/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion2/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion1/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion5/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion3/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion7/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion8/sensor] → 21.0
[2025-07-09 22:20:55] [myhome/habitacion10/sensor] → 22.0
[2025-07-09 22:20:55] [myhome/habitacion9/sensor] → 21.0
[2025-07-09 22:20:58] [myhome/habitacion10/consigna] → calentar
[2025-07-09 22:21:00] [myhome/habitacion1/temp] → 21.0
```

Ilustración 84. Logs de la consigna "calentar" de la habitación 10.

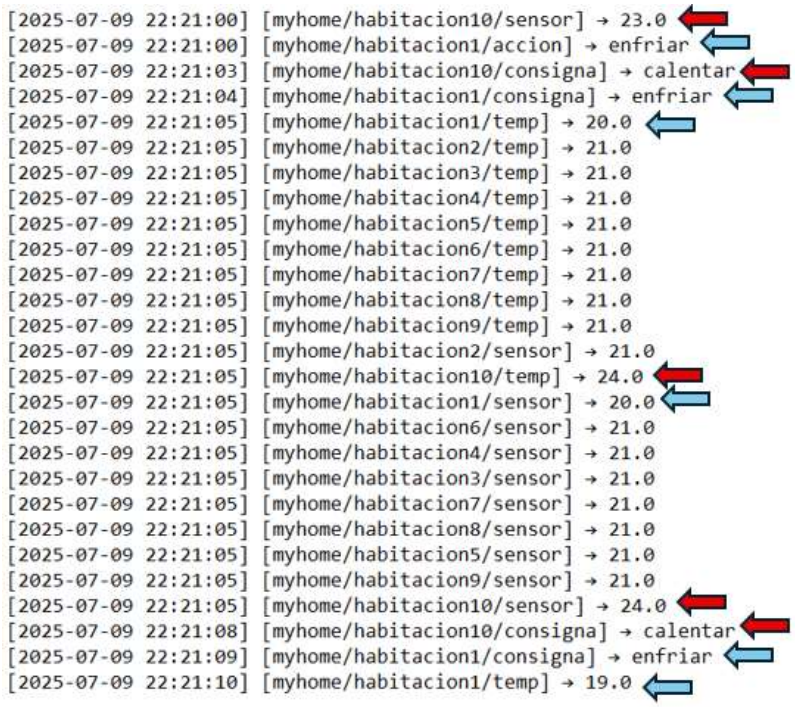


Ilustración 85. Logs de la consigna "enfriar" de la habitación 1.

En el modelo F2, el retraso entre la consigna enfriar y la actualización de la temperatura desaparece, y ambas temperaturas son actualizadas en el ciclo posterior a la consigna. En este caso, también se puede confirmar que se cumple el tiempo de reacción establecido como óptimo de 5 segundos, al igual que en el modelo F1.

6.4.3 ESCALADO 2 (25 HABITACIONES)

En primer lugar, se ha tratado de llevar a cabo el escalado del modelo F1 a 25 habitaciones, aunque sin éxito. En primer lugar, se trató de escalar de la misma manera que para las 10 habitaciones, obteniéndose un error en la carga de la página de la API, mostrado en la Ilustración 86.

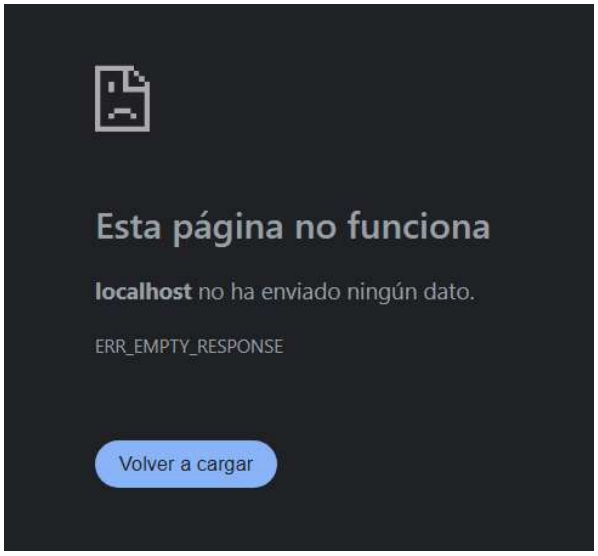


Ilustración 86. Respuesta del local host al iniciar la consola.

Para comprobar si es cuestión de la conexión o del sistema, se trataron de hacer diversas modificaciones para mejorar el rendimiento, como, por ejemplo, ejecutar zenoh en un hilo secundario separado del arranque de la API. Sin embargo, se seguía sin obtener respuesta, por lo que se pasó a construir ZENOH como un contenedor independiente, añadiéndolo de la siguiente manera al docker compose:

```
zenoh:
    image: eclipse/zenoh:latest
    container_name: zenoh
    ports:
      - "7447:7447"
      - "8001:8001"
    networks:
      - zenoh-net
```

Se añade el puerto 8001 dado que es un puerto libre de conexiones, que no dará problemas. Sin embargo, al consultar los logs internos del contenedor, se obtuvo la respuesta mostrada en la Ilustración 87:

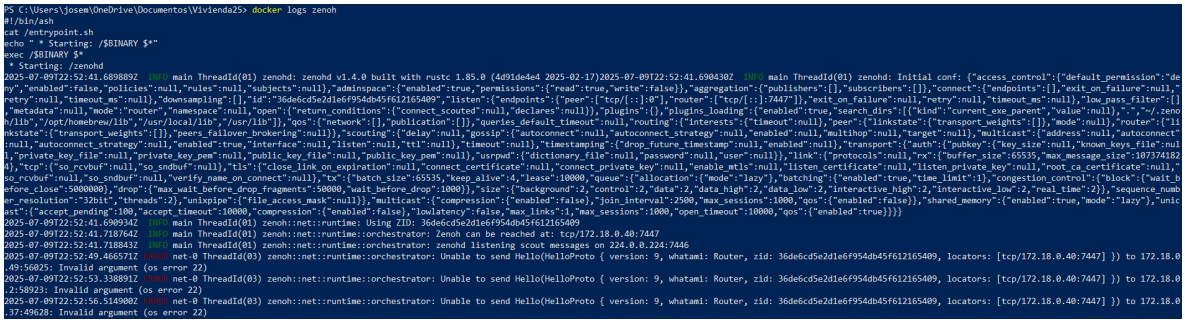


Ilustración 87. Error en los logs del contenedor "zenoh".

El mensaje de error "unable to send Hello(HelloProto)" implica que zenoh está tratando de establecer conexiones con los elementos del sistema, pero sin resultado. Por tanto, queda claro que es un error de la red docker y como se está (o no se está) montando. El motivo de este error se debe a una de las bases del ensamblado de este proyecto: aunque apoyado en Docker, el sistema operativo sobre el que se cimenta todo es Windows, y, por tanto, para poder operar con Docker se crea una máquina virtual de Linux en el interior del sistema operativo. Sin embargo, el soporte que el método multicast tiene dentro de estas máquinas virtuales es muy pobre y, debido a eso, a la hora de escalarlo y exigir mayores requerimientos, este se ve forzado a parar. El tráfico multicast entonces ya no llega ni al propio host de Windows, ni a los contenedores (el contenedor zenoh no puede iniciar el protocolo de conexión). Todas las soluciones a este problema se plantean entonces desde construir el proyecto desde cero en Linux, ya que no hay soluciones posibles en Windows. Debido a la imposibilidad de dicha acometida, se plantea entonces el escalado a 25 habitaciones del modelo F1 como un escalado fallido.

Por otro lado, el modelo F2 que no soporta ninguna clase de conexión multicast sí que ha sido posible de escalar, y se han realizado las distintas consignas (Tabla 11) para comprobar que la velocidad de respuesta sigue estando dentro del rango óptimo.

Habitación	**Consigna**	**Hora**
Habitación 13	Enfriar	22:49:17

Habitación 24	Calentar	22:49:30

Tabla 11. Consignas por habitación y fecha del modelo F2 para el escalado 2.

A continuación, en la Ilustración 88 y la Ilustración 89 se muestran los logs del sniffer para las consignas determinadas, y, al igual que en el escalado anterior, se mostrará en rojo el proceso de calentamiento y en azul el de enfriamiento:

```
[2025-07-09 22:49:17] [myhome/habitacion13/accion] → enfriar
[2025-07-09 22:49:20] [myhome/habitacion13/consigna] → enfriar
[2025-07-09 22:49:21] [myhome/habitacion1/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion2/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion3/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion4/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion1/sensor] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion5/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion6/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion7/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion8/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion9/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion10/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion11/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion12/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion13/temp] → 20.0
[2025-07-09 22:49:21] [myhome/habitacion14/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion15/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion16/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion17/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion18/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion19/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion20/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion21/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion22/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion23/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion2/sensor] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion24/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion25/temp] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion23/sensor] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion24/sensor] → 21.0
[2025-07-09 22:49:21] [myhome/habitacion25/sensor] → 21.0
```

Ilustración 88. Logs de la consigna "enfriar" de la habitación 13.

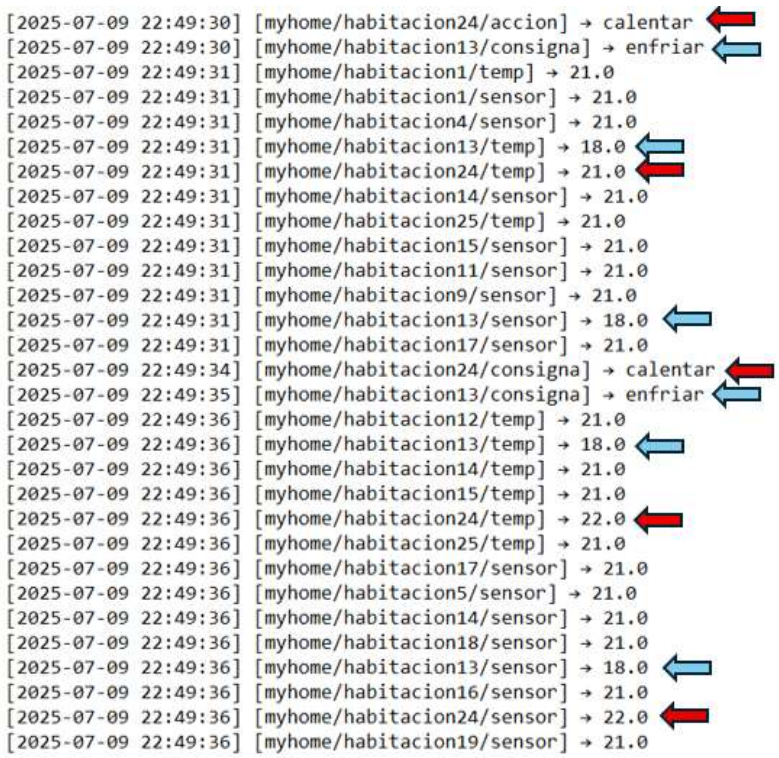


Ilustración 89. Logs de la consigna "calentar" de la habitación 24.

Con el propósito de mostrar correctamente la mayor cantidad de logs, se han eliminado algunos logs intermedios que contenían información poco útil (temperaturas y sensores de las demás habitaciones, que se mantienen a 21 ºC y que no aportan información). Se ha tomado esta determinación debido a que la lista de logs es muy larga, al tratarse de 50 instancias por ciclo. Como se puede comprobar, en el caso de calentar la habitación, la consigna queda registrada en un punto posterior al registro de la acción en el tópico propio, y, por tanto, no queda actualizada la temperatura en la primera remesa tras la consigna, sino en la segunda. Sin embargo, en el caso del enfriamiento de la habitación 13, la actualización de temperatura se realiza en el ciclo inmediatamente posterior a la consigna. En ambos casos, se sigue cumpliendo la condición de ventana de operabilidad de 5 segundos, y, por tanto, el rendimiento del sistema sigue siendo óptimo.

6.4.4 ESCALADO 3 (50 HABITACIONES)

Con el modelo F1 fuera de combate, el único modelo disponible para escalar es el modelo F2. Sin embargo, antes de llegar a las 50 habitaciones, la red Docker ha cedido de la misma forma que cedió en el caso anterior. Como se puede observar en la Ilustración 90, el error obtenido en el local host es el mismo.

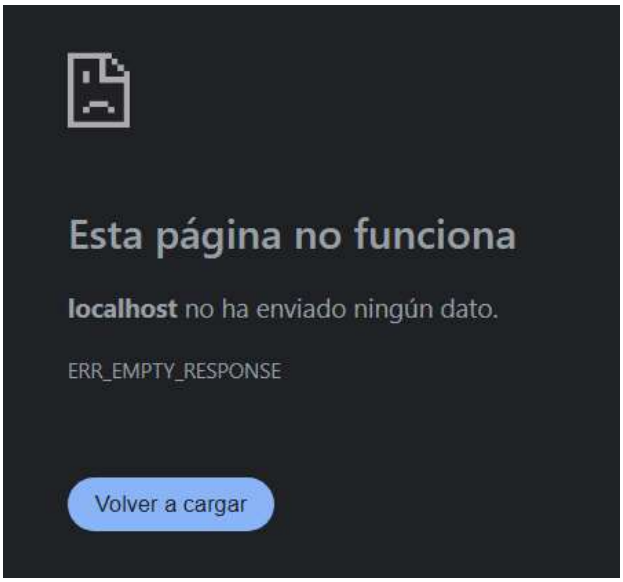


Ilustración 90. Error del local host al intentar iniciar la consola.

Sin embargo, el error ahora no se produce debido a la falta de soporte para la red multicast de Docker, sino debido a que el volumen de la red que se debía generar para el funcionamiento del sistema sobrepasaba las capacidades del ordenador. Habitualmente, Docker emplea entorno al 10% de la CPU disponible para simplemente la construcción de la red. Sin embargo, en este caso, se llegó al 32%, es decir, aumentó en más de tres veces el consumo de recurso únicamente en la construcción de los contenedores (Ilustración 91).

Ilustración 91. Gráfico de uso de CPU por parte de Docker.

Hay que tener en cuenta, que, para este punto, el número de contenedores supera los 100, y esto implica el mantener más de 100 conexiones distintas en un entorno de máquina virtual. Las capacidades físicas del ordenador simplemente cedieron ante los requerimientos de la red, impidiendo el montaje de la misma.

6.4.5 CONCLUSIONES

Como se ha podido comprobar, el funcionamiento de ambos modelos al escalado es adecuado cuando este es pequeño. Sin embargo, debido al sistema operativo en que se desarrolla el proyecto, el modelo F1 termina teniendo muchos problemas para lanzar su red docker interna, debido al poco soporte que recibe el multicast en las máquinas virtuales. Por otro lado, el escalado en el modelo F2 se comporta de manera adecuada y óptima, similar al de sus predecesores más pequeños. Sin embargo, conforme se va ampliando más y más el sistema, Docker pierde la capacidad para montar la red y poder poner en funcionamiento el protocolo.

Sin embargo, hasta el fallo de los dos modelos por culpa de la red Docker, se observaba un comportamiento normal y óptimo del sistema, lo que lleva a pensar que realmente el proyecto de escalado es un "fracaso" a grandes escalas debido a las limitaciones del hardware y software de simulación empleado, y no debido a las limitaciones del protocolo.

En la Tabla 12 se resume el desempeño de ambos modelos para cada tipo de escalado.

	Modelo F1	**Modelo F2**
5 Habitaciones	Cumple	Cumple
10 Habitaciones	Cumple	Cumple
25 Habitaciones	No es capaz de lanzarlo	Cumple
50 Habitaciones	No es capaz de lanzarlo	No es capaz de lanzarlo

Tabla 12. Desempeño de cada modelo en cada escalado.

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

Los objetivos que se marcaron para este proyecto, mencionados en la sección 4.2, fueron los siguientes:

- Investigar y estudiar sobre las distintas funcionalidades del protocolo Zenoh, explorando tanto su modo multicast como su modo unicast, y ponerlas en marcha en un sistema diseñado para ello (en este caso, el sistema de climatización de una vivienda)
- Como consecuencia del estudio, analizar los resultados en cuanto a eficiencia y rendimiento del protocolo, para poder determinar las ventajas y desventajas de su uso.

En lo referente al primer objetivo, este se ha cumplido con creces. Se ha diseñado desde cero en Python un modelo de vivienda virtual en el que se ha implementado el protocolo Zenoh, y se ha hecho uso de todas las funcionalidades que se han podido: modos de conexión multicast (modelo F1) y unicast (modelo F2), plugins (REST y API) y seguridad (autenticación y TLS). El desarrollo de dos modelos de comunicación distintos con la misma base ha permitido poder estudiar las ventajas y las desventajas de ambos modelos, de cara a la comparativa entre ambos.

Para el segundo objetivo, se dividirá este capítulo en tres secciones distintas, para abarcar las conclusiones sobre Zenoh como protocolo de comunicación, las conclusiones sobre los distintos modelos Zenoh empleados, y una última sección con los futuros trabajos derivados de este proyecto.

7.1 CONCLUSIONES SOBRE ZENOH COMO PROTOCOLO

En cuanto al segundo objetivo, en base a los resultados obtenidos en el Capítulo 6. y lo aprendido durante el desarrollo del propio sistema en el Capítulo 5. se pueden llegar a las siguientes conclusiones respecto a las ventajas y desventajas de Zenoh como protocolo de comunicación.

7.1.1 VENTAJAS DE ZENOH

En cuanto a las ventajas, se observa:

- Simplicidad: Como se indica en la página web, Zenoh es un protocolo bastante sencillo tanto de implementar dentro de un sistema, como de comprender. Sin tener conocimientos previos sobre teoría de las comunicaciones, protocolos de comunicaciones, programación en Python y protocolos de red, se ha conseguido desarrollar un sistema operativo con distintas funcionalidades, con integración de distintos elementos y cuyas comunicaciones están enteramente soportadas por Zenoh.
- Eficiencia: a la vista de los resultados está que las comunicaciones entran siempre dentro del rango de tiempo óptimo de 5 segundos que se marcó en la simulación del proceso. Los tópicos en donde se publican las consignas eran actualizados en cuestión de milisegundos (en ningún caso se superó el segundo), y el sistema recibía la información en vivo y al momento. Además, desde la consola también se podía hacer un seguimiento en vivo de las temperaturas de las habitaciones, observando cómo se actualizaban las temperaturas cada 5 segundos tras haber realizado una consigna.
- Escalabilidad: Los modelos han sido fácilmente escalables. Si bien es cierto que se ha hecho uso de una herramienta como Docker, el único requerimiento para la escalabilidad del sistema ha sido modificar ligeramente los scripts para poder hacer frente a la lectura de archivos tipo json. Por lo demás, al contar con archivos de definición de comportamiento (peer, cliente o router), simplemente ha sido necesario indicar el número de habitaciones que se quería a la hora de aumentar el sistema. Por otro lado, cuando se han escalado ambos modelos, no se ha registrado un empeoramiento de la respuesta del sistema, sino que la eficiencia seguía siendo la misma hasta el colapso de la red. Por tanto, se puede razonar que el protocolo seguía funcionando de manera correcta, y que han sido en este caso las capacidades del hardware las que han puesto las limitaciones.

- Compatibilidad: Se ha utilizado Zenoh en conjunto con otras herramientas como pueden ser Docker, Tcpdump, FastAPI y plugins REST, y el resultado ha sido excelente. La sinergia que ha mantenido con todas las herramientas, especialmente con Docker, ha resultado crucial a la hora de poder desarrollar el proyecto de forma ordenada e informada, puesto que no se ha necesitado de ningún proceso de ensamblado o similar para hacer funcionar las distintas herramientas.

7.1.2 DESVENTAJAS DE ZENOH

Pese a que las ventajas superan en número y peso a las desventajas, es importante también comentarlas. Cabe resaltar que las desventajas se centran, principalmente, en la amabilidad de Zenoh con el nuevo usuario:

- Instalación: La instalación en sí no es complicada, pero, como se ha mencionado, la falta de conocimientos previos en asuntos de programación volvió una tarea a priori sencilla en un proceso lento y complicado. La guía de instalación no es especialmente profunda, y el hecho de que se requieran diversos elementos como Cargo o Rust para su funcionamiento hace que pueda resultar liosa su instalación si no se tienen conocimientos sobre ello.
- Poca claridad: Este punto puede parecer incongruente con el apartado anterior, en el que se ha comentado que una de las ventajas de Zenoh es su simplicidad. Si bien es cierto que el protocolo en sí es muy simple y su forma de implementarse también, la página web oficial de Zenoh no es especialmente amigable con los nuevos usuarios. La información se muestra directamente, sin ningún tipo de introducción o contexto, y por tanto si no se tienen los conocimientos previos necesarios se puede volver complicada la tarea de entender todas las funcionalidades que el protocolo ofrece. Este punto no tiene tanto que ver con el protocolo en sí, sino con la forma en la que se muestra al público, y no presenta una desventaja en cuanto a factores como el rendimiento o la operabilidad, pero se ha considerado necesario comentarlo, puesto que, en el caso de este proyecto, la falta de conocimientos previos en el ámbito de las comunicaciones tuvo como consecuencia un inicio algo lento y complicado.

- Documentación: Zenoh es un protocolo de vanguardia, que se encuentra en actualizaciones constantes. Al inicio de este proyecto (octubre de 2024), la versión de Zenoh era la 1.2.1, y para la conclusión del mismo (julio de 2025), la versión actual es la 1.4.0. Esto es una muestra de las constantes mejoras que se implementan al protocolo, lo cual se podría considerar (porque lo es) como una ventaja. Sin embargo, el hecho de las constantes actualizaciones es que, en ocasiones, términos o líneas de código que funcionaban en versiones antiguas no funcionan en las versiones modernas. Por tanto, el proyecto comienza a dar errores y es complicado determinar la causa del error cuando, a priori, nada ha cambiado y ha pasado de funcionar a no funcionar. Si bien esto es fácilmente solucionable simplemente fijando la versión de Zenoh en la que se quiere trabajar, a la hora de realizar proyectos a largo plazo en los que se tenga interés en actualizar las versiones de Zenoh conforme van saliendo tiene importancia. Las líneas de error en estos casos no muestran información indicando que el comando utilizado pertenece a una versión anterior, y simplemente informan de "unknown key".
- Novedad: El hecho de que Zenoh sea un protocolo tan nuevo implica también que la adaptación por parte del resto de tecnologías no está suficientemente establecida. Se ha hablado de la compatibilidad de Zenoh con herramientas como Docker, en el caso de Wireshark, si bien en el caso de este último las conexiones eran fácilmente identificables, aún no cuenta con una funcionalidad que permita descifrar ni el tipo de protocolo ni el contenido de los datagramas. No se ha conseguido encontrar aún un archivo Lua con los comandos para que Wireshark sea capaz de leer los paquetes de datos, y, para poder comprobar que las actualizaciones estaban siendo realizadas correctamente, se ha tenido que implementar en el sistema un sniffer.

7.2 CONCLUSIONES SOBRE LOS DISTINTOS MODELOS ZENOH

Si bien en cuanto a términos de eficiencia, ambos han cumplido con el objetivo de mantenerse en el rango óptimo de respuesta, ambos modelos presentan diferencias. En este

