
This is the **published version** of the bachelor thesis:

Moreno Lorente, Marc. *Arquitectura de Microserveis Asíncrons per a la Ingesta de Dades en Temps Real*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Enginyeria Informàtica)

This version is available at <https://ddd.uab.cat/record/334464>

under the terms of the  license.

Arquitectura de Microserveis Asíncrons per a la Ingesta de Dades en Temps Real

Marc Moreno Lorente

27 de juny de 2026

Resum – En aquest treball s'ha dissenyat, implementat i validat una arquitectura de microserveis asíncrons per a la ingesta de dades en temps real. El sistema simula el monitoratge d'una flota de neveres industrials que generen esdeveniments concurrents amb informació de temperatura, estat operatiu, consum energètic i alarmes. L'arquitectura s'ha desplegat sobre un clúster de Kubernetes format per dispositius Raspberry Pi 5, utilitzant contenidors Docker, Helm per al desplegament, Redis com a broker de missatges, PostgreSQL com a sistema de persistència i Grafana per a la visualització dels resultats. La validació s'ha realitzat mitjançant proves funcionals i proves de càrrega bàsiques amb diferents volums de dispositius simulats. Els resultats mostren que el sistema és capaç d'ingerir, processar, persistir i visualitzar dades de manera asíncrona, tot i que també s'ha observat un increment del temps de processament quan el nombre de dispositius simulats és elevat.

Paraules clau – microserveis, processament asíncron, Kubernetes, Redis, PostgreSQL, Grafana, ingesta de dades, simulació de dispositius, sistemes distribuïts

Abstract – This project presents the design, implementation and validation of an asynchronous microservices architecture for real-time data ingestion. The system simulates the monitoring of a fleet of industrial refrigerators that generate concurrent events containing temperature, operational status, energy consumption and alarm information. The architecture has been deployed on a Kubernetes cluster composed of Raspberry Pi 5 devices, using Docker containers, Helm for deployment, Redis as a message broker, PostgreSQL as the persistence system and Grafana for data visualization. The validation has been carried out through functional tests and basic load tests with different volumes of simulated devices. The results show that the system is able to ingest, process, persist and visualize data asynchronously, although an increase in processing time was also observed when the number of simulated devices was high.

Keywords – microservices, asynchronous processing, Kubernetes, Redis, PostgreSQL, Grafana, data ingestion, device simulation, distributed systems



1 INTRODUCCIÓ - CONTEXT DEL TREBALL

EN els darrers anys, l'augment de dispositius connectats i sistemes distribuïts ha generat la necessitat de plataformes capaces d'ingerir i gestionar grans volums de dades en temps real. Aquest tipus de sistemes planteja reptes importants relacionats amb la concurrència, la comunicació entre serveis, la persistència eficient i la to-

lerància a fallades.

La idea del projecte neix de voler aplicar aquests conceptes en un cas pràctic que es pogués desenvolupar i provar dins de l'abast d'un Treball Final de Grau. Per això, el treball no s'ha orientat cap a l'anàlisi avançada de dades, ni cap a tecnologies de Big Data o Machine Learning, sinó cap a la construcció d'una arquitectura funcional que permeti moure dades entre diferents components i comprovar-ne el comportament.

El cas d'ús escollit és el monitoratge d'una flota de neveres industrials simulades. Cada nevera actua com un dispositiu productor d'esdeveniments i envia periòdicament informació com la temperatura interna, l'estat de la porta, el consum energètic, l'estat de l'alarma i l'estat operatiu del

- E-mail de contacte: Marc.MorenoL@autonoma.cat
- Menció: Enginyeria de Computadors
- Treball tutoritzat per: Joan Josep Piedrafita Farras
- Curs 2025/26

dispositiu. Aquest escenari permet representar de manera simplificada un sistema IoT distribuït, sense fer que el cas d'ús sigui massa gran per al projecte.

Amb aquest cas d'ús es volen posar a prova les parts principals del sistema: la recepció concurrent d'esdeveniments, l'ús d'un broker de missatges, el processament posterior, la persistència a base de dades i la visualització dels resultats. A més, el desplegament sobre un clúster físic de Raspberry Pi permet plantejar el projecte en un entorn més realista, on també apareixen aspectes d'infraestructura, xarxa i desplegament que no es veurien igual en una execució local.

La resta del document presenta primer els objectius i l'abast del projecte. Després es descriuen les tecnologies utilitzades i la metodologia seguida, s'exposa la infraestructura desplegada, l'arquitectura funcional i els components desenvolupats. Finalment, es presenten les proves realitzades, la discussió dels resultats, les conclusions i les possibles línies de treball futur.

2 OBJECTIUS I ABAST

2.1 Objectius

L'objectiu principal d'aquest Treball Final de Grau és dissenyar, implementar i validar una arquitectura de microserveis asíncrons per a la ingesta de dades en temps real. El sistema utilitza múltiples neveres industrials simulades, que generen esdeveniments de manera periòdica i concurrent.

El treball se centra sobretot en la infraestructura i en el flux de dades entre components: recepció dels esdeveniments, publicació al broker, processament posterior, persistència a base de dades i visualització dels resultats. Per tant, el focus no és només desenvolupar una API o un simulador, sinó integrar tots els elements en un sistema distribuït funcional.

A partir d'aquest objectiu general, el treball planteja els objectius específics següents:

- Dissenyar una arquitectura de microserveis desacoblada, on cada component tingui una responsabilitat concreta dins del flux de dades.
- Implementar un servei d'ingesta mitjançant una API asíncrona capaç de rebre esdeveniments generats per diferents dispositius simulats.
- Utilitzar un broker de missatges per desacoblar la recepció dels esdeveniments del seu processament posterior.
- Desenvolupar un servei de processament encarregat de consumir els esdeveniments del broker, tractar les dades rebudes i persistir-les a la base de dades.
- Implementar un sistema de persistència que permeti emmagatzemar tant els esdeveniments generats com l'estat i la configuració de les neveres simulades.
- Desenvolupar un simulador de neveres capaç de generar esdeveniments periòdics i concurrents amb diferents volums de càrrega.
- Desplegar els serveis del sistema en una infraestructura distribuïda basada en contenidors i Kubernetes, utilitzant eines de desplegament com Helm.

- Crear panells de Grafana que permetin consultar el comportament del sistema i validar el flux complet de dades.
- Validar el funcionament del sistema mitjançant proves funcionals i proves de càrrega bàsiques, observant el comportament amb diferents quantitats de neveres simulades.

2.2 Abast i limitacions

L'abast del projecte queda limitat a la construcció i validació d'una arquitectura funcional dins d'un entorn local controlat. El sistema no pretén ser una plataforma final preparada per a producció, sinó una implementació pràctica que permeti provar el comportament d'un pipeline de dades asíncron.

Queden fora de l'abast del treball les tècniques d'anàlisi avançada de dades, les tecnologies específiques de Big Data i els mecanismes de Machine Learning. La visualització a Grafana s'ha utilitzat principalment per consultar i validar les dades generades durant les proves. En un escenari més proper a una empresa, caldria ampliar aquesta part amb visualitzacions més específiques per a cada tipus de dada, alertes operatives reals i mecanismes de notificació, per exemple mitjançant Grafana Alerting o Alertmanager.

