UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

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

Revaliente Torres, Lucía. *Arquitectura serverless per al processament quasi en temps real de dades biomèdiques wearable en àrees hospitalàries no crítiques.* Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Grau en Enginyeria de Dades)

This version is available at https://ddd.uab.cat/record/337319

under the terms of the CC BY-SA license.

Arquitectura *serverless* per al processament quasi en temps real de dades biomèdiques *wearable* en àrees hospitalàries no crítiques

Lucía Revaliente Torres

Resum— Aquest treball presenta el disseny, la implementació i l'avaluació d'una arquitectura *cloud* distribuïda per a la ingestió, el processament, la persistència i la monitorització quasi en temps real de dades biomèdiques provinents de dispositius *wearable*, considerant com a cas d'ús la monitorització contínua de pacients hospitalaris en àrees no crítiques. L'arquitectura, desplegada a AWS mitjançant un model completament *serverless*, integra Amazon Kinesis Data Streams per a la ingestió, AWS Lambda per al processament, Amazon S3 i DynamoDB per a la persistència i Amazon CloudWatch per a la monitorització. El sistema s'ha avaluat mitjançant tres escenaris de càrrega creixent (de ~8 ev/s a ~4.304 ev/s), definits a partir de les freqüències reals del *dataset* WESAD, amb un SLO de P95<10 s i una taxa de pèrdua inferior a l'1%. Els resultats confirmen que el paradigma *serverless* és una base arquitectònica viable per a la monitorització biomèdica contínua en àrees no crítiques: els escenaris base i sostingut compleixen la totalitat dels SLOs (P95 màxim 1.074 ms, pèrdua 0,00%), mentre que l'escenari de pic revela el subdimensionament de la capa d'ingestió com a factor limitant. El diagnosi experimental identifica el *throttling* de Kinesis com a causa principal i demostra que la totalitat del retard mesurat s'acumula al *buffer* de Kinesis i no al processament intern de Lambda, orientant així les optimitzacions futures cap a la capa d'ingestió i al disseny de les *partition keys*.

Paraules clau— Arquitectures *serverless*, processament en quasi temps real, monitorització biomèdica, dispositius *wearable*, computació en el núvol, ingestió de dades en *streaming*, AWS, Kinesis, Lambda.

Abstract— This work presents the design, implementation and evaluation of a distributed cloud architecture for the near real-time ingestion, processing, persistence and monitoring of biomedical data from wearable devices, using the continuous monitoring of hospital patients in non-critical areas as a use case. The architecture, deployed on AWS following a fully serverless model, integrates Amazon Kinesis Data Streams for ingestion, AWS Lambda for processing, Amazon S3 and DynamoDB for persistence, and Amazon CloudWatch for monitoring. The system has been evaluated through three experimental scenarios of increasing load (from ~8 ev/s to ~4,304 ev/s), defined from the real sampling frequencies of the WESAD dataset, with an end-to-end latency SLO of P95<10 s and a target data loss rate below 1%. Results confirm that the serverless paradigm is a viable architectural foundation for continuous biomedical monitoring in non-critical areas: the base and sustained scenarios meet all SLOs (maximum P95 of 1,074 ms, 0.00% data loss), while the peak scenario reveals the undersizing of the ingestion layer as the limiting factor. The experimental diagnosis identifies Kinesis throttling as the main cause and demonstrates that the entire measured delay accumulates in the Kinesis buffer and not in the internal Lambda process, guiding future optimizations to the ingestion layer and the design of the partition keys.

Index Terms—Serverless architectures, near real-time processing, biomedical monitoring, wearable devices, cloud computing, streaming data ingestion, AWS, Kinesis, Lambda.

1 Introducció

En entorns hospitalaris, especialment en àrees no crítiques, la monitorització dels pacients es basa habitualment en mesures de constants vitals realitzades a intervals determinats. Aquest model de seguiment discontinu pot dificultar la detecció precoç de canvis en l'estat fisiològic dels pacients, de manera que determinades alteracions poden passar desapercebudes o ser detectades amb retard [1]. Davant d'aquest escenari, els dispositius *wearable* han emergit com una alternativa que permet avançar cap a models de monitorització contínua, generant fluxos continus de dades que descriuen l'evolució temporal de múltiples variables fisiològiques [2][3][4].