punto, se analizarán las ventajas de cada modelo con respecto del otro, a fin de compararlos y determinar en qué situación es útil cada uno.

7.2.1 VENTAJAS DEL MODELO F1 RESPECTO AL MODELO F2

En primer lugar, respecto a las ventajas del modelo F1, se observan:

- Simplicidad: El modelo F1 es mucho más sencillo de implementar que el modelo F2. Debido a que es el modo predeterminado de Zenoh, la gran mayoría del código está pensado de forma predeterminada para este tipo de comunicaciones, y, por tanto, es más fácil implementarlo y avanzar en él. No ha sido necesario, por ejemplo, la implementación de un archivo de configuración para los clientes, o determinar los puertos en los que tenían que establecerse las comunicaciones, puesto que todo eso lo hace Zenoh de forma predeterminada.
- Ligereza: al no tener que establecer los certificados de autenticación y TLS de cada uno de los miembros, el sistema es ligero en cuanto al número de archivos abiertos se refiere. Esto no puede decirse del modelo F2, en el que se tuvo que realizar una modificación en el docker-compose para aumentar el límite máximo de archivos abiertos de forma simultánea, ya que este había sido excedido.
- Automatización: El hecho del descubrimiento multicast hace que las conexiones se establezcan de forma automática, y, por tanto, no se requiera determinarlas en ningún momento manualmente. Esto, de cara a la escalabilidad, es una gran ventaja, dado que supone un gran ahorro de tiempo.

En general, el modelo F1 es más sencillo puesto que implementa menos características de Zenoh, lo cual hace que también sea más ligero y fácil de manejar. En casos en los que la seguridad no sea un problema (redes cerradas), y que se tenga intención de poder variar el número de elementos del sistema, será la opción preferente.

7.2.2 VENTAJAS DEL MODELO F2 RESPECTO AL MODELO F1

Por el lado de las ventajas del modelo F2, se puede observar:

- Seguridad: Este punto es bastante evidente, pero destacable debido a que para sistemas que requieran de esta característica, es importante. La seguridad que aporta el cifrado TLS y la autenticación permite que los datos no puedan ser accedidos mediante las tecnologías de software de captura de tráfico disponibles en el mercado, y se muestra como una gran ventaja a la hora de aplicar este modelo.
- Unicast: El hecho de no contar con descubrimiento multicast permite que este modelo sea aplicable en redes que no pueden las comunicaciones multicast, o que su capacidad para soportar este tipo de redes está limitada (como la red Docker). Unicast es la forma en la que la gran mayoría de redes están configuradas para soportar, y, por tanto, Zenoh en este formato estará correctamente soportado.

El modelo F2 es más complejo debido a que implementa una malla de comunicaciones mayor, y capas de seguridad y cifrad. Será la opción más adecuada en casos en los que la seguridad sea un factor fundamental, o las capacidades del sistema no permitan el descubrimiento multicast.

7.3 TRABAJOS FUTUROS

En este proyecto se ha investigado e implementado Zenoh sin haber tenido, previamente, una base de conocimientos en protocolos, comunicaciones y programación. Aunque los resultados son más que satisfactorios, este hecho ha sido fundamental a la hora de los objetivos que se han marcado para el proyecto y como este ha sido desarrollado. De cara a realizar futuras investigaciones sobre el asunto, y a desarrollar aún más el proyecto, se plantean distintas vías de trabajo:

- Desarrollo en Linux: El motivo fundamental del uso del sistema operativo Windows ha sido su disponibilidad, pues es el sistema operativo del ordenador en el que se ha desarrollado el sistema. Por ello, para el funcionamiento de Docker, se ha tenido que emplear una máquina virtual, que ha menguado las capacidades de este para rendir al máximo. De cara al futuro, se puede plantear la realización de este mismo

proyecto, pero en un sistema operativo Linux, en el cual se tengan menos restricciones de redes.

- Aplicación real a un entorno industrial: En este proyecto se ha simulado un sistema de climatización, que ha debido ser construido desde cero. Una segunda etapa del proyecto podría incluir la aplicación de Zenoh a un sistema de climatización real, en el que los elementos del sistema sean componentes reales y no scripts que las emulen.
- Sinergia con protocolos: En vista de la compatibilidad que ha presentado Zenoh con las herramientas utilizadas, y la compatibilidad planteada en la web, un camino para seguir desarrollando el proyecto es la implementación conjunta de Zenoh con otros protocolos del mercado, como PROFINET u OPC UA.

A modo de cierre de memoria, y como conclusión última, se puede decir que en este proyecto ha realizado un estudio sobre Zenoh, que es un protocolo de comunicación pub/sub/query lanzado recientemente al mercado, aplicándolo sobre un sistema de climatización de un modelo de vivienda virtual que ha sido diseñado en Python desde cero. El proyecto, dividido en fases, ha tenido como objetivo abarcar las distintas funcionalidades que presenta Zenoh, entregando, al finalizar el mismo, unos resultados satisfactorios. La compatibilidad, eficiencia y escalabilidad se plantean como los puntos más fuertes de un protocolo Zenoh, que, sin embargo, también tiene puntos débiles, centrados principalmente en su novedad y falta de amabilidad con los nuevos usuarios. De cara al futuro, como vías de desarrollo del proyecto, la implementación de este protocolo en entornos reales, o junto a otros protocolos de comunicación ya establecidos en el mercado, se plantea como una posibilidad interesante y que debe ser, sin duda, tenida en cuenta.

BIBLIOGRAFÍA

[1] Zenoh, «What is Zenoh?,» [En línea]. Available: https://zenoh.io/docs/overview/what-is-zenoh/. [Último acceso: 18 Julio 2025].

[2] Docker, «What is Docker?,» Docker, [En línea]. Available: https://docs.docker.com/get-started/docker-overview/. [Último acceso: 18 Julio 2025].

[3] Wireshark, «Wireshark's User Guide,» Wireshark, [En línea]. Available: https://www.wireshark.org/docs/wsug_html_chunked/. [Último acceso: 18 Julio 2025].

[4] M. Vergara Carmona, «Guía del comando tcpdump,» SYSADMIN, 3 Enero 2024. [En línea]. Available: https://vergaracarmona.es/guia-del-comando-tcpdump/. [Último acceso: 18 Julio 2025].

[5] Repsol, «Todo sobre la industria 4.0,» Repsol, [En línea]. Available: https://www.repsol.com/es/energia-futuro/tecnologia-innovacion/cuarta-revolucion-industrial/index.cshtml. [Último acceso: 18 Julio 2025].

[6] Equipo Editorial de IONOS, «¿Qué es el modelo OSI?,» 3 Noviembre 2022. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/unicast/. [Último acceso: 18 Julio 2025].

[7] Cloudfare, «¿Qué es el modelo OSI?,» [En línea]. Available: https://www.cloudflare.com/es-es/learning/ddos/glossary/open-systems-interconnection-model-osi/. [Último acceso: 18 Julio 2025].

[8] J. L. Martínez, «El modelo OSI,» 15 Noviembre 2018. [En línea]. Available: https://www.prored.es/el-modelo-osi/. [Último acceso: 18 Julio 2025].

[9] castr, «Unicast vs Multicast vs Broadcast: What's the Difference?,» 9 Mayo 2023. [En línea]. Available: https://castr.com/blog/unicast-vs-multicast-vs-broadcast/. [Último acceso: 18 Julio 2025].

[10] Equipo Editorial de IONOS, «Unicast: conexión dirigida entre dos puntos.,» 26 3 2019. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/unicast/. [Último acceso: 18 Julio 2025].

[11] Fortinet, «¿Qué es el protocolo de control de transmisión TCP/IP?,» [En línea]. Available: https://www.fortinet.com/lat/resources/cyberglossary/tcp-ip. [Último acceso: 18 Julio 2025].

[12] Bunny.net, «What Does the TCP Protocol Stand for and What is the Three Way Handshake?,» [En línea]. Available: https://bunny.net/academy/network/what-is-transmission-control-protocol-tcp/. [Último acceso: 18 Julio 2025].

[13] S. De Luz, «Tráfico Multicast: Qué es y por qué es tan importante.,» Redes zone, 2 Junio 2025. [En línea]. Available: https://www.redeszone.net/tutoriales/internet/que-es-trafico-multicast/. [Último acceso: 18 Julio 2025].

[14] Bunny.net, «What is the User Datagram Protocol (UDP) in Computer Networks?,» [En línea]. Available: https://bunny.net/academy/network/what-is-user-datagram-protocol-udp-and-how-does-it-work/. [Último acceso: 18 Julio 2025].

[15] Lowi, «Glosario Lowi,» [En línea]. Available: https://www.lowi.es/glosario/que-es-protocolo-de-comunicacion/. [Último acceso: 18 Julio 2025].

[16] «¿Qué es Pub/Sub?,» 16 Junio 2025. [En línea]. Available: https://cloud.google.com/pubsub/docs/overview?hl=es-419. [Último acceso: 18 Julio 2025].

[17] AWS, «¿Qué son los mensajes de publicación y suscripción?,» AWS, [En línea]. Available: https://aws.amazon.com/es/what-is/pub-sub-messaging/. [Último acceso: 18 Julio 2025].

[18] Zenoh, «Getting Started - Deployment,» [En línea]. Available: https://zenoh.io/docs/getting-started/deployment/. [Último acceso: 18 Julio 2025].

[19] Equipo Editorial de IONOS, «¿Cómo funciona el modelo cliente-servidor?,» 31 Enero 2023. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/modelo-cliente-servidor/. [Último acceso: 18 Julio 2025].

[20] Equipo editorial de IONOS, «¿Qué es P2P (peer to peer)?,» 12 Mayo 2023. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-p2p-peer-to-peer/. [Último acceso: 18 Julio 2025].

[21] Equipo Editorial IONOS, «¿Qué es un plugin y para qué se usa?,» IONOS Digital Guide, 9 Octubre 2020. [En línea]. Available: https://www.ionos.es/digitalguide/servidores/know-how/que-es-un-plugin/. [Último acceso: 18 Julio 2025].

[22] Zenoh, «Repositorio eclipse-zenoh,» Github, [En línea]. Available: https://github.com/orgs/eclipse-zenoh/repositories?type=all. [Último acceso: 18 Julio 2025].

[23] Aula 21, «Profinet: Qué es y cómo funciona,» Aula 21, [En línea]. Available: https://www.cursosaula21.com/profinet-que-es-y-como-funciona/. [Último acceso: 18 Julio 2025].

[24] Profibus, «PROFINET: ¿Qué es y como funciona?,» Profibus, 15 Noviembre 2023. [En línea]. Available: https://profibus.com.ar/profinet-que-es-y-como-funciona/. [Último acceso: 18 Julio 2025].

[25] WIT Automatización, «¿Qué es PROFINET y para qué sirve?,» WIT Automatización, [En línea]. Available: https://witautomatizacion.es/que-es-profinet-y-para-que-sirve/. [Último acceso: 18 Julio 2025].

[26] Paessler, «¿Qué es OPC UA?,» Paessler, [En línea]. Available: https://www.paessler.com/es/it-explained/opc-ua. [Último acceso: 18 Julio 2025].

[27] INCIBE, «OPC UA, equilibrio entre ciberseguridad y rendimiento,» INCIBE, 11 Enero 2024. [En línea]. Available: https://www.incibe.es/incibe-cert/blog/opc-ua-equilibrio-entre-ciberseguridad-y-rendimiento. [Último acceso: 18 Julio 2025].

[28] Organización de las Naciones Unidas, «Objetivos de Desarrollo Sostenible,» Organización de las Naciones Unidas, [En línea]. Available: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Último acceso: 18 Julio 2025].

[29] J. A. Soto, «¿Qué es el kernel y para qué sirve?,» Geeknetik, 26 Julio 2020. [En línea]. Available: https://www.geeknetic.es/Kernel/que-es-y-para-que-sirve. [Último acceso: 18 Julio 2025].

[30] MrDevSecOps, «Docker Objects,» 29 Septiembre 2021. [En línea]. Available: https://medium.com/@mrdevsecops/docker-objects-e561f0ce3365. [Último acceso: 18 Julio 2025].

[31] Don Docker, «Como hacer redes con Docker,» 25 Mayo 2016. [En línea]. Available: https://dondocker.com/como-hacer-redes-con-docker/. [Último acceso: 18 Julio 2025].

[32] auth0, «¿Qué es la autenticación?,» auth0, [En línea]. Available: https://auth0.com/es/intro-to-iam/what-is-authentication. [Último acceso: 18 Julio 2025].

[33] Aratek, «5 factores de autenticación: una guía desde las contraseñas hasta la biometría,» Aratek, 31 Agosto 2023. [En línea]. Available: https://www.aratek.co/es/news/5-authentication-factors-a-guide-from-passwords-to-biometrics. [Último acceso: 18 Julio 2025].

[34] M. Balba, «¿Qué es el handshake?,» ninjaOne, 14 Noviembre 2024. [En línea]. Available: https://www.ninjaone.com/es/it-hub/endpoint-management/handshake/. [Último acceso: 18 Julio 2025].

[35] J. A. Castillo, «Protocolo TCP/IP - Qué es y cómo funciona,» 21 Marzo 2020. [En línea]. Available: https://www.profesionalreview.com/2020/03/21/protocolo-tcp-ip/. [Último acceso: 18 Julio 2025].

[36] T. Dierks, «The Transport Layer Security (TLS) Protocol version 1.2.,» Datatracker, Agosto 2008. [En línea]. Available: https://datatracker.ietf.org/doc/html/rfc5246. [Último acceso: 18 Julio 2025].

[37] Cloudfare, «¿Qué es TLS (Transport Layer Security)?,» Cloudfare, [En línea]. Available: https://www.cloudflare.com/es-es/learning/ssl/transport-layer-security-tls/. [Último acceso: 18 Julio 2025].

[38] Cloudfare, «¿Qué es una clave criptográfica?,» Cloudfare, [En línea]. Available: https://www.cloudflare.com/es-es/learning/ssl/what-is-a-cryptographic-key/. [Último acceso: 18 Julio 2025].

[39] EduEscapeRoom, «Cifrado César,» EduEscapeRoom, [En línea]. Available: https://eduescaperoom.com/cifrado-cesar/. [Último acceso: 18 Julio 2025].

[40] Cloudfare, «¿Qué es una clave de sesión? Claves de sesión y protocolos de enlace TLS.,» Cloudfare, [En línea]. Available: https://www.cloudflare.com/es-es/learning/ssl/what-is-a-session-key/. [Último acceso: 18 Julio 2025].

[41] Alice, «Cifrado César,» 9 Febrero 2018. [En línea]. Available: https://alieslutic.wordpress.com/2018/02/09/cifrado-cesar/. [Último acceso: 18 Julio 2025].

[42] OpenSSL, «OpenSSL Library,» OpenSSL, [En línea]. Available: https://openssl-library.org/. [Último acceso: 18 Julio 2025].

ANEXO I: CÓDIGO DE LA FASE I

En este anexo se encuentran recogidos los códigos de la Fase 1.

Actuador.py:

```
import zenoh
import os
import time
import threading

def main():
    room = os.getenv("ROOM", "unknown_room")

    accion_topic = f"myhome/{room}/accion"
    consigna_topic = f"myhome/{room}/consigna"

    print(f" El actuador de '{room}' escuchando acciones en: {accion_topic} y
reenviando a: {consigna_topic}")

    modo_actual = None
    modo_lock = threading.Lock()

    with zenoh.open(zenoh.Config()) as session:

        def accion_callback(sample):
            nonlocal modo_actual
            try:
                modo = bytes(sample.payload).decode("utf-8").strip().lower()
                print(f"[{room}] Acción recibida: {modo}")
                with modo_lock:
                    modo_actual = modo
            except Exception as e:
                print(f"[{room}] Error en callback_accion: {e}")

        session.declare_subscriber(accion_topic, accion_callback)

        def reenviar_loop():
            while True:
                with modo_lock:
                    modo = modo_actual
                if modo:
                    session.put(consigna_topic, modo.encode("utf-8"))
                    print(f"[{room}] Reenviando comando: {modo}")
                time.sleep(5)

        hilo_reenvio = threading.Thread(target=reenviar_loop, daemon=True)
        hilo_reenvio.start()
```

```
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(f" El actuador de '{room}' detenido.")

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

Dockerfile:

```
# Dockerfile para actuador

# Imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*

# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip

# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY actuador.py .

# Exponer puerto
```

```
EXPOSE 8080

CMD ["python", "actuador.py"]
```

Requirements.txt:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
paho-mqtt
requests
```

Consola:

```
from fastapi import FastAPI, Request, Form, HTTPException, status
from fastapi.responses import HTMLResponse, PlainTextResponse
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
import zenoh
import threading

# Crear app FastAPI
app = FastAPI()

# Permitir peticiones desde cualquier origen
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# Configurar Jinja2 para usar templates desde carpeta 'templates'
templates = Jinja2Templates(directory="templates")

habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]
temperaturas = {hab: "—" for hab in habitaciones}
temperaturas_lock = threading.Lock()

# Abrir sesión Zenoh para recibir datos y enviar accions
zsession = zenoh.open(zenoh.Config())

# Callback para actualizar temperaturas al recibir datos de Zenoh
def listener(room):
    def callback(sample):
        try:
            payload_bytes = bytes(sample.payload)
            payload_str = payload_bytes.decode("utf-8")

            with temperaturas_lock:
                temperaturas[room] = payload_str

            print(f"[{room}] Temperatura actualizada: {payload_str}°C")
        except Exception as e:
```

```
            print(f"Error al procesar muestra para {room}: {e}")
    return callback

# Suscribirse a las temperaturas de cada habitación en Zenoh
def suscribir_temperaturas():
    for hab in habitaciones:
        topic = f"myhome/{hab}/temp"
        zsession.declare_subscriber(topic, listener(hab))
        print(f"Suscrito a: {topic}")

# Ejecutar la suscripción en hilo aparte para no bloquear FastAPI
threading.Thread(target=suscribir_temperaturas, daemon=True).start()

# Endpoint para mostrar la página principal con el estado y el formulario
@app.get("/", response_class=HTMLResponse)
async def index(request: Request):
    with temperaturas_lock:
        temp_copy = temperaturas.copy()

    return templates.TemplateResponse("index.html", {
        "request": request,
        "temperaturas": temp_copy,
        "habitaciones": habitaciones,
    })

# Endpoint para recibir la accion enviada desde el formulario
@app.post("/accion", response_class=HTMLResponse)
async def set_accion(request: Request, habitacion: str = Form(...), accion: str =
Form(...)):
    try:
        if habitacion not in habitaciones:
            mensaje = f" Habitación no reconocida: {habitacion}"
        else:
            key = f"myhome/{habitacion}/accion"
            zsession.put(key, accion.encode("utf-8"))
            mensaje = f"Acción '{accion}' enviada a {habitacion}"
            print(f"[consola] Enviada acción: {accion} → {key}")

        with temperaturas_lock:
            temp_copy = temperaturas.copy()

        return templates.TemplateResponse("index.html", {
            "request": request,
            "temperaturas": temp_copy,
            "habitaciones": habitaciones,
            "mensaje": mensaje,
        })

    except Exception as e:
        print(f"Error en set_accion: {e}")
        return PlainTextResponse("Internal Server Error",
status_code=status.HTTP_500_INTERNAL_SERVER_ERROR)
```

```
# Endpoint para consultar temperaturas en una habitación concreta
@app.get("/temperatura/{habitacion}")
async def get_temperatura(habitacion: str):
    habitacion = habitacion.lower()
    if habitacion not in habitaciones:
        raise HTTPException(status_code=404, detail="Habitación no encontrada")
    with temperaturas_lock:
        temp = temperaturas.get(habitacion, "—")
    return {"habitacion": habitacion, "temperatura": temp}
```

Dockerfile:

```
# Dockerfile para consola

# Imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*

# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip

# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY consola.py .

# Exponer el puerto en el que FastAPI escuchará
```

```
EXPOSE 8000

# Ejecutar el servidor Uvicorn para FastAPI
CMD ["uvicorn", "consola:app", "--host", "0.0.0.0", "--port", "8000"]
```

Requirements.txt:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
python-multipart
fastapi
uvicorn[standard]
flask
requests
```

Proceso:

```
import time
import threading
import zenoh
import os

# Lista de habitaciones
habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]

# Estado de cada habitación: temperatura actual y modo
estado_habitaciones = {
    hab: {"temperatura": 21.0, "modo": "ninguno"} for hab in habitaciones
}

# Crear un lock para acceso seguro en hilos
estado_lock = threading.Lock()

def consigna_listener(room):
    def callback(sample):
        try:
            payload = bytes(sample.payload).decode("utf-8").strip().lower()
            print(f"[{room}] Consigna recibida: {payload}")

            with estado_lock:
                if payload in ["calentar", "enfriar"]:
                    estado_habitaciones[room]["modo"] = payload
                else:
                    estado_habitaciones[room]["modo"] = "ninguno"
        except Exception as e:
            print(f"[{room}] Error al procesar consigna: {e}")
    return callback

def simulador_clima(session):
    while True:
        with estado_lock:
            for room, datos in estado_habitaciones.items():
                modo = datos["modo"]
```

```
                temp = datos["temperatura"]

                if modo == "calentar" and temp < 25:
                    datos["temperatura"] += 1
                elif modo == "enfriar" and temp > 18:
                    datos["temperatura"] -= 1

                # Publicar temperatura
                temp_key = f"myhome/{room}/temp"
                session.put(temp_key, str(round(datos["temperatura"], 2)))
                print(f"[{room}] Temp publicada: {datos['temperatura']}°C")
        time.sleep(5)

if __name__ == "__main__":

    config = zenoh.Config()
    with zenoh.open(config) as session:
        # Subscribirse a todas las consignas
        for room in habitaciones:
            key = f"myhome/{room}/consigna"
            session.declare_subscriber(key, consigna_listener(room))
            print(f" Subscrito a: {key}")

        # Lanzar simulador en hilo separado
        threading.Thread(target=simulador_clima, args=(session,),
daemon=True).start()

        # Mantener el programa corriendo
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(" Proceso detenido.")
```

Dockerfile:

```
# Dockerfile para proceso

# Usamos una imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*
```

```
# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip

# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY proceso.py .

# Exponer puerto si es necesario
EXPOSE 8080

CMD ["python", "proceso.py"]
```

Requirements.txt:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
paho-mqtt
requests
```

Sensor:

```
import time
import zenoh
import os

def main():

    room = os.getenv("ROOM", "unknown_room")

    input_topic = f"myhome/{room}/temp"
    output_topic = f"myhome/{room}/sensor"

    print(f" El sensor de '{room}' escuchando en: {input_topic}, y reenviando a:
{output_topic}")

    with zenoh.open(zenoh.Config()) as session:
```

```
        def callback(sample):
            try:
                temp = bytes(sample.payload).decode("utf-8")
                print(f"[{room}] Temperatura recibida: {temp}°C")
                session.put(output_topic, temp)

            except Exception as e:
                print(f"[{room}] Error en reenvío: {e}")

        session.declare_subscriber(input_topic, callback)

        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            print(f" Sensor '{room}' detenido.")

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

Dockerfile:

```
# Dockerfile para sensor

# Usamos una imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*

# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip
```

```
# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY sensor.py .

# Exponer puerto si es necesario
EXPOSE 8080

CMD ["python", "sensor.py"]
```

Requirements.txt:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
paho-mqtt
requests
```

Sniffer:

```
import zenoh
import time

LOG_FILE = "/app/logs/capturas_zenoh.txt"

def main():
    with zenoh.open(zenoh.Config()) as session:
        with open(LOG_FILE, "a", encoding="utf-8") as f:

            def callback(sample):

                try:
                    texto = bytes(sample.payload).decode("utf-8",
errors="replace")
                    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
                    linea = f"[{timestamp}] [{sample.key_expr}] → {texto}"
                    print(linea)
                    f.write(linea + "\n")
                    f.flush()
                except Exception as e:
                    print(f"Error al procesar muestra: {e}")

            session.declare_subscriber("myhome/**", callback)

            try:
                while True:
                    time.sleep(1)
            except KeyboardInterrupt:
                print("\nSniffer detenido por el usuario.")
```

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

Dockerfile:

```
# Dockerfile para sniffer

# Usamos una imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*

# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip

# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY sniffer.py .