Pel que fa a la gestió d'errors, el projecte no incorpora mecanismes avançats de recuperació automàtica ni tolerància a fallades a nivell d'aplicació. La detecció de problemes s'ha basat principalment en els logs dels serveis, els esdeveniments de Kubernetes i la informació disponible als panells de Grafana.

Finalment, les proves de càrrega realitzades són bàsiques i tenen com a objectiu observar el comportament general del sistema amb diferents volums de neveres simulades. No es plantegen com una comparativa exhaustiva de rendiment, sinó com una manera d'identificar els límits principals de la configuració utilitzada.

3 ESTAT DE L'ART I TECNOLOGIES RELACIONADES

El treball s'emmarca dins l'àmbit dels sistemes distribuïts, els microserveis, la ingesta de dades en temps real i el processament asíncron d'events. En molts sistemes actuals, especialment en entorns industrials, IoT o de monitorització, les dades no provenen d'una única font, sinó de múltiples dispositius que generen informació de manera contínua. En aquests casos, no és suficient amb rebre les dades, sinó que també cal poder absorbir pics de càrrega, processar els events de manera ordenada, persistir-los i visualitzar-ne l'estat amb una latència raonable.

Una arquitectura monolítica o excessivament acoblada pot funcionar en escenaris petits, però pot presentar limitacions quan augmenta el nombre de dispositius o la freqüència d'enviament de dades. Si el mateix servei s'encarrega de rebre, validar, processar i guardar cada event, qualsevol retard en una d'aquestes fases pot afectar directament la resta del sistema. Per aquest motiu, en arquitectures orientades a events és habitual separar la ingesta del processament mitjançant cues o brokers de missatges.

Els microserveis permeten dividir el sistema en components amb responsabilitats concretes, com ara la recepció de dades, el processament, la persistència o la visualització. Aquesta separació facilita que cada component pugui evolucionar, desplegar-se o escalar-se de manera independent. A més, l'ús d'un broker de missatges permet desacoblar els productors dels consumidors, de manera que l'API d'entrada no ha d'esperar que tot el processament posterior finalitzi per poder acceptar nous events.

En aquest projecte, aquest plantejament s'aplica a un escenari de neveres industrials simulades. Tot i que els dispositius no són físics, el cas d'ús representa una situació habitual en sistemes de monitorització: múltiples fonts independents generen events periòdics, canvis d'estat i alarmes que han de ser processats i emmagatzemats per poder-ne consultar l'evolució. Aquesta aproximació permet validar una arquitectura propera a un sistema real, però dins d'un entorn controlat i assumible per a l'abast del treball.

Per implementar aquesta arquitectura, la ingesta s'ha implementat amb FastAPI, una eina adequada per desenvolupar APIs en Python de manera ràpida i amb suport per a validació de dades [1]. Per desacoblar els components s'ha utilitzat Redis Streams, que permet publicar i consumir missatges de manera ordenada mitjançant streams i grups de consumidors [2]. Les dades processades es desen finalment a PostgreSQL, que actua com a base de dades principal del sistema [3].

Per al desplegament s'han utilitzat contenidors Docker i Kubernetes. Els contenidors permeten empaquetar cada servei amb les seves dependències, mentre que Kubernetes facilita la gestió dels serveis desplegats, la xarxa interna i la configuració dels diferents components [4]. En aquest treball també s'ha utilitzat Helm per mantenir una estructura més clara en el desplegament dels serveis dins del clúster [6]. En aquest projecte, l'ús de Kubernetes no respon a una necessitat estricta de producció, sinó a la voluntat de validar l'arquitectura en un entorn distribuït real, format per diversos dispositius Raspberry Pi.

La part de xarxa i accés s'ha resolt amb Calico, Traefik, WireGuard i Cloudflare Tunnel. Calico s'ha utilitzat com a CNI del clúster, Traefik com a controlador d'entrada, WireGuard per crear una xarxa privada d'accés i Cloudflare Tunnel per publicar de manera controlada alguns serveis concrets [8] [9] [12] [13].

Finalment, per observar el comportament del sistema s'han utilitzat Prometheus i Grafana. Prometheus permet recollir mètriques del clúster i Grafana s'ha utilitzat per crear panells de visualització connectats a les dades generades pel sistema [10] [11].

3.1 Comparació amb alternatives tecnològiques

Durant el plantejament del sistema també es podrien haver utilitzat altres tecnologies per als components principals. En el cas del broker de missatges, Kafka hauria estat una alternativa més orientada a escenaris amb volums elevats de dades, particionat i persistència del log [14]. RabbitMQ, en canvi, hauria encaixat en un model de missatgeria més clàssic basat en cues i encaminament de missatges [15]. En aquest projecte s'ha escollit Redis Streams perquè ofereix una solució més lleugera, senzilla de desplegar i suficient

per al volum d'events generat pel simulador.

Pel que fa a la persistència, TimescaleDB podria haver estat una opció interessant per tractar dades de sèrie temporal, especialment en un sistema amb més volum històric o consultes temporals més avançades [16]. Tot i això, PostgreSQL s'ha considerat suficient per a l'abast del treball, ja que permet persistir tant els events com l'estat i la configuració de les neveres sense afegir més complexitat. A més, l'experiència prèvia amb aquesta tecnologia ha permès reduir el risc d'implementació i centrar l'esforç en l'arquitectura distribuïda, la ingesta asíncrona i el desplegament del sistema.

Finalment, el desplegament també s'hauria pogut fer amb Docker Compose, especialment per validar ràpidament el sistema en un entorn local [17]. Tot i això, s'ha optat per Kubernetes perquè és l'estàndard en molts entorns cloud-native i empresarials per al desplegament d'aplicacions distribuïdes. En aquest sentit, Kubernetes permet aproximar el projecte a un escenari més realista, amb serveis independents, xarxa interna, ingressos, desplegaments declaratius, observabilitat i execució sobre un clúster físic. Aquesta decisió diferencia el projecte d'un desplegament purament local o domèstic i l'apropa més a la manera com una empresa podria desplegar i operar un sistema d'aquest tipus.

TAULA 1: COMPARACIÓ AMB ALTERNATIVES TECNOLÒGIQUES

Component	Alternatives	Decisió presa
Broker de missatges	Kafka, RabbitMQ, Redis Streams	Redis Streams, per ser més lleuger i suficient per al volum d'events del projecte.
Persistència	PostgreSQL, TimescaleDB	PostgreSQL, perquè és suficient per a l'abast del projecte, permet gestionar events, estat i configuració, i redueix el risc gràcies a l'experiència prèvia amb aquesta tecnologia.
Desplegament	Docker Compose, Kubernetes	Kubernetes, perquè és l'estàndard de facto en entorns cloud-native i permet validar el sistema en un escenari distribuït més proper a un entorn real.

TAULA 2: TECNOLOGIES PRINCIPALS UTILITZADES AL PROJECTE

Tecnologia	Funció dins del projecte
FastAPI	API d'ingesta dels esdeveniments generats per les neveres.
Redis	Broker de missatges entre l'API i el servei de processament.
PostgreSQL	Persistència dels esdeveniments, estat i configuració de les neveres.
Kubernetes i Helm	Desplegament i gestió dels serveis dins del clúster.
Calico i Traefik	Xarxa interna del clúster i gestió de l'accés als serveis desplegats.
Grafana i Prometheus	Visualització i monitorització del comportament del sistema.
WireGuard i Cloudflare Tunnel	Accés privat a la infraestructura i publicació controlada de serveis.