Tanmateix, la disponibilitat d'aquestes dades planteja nous reptes tecnològics. En escenaris amb múltiples pacients monitoritzats simultàniament, els sistemes han de gestionar fluxos continus de dades heterogènies, processar grans volums d'informació amb baixa latència i adaptar-se a increments de càrrega sense comprometre la disponibilitat ni la integritat de la informació [5][6][7]. En aquest context, la computació en el núvol ofereix una base adequada per afrontar aquests requisits, proporcionant serveis gestionats per a la ingestió en *streaming*, el processament distribuït i l'emmagatzematge escalable [6][7]. Dins d'aquest paradigma, les arquitectures *serverless* permeten reduir la complexitat operativa i facilitar l'adaptació dinàmica dels recursos a la càrrega de treball [8][9].

Partint d'aquesta motivació, aquest treball té com a objectiu demostrar experimentalment si el model *serverless* és una base arquitectònica viable per a la monitorització contínua de pacients hospitalaris en àrees no crítiques. Per assolir-ho, el sistema s'avalua sota condicions de càrrega creixent derivades de freqüències reals de dispositius *wearable*, analitzant-ne el comportament en termes de latència, rendiment, integritat, escalabilitat i cost operatiu. Els resultats permeten identificar els límits de la configuració actual i els factors limitants sota càrrega extrema.

- *E-mail de contacte: lucia.revaliente@autonoma.cat*

- *Treball tutoritzat per: Esteve Alquezar Mora (Àrea d'Arquitectura i de Tecnologia de Computadors)*
- *Curs 2025/26*

La memòria s'organitza en nou capítols, des de l'estat de l'art i el disseny de l'arquitectura fins a l'avaluació experimental i les conclusions.

2 ESTAT DE L'ART

2.1 Monitorització biomèdica basada en dispositius *wearable*

La literatura mostra un interès creixent en l'ús de dispositius *wearable* per al seguiment continu de pacients tant en entorns clínics com domiciliars. Diversos estudis han explorat la seva aplicació en la monitorització cardiovascular, la detecció precoç d'alteracions fisiològiques i el suport a la gestió de malalties cròniques [2][3][4]. Aquests treballs assenyalen la necessitat d'infraestructures capaces de processar eficientment els fluxos continus de dades generats [2][5].

2.2 Arquitectures cloud per al processament de dades biomèdiques

Per donar resposta a aquest repte, diversos treballs han proposat arquitectures basades en la computació en el núvol. Bahmani et al. [6] i Orro et al. [7] proposen plataformes per a la gestió i monitorització de dades de salut, evidenciant els avantatges en escalabilitat i disponibilitat, però també la necessitat de controlar la latència, la integritat i la supervisió operativa.

2.3 Arquitectures *serverless*

En aquest context, les arquitectures *serverless* han emergit com una alternativa per al desplegament de sistemes distribuïts de processament quasi en temps real. Aquest paradigma delega la gestió de la infraestructura al proveïdor *cloud* i adapta els recursos a la càrrega de manera automàtica. Estudis recents en destaquen l'escalabilitat i la simplicitat operativa, però també assenyalen limitacions en latència, observabilitat i dependència del proveïdor [8][9][10].

2.4 Posicionament del treball

La literatura revisada posa de manifest el potencial dels dispositius *wearable*, les plataformes *cloud* i les arquitectures *serverless* per a la monitorització biomèdica. Tot i això, el comportament d'aquests sistemes sota càrrega creixent en escenaris quasi en temps real ha estat poc estudiat. Aquest treball contribueix a cobrir aquesta mancança mitjançant el disseny, la implementació i l'avaluació experimental d'una arquitectura *serverless* sobre AWS, en tres escenaris de càrrega derivats de freqüències reals de dispositius wearable.

3 OBJECTIUS I METODOLOGIA

3.1 Objectiu general

L'objectiu del treball és dissenyar, implementar i avaluar experimentalment una arquitectura *cloud* distribuïda basada en serveis *serverless* per al processament quasi en temps real de dades biomèdiques provinents de dispositius *wearable*, analitzant-ne el comportament en termes de latència, rendiment, integritat, escalabilitat i cost operatiu. L'objectiu final és determinar la viabilitat d'aquest model com a base arquitectònica per a la monitorització contínua de pacients hospitalaris en àrees no crítiques.