# Exponer puerto si es necesario
EXPOSE 8080

CMD ["python", "sniffer.py"]
```

Requirements.txt:

```
git+https://github.com/eclipse-zenoh/zenoh-python.git
```

```
paho-mqtt
requests
```

Zenoh-myhome.json:

```
{
plugins: {
    rest: {                          // Activa el plugin REST (consultas por HTTP)
    http_port: 8000                 // Escucha en el puerto 8000
    },
    storage_manager: {               // Activa el plugin de almacenamiento
      storages: {
        myhome: {                    // Define un almacenamiento llamado "myhome"
          key_expr: "myhome/**",     // Guarda y permite consultar todo lo que
empieza con "myhome/"
          volume: {
          id: "memory"               // Usa memoria RAM para guardar los datos
          }
       }
     }
   }
 }
}
```

Docker-compose.yml:

```
version: '3.8'

services:
  # Servicio para la consola
  consola:
    build:
      context: ./consola  # Directorio actual donde está el Dockerfile de la
consola
      dockerfile: Dockerfile  # Ruta al Dockerfile de la consola
    volumes:
      - ./zenoh-config:/app/zenoh-config  # Montar la configuración de Zenoh
      - ./ZENOH/zenohd.exe:/app/zenohd   # Montar la instalación de Zenoh
    environment:
      - ROOM=consola  # Establece5 la variable de entorno para identificar este
servicio como "consola"
    networks:
      - zenoh-net  # Red que conecta todos los servicios
    ports:
      - "8000:8000"  # Exponer el puerto 8000 para poder acceder a la API desde
el host
    restart: always  # El contenedor se reinicia automáticamente si falla

  # Servicio para el sensor en el salón
  sensor_salon:
    build:
      context: ./sensor
      dockerfile: Dockerfile
    volumes:
```

```
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=salon
    networks:
      - zenoh-net
    restart: always

  # Servicio para el sensor en la cocina
  sensor_cocina:
    build:
      context: ./sensor
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=cocina
    networks:
      - zenoh-net
    restart: always

  #Servicio para el sensor del baño
  sensor_bano:
    build:
      context: ./sensor
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=bano
    networks:
      - zenoh-net
    restart: always

  # Servicio para el sensor en el dormitorio 1
  sensor_dormitorio1:
    build:
      context: ./sensor
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=dormitorio1
    networks:
      - zenoh-net
    restart: always

  # Servicio para el sensor en el dormitorio 2
  sensor_dormitorio2:
    build:
```

```
    context: ./sensor
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config
    - ./ZENOH/zenohd.exe:/app/zenohd
  environment:
    - ROOM=dormitorio2
  networks:
    - zenoh-net
  restart: always

# Servicio para el actuador en el salón
actuador_salon:
  build:
    context: ./actuador
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config
    - ./ZENOH/zenohd.exe:/app/zenohd
  environment:
    - ROOM=salon
  networks:
    - zenoh-net
  restart: always

# Servicio para el actuador en la cocina
actuador_cocina:
  build:
    context: ./actuador
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config
    - ./ZENOH/zenohd.exe:/app/zenohd
  environment:
    - ROOM=cocina
  networks:
    - zenoh-net
  restart: always

  # Servicio para el actuador en el baño
actuador_bano:
  build:
    context: ./actuador
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config
    - ./ZENOH/zenohd.exe:/app/zenohd
  environment:
    - ROOM=bano
  networks:
    - zenoh-net
  restart: always
```

```
  # Servicio para el actuador en el dormitorio 1
  actuador_dormitorio1:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=dormitorio1
    networks:
      - zenoh-net
    restart: always

  # Servicio para el actuador en el dormitorio 2
  actuador_dormitorio2:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    environment:
      - ROOM=dormitorio2
    networks:
      - zenoh-net
    restart: always

  # Servicio para el proceso)
  proceso:
    build:
      context: ./proceso
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config
      - ./ZENOH/zenohd.exe:/app/zenohd
    networks:
      - zenoh-net
    restart: always
  #Servicio para el sniffer
  sniffer:
    build:
      context: ./sniffer
      dockerfile: Dockerfile
    volumes:
      - ./sniffer/logs:/app/logs  # Carpeta local para guardar los logs de
captura
    networks:
      - zenoh-net
    stdin_open: true
    tty: true
    restart: unless-stopped
```

```
# Definición de la red a utilizar por los contenedores
networks:
  zenoh-net:
    driver: bridge
```

ANEXO II: CÓDIGO DE LA FASE II

En este Anexo se encuentran recogidos los códigos de la Fase 2. Respecto a los Dockerfile y Requirements.txt, solo se ha añadido el Dockerfile de la Consola, dado que es el único que cambia.

Gen_certs.sh (archivo de generación de certificados):

```
#!/bin/bash

export MSYS_NO_PATHCONV=1
set -e

# Crear directorio de certificados si no existe
mkdir -p certs
cd certs

# Limpiar certificados existentes
rm -f *.pem *.csr *.txt *.srl

echo " Generando certificados TLS para Zenoh"

# 1. Generar clave privada de la CA (Autoridad Certificadora)
echo " Generando clave privada de la CA"
openssl genrsa -out ca-key.pem 4096

# 2. Generar certificado de la CA
echo " Generando certificado de la CA"
openssl req -new -x509 -days 3650 -key ca-key.pem -sha256 -out ca-cert.pem \
  -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=MyHomeCA"

# 3. Generar clave privada del servidor (router Zenoh)
echo " Generando clave privada del servidor"
openssl genrsa -out server-key.pem 4096

# 4. Generar solicitud de certificado del servidor
echo " Generando solicitud de certificado del servidor"
openssl req -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=zenoh-router" \
  -sha256 -new -key server-key.pem -out server.csr

# 5. Crear archivo de extensiones para el servidor
cat > server-extensions.txt << 'EOF'
[v3_req]
subjectAltName = @alt_names
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
```

```
extendedKeyUsage = serverAuth

[alt_names]
DNS.1 = zenoh-router
DNS.2 = zenoh
DNS.3 = localhost
DNS.4 = router
IP.1 = 127.0.0.1
IP.2 = 0.0.0.0
EOF

# 6. Generar certificado del servidor firmado por la CA
echo " Generando certificado del servidor"
openssl x509 -req -days 3650 -in server.csr -CA ca-cert.pem -CAkey ca-key.pem \
  -out server-cert.pem -extensions v3_req -extfile server-extensions.txt -
CAcreateserial

# 7. Generar clave privada del cliente
echo " Generando clave privada del cliente"
openssl genrsa -out client-key.pem 4096

# 8. Generar solicitud de certificado del cliente
echo " Generando solicitud de certificado del cliente"
openssl req -subj "/C=ES/ST=Madrid/L=Madrid/O=MyHome/OU=IoT/CN=zenoh-client" \
  -new -key client-key.pem -out client.csr

# 9. Crear archivo de extensiones para el cliente
cat > client-extensions.txt << 'EOF'
[v3_req]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth
EOF

# 10. Generar certificado del cliente firmado por la CA
echo " Generando certificado del cliente"
openssl x509 -req -days 3650 -in client.csr -CA ca-cert.pem -CAkey ca-key.pem \
  -out client-cert.pem -extensions v3_req -extfile client-extensions.txt -
CAcreateserial

# 11. Crear archivo de credenciales
echo " Creando archivo de credenciales"
cat > zenoh_credentials.txt << 'EOF'
router:routerpass
sensor:sensorpass
actuador:actpass
sniffer:sniffpass
proceso:procpass
consola:conspass
EOF

# 12. Limpiar archivos temporales
rm -f server.csr client.csr server-extensions.txt client-extensions.txt
```

```
# 13. Establecer permisos apropiados
chmod 600 *-key.pem
chmod 644 *-cert.pem ca-cert.pem zenoh_credentials.txt

# 14. Verificar certificados
echo ""
echo "Verificando certificados"
echo "Verificando certificado del servidor:"
if openssl verify -CAfile ca-cert.pem server-cert.pem; then
    echo " Certificado del servidor: VÁLIDO"
else
    echo " Certificado del servidor: INVÁLIDO"
    exit 1
fi

echo "Verificando certificado del cliente:"
if openssl verify -CAfile ca-cert.pem client-cert.pem; then
    echo " Certificado del cliente: VÁLIDO"
else
    echo " Certificado del cliente: INVÁLIDO"
    exit 1
fi

# 15. Mostrar información detallada de los certificados
echo ""
echo " Información del certificado del servidor:"
openssl x509 -in server-cert.pem -text -noout | grep -A 5 "Subject Alternative
Name" || echo "Sin SAN encontrado"

echo ""
echo " Información del certificado del cliente:"
openssl x509 -in client-cert.pem -text -noout | grep -A 3 "Subject:"

echo ""
echo " Certificados generados exitosamente:"
echo "    CA Certificate: ca-cert.pem"
echo "    Server Certificate: server-cert.pem"
echo "    Server Key: server-key.pem"
echo "    Client Certificate: client-cert.pem"
echo "    Client Key: client-key.pem"
echo "    Credentials: zenoh_credentials.txt"
echo ""
echo " Certificados listos para usar con TLS"

# 16. Información adicional de depuración
echo ""
echo "Información de depuración:"
echo "Tamaños de archivos:"
ls -la *.pem zenoh_credentials.txt

echo ""
echo "Fechas de validez del certificado del servidor:"
```

```
openssl x509 -in server-cert.pem -noout -dates
```

Test_certs.sh (archive de comprobación de certificados):

```
#!/bin/bash

echo " Probando configuración TLS de Zenoh..."

CERT_DIR="./certs"

# Verificar que existen los certificados
echo "1. Verificando existencia de certificados..."
for cert in ca-cert.pem server-cert.pem server-key.pem client-cert.pem client-
key.pem; do
    if [ -f "$CERT_DIR/$cert" ]; then
        size=$(stat -c%s "$CERT_DIR/$cert" 2>/dev/null || stat -f%z
"$CERT_DIR/$cert" 2>/dev/null)
        echo " $cert: $size bytes"
    else
        echo " $cert: NO EXISTE"
        exit 1
    fi
done

# Verificar validez de certificados
echo ""
echo "2. Verificando validez de certificados..."
cd "$CERT_DIR"

if openssl verify -CAfile ca-cert.pem server-cert.pem >/dev/null 2>&1; then
    echo " Certificado del servidor: VÁLIDO"
else
    echo " Certificado del servidor: INVÁLIDO"
    exit 1
fi

if openssl verify -CAfile ca-cert.pem client-cert.pem >/dev/null 2>&1; then
    echo " Certificado del cliente: VÁLIDO"
else
    echo " Certificado del cliente: INVÁLIDO"
    exit 1
fi

# Verificar nombres alternativos del servidor
echo ""
echo "3. Verificando Subject Alternative Names del servidor..."
san_info=$(openssl x509 -in server-cert.pem -text -noout | grep -A 5 "Subject
Alternative Name")
if echo "$san_info" | grep -q "zenoh-router\|zenoh\|localhost"; then
    echo " SAN configurados correctamente:"
```

```
    echo "$san_info"
else
    echo " SAN no configurados correctamente"
    echo "$san_info"
fi

# Probar conexión TLS manual (si openssl s_client está disponible)
echo ""
echo "4. Probando handshake TLS..."
if command -v timeout >/dev/null 2>&1; then
    # Iniciar el router en background para la prueba
    docker-compose up -d zenoh-router 2>/dev/null || echo "  No se pudo iniciar
el router para la prueba"

    sleep 5  # Esperar a que el router se inicie

    # Probar conexión TLS
    if timeout 10 openssl s_client -connect localhost:7448 -CAfile ca-cert.pem -
cert client-cert.pem -key client-key.pem -verify_return_error -quiet </dev/null
>/dev/null 2>&1; then
        echo " Handshake TLS: EXITOSO"
    else
        echo " Handshake TLS: FALLIDO"
        echo "   Verificar que el router esté corriendo y configurado
correctamente"
    fi

    docker-compose stop zenoh-router 2>/dev/null || true
else
    echo "  timeout no disponible"
fi