4 DISSENY I METODOLOGIA DEL SISTEMA

4.1 Metodologia de desenvolupament

La metodologia seguida durant el desenvolupament del projecte ha estat incremental i orientada a la integració progressiva dels components. En lloc de desenvolupar tots els ser-

veis de manera aïllada i integrar-los al final, s'ha anat construint el sistema per fases, validant cada part abans d'afegir la següent.

Les fases principals han estat les següents:

- **Desplegament de la infraestructura base:** configuració del clúster de Kubernetes, la xarxa privada, l'accés remot i les eines bàsiques d'observabilitat.
- **Instal·lació dels serveis de suport:** desplegament de Redis i PostgreSQL, necessaris per al funcionament del pipeline de dades.
- **Desenvolupament dels microserveis propis:** implementació de l'API d'ingesta i del servei de processament en Python, empaquetats en contenidors Docker i desplegats mitjançant Helm.
- **Integració del simulador i validació:** desenvolupament del simulador de neveres, creació dels panells de Grafana i execució de les proves funcionals i de càrrega.

Aquesta manera de treballar ha permès avançar de forma progressiva i detectar problemes en cada fase, especialment en la part d'infraestructura i en la integració entre serveis.

4.2 Infraestructura desplegada

La infraestructura del projecte s'ha desplegat sobre un clúster físic format per tres dispositius Raspberry Pi 5 de 4GB de memòria RAM, executant Ubuntu Server 22.04 LTS. El clúster s'ha configurat mitjançant kubernetes, amb un node actuant com a control plane i dos nodes worker. Aquesta decisió ha permès validar el sistema en un entorn distribuït real i no únicament en una execució local sobre una sola màquina.

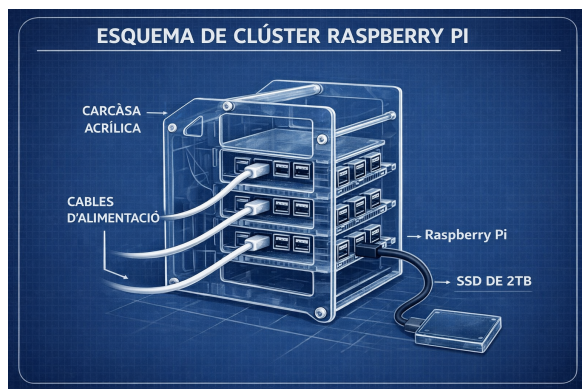


Fig. 1: Representació visual de la infraestructura física del clúster desplegat.

Per a la xarxa interna del clúster s'ha utilitzat Calico com a CNI, mentre que Traefik s'ha desplegat com a Ingress Controller per gestionar l'accés als serveis exposats. Aquesta combinació ha permès disposar d'una xarxa funcional entre pods i serveis, així com d'un mecanisme centralitzat per publicar les aplicacions desplegades dins del clúster.

L'accés remot a la infraestructura s'ha resolt mitjançant una xarxa privada virtual basada en WireGuard. Per simplificar-ne la gestió s'ha utilitzat wg-easy, desplegat

amb Docker Compose directament al node control plane i fora del clúster de Kubernetes. Així, l'accés VPN queda disponible de manera independent als serveis desplegats dins del clúster, fet que resulta útil en cas de reinicis, canvis o problemes dins de Kubernetes.

S'ha utilitzat el domini propi *pizzapesto.eu* per facilitar l'accés als serveis principals de la infraestructura. El subdomini *vpn.pizzapesto.eu* s'ha utilitzat per accedir a la VPN mitjançant port forwarding, mentre que *manage-vpn.pizzapesto.eu* s'ha reservat per a la consola de gestió de wg-easy. Aquesta consola s'ha publicat mitjançant Cloudflare Tunnel i s'ha protegit amb Cloudflare Zero Trust, utilitzant autenticació basada en GitHub per limitar-ne l'accés.

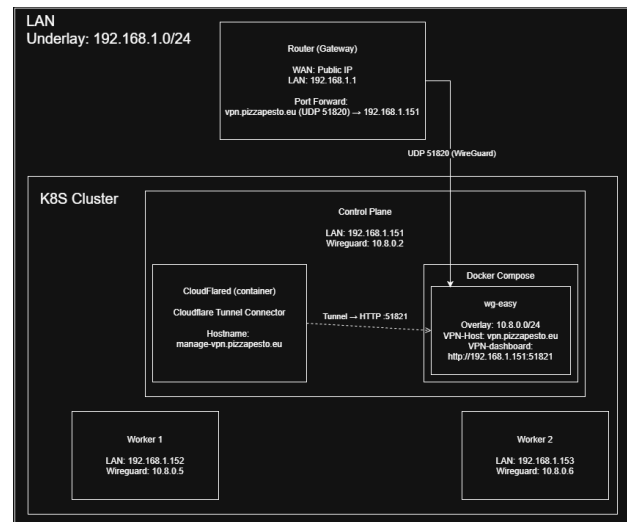


Fig. 2: Arquitectura de la infraestructura desplegada.

Amb aquesta separació, la major part de la infraestructura es manté dins de la xarxa privada, exposant cap a Internet únicament els components que ho requereixen. A més, s'ha configurat Traefik per generar certificats Let's Encrypt mitjançant Cloudflare, permetent l'accés HTTPS als serveis interns publicats dins de la xarxa privada.

Finalment, s'ha desplegat una base d'observabilitat formada per Prometheus, Grafana i Alertmanager. Prometheus s'encarrega de recollir mètriques del clúster i Grafana permet visualitzar tant l'estat de la infraestructura com les dades generades pel sistema de neveres. Alertmanager també s'ha desplegat com a part de la pila de monitorització, però en aquest projecte no s'ha utilitzat activament per generar alertes reals, quedant configurat com a component de suport per a una possible ampliació futura.

4.3 Arquitectura funcional

L'arquitectura funcional del sistema segueix un model de pipeline de dades asíncron. El flux principal comença al simulador de neveres, que genera esdeveniments periòdics per a cada dispositiu simulat. Aquests esdeveniments s'envien a l'API d'ingesta, desenvolupada amb FastAPI, que valida les dades rebudes i les publica a Redis mitjançant streams.

Redis actua com a punt intermedi entre la ingesta i el processament. Aquesta decisió permet que l'API no hagi de processar ni persistir directament cada esdeveniment rebut, sinó únicament validar-lo i publicar-lo al broker. Posterior-

ment, el servei de processament consumeix els missatges de Redis, prepara les dades i les persisteix a PostgreSQL. Finalment, Grafana consulta la base de dades per representar l'estat de les neveres, els esdeveniments generats i el temps de processament.

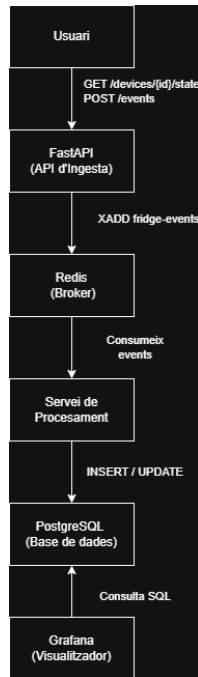


Fig. 3: Flux principal d'esdeveniments del sistema.