3.2 Objectius específics

Per assolir aquest objectiu general, es plantegen els objectius específics següents:

- Identificar els requisits d'un sistema de processament quasi en temps real de dades biomèdiques.
- Dissenyar una arquitectura distribuïda basada en serveis *cloud* gestionats.
- Implementar un prototip funcional desplegat sobre AWS.
- Definir una metodologia experimental per avaluar el comportament del sistema.
- Analitzar experimentalment el comportament del sistema en termes de latència, rendiment, integritat i escalabilitat, identificar els principals colls d'ampolla i determinar els límits de la configuració sota càrrega extrema.

3.3 Abast

L'abast del treball inclou la ingestió, el processament, la persistència i la monitorització de dades biomèdiques, així com l'avaluació experimental de l'arquitectura proposada. Queden fora de l'abast del projecte la validació clínica dels resultats, l'ús de dispositius reals en entorns hospitalaris, l'aplicació de models de *machine learning* (ML) sobre les dades processades i el desenvolupament d'interfícies d'usuari destinades a personal sanitari o pacients.

3.4. Metodologia

El treball segueix una metodologia iterativa. Aquest enfocament permet analitzar progressivament les decisions arquitectòniques adoptades i introduir millores sobre el prototip a mesura que es detecten limitacions o oportunitats d'optimització.

En una primera fase s'estableixen els requisits del sistema i el model de dades de referència. A continuació, es dissenya una arquitectura basada en serveis *cloud* gestionats. Posteriorment, s'implementa un prototip funcional. Finalment, el sistema s'avalua sota diferents nivells de càrrega per caracteritzar-ne el comportament.

4 REQUISITS I MODEL DE DADES

4.1 Requisits del sistema

A partir dels objectius definits prèviament, s'estableixen un conjunt de requisits no funcionals que guien tant el disseny de l'arquitectura com la seva posterior avaluació experimental. Atès que el sistema està orientat a la monitorització contínua de pacients en àrees hospitalàries no crítiques, es prioritza la capacitat de processar dades amb baixa latència, preservar-ne la integritat i adaptar-se a increments de càrrega sense comprometre el funcionament global del sistema. Per avaluar aquests aspectes es defineixen els requisits recollits a la Taula 1.

Taula 1. Requisits no funcionals de l'arquitectura.

Categoria	Requisit
Latència	Temps de processament inferior a 10 segons per al P95
Integritat	Pèrdua inferior a l'1% dels esdeveniments processats
Escalabilitat	Capacitat d'adaptació a increments de càrrega sense degradació significativa del servei
Observabilitat	Disponibilitat de mètriques i registres per monitoritzar el comportament del sistema
Cost	Cost operatiu reduït mitjançant l'ús de serveis cloud gestionats

El SLO principal s'estableix en P95 < 10.000 ms, llindar derivat del context clínic: en àrees no crítiques, les alteracions fisiològiques rellevants es produeixen en escales temporals de minuts [1]. Per tant, una latència de l'ordre de segons no compromet la utilitat clínica del sistema. A més, s'utilitza el P95 com a indicador de referència per evitar que valors extrems puntuals condicionin l'avaluació [12].

4.2 *Dataset* de referència

Per avaluar els requisits definits, s'utilitza com a referència el *dataset* WESAD (*Wearable Stress and Affect Detection*) [13]. Aquest és àmpliament utilitzat en treballs de recerca relacionats amb l'anàlisi de senyals fisiològics. Conté dades multimodals recollides mitjançant dispositius *wearable*, incloent senyals fisiològiques i de moviment representades com a sèries temporals contínues amb freqüències de mostreig heterogènies.

Tanmateix, aquestes dades no s'utilitzen com a font directa d'esdeveniments durant els experiments. WESAD serveix com a referència estructural per definir l'esquema de dades, les freqüències de mostreig i la naturalesa multimodal dels registres. A partir d'aquesta informació, es desenvolupa un simulador capaç de generar fluxos continus d'esdeveniments biomèdics. El detall complet dels sensors i les seves freqüències es presenta a l'Apèndix 1.

4.3 Model de dades

Els events generats pel simulador segueixen un model de dades comú que defineix l'estructura de la informació al llarg de tot el *pipeline*. Atesa l'heterogeneïtat de freqüències de mostreig entre sensors, s'adopta un registre per mostra: cada sensor genera esdeveniments independents enviats segons la seva freqüència específica. Aquest fet permet representar de manera homogènia fluxos de dades multimodals i simplifica el seu processament distribuït.