# Verificar formato de certificados
echo ""
echo "5. Verificando formato de certificados..."
for cert in ca-cert.pem server-cert.pem client-cert.pem; do
    if head -1 "$cert" | grep -q "BEGIN CERTIFICATE"; then
        echo " $cert: Formato PEM correcto"
    else
        echo " $cert: Formato PEM incorrecto"
    fi
done

for key in server-key.pem client-key.pem; do
    if head -1 "$key" | grep -q "BEGIN.*PRIVATE KEY"; then
        echo " $key: Formato PEM correcto"
    else
        echo " $key: Formato PEM incorrecto"
    fi
done

# Mostrar fechas de validez
echo ""
```

```
echo "6. Fechas de validez de certificados..."
echo "Servidor:"
openssl x509 -in server-cert.pem -noout -dates | sed 's/^/  /'
echo "Cliente:"
openssl x509 -in client-cert.pem -noout -dates | sed 's/^/  /'

echo ""
echo " Verificación de certificados completada"

cd ..
```

Actuador.py:

```
import zenoh
import os
import time
import threading
import logging
import json

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def create_actuador_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"actuador"')
    config.insert_json5("transport/auth/usrpwd/password", '"actpass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"

    if tls_enabled:
        logger.info("TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        # Usar las claves correctas que coinciden con tu archivo JSON
        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")

        # Deshabilitar verificación del nombre del servidor si es necesario
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
```

```
        logger.info("TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)

def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error("Faltan certificados TLS:")
        for m in missing:
            logger.error(f" - {m}")
        return False

    # Verificar que los archivos no estén vacíos
    for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f"El archivo {cert_file} está vacío")
                    return False
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f"El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
            logger.error(f"Error leyendo {cert_file}: {e}")
            return False

    logger.info("Todos los certificados TLS verificados correctamente")
    return True

def main():
    # Verificar certificados al inicio
    if not verify_certificates():
        logger.error("No se pueden encontrar los certificados necesarios para
TLS")
        logger.error("   Ejecuta generate_certificates.sh para generarlos")
        exit(1)

    room = os.getenv("ROOM", "unknown_room")

    accion_topic = f"myhome/{room}/accion"
```

```
    consigna_topic = f"myhome/{room}/consigna"

    tls_status = "TLS" if os.getenv("ZENOH_TLS_ENABLED", "false").lower() ==
"true" else "TCP"
    logger.info(f"Actuador '{room}' ({tls_status}) escuchando acciones en:
{accion_topic} y reenviando a: {consigna_topic}")

    modo_actual = None
    modo_lock = threading.Lock()

    try:
        # Usamos la nueva función de sesión con soporte TLS
        with create_actuador_session() as session:
            logger.info("Sesión Zenoh creada exitosamente")

            def accion_callback(sample):
                nonlocal modo_actual
                try:
                    modo = bytes(sample.payload).decode("utf-8").strip().lower()
                    logger.info(f"[{room}] Acción recibida: {modo}")
                    with modo_lock:
                        modo_actual = modo
                except Exception as e:
                    logger.error(f"[{room}] Error en callback acción: {e}")

            session.declare_subscriber(accion_topic, accion_callback)
            logger.info(f"Suscriptor creado para: {accion_topic}")

            def reenviar_loop():
                while True:
                    with modo_lock:
                        modo = modo_actual
                    if modo:
                        try:
                            session.put(consigna_topic, modo.encode("utf-8"))
                            logger.info(f"[{room}] Reenviando comando: {modo}")
                        except Exception as e:
                            logger.error(f"[{room}] Error reenviando comando:
{e}")
                    time.sleep(5)  # Reenviar cada 5 segundos para asegurar que
se mantenga activo

            hilo_reenvio = threading.Thread(target=reenviar_loop, daemon=True)
            hilo_reenvio.start()

            try:
                logger.info("Actuador iniciado. Presiona Ctrl+C para detener.")
                while True:
                    time.sleep(1)
            except KeyboardInterrupt:
                logger.info(f"Actuador '{room}' detenido.")

    except Exception as e:
```

```
        logger.error(f"Error creando sesión Zenoh: {e}")
        logger.error("   Verifica que el router esté ejecutándose y los
certificados sean válidos")
        exit(1)

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

Consola.py:

```
from fastapi import FastAPI, Request, Form, HTTPException, status
from fastapi.responses import HTMLResponse, PlainTextResponse
from fastapi.templating import Jinja2Templates
from fastapi.middleware.cors import CORSMiddleware
import zenoh
import threading
import os
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def create_console_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"consola"')
    config.insert_json5("transport/auth/usrpwd/password", '"conspass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"

    if tls_enabled:
        logger.info("TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
        logger.info("TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)
```

```
def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error(" Faltan certificados TLS:")
        for m in missing:
            logger.error(f" - {m}")
        return False

    # Verificar que los archivos no estén vacíos
    for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f"El archivo {cert_file} está vacío")
                    return False
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f"El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
            logger.error(f"Error leyendo {cert_file}: {e}")
            return False

    logger.info("Todos los certificados TLS verificados correctamente")
    return True

# Crear app FastAPI
app = FastAPI(title="MyHome Console", version="2.0.0")

# Permitir peticiones desde cualquier origen (útil para desarrollo)
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_methods=["*"],
    allow_headers=["*"],
)

# Configurar Jinja2 para usar templates desde carpeta 'templates'
templates = Jinja2Templates(directory="templates")

habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]
```

```
temperaturas = {hab: "—" for hab in habitaciones}
temperaturas_lock = threading.Lock()

# Variables de estado de conexión
connection_status = {
    "connected": False,
    "tls_enabled": os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true",
    "last_error": None
}

# Crear sesión Zenoh con la nueva función que soporta TLS
try:
    zsession = create_console_session()
    connection_status["connected"] = True
    logger.info("Consola MyHome iniciada correctamente")
except Exception as e:
    logger.error(f" Error crítico al inicializar: {e}")
    connection_status["last_error"] = str(e)
    zsession = None

# Callback para actualizar temperaturas al recibir datos de Zenoh
def listener(room):
    def callback(sample):
        try:
            payload_bytes = bytes(sample.payload)
            payload_str = payload_bytes.decode("utf-8")

            with temperaturas_lock:
                temperaturas[room] = payload_str

            logger.info(f"[{room}] Temperatura actualizada: {payload_str}°C")
        except Exception as e:
            logger.error(f" Error al procesar muestra para {room}: {e}")
    return callback

# Suscribirse a las temperaturas de cada habitación en Zenoh
def suscribir_temperaturas():
    if not zsession:
        logger.error(" No hay sesión Zenoh disponible para suscripciones")
        return

    try:
        for hab in habitaciones:
            topic = f"myhome/{hab}/temp"
            zsession.declare_subscriber(topic, listener(hab))
            logger.info(f" Suscrito a: {topic}")

        logger.info("Todas las suscripciones configuradas correctamente")
    except Exception as e:
        logger.error(f" Error en suscripciones: {e}")
        connection_status["last_error"] = str(e)

# Ejecutar la suscripción en hilo aparte para no bloquear FastAPI
```

```
if zsession:
    threading.Thread(target=suscribir_temperaturas, daemon=True).start()

# Endpoint para mostrar información del estado del sistema
@app.get("/status")
async def get_status():
    """Endpoint para verificar el estado de la conexión y configuración"""
    return {
        "status": "online" if connection_status["connected"] else "offline",
        "tls_enabled": connection_status["tls_enabled"],
        "last_error": connection_status["last_error"],
        "habitaciones": habitaciones,
        "zenoh_session": zsession is not None
    }

# Endpoint para mostrar la página principal con el estado y el formulario
@app.get("/", response_class=HTMLResponse)
async def index(request: Request):
    try:
        with temperaturas_lock:
            temp_copy = temperaturas.copy()

        # Añadir información de estado para mostrar en la interfaz
        context = {
            "request": request,
            "temperaturas": temp_copy,
            "habitaciones": habitaciones,
            "tls_enabled": connection_status["tls_enabled"],
            "connected": connection_status["connected"]
        }

        return templates.TemplateResponse("index.html", context)
    except Exception as e:
        logger.error(f" Error en index: {e}")
        return PlainTextResponse("Internal Server Error", status_code=500)

# Endpoint para recibir la accion enviada desde el formulario
@app.post("/accion", response_class=HTMLResponse)
async def set_accion(request: Request, habitacion: str = Form(...), accion: str =
Form(...)):
    try:
        if not zsession:
            mensaje = "No hay conexión con Zenoh"
        elif habitacion not in habitaciones:
            mensaje = f"Habitación no reconocida: {habitacion}"
        else:
            key = f"myhome/{habitacion}/accion"
            zsession.put(key, accion.encode("utf-8"))
            mensaje = f" Acción '{accion}' enviada a {habitacion}"
            logger.info(f" [consola] Enviada acción: {accion} → {key}")

        with temperaturas_lock:
            temp_copy = temperaturas.copy()
```

```
        context = {
            "request": request,
            "temperaturas": temp_copy,
            "habitaciones": habitaciones,
            "mensaje": mensaje,
            "tls_enabled": connection_status["tls_enabled"],
            "connected": connection_status["connected"]
        }

        return templates.TemplateResponse("index.html", context)

    except Exception as e:
        logger.error(f" Error en set_accion: {e}")
        return PlainTextResponse("Internal Server Error",
status_code=status.HTTP_500_INTERNAL_SERVER_ERROR)

# Endpoint para consultar temperaturas en una habitación concreta
@app.get("/temperatura/{habitacion}")
async def get_temperatura(habitacion: str):
    habitacion = habitacion.lower()
    if habitacion not in habitaciones:
        raise HTTPException(status_code=404, detail="Habitación no encontrada")

    with temperaturas_lock:
        temp = temperaturas.get(habitacion, "—")

    return {
        "habitacion": habitacion,
        "temperatura": temp,
        "tls_enabled": connection_status["tls_enabled"],
        "timestamp": threading.current_thread().name
    }

# Healthcheck endpoint
@app.get("/health")
async def health_check():
    """Endpoint de verificación de salud del servicio"""
    return {
        "status": "healthy" if zsession else "unhealthy",
        "tls": connection_status["tls_enabled"],
        "zenoh_connected": connection_status["connected"]
    }

if __name__ == "__main__":
    import uvicorn
    logger.info(" Iniciando servidor FastAPI...")
    uvicorn.run(app, host="0.0.0.0", port=8010)
```

Dockerfile:

```
# Dockerfile para consola

# Usamos una imagen base de Python
FROM python:3.10-slim

# Establecer el directorio de trabajo dentro del contenedor
WORKDIR /app

# Instalar dependencias del sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    curl \
    git \
    tcpdump \
    && rm -rf /var/lib/apt/lists/*

# Instalar Rust y Cargo (necesarios para algunas dependencias de Python como
zenoh-python)
RUN curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh -s -- -y

# Establecer el PATH de Cargo de forma global para que esté disponible en el
contenedor
ENV PATH="/root/.cargo/bin:$PATH"

# Copiar el archivo de requerimientos al contenedor
COPY requirements.txt .

# Actualizar pip
RUN pip install --upgrade pip

# Instalar las dependencias de Python
RUN pip install -r requirements.txt

# Copiar el archivo zenohd.exe desde la carpeta ZENOH dentro de cada servicio
COPY ZENOH/zenohd.exe /app/zenohd

# Copiar el archivo de código fuente
COPY consola.py .

# Copiar la carpeta de templates
COPY templates ./templates

# Exponer el puerto en el que FastAPI escuchará (por defecto 8001)
EXPOSE 8010

# Ejecutar el servidor Uvicorn para FastAPI (en lugar de consola.py directamente)
CMD ["uvicorn", "consola:app", "--host", "0.0.0.0", "--port", "8010"]
```