A més del flux principal d'esdeveniments, el sistema incorpora un segon flux orientat a la gestió bàsica de les neveres simulades. Mitjançant l'API es poden enviar comandes o modificar la configuració d'un dispositiu, com ara apagar una nevera, forçar el compressor, modificar el setpoint de temperatura o silenciar una alarma. Aquestes accions també es publiquen a Redis i són processades posteriorment pel servei de processament, que actualitza l'estat o la configuració corresponent a PostgreSQL.

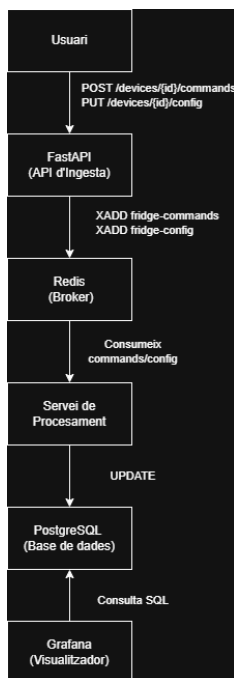


Fig. 4: Flux de comandes i configuració.

Les Figures 3 i 4 representen de manera esquemàtica aquests dos fluxos funcionals.

4.4 Components desenvolupats

Els components desenvolupats específicament per al projecte han estat l'API d'ingesta, el servei de processament i el simulador de neveres industrials. Tots tres s'han implementat en Python, s'han empaquetat en imatges Docker i s'han desplegat al clúster mitjançant charts de Helm propis.

- **API d'ingesta:** desenvolupada amb FastAPI, actua com a punt d'entrada del sistema. Exposa els endpoints necessaris per rebre esdeveniments, consultar l'estat d'una nevera, enviar comandes i modificar-ne la configuració. Les peticions rebudes es validen i es publiquen a Redis mitjançant streams diferenciats per a esdeveniments, comandes i canvis de configuració.
- **Servei de processament:** és l'encarregat de consumir els missatges publicats a Redis. Processa els esdeveniments generats per les neveres i els persisteix a PostgreSQL. També tracta les comandes i els canvis de configuració, actualitzant l'estat persistent de cada dispositiu a la base de dades.
- **Simulador de neveres industrials:** permet generar una càrrega controlada sobre el sistema. En iniciar-lo es defineix la URL de l'API, el nombre de neveres simulades i l'interval d'enviament dels esdeveniments. Cada nevera s'executa de manera concurrent, consulta periòdicament el seu estat actual i genera dades com la temperatura, l'estat de la porta, el consum energètic, l'estat de l'alarma i l'estat operatiu.

5 RESULTATS I VALIDACIÓ

5.1 Validació funcional

La validació funcional del sistema s'ha realitzat comprovant que cada part del flux de dades funcionava correctament, des de la generació dels esdeveniments fins a la seva persistència i visualització. Per fer-ho, s'han revisat els diferents components del sistema de manera progressiva:

1. S'ha verificat que el simulador era capaç de generar esdeveniments periòdics per a múltiples neveres simulades, revisant els logs del mateix simulador.
2. S'ha comprovat que l'API d'ingesta rebia correctament els esdeveniments i retornava una resposta adequada als diferents endpoints.
3. S'ha validat que els missatges arribaven a Redis consultant els streams mitjançant `redis-cli` des del pod master de Redis.
4. S'ha comprovat que el servei de processament consumia els missatges del broker i persistia les dades a PostgreSQL, revisant tant els logs del servei com les taules de la base de dades.
5. S'ha verificat que les comandes i els canvis de configuració afectaven el comportament posterior de les neveres simulades.

Amb aquestes comprovacions s'ha pogut validar el funcionament complet del sistema, incloent la ingesta, el processament asíncron, la persistència de dades i la resposta del simulador davant canvis d'estat o configuració.

5.2 Visualització a Grafana

Per facilitar la representació del comportament del sistema, s'han creat diferents panells a Grafana connectats a PostgreSQL. Aquests panells s'han separat en dues categories principals:

- **Panells globals:** mostren informació agregada del sistema, com els esdeveniments per dispositiu, els esdeveniments per minut, el total d'esdeveniments registrats i el temps mitjà de processament de les diferents peticions.
- **Panells específics per nevera:** mostren informació concreta d'un dispositiu, com la configuració actual, l'últim event registrat, l'històric de temperatures, la temperatura mitjana, l'última temperatura registrada i el registre d'esdeveniments generats on podem veure tots els events sense necessitat d'anar a la base de dades.

Aquesta separació ha permès consultar tant l'estat general del sistema com el comportament individual de cada nevera simulada. A la Figura 5 es mostra una vista dels panells globals, mentre que a la Figura 6 es mostra un exemple de panell específic per a una nevera concreta.

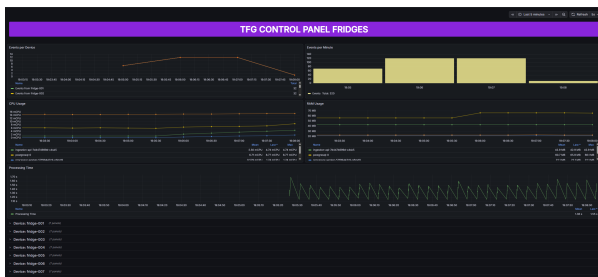


Fig. 5: Panells globals de Grafana.

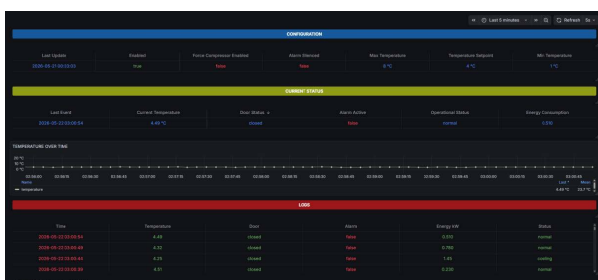


Fig. 6: Panell específic d'una nevera simulada.

5.3 Proves de càrrega

Per observar el comportament del sistema sota concurrència s'han realitzat proves de càrrega bàsiques utilitzant el simulador de neveres industrials. En aquestes proves s'ha mantingut una cadència d'enviament de 5 segons i s'ha modificat el nombre de neveres simulades, començant amb una càrrega baixa i augmentant progressivament el volum d'esdeveniments generats.

L'objectiu d'aquestes proves no ha estat obtenir una comparativa exhaustiva de rendiment, sinó comprovar fins a quin punt el sistema podia mantenir el flux d'ingesta, processament, persistència i visualització amb diferents nivells de concurrència.

TAULA 3: MÈTRIQES OBSERVADES DURANT LES PROVES DE CÀRREGA

Neveres	Events/min esperats	Events/min observats	Temps de processament
10	120	~110-120	Mitjana 1,36 s; últim 1,56 s.
25	300	~275-300	Mitjana 1,68 s; últim 1,24 s.
50	600	~560-600	Mitjana 2,31 s; últim 1,92 s.
100	1200	~1120-1200	Mitjana 49,1 s; últim 1,45 min.