Per preservar la traçabilitat de la informació, es defineixen dos nivells de persistència: dades crues i processades. Les dades *raw* corresponen als esdeveniments originals generats pel simulador. En canvi, les dades processades incorporen la informació temporal i els atributs generats durant el processament. Aquesta separació garanteix la reproducibilitat dels resultats, facilita la verificació d'integritat al llarg del *pipeline* i permet identificar discrepàncies entre les dades originals i les processades. L'estructura detallada dels camps i exemples de registres es recullen a l'Apèndix 1.

5 Disseny de l'arquitectura

5.1 Arquitectura conceptual

A partir dels requisits definits prèviament, es dissenya una arquitectura distribuïda basada en un model de processament orientat a esdeveniments, estructurada en cinc blocs principals: generació de registres, ingestió, processament, persistència i monitorització. La Figura 1 presenta una vista conceptual del sistema.

Cada bloc assumeix una responsabilitat específica dins del *pipeline*, de manera que la seva separació facilita l'escalabilitat del sistema, simplifica l'evolució independent dels components i permet incorporar mecanismes de monitorització sense afectar el flux principal de processament.

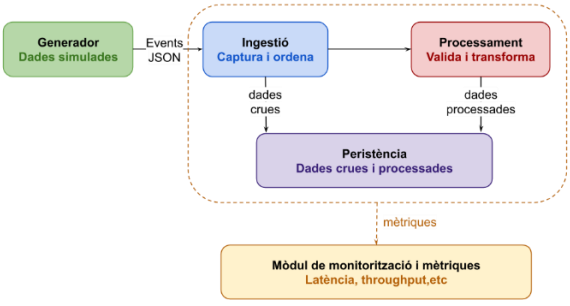

Figura 1. Model conceptual de l'arquitectura i flux de dades.

5.2 Arquitectura AWS

A partir del model conceptual, es concreta una implementació completament *serverless* basada en serveis gestionats d'Amazon Web Services (AWS). AWS s'ha escollit per la maduresa dels seus serveis de *streaming*, la integració nativa entre components *serverless* i la disponibilitat de Free Tier. L'arquitectura resultant estructura el flux de dades en cinc blocs: generació, ingestió, processament, persistència i monitorització, tal com es pot observar a la Figura 2.

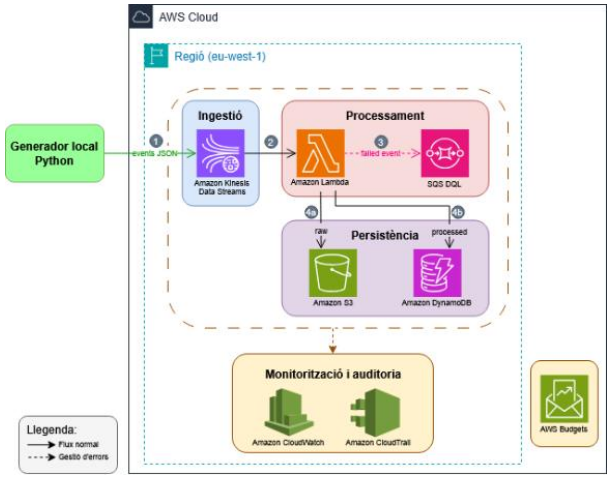

Figura 2. Diagrama de l'arquitectura AWS i flux de dades.

Com es pot observar, els esdeveniments biomèdics generats pel simulador són ingerits mitjançant Kinesis Data Streams, processats per Lambda i persistits per S3 (dades *raw)* i DynamoDB (dades processades). Els events fallits s'envien a una DLQ de SQS. Paral·lelament, CloudWatch, CloudTrail i Budgets cobreixen la monitorització, auditoria i control de cost.

5.3 Decisions arquitectòniques

A continuació, es detallen les decisions arquitectòniques adoptades per a cada component del sistema. Per a cada servei seleccionat s'inclou la principal alternativa descartada, el detall complet es presenta a l'Apèndix 2.

5.3.1 Ingestió

S'ha seleccionat Amazon Kinesis Data Streams com a servei d'ingestió. En el context d'aquest sistema, on múltiples sensors generen fluxos continus a freqüències heterogènies (de 4 Hz a 700 Hz), Kinesis ofereix les propietats necessàries: *streaming* continu d'alta freqüència, preservació de l'ordre temporal per *partition key* (`subject_id + sensor_type`), gestió de pics mitjançant *buffering* al *stream* i retenció de 24 hores per a reprocessament en cas de fallada [14]. La seva integració nativa amb Lambda via *event source mapping* permet desacoblar la ingestió del processament sense configuració addicional.