Proceso.py:

```
import time
import threading
import zenoh
import os
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def create_proceso_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"proceso"')
    config.insert_json5("transport/auth/usrpwd/password", '"procpass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"

    if tls_enabled:
        logger.info(" TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        # Usar las claves correctas que coinciden con tu archivo JSON
        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")

        # Deshabilitar verificación del nombre del servidor si es necesario
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
        logger.info(" TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)

def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error(" Faltan certificados TLS:")
```

```
        for m in missing:
            logger.error(f" - {m}")
        return False

    # Verificar que los archivos no estén vacíos
    for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f" El archivo {cert_file} está vacío")
                    return False
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f" El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
            logger.error(f" Error leyendo {cert_file}: {e}")
            return False

    logger.info(" Todos los certificados TLS verificados correctamente")
    return True

# Lista de habitaciones
habitaciones = ["salon", "cocina", "bano", "dormitorio1", "dormitorio2"]

# Estado de cada habitación: temperatura actual y modo
estado_habitaciones = {
    hab: {"temperatura": 21.0, "modo": "ninguno"} for hab in habitaciones
}

# Crear un lock para acceso seguro en hilos
estado_lock = threading.Lock()

def consigna_listener(room):
    def callback(sample):
        try:
            payload = bytes(sample.payload).decode("utf-8").strip().lower()
            logger.info(f"[{room}] Consigna recibida: {payload}")

            with estado_lock:
                if payload in ["calentar", "enfriar"]:
                    estado_habitaciones[room]["modo"] = payload
                else:
                    estado_habitaciones[room]["modo"] = "ninguno"
        except Exception as e:
            logger.error(f"[{room}] Error al procesar consigna: {e}")
    return callback

def simulador_clima(session):
    while True:
```

```
        with estado_lock:
            for room, datos in estado_habitaciones.items():
                modo = datos["modo"]
                temp = datos["temperatura"]

                if modo == "calentar" and temp < 25:
                    datos["temperatura"] += 1
                elif modo == "enfriar" and temp > 18:
                    datos["temperatura"] -= 1

                # Publicar temperatura
                temp_key = f"myhome/{room}/temp"
                session.put(temp_key, str(round(datos["temperatura"], 2)))
                logger.info(f"[{room}]  Temp publicada:
{datos['temperatura']}°C")
        time.sleep(5)

def main():
    if not verify_certificates():
        logger.error(" No se pueden encontrar los certificados necesarios para
TLS")
        logger.error("   Ejecuta generate_certificates.sh para generarlos")
        exit(1)

    tls_status = " TLS" if os.getenv("ZENOH_TLS_ENABLED", "false").lower() ==
"true" else " TCP"
    logger.info(f" Iniciando proceso central de simulación de temperatura
({tls_status})...")

    with create_proceso_session() as session:
        for room in habitaciones:
            key = f"myhome/{room}/consigna"
            session.declare_subscriber(key, consigna_listener(room))
            logger.info(f" Subscrito a: {key}")

        threading.Thread(target=simulador_clima, args=(session,),
daemon=True).start()

        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            logger.info(" Proceso detenido.")

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

Sensor.py:

```
import time
```

```
import zenoh
import os
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

def create_sensor_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"sensor"')
    config.insert_json5("transport/auth/usrpwd/password", '"sensorpass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"

    if tls_enabled:
        logger.info(" TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        # Usar las claves correctas que coinciden con tu archivo JSON
        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")

        # Deshabilitar verificación del nombre del servidor si es necesario
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
        logger.info("TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)

def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error(" Faltan certificados TLS:")
        for m in missing:
            logger.error(f" - {m}")
```

```
        return False

    # Verificar que los archivos no estén vacíos
    for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f" El archivo {cert_file} está vacío")
                    return False
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f" El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
            logger.error(f" Error leyendo {cert_file}: {e}")
            return False

    logger.info(" Todos los certificados TLS verificados correctamente")
    return True

# Función principal del sensor
def main():
    # Verificar certificados al inicio
    if not verify_certificates():
        logger.error(" No se pueden encontrar los certificados necesarios para
TLS")
        logger.error("   Ejecuta generate_certificates.sh para generarlos")
        exit(1)

    # Obtenemos el nombre de la habitación desde la variable de entorno ROOM (por
defecto: "unknown_room")
    room = os.getenv("ROOM", "unknown_room")

    # Tópico desde donde el sensor recibirá la temperatura enviada por el proceso
    input_topic = f"myhome/{room}/temp"

    # Tópico al que el sensor reenviará esa temperatura
    output_topic = f"myhome/{room}/sensor"

    tls_status = " TLS" if os.getenv("ZENOH_TLS_ENABLED", "false").lower() ==
"true" else " TCP"
    logger.info(f" Sensor '{room}' ({tls_status}) escuchando en: {input_topic},
reenviando a: {output_topic}")

    # Usamos la nueva función de sesión con soporte TLS
    with create_sensor_session() as session:

        # Esta función será llamada cada vez que se reciba un mensaje en
input_topic
        def callback(sample):
```

```
            try:
                # Decodificamos el payload recibido (bytes) a texto
                temp = bytes(sample.payload).decode("utf-8")

                # Mostramos por consola lo recibido y que será reenviado
                logger.info(f"[{room}]  Recibida: {temp}°C → reenviando...")

                # Publicamos el mismo valor en el nuevo tópico
                session.put(output_topic, temp)

            except Exception as e:
                logger.error(f"[{room}]  Error en reenvío: {e}")

        # Nos suscribimos al tópico original donde el proceso publica
temperaturas
        session.declare_subscriber(input_topic, callback)

        # Bucle infinito para mantener el proceso activo
        try:
            while True:
                time.sleep(1)
        except KeyboardInterrupt:
            logger.info(f" Sensor '{room}' detenido.")

# Punto de entrada
if __name__ == "__main__":
    main()
```

Sniffer.py:

```
import zenoh
import time
import os
import logging

# Configurar logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

LOG_FILE = "/app/logs/capturas_zenoh.txt"

def create_sniffer_session():
    config = zenoh.Config()
    config.insert_json5("mode", '"client"')

    # Configuración de autenticación
    config.insert_json5("transport/auth/usrpwd/user", '"sniffer"')
    config.insert_json5("transport/auth/usrpwd/password", '"sniffpass"')

    tls_enabled = os.getenv("ZENOH_TLS_ENABLED", "false").lower() == "true"
```

```
    if tls_enabled:
        logger.info(" TLS habilitado - configurando certificados")
        config.insert_json5("connect/endpoints", '["tls/zenoh-router:7448"]')

        # Usar las claves correctas que coinciden con tu archivo JSON
        config.insert_json5("transport/link/tls/root_ca_certificate",
'"/certs/ca-cert.pem"')
        config.insert_json5("transport/link/tls/connect_certificate",
'"/certs/client-cert.pem"')
        config.insert_json5("transport/link/tls/connect_private_key",
'"/certs/client-key.pem"')
        config.insert_json5("transport/link/tls/enable_mtls", "true")

        # Deshabilitar verificación del nombre del servidor si es necesario
        config.insert_json5("transport/link/tls/verify_name_on_connect", "false")
    else:
        logger.info(" TLS deshabilitado - usando TCP")
        config.insert_json5("connect/endpoints", '["tcp/zenoh-router:7447"]')

    return zenoh.open(config)

def verify_certificates():
    if os.getenv("ZENOH_TLS_ENABLED", "false").lower() != "true":
        return True

    certs = [
        "/certs/ca-cert.pem",
        "/certs/client-cert.pem",
        "/certs/client-key.pem"
    ]
    missing = [c for c in certs if not os.path.exists(c)]
    if missing:
        logger.error(" Faltan certificados TLS:")
        for m in missing:
            logger.error(f" - {m}")
        return False

    # Verificar que los archivos no estén vacíos
    for cert_file in certs:
        try:
            with open(cert_file, 'r') as f:
                content = f.read().strip()
                if not content:
                    logger.error(f" El archivo {cert_file} está vacío")
                    return False
                if cert_file.endswith('.pem'):
                    if not (content.startswith('-----BEGIN') and
content.endswith('-----')):
                        logger.error(f" El archivo {cert_file} no tiene formato
PEM válido")
                        return False
        except Exception as e:
```

```
            logger.error(f" Error leyendo {cert_file}: {e}")
            return False

    logger.info(" Todos los certificados TLS verificados correctamente")
    return True

def main():
    if not verify_certificates():
        logger.error(" No se pueden encontrar los certificados necesarios para
TLS")
        logger.error("   Ejecuta generate_certificates.sh para generarlos")
        exit(1)

    tls_status = " TLS" if os.getenv("ZENOH_TLS_ENABLED", "false").lower() ==
"true" else "🔓 TCP"
    logger.info(f" Sniffer iniciado ({tls_status}). Capturando tráfico de
myhome/**")

    with create_sniffer_session() as session:
        with open(LOG_FILE, "a", encoding="utf-8") as f:
            def callback(sample):
                try:
                    texto = bytes(sample.payload).decode("utf-8",
errors="replace")
                    timestamp = time.strftime("%Y-%m-%d %H:%M:%S")
                    linea = f"[{timestamp}] [{sample.key_expr}] → {texto}"
                    print(linea)
                    f.write(linea + "\n")
                    f.flush()
                except Exception as e:
                    logger.error(f" Error al procesar muestra: {e}")

            session.declare_subscriber("myhome/**", callback)

            try:
                while True:
                    time.sleep(1)
            except KeyboardInterrupt:
                logger.info(" Sniffer detenido por el usuario.")

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

Zenoh-client.json:

```
{
  "mode": "client",
  "connect": {
    "endpoints": ["tls/zenoh:7448"]
```

```
  },
  "transport": {
    "link": {
      "tls": {
        "root_ca_certificate": "/certs/ca-cert.pem",
        "connect_certificate": "/certs/client-cert.pem",
        "connect_private_key": "/certs/client-key.pem",
        "enable_mtls": true,
        "verify_name_on_connect": false
      }
    }
  }
}
```

Zenoh-myhome.json:

```
{
  "mode": "router",
  "plugins": {
    "rest": {
      "http_port": 8010
    },
    "storage_manager": {
      "storages": {
        "myhome": {
          "key_expr": "myhome/**",
          "volume": {
            "id": "memory"
          }
        }
      }
    }
  },
  "listen": {
    "endpoints": [
      "tcp/0.0.0.0:7447",
      "tls/0.0.0.0:7448"
    ]
  },
  "transport": {
    "unicast": {
      "accept_timeout": 10000,
      "accept_pending": 100,
      "max_sessions": 1000,
      "max_links": 1
    },
    "link": {
      "tls": {
        "root_ca_certificate": "/certs/ca-cert.pem",
        "listen_certificate": "/certs/server-cert.pem",
```

```
        "listen_private_key": "/certs/server-key.pem",
        "enable_mtls": true,
        "verify_name_on_connect": true
      }
    },
    "auth": {
      "usrpwd": {
        "user": "router",
        "password": "routerpass",
        "dictionary_file": "/certs/zenoh_credentials.txt"
      }
    }
  }
}
```