Els resultats mostren que el volum d'events processats per minut s'aproxima al volum esperat generat pel simulador en totes les proves. Amb 10, 25 i 50 neveres, el sistema manté un temps de processament baix, amb valors mitjans entre 1,36 s i 2,31 s. En canvi, amb 100 neveres simulades, tot i que el nombre d'events per minut continua sent proper als 1200 events/min esperats, el temps de processament augmenta de manera clara, arribant a una mitjana de 49,1 segons i a un últim valor observat d'1,45 minuts. Aquest resultat indica que el sistema continua ingerint dades, però el servei de processament comença a acumular retard quan la càrrega arriba als 20 events/s.

TAULA 4: COMPARACIÓ D'ÚS DE RECURSOS ENTRE CÀRREGA BAIXA I CÀRREGA ALTA

Component	CPU 10	CPU 100	RAM 10	RAM 100
API	6,76 mCPU	39,6 mCPU	40,9 MB	43,0 MB
PostgreSQL	13,3 mCPU	24,9 mCPU	59,4 MB	63,8 MB
Processor	3,09 mCPU	18,8 mCPU	21,1 MB	22,3 MB
Redis	15,5 mCPU	19,1 mCPU	19,1 MB	22,1 MB

La comparació de recursos mostra que, en passar de 10 a 100 neveres simulades, el consum de CPU augmenta sobretot a l'API d'ingesta i al servei de processament. Tot i això, els valors observats continuen sent baixos en termes absoluts i l'ús de memòria es manté força estable. Per tant, la degradació observada en la prova de 100 neveres no sembla estar causada per una manca directa de CPU o memòria als pods, sinó per l'acumulació progressiva de retard en el processament dels missatges.

A més per a la prova amb 100 neveres també s'ha calculat la latència real a partir de la diferència entre la generació de l'event i el seu processament a PostgreSQL. La latència mitjana ha estat de 113 s, amb una p95 de 120 s i una p99 de 122 s. Aquest resultat confirma que, en l'escenari de càrrega més alt, el sistema continua ingerint dades però acumula retard en el processament dels missatges.



Fig. 7: Càrrega amb 100 neveres simulades.

5.4 Proves funcionals

S’han realitzat també proves funcionals sobre el comportament de les neveres simulades. Aquestes proves s’han fet amb una càrrega base de 10 neveres i una cadència d’enviament de 5 segons, per poder observar de manera clara l’efecte de cada acció sobre els esdeveniments generats posteriorment.

TAULA 5: PROVES FUNCIONALS REALITZADES

Prova	Neveres simulades	Cadència d’enviament	Objectiu	Resultat observat
1	10	5s	Apagar nevera	La nevera continua enviant events, però passa a estat <code>off</code> , el consum queda a 0 kW i la temperatura va augmentant.
2	10	5s	Forçar compressor	La nevera passa a estat <code>cooling</code> , augmenta el consum energètic i la temperatura va disminuint.
3	10	5s	Modificar setpoint	El nou setpoint queda persistent i afecta el comportament posterior del compressor.
4	10	5s	Silenciar alarma	L’alarma queda silenciada i els events posteriors no reporten alarma encara que es compleixi la condició.

Aquestes proves han permès comprovar que les comandes i els canvis de configuració quedaven persistits correctament i afectaven el comportament posterior del simulador. Per tant, el sistema no només rep esdeveniments, sinó que també manté l’estat de cada nevera i permet modificar-ne el funcionament durant l’execució.

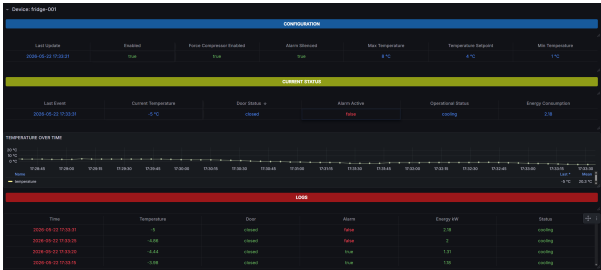


Fig. 8: Prova funcional d’apagar una nevera.

5.5 Discussió dels resultats

Els resultats obtinguts mostren que el sistema compleix el flux principal plantejat al projecte. Els esdeveniments generats pel simulador arriben correctament a l’API d’ingesta, es publiquen a Redis, són processats pel servei corresponent i

finalment queden persistits a PostgreSQL. A partir d’aquestes dades, Grafana pot representar tant l’estat general del sistema com el comportament individual de cada nevera.

Les proves funcionals també han permès comprovar que el sistema manté correctament l’estat i la configuració de les neveres simulades. Les accions d’apagar una nevera, forçar el compressor, modificar el setpoint o silenciar una alarma no només queden registrades, sinó que afecten els esdeveniments generats posteriorment. Això confirma que el sistema no funciona únicament com un receptor d’events, sinó que també incorpora una gestió bàsica de l’estat dels dispositius.

Pel que fa a les proves de càrrega, el comportament ha estat estable amb 10 i 25 neveres simulades. En aquests casos, el sistema ha pogut mantenir el ritme d’ingesta i processament sense problemes visibles. Amb 50 neveres, el sistema ha continuat funcionant correctament, però s’ha començat a observar un increment del temps de processament. Finalment, amb 100 neveres simulades, el sistema encara ha pogut rebre i persistir dades, però el processament ha deixat de ser òptim i alguns missatges han quedat pendents per a iteracions posteriors.

Aquest comportament no s’ha considerat un error funcional del sistema, sinó una limitació de la configuració utilitzada. El projecte s’ha desplegat sobre un clúster físic de Raspberry Pi i amb una configuració simple dels serveis, per tant és normal que apareguin límits quan s’incrementa la càrrega. Per millorar aquest punt es podrien afegir més rèpliques del servei de processament, ajustar el nombre de missatges consumits per iteració o revisar els recursos assignats als serveis dins de Kubernetes.

També cal tenir en compte que l’ús de Kubernetes, Helm, Ingress, VPN i eines d’observabilitat ha afegit complexitat al projecte. Per a un sistema funcionalment senzill, aquesta infraestructura pot semblar sobredimensionada, però en aquest cas ha servit per treballar amb un entorn més proper a una arquitectura distribuïda real. Això ha permès validar no només el codi dels microserveis, sinó també el desplegament, la comunicació entre components, l’accés als serveis i la monitorització.

6 CONCLUSIONS I TREBALL FUTUR

El projecte s’ha completat correctament i s’ha pogut implementar el sistema plantejat inicialment. El resultat final és una arquitectura funcional que permet generar esdeveniments des d’un simulador de neveres, ingerir-los mitjançant una API, publicar-los a Redis Streams, processar-los i guardar-los a PostgreSQL per poder-los consultar i visualitzar posteriorment a Grafana.

Pel que fa als objectius inicials, s’han assolit els punts principals del treball. S’ha desenvolupat un simulador concurrent de neveres industrials, una API d’ingesta, un sistema de missatgeria asíncron, un servei de processament i una base de dades per persistir tant els events com l’estat i la configuració dels dispositius. A més, tot el sistema s’ha desplegat sobre un clúster físic de Kubernetes i s’han creat panells de Grafana per comprovar-ne el comportament.

Les proves realitzades mostren que l’arquitectura és vàlida per representar un sistema distribuït simplificat. Amb càrregues baixes i mitjanes, el sistema ha pogut ingerir i processar els events sense problemes visibles. En canvi,

amb 100 neveres simulades s'ha observat una limitació important: la ingesta continua funcionant i el volum d'events per minut es manté proper al valor esperat, però el temps de processament augmenta progressivament. Per tant, el principal coll d'ampolla detectat no es troba tant en l'entrada de dades, sinó en la capacitat del servei de processament per consumir i persistir els missatges al mateix ritme en què es generen.

Durant el desenvolupament també han aparegut diversos problemes tècnics, sobretot relacionats amb la infraestructura. L'ús inicial d'una versió no LTS d'Ubuntu i alguns problemes amb containerd van provocar errors difícils de diagnosticar, fet que va portar a utilitzar una versió LTS més estable. També hi va haver problemes amb UFW i WireGuard, que es van acabar resolent simplificant la configuració amb wg-easy. La gestió dels volums persistents també va requerir diverses proves fins a optar per un NFS al node control plane. Finalment, en la part de xarxa del clúster, es va descartar Canal i es va acabar utilitzant Calico mitjançant manifest directe per reduir complexitat.

Una de les decisions que ha funcionat millor ha estat separar la ingesta del processament mitjançant Redis Streams. Això ha evitat que l'API hagués d'assumir tota la feina i ha fet més fàcil observar el comportament del sistema sota càrrega. També ha estat encertat utilitzar PostgreSQL, ja que era suficient per a l'abast del projecte i, a més, era una tecnologia amb la qual ja tenia experiència prèvia. Això ha reduït el risc d'implementació i ha permès centrar més temps en l'arquitectura i el desplegament.

Un aspecte important del treball és que la infraestructura ha acabat tenint més pes del previst inicialment. Tot i que l'objectiu principal era desenvolupar una arquitectura d'ingesta i processament d'events, bona part de la complexitat real del projecte ha aparegut en el desplegament del clúster, la configuració de xarxa, la persistència, l'accés remot i la monitorització. Això ha fet que el projecte no només consistís a implementar els microserveis, sinó també a construir i estabilitzar l'entorn on aquests serveis havien d'executar-se.

La decisió més discutible ha estat començar directament amb una infraestructura basada en Kubernetes. Tot i que ha aportat realisme al projecte i l'ha apropiat més a un entorn d'empresa, també ha afegit força complexitat des de les primeres fases. Si tornés a repetir el treball, probablement validaria primer tot el flux funcional amb Docker Compose i migraria després a Kubernetes un cop els serveis principals ja estiguessin estabilitzats. També hauria estat útil plantejar abans una estratègia d'escalat del servei de processament, especialment després d'observar el comportament de la prova amb 100 neveres.

Com a treball futur, es podria millorar el rendiment del sistema escalant el servei de processament, ajustant el consum de missatges de Redis i revisant els recursos assignats als diferents components. També es podrien afegir alertes reals, millorar els panells de Grafana i ampliar el simulador amb nous tipus de dispositius o comportaments més complexos. Finalment, seria interessant validar l'arquitectura amb dispositius físics reals o amb un volum de dades més proper a un entorn productiu.

ÚS DE LA IA GENERATIVA

Durant el desenvolupament del treball s'han utilitzat eines d'intel·ligència artificial generativa com a suport puntual en algunes parts del procés. Principalment, s'han fet servir per revisar la redacció dels documents, millorar la claredat d'alguns apartats, estructurar millor determinades seccions i resoldre dubtes tècnics concrets relacionats amb la implementació i el desplegament del sistema.

En l'àmbit tècnic, aquestes eines s'han utilitzat com a suport per consultar exemples de referència, revisar fragments de codi auxiliar, resoldre errors puntuals i contrastar l'ús d'algunes tecnologies emprades en el projecte, com ara FastAPI, Redis Streams, PostgreSQL, Docker, Kubernetes i l'ús de programació asíncrona en Python. El codi incorporat al projecte ha estat revisat, adaptat, integrat i validat per l'estudiant dins de l'arquitectura desenvolupada.

També s'ha utilitzat una eina d'IA generativa per modificar una imatge de la infraestructura física del clúster amb una finalitat visual, per tal que encaixés millor dins del format del document.

L'ús d'aquestes eines no ha substituït el treball propi de disseny, implementació, desplegament, validació ni interpretació dels resultats. Les decisions tècniques, el desenvolupament dels components, la configuració de la infraestructura, les proves realitzades i l'anàlisi dels resultats han estat duts a terme per l'estudiant.

AGRAÏMENTS

Vull agrair al meu tutor del Treball Final de Grau el seguiment realitzat durant el projecte i, especialment, la llibertat que m'ha donat a l'hora de plantejar-ne el desenvolupament. Aquesta flexibilitat m'ha permès adaptar la idea inicial del treball a una proposta més personal, centrada en tecnologies que volia utilitzar, investigar i posar en pràctica.

També li vull agrair haver-me permès prendre decisions pròpies tant en la redacció i l'enfocament del document com en el disseny de l'arquitectura i la metodologia de desplegament del clúster. Aquesta llibertat ha fet que el projecte fos més motivador i m'ha permès aprendre no només sobre el sistema desenvolupat, sinó també sobre la infraestructura i les eines utilitzades per desplegar-lo.

REFERÈNCIES

- [1] FastAPI. *FastAPI documentation*. Disponible a: <https://fastapi.tiangolo.com/>. Consultat: juny de 2026.
- [2] Redis. *Redis Streams documentation*. Disponible a: <https://redis.io/docs/latest/develop/data-types/streams/>. Consultat: juny de 2026.
- [3] PostgreSQL. *PostgreSQL documentation*. Disponible a: <https://www.postgresql.org/docs/>. Consultat: juny de 2026.
- [4] Kubernetes. *Kubernetes documentation*. Disponible a: <https://kubernetes.io/docs/>. Consultat: juny de 2026.

- [5] Kubernetes. *Creating a cluster with kubeadm*. Disponible a: <https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/>. Consultat: juny de 2026.
- [6] Helm. *Helm documentation*. Disponible a: <https://helm.sh/docs/>. Consultat: juny de 2026.
- [7] Docker. *Docker documentation*. Disponible a: <https://docs.docker.com/>. Consultat: juny de 2026.
- [8] Tigera. *Calico documentation*. Disponible a: <https://docs.tigera.io/>. Consultat: juny de 2026.
- [9] Traefik Labs. *Traefik Proxy documentation*. Disponible a: <https://doc.traefik.io/traefik/>. Consultat: juny de 2026.
- [10] Prometheus. *Prometheus documentation*. Disponible a: <https://prometheus.io/docs/introduction/overview/>. Consultat: juny de 2026.
- [11] Grafana Labs. *Grafana documentation*. Disponible a: <https://grafana.com/docs/>. Consultat: juny de 2026.
- [12] WireGuard. *WireGuard: fast, modern, secure VPN tunnel*. Disponible a: <https://www.wireguard.com/>. Consultat: juny de 2026.
- [13] Cloudflare. *Cloudflare Tunnel documentation*. Disponible a: <https://developers.cloudflare.com/cloudflare-one/networks/connectors/cloudflare-tunnel/>. Consultat: juny de 2026.
- [14] Apache Kafka. *Apache Kafka documentation*. Disponible a: <https://kafka.apache.org/documentation/>. Consultat: juny de 2026.
- [15] RabbitMQ. *RabbitMQ documentation*. Disponible a: <https://www.rabbitmq.com/docs>. Consultat: juny de 2026.
- [16] Timescale. *TimescaleDB documentation*. Disponible a: <https://docs.timescale.com/>. Consultat: juny de 2026.
- [17] Docker. *Docker Compose documentation*. Disponible a: <https://docs.docker.com/compose/>. Consultat: juny de 2026.

APÈNDIX

A.1 Mètriques completes de les proves de càrrega

Aquest annex recull les mètriques completes de les proves de càrrega realitzades amb 10, 25, 50 i 100 neveres simulades. S'hi inclouen tant les mètriques de rendiment del sistema, com el volum d'events generats, els events per segon esperats, el total d'events observats i els temps de processament, com les mètriques d'ús de CPU i memòria dels components principals. Les dades provenen dels panells de Grafana connectats a PostgreSQL i Prometheus durant la finestra de cada prova.

TAULA 6: MÈTRIQES COMPLETES DE LES PROVES DE CÀRREGA

Neveres	Cadència	Events/s esp.	Events/min esp.	Events/min obs.	Total events	Mitjana	Últim	p95/p99
10	5 s	2	120	~110–120	589	1,36 s	1,56 s	–
25	5 s	5	300	~275–300	1470	1,68 s	1,24 s	–
50	5 s	10	600	~560–600	2936	2,31 s	1,92 s	–
100	5 s	20	1200	~1120–1200	4169	49,1 s	1,45 min	120 s / 122 s

TAULA 7: MÈTRIQES DE CPU OBSERVADES DURANT LES PROVES DE CÀRREGA

Neveres	Component	Mitjana	Últim valor	Màxim
10	API d'ingesta	6,76 mCPU	6,67 mCPU	6,86 mCPU
10	PostgreSQL	13,3 mCPU	15,2 mCPU	15,2 mCPU
10	Processor	3,09 mCPU	3,10 mCPU	3,12 mCPU
10	Redis	15,5 mCPU	15,5 mCPU	15,6 mCPU
25	API d'ingesta	13,7 mCPU	14,0 mCPU	14,0 mCPU
25	PostgreSQL	14,1 mCPU	13,6 mCPU	16,7 mCPU
25	Processor	7,33 mCPU	7,45 mCPU	7,45 mCPU
25	Redis	16,4 mCPU	16,3 mCPU	16,5 mCPU
50	API d'ingesta	23,7 mCPU	24,9 mCPU	25,6 mCPU
50	PostgreSQL	20,7 mCPU	19,6 mCPU	22,4 mCPU
50	Processor	13,8 mCPU	14,6 mCPU	14,8 mCPU
50	Redis	17,8 mCPU	18,0 mCPU	18,1 mCPU
100	API d'ingesta	39,6 mCPU	49,2 mCPU	49,5 mCPU
100	PostgreSQL	24,9 mCPU	26,4 mCPU	26,4 mCPU
100	Processor	18,8 mCPU	22,1 mCPU	22,1 mCPU
100	Redis	19,1 mCPU	19,9 mCPU	20,1 mCPU

TAULA 8: MÈTRIQES DE MEMÒRIA OBSERVADES DURANT LES PROVES DE CÀRREGA

Neveres	Component	Mitjana	Últim valor	Màxim
10	API d'ingesta	40,9 MB	40,9 MB	40,9 MB
10	PostgreSQL	59,4 MB	59,5 MB	59,5 MB
10	Processor	21,1 MB	21,2 MB	21,2 MB
10	Redis	19,1 MB	19,0 MB	19,4 MB
25	API d'ingesta	41,1 MB	41,1 MB	41,6 MB
25	PostgreSQL	62,0 MB	62,1 MB	62,3 MB
25	Processor	22,0 MB	22,0 MB	22,0 MB
25	Redis	20,1 MB	21,7 MB	n.d.
50	API d'ingesta	41,8 MB	41,8 MB	42,3 MB
50	PostgreSQL	62,9 MB	63,2 MB	63,7 MB
50	Processor	22,2 MB	22,5 MB	22,5 MB
50	Redis	20,9 MB	21,2 MB	21,2 MB
100	API d'ingesta	43,0 MB	42,9 MB	43,2 MB
100	PostgreSQL	63,8 MB	64,6 MB	64,6 MB
100	Processor	22,3 MB	22,6 MB	22,6 MB
100	Redis	22,1 MB	22,9 MB	22,9 MB

En general, les dades mostren que el consum de memòria es manté força estable entre les diferents proves, mentre que el consum de CPU augmenta de manera més visible a mesura que creix el nombre de neveres simulades. Tot i això, els valors absoluts observats continuen sent baixos, fet que reforça la idea que la degradació detectada en l'escenari de 100 neveres està més relacionada amb l'acumulació de retard en el processament que no pas amb un esgotament directe de CPU o memòria.

A.2 Captures de proves funcionals

Aquest apartat recull captures complementàries de les proves funcionals realitzades sobre les neveres simulades. Les imatges mostren el comportament observat en accions com apagar una nevera, forçar el compressor, modificar el setpoint o silenciar una alarma.

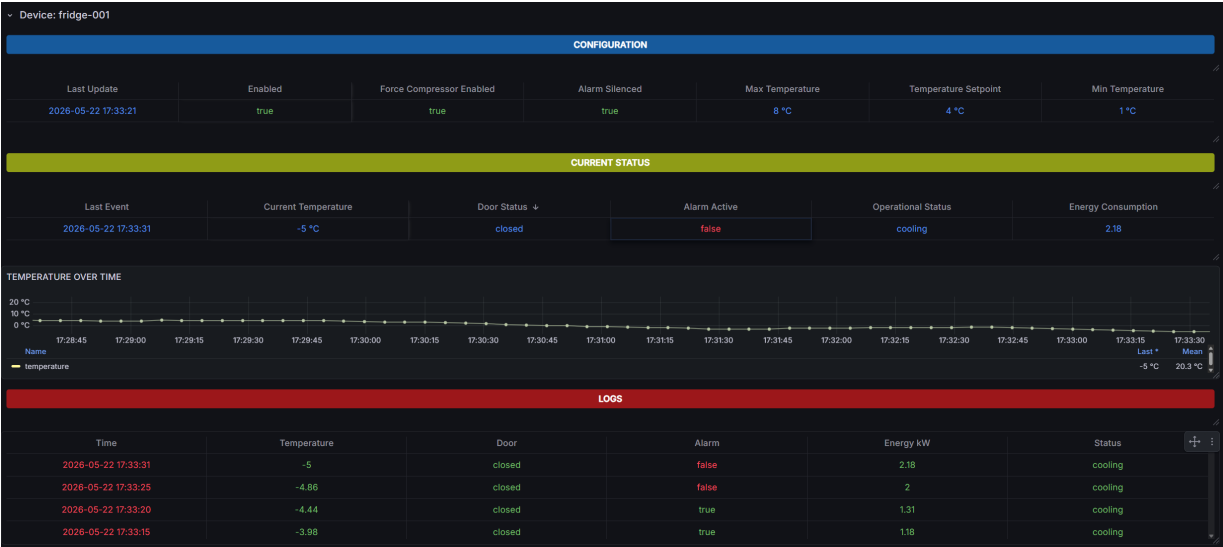


Fig. 9: Prova funcional d’apagar una nevera simulada.

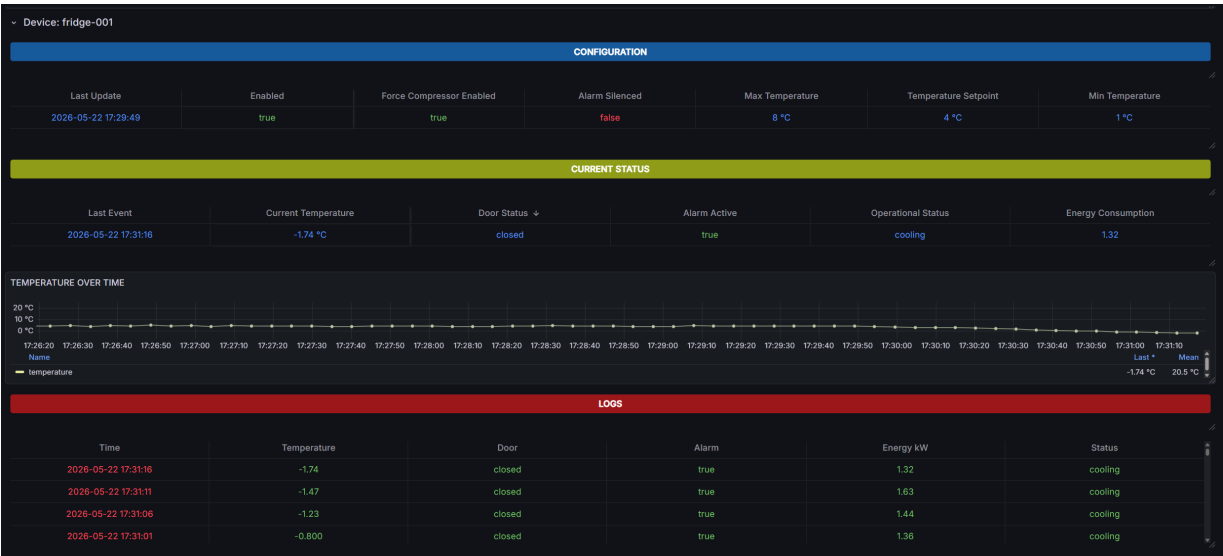


Fig. 10: Prova funcional de forçar el compressor d’una nevera simulada.

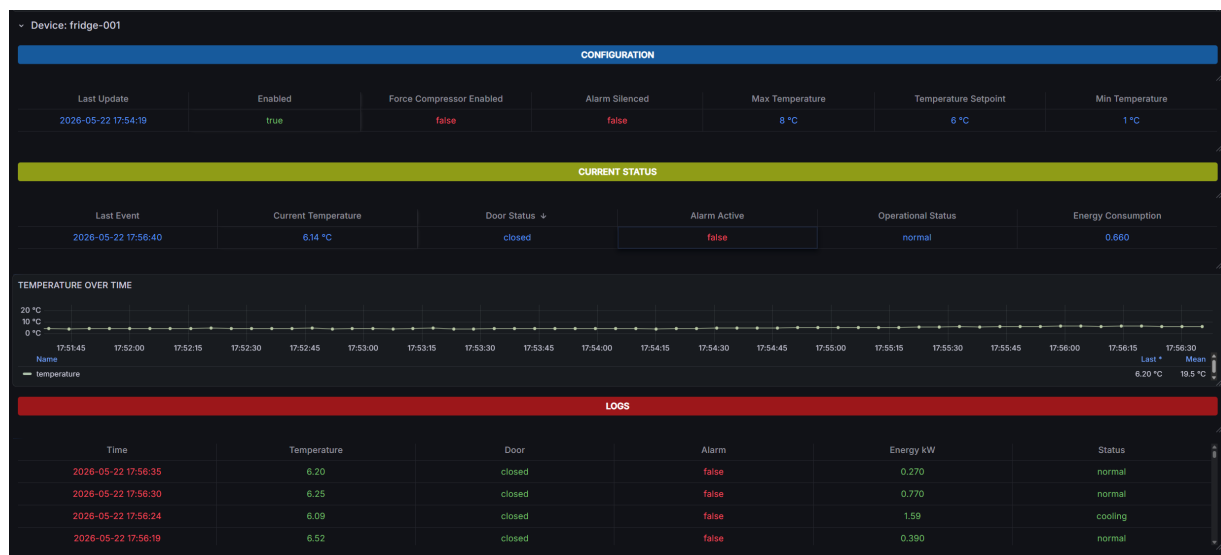


Fig. 11: Prova funcional de modificació del setpoint de temperatura.

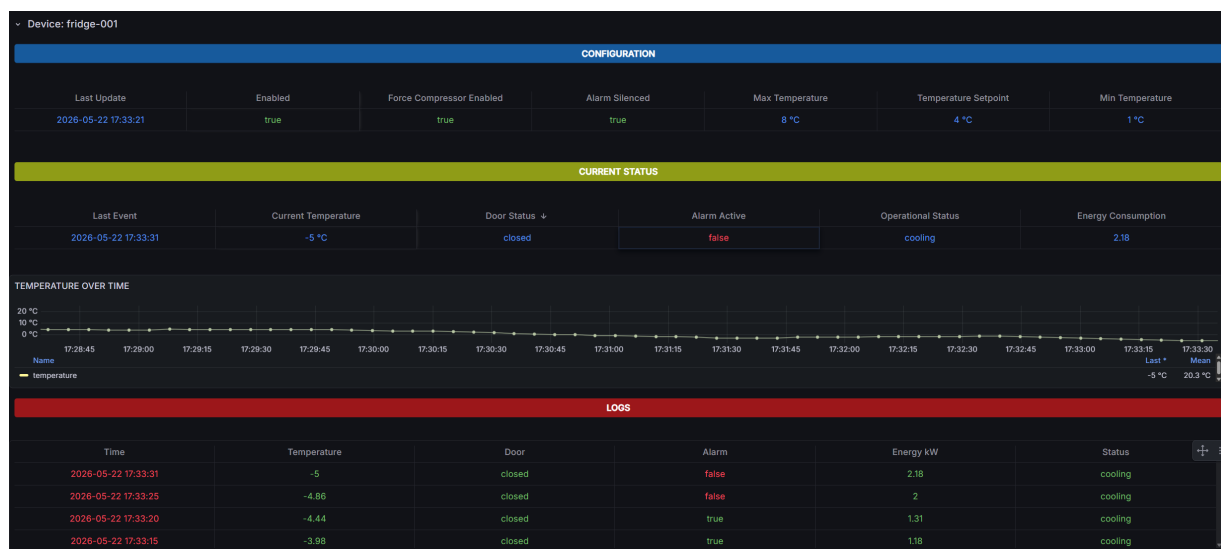


Fig. 12: Prova funcional de silenciar una alarma.

A.3 Captures de proves de càrrega

Aquest apartat recull captures complementàries de les proves de càrrega executades amb diferents volums de neveres simulades. Aquestes proves permeten observar el comportament del sistema en escenaris de càrrega baixa, moderada i alta.



Fig. 13: Prova de càrrega amb 10 neveres simulades.



Fig. 14: Prova de càrrega amb 25 neveres simulades.



Fig. 15: Prova de càrrega amb 50 neveres simulades.



Fig. 16: Prova de càrrega amb 100 neveres simulades.