La principal alternativa descartada és Amazon MSK (Managed *S*treaming for Apache Kafka). Tot i ser la referència estàndard del sector per a *pipelines* de *streaming* d'alta freqüència, l'absència de Free Tier, la complexitat operativa desproporcionada per a un prototip individual i la necessitat de configuració addicional per integrar-se amb Lambda l'han fet inadequat per a aquest context. En un escenari de producció amb centenars de pacients simultanis, Kafka seria la decisió arquitectònica preferida.

5.3.2 Processament

S'ha seleccionat AWS Lambda com a servei de processament principal. Lambda actua com a punt d'escriptura del sistema: és responsable de persistir les dades crues a Amazon S3 i les dades processades a Amazon DynamoDB, consolidant en un únic component les operacions de processament i escriptura [15]. Per a cada event realitza la validació de camps, la conversió d'unitats i l'enriquiment amb *timestamps*. Pel que fa el rendiment, cal tenir en compte el fenomen de *cold start*, que pot introduir una latència addicional puntual de 100–500 ms a l'inici dels escenaris, sense comprometre el SLO [15].

Per a la gestió d'errors s'afegeix Amazon SQS com a Dead Letter Queue (DLQ). Els events no procesats correctament s'envien per a anàlisi posterior.

La principal alternativa descartada és Apache Flink (Amazon Managed Service for Apache Flink). Tot i ser tècnicament idoni per al processament de *streams* d'alta freqüència, introdueix una complexitat operativa alta i un cost base sense Free Tier incompatibles amb el prototip [16]. En un escenari de producció amb requisits de transformació complexa o centenars de pacients simultanis, Flink seria el servei escollit.

5.3.3 Persistència

La capa de persistència s'organitza en dos nivells amb serveis diferenciats per a cada finalitat, amb Lambda com a únic escriptor del sistema.

Per a dades crues s'utilitza Amazon S3. Atès que els registres *raw* es conserven per a reprocessament futur i no per a consultes operatives, l'absència d'estructura no representa cap limitació: S3 permet conservar els events exactament com arriben amb durabilitat de 99,999999999% i escalabilitat il·limitada [17].

Per a dades processades s'utilitza Amazon DynamoDB. La monitorització del *pipeline* requereix accés directe als registres per clau (`subject_id+event_id`) amb latència de mil·lisegons, un requisit que DynamoDB cobreix de forma nativa amb un Free Tier generós [18].

El disseny de les claus de partició és crític per evitar el fenomen de *hot partitions*, que es produeix quan massa escriptures es concentren en una mateixa partició i pot causar *throttling*. Per mitigar aquest risc, la partition key es defineix com la concatenació `subject_id#sensor_type`, distribuint les escriptures entre tantes particions com tipus de sensors actius per subjecte. Com a sort key s'utilitza event_id (UUID), garantint la unicitat absoluta de cada registre i la idempotència de les escriptures.

La principal alternativa descartada per a les dades processades és S3+Athena: tot i ser més econòmic per a anàlisi massiu ocasional, introdueix latències de consulta d'entre 2 i 30 segons depenent del volum i la complexitat, inadequades per a la monitorització operativa del *pipeline*.

5.3.4 Monitorització i auditoria

S'ha seleccionat Amazon CloudWatch per a la recollida de mètriques custom de cada bloc del pipeline, amb alarmes automàtiques sobre mètriques clau i verificació d'integritat comparant events ingestats i registres emmagatzemats [20]. S'ha descartat AWS X-Ray per absència de sistema d'alarmes natiu i inhabilitat per comparar recomptes entre blocs [21].

AWS CloudTrail s'incorpora com a mecanisme d'auditoria complementari per registrar les accions sobre els recursos del sistema. Tot i que el projecte no aborda el compliment complet de normatives clíniques, la traçabilitat sobre accessos i modificacions és rellevant en entorns amb dades potencialment sensibles. Per al control del cost operatiu, AWS Budgets configura alertes automàtiques al 80-100% del pressupost mensual de 10$.

5.4. *Trade-off* arquitectònics

A continuació, es presenten els principals *trade-offs* arquitectònics del sistema, analitzant el cost i el benefici de cada decisió de disseny tant en el context del prototip com en la seva projecció cap a un escenari de producció.

5.4.1 Near real-time vs real-time

L'adopció d'un model quasi en