Docker-compose.yml:

```
version: '3.8'

services:
  # Servicio para Zenoh Router (debe arrancar primero)
  zenoh:
    image: eclipse/zenoh:latest
    container_name: zenoh-router
    restart: unless-stopped
    ports:
      - "7447:7447"  # Puerto TCP para el protocolo Zenoh
      - "7448:7448"  # Puerto TLS para el protocolo Zenoh
      - "8011:8010"  # Puerto para la API REST
    volumes:
      - ./seguridad/certs:/certs:ro  # Solo lectura para los certificados
      - ./zenoh-config:/root/.zenoh:ro
    environment:
      - RUST_LOG=debug
      - ZENOH_LOG=debug
    command: ["-c", "/root/.zenoh/zenoh-myhome.json"]
    networks:
      zenoh-net:
        aliases:
          - zenoh-router
          - zenoh
          - router
    ulimits:
      nofile:
        soft: 65536
        hard: 65536
    # Configuración adicional del kernel
    sysctls:
      - net.core.somaxconn=1024
      - net.ipv4.tcp_max_syn_backlog=1024
```

```
# Servicio para la consola
consola:
  build:
    context: ./consola
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config:ro
    - ./ZENOH/zenohd.exe:/app/zenohd:ro
    - ./seguridad/certs:/certs:ro
    - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
  environment:
    - ROOM=consola
    - ZENOH_TLS_ENABLED=true
    - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
  networks:
    - zenoh-net
  ports:
    - "8010:8010"
  restart: always
  depends_on:
    - zenoh
  ulimits:
    nofile:
      soft: 4096
      hard: 4096

# Servicio para el sensor en el salón
sensor_salon:
  build:
    context: ./sensor
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config:ro
    - ./ZENOH/zenohd.exe:/app/zenohd:ro
    - ./seguridad/certs:/certs:ro
    - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
  environment:
    - ROOM=salon
    - ZENOH_TLS_ENABLED=true
    - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
  networks:
    - zenoh-net
  restart: always
  depends_on:
    - zenoh
  ulimits:
    nofile:
      soft: 4096
      hard: 4096

# Servicio para el sensor en la cocina
sensor_cocina:
```

```
  build:
    context: ./sensor
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config:ro
    - ./ZENOH/zenohd.exe:/app/zenohd:ro
    - ./seguridad/certs:/certs:ro
    - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
  environment:
    - ROOM=cocina
    - ZENOH_TLS_ENABLED=true
    - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
  networks:
    - zenoh-net
  restart: always
  depends_on:
    - zenoh
  ulimits:
    nofile:
      soft: 4096
      hard: 4096

# Servicio para el sensor del baño
sensor_bano:
  build:
    context: ./sensor
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config:ro
    - ./ZENOH/zenohd.exe:/app/zenohd:ro
    - ./seguridad/certs:/certs:ro
    - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
  environment:
    - ROOM=bano
    - ZENOH_TLS_ENABLED=true
    - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
  networks:
    - zenoh-net
  restart: always
  depends_on:
    - zenoh
  ulimits:
    nofile:
      soft: 4096
      hard: 4096

# Servicio para el sensor en el dormitorio 1
sensor_dormitorio1:
  build:
    context: ./sensor
    dockerfile: Dockerfile
  volumes:
    - ./zenoh-config:/app/zenoh-config:ro
```

```
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=dormitorio1
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el sensor en el dormitorio 2
  sensor_dormitorio2:
    build:
      context: ./sensor
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=dormitorio2
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el actuador en el salón
  actuador_salon:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=salon
```

```
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el actuador en la cocina
  actuador_cocina:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=cocina
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el actuador en el baño
  actuador_bano:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=bano
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
```

```
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el actuador en el dormitorio 1
  actuador_dormitorio1:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=dormitorio1
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el actuador en el dormitorio 2
  actuador_dormitorio2:
    build:
      context: ./actuador
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ROOM=dormitorio2
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
```

```
        hard: 4096

  # Servicio que simula el "proceso" ambiental
  proceso:
    build:
      context: ./proceso
      dockerfile: Dockerfile
    volumes:
      - ./zenoh-config:/app/zenoh-config:ro
      - ./ZENOH/zenohd.exe:/app/zenohd:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    restart: always
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

  # Servicio para el sniffer
  sniffer:
    build:
      context: ./sniffer
      dockerfile: Dockerfile
    volumes:
      - ./sniffer/logs:/app/logs
      - ./zenoh-config:/app/zenoh-config:ro
      - ./seguridad/certs:/certs:ro
      - ./seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro
    environment:
      - ZENOH_TLS_ENABLED=true
      - ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json
    networks:
      - zenoh-net
    stdin_open: true
    tty: true
    restart: unless-stopped
    depends_on:
      - zenoh
    ulimits:
      nofile:
        soft: 4096
        hard: 4096

# Definición de la red
networks:
  zenoh-net:
```

```
    driver: bridge
```

ANEXO III: CÓDIGO DE LA FASE III

En este Anexo se encuentran recogidos los códigos de la Fase 3. Únicamente se ha incluido el archivo de generación del docker-compose y del habitaciones.json.

Gen_docker.py para el modelo F1:

```
import json

N_HABITACIONES = N

docker_compose = {
    'version': '3.8',
    'services': {},
    'networks': {
        'zenoh-net': {
            'driver': 'bridge'
        }
    }
}

# Consola
docker_compose['services']['consola'] = {
    'build': {
        'context': './consola',
        'dockerfile': 'Dockerfile'
    },
    'volumes': [
        './zenoh-config:/app/zenoh-config',
        './ZENOH/zenohd.exe:/app/zenohd',
        './habitaciones.json:/app/habitaciones.json'
    ],
    'environment': ['ROOM=consola'],
    'networks': ['zenoh-net'],
    'ports': ['8000:8000'],
    'restart': 'always'
}

# Proceso ambiental
docker_compose['services']['proceso'] = {
    'build': {
        'context': './proceso',
        'dockerfile': 'Dockerfile'
    },
    'volumes': [
        './zenoh-config:/app/zenoh-config',
        './ZENOH/zenohd.exe:/app/zenohd',
```

```
            './habitaciones.json:/app/habitaciones.json'
        ],
        'networks': ['zenoh-net'],
        'restart': 'always'
    }

    # Sniffer
    docker_compose['services']['sniffer'] = {
        'build': {
            'context': './sniffer',
            'dockerfile': 'Dockerfile'
        },
        'volumes': [
            './sniffer/logs:/app/logs'
        ],
        'networks': ['zenoh-net'],
        'stdin_open': True,
        'tty': True,
        'restart': 'unless-stopped'
    }

    # Generar sensores y actuadores
    habitaciones = []
    for i in range(1, N_HABITACIONES + 1):
        nombre = f"habitacion{i}"
        habitaciones.append(nombre)

        # Sensor
        docker_compose['services'][f"sensor_{nombre}"] = {
            'build': {
                'context': './sensor',
                'dockerfile': 'Dockerfile'
            },
            'volumes': [
                './zenoh-config:/app/zenoh-config',
                './ZENOH/zenohd.exe:/app/zenohd'
            ],
            'environment': [f"ROOM={nombre}"],
            'networks': ['zenoh-net'],
            'restart': 'always'
        }

        # Actuador
        docker_compose['services'][f"actuador_{nombre}"] = {
            'build': {
                'context': './actuador',
                'dockerfile': 'Dockerfile'
            },
            'volumes': [
                './zenoh-config:/app/zenoh-config',
                './ZENOH/zenohd.exe:/app/zenohd'
            ],
            'environment': [f"ROOM={nombre}"],
```

```
        'networks': ['zenoh-net'],
        'restart': 'always'
    }

# Guardar docker-compose
with open('docker-compose.generated.yml', 'w') as f:
    import yaml
    yaml.dump(docker_compose, f, sort_keys=False)

# Guardar habitaciones.json
with open('habitaciones.json', 'w') as f:
    json.dump(habitaciones, f, indent=4)

print(f" docker-compose.generated.yml y habitaciones.json generados para
{N_HABITACIONES} habitaciones.")
```

Gen_docker.py para el modelo F2:

```
import json
import yaml
import copy

N_HABITACIONES = 10

# Base del docker-compose
docker_compose = {
    'version': '3.8',
    'services': {},
    'networks': {
        'zenoh-net': {
            'driver': 'bridge'
        }
    }
}

# Servicio zenoh-router
docker_compose['services']['zenoh'] = {
    'image': 'eclipse/zenoh:latest',
    'container_name': 'zenoh-router',
    'restart': 'unless-stopped',
    'ports': [
        '7447:7447',
        '7448:7448',
        '8013:8012'
    ],
    'volumes': [
        './seguridad/certs:/certs:ro',
        './zenoh-config:/root/.zenoh:ro'
    ],
    'environment': [
```

```
            'RUST_LOG=debug',
            'ZENOH_LOG=debug'
        ],
        'command': ['-c', '/root/.zenoh/zenoh-myhome.json'],
        'networks': {
            'zenoh-net': {
                'aliases': ['zenoh-router', 'zenoh', 'router']
            }
        },
        'ulimits': {
            'nofile': {'soft': 65536, 'hard': 65536}
        },
        'sysctls': [
            'net.core.somaxconn=1024',
            'net.ipv4.tcp_max_syn_backlog=1024'
        ]
    }

    # Servicio consola
    docker_compose['services']['consola'] = {
        'build': {
            'context': './consola',
            'dockerfile': 'Dockerfile'
        },
        'volumes': [
            './zenoh-config:/app/zenoh-config:ro',
            './ZENOH/zenohd.exe:/app/zenohd:ro',
            './seguridad/certs:/certs:ro',
            './seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro',
            './habitaciones.json:/app/habitaciones.json:ro'
        ],
        'environment': [
            'ROOM=consola',
            'ZENOH_TLS_ENABLED=true',
            'ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json'
        ],
        'networks': ['zenoh-net'],
        'ports': ['8012:8012'],
        'restart': 'always',
        'depends_on': ['zenoh'],
        'ulimits': {
            'nofile': {'soft': 4096, 'hard': 4096}
        }
    }

    # Habitaciones
    habitaciones = []

    for i in range(1, N_HABITACIONES + 1):
        nombre = f"habitacion{i}"
        habitaciones.append(nombre)

        base_service = {
```

```
        'build': {
            'context': '',
            'dockerfile': 'Dockerfile'
        },
        'volumes': [
            './zenoh-config:/app/zenoh-config:ro',
            './ZENOH/zenohd.exe:/app/zenohd:ro',
            './seguridad/certs:/certs:ro',
            './seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:r
o'
        ],
        'environment': [
            f'ROOM={nombre}',
            'ZENOH_TLS_ENABLED=true',
            'ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json'
        ],
        'networks': ['zenoh-net'],
        'restart': 'always',
        'depends_on': ['zenoh'],
        'ulimits': {
            'nofile': {'soft': 4096, 'hard': 4096}
        }
    }

    sensor = copy.deepcopy(base_service)
    sensor['build']['context'] = './sensor'
    docker_compose['services'][f'sensor_{nombre}'] = sensor

    actuador = copy.deepcopy(base_service)
    actuador['build']['context'] = './actuador'
    docker_compose['services'][f'actuador_{nombre}'] = actuador

# Servicio proceso ambiental
docker_compose['services']['proceso'] = {
    'build': {
        'context': './proceso',
        'dockerfile': 'Dockerfile'
    },
    'volumes': [
        './zenoh-config:/app/zenoh-config:ro',
        './ZENOH/zenohd.exe:/app/zenohd:ro',
        './seguridad/certs:/certs:ro',
        './seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro',
        './habitaciones.json:/app/habitaciones.json:ro'
    ],
    'environment': [
        'ZENOH_TLS_ENABLED=true',
        'ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json'
    ],
    'networks': ['zenoh-net'],
    'restart': 'always',
    'depends_on': ['zenoh'],
    'ulimits': {
```

```
        'nofile': {'soft': 4096, 'hard': 4096}
    }
}

# Servicio sniffer
docker_compose['services']['sniffer'] = {
    'build': {
        'context': './sniffer',
        'dockerfile': 'Dockerfile'
    },
    'volumes': [
        './sniffer/logs:/app/logs',
        './zenoh-config:/app/zenoh-config:ro',
        './seguridad/certs:/certs:ro',
        './seguridad/certs/zenoh_credentials.txt:/app/zenoh_credentials.txt:ro'
    ],
    'environment': [
        'ZENOH_TLS_ENABLED=true',
        'ZENOH_CONFIG=/app/zenoh-config/zenoh-client.json'
    ],
    'networks': ['zenoh-net'],
    'stdin_open': True,
    'tty': True,
    'restart': 'unless-stopped',
    'depends_on': ['zenoh'],
    'ulimits': {
        'nofile': {'soft': 4096, 'hard': 4096}
    }
}

# Guardar docker-compose.yml
with open('docker-compose.yml', 'w') as f:
    yaml.dump(docker_compose, f, sort_keys=False)

# Guardar habitaciones.json
with open('habitaciones.json', 'w') as f:
    json.dump(habitaciones, f, indent=4)

print(f"docker-compose.yml y habitaciones.json generados para {N_HABITACIONES}
habitaciones.")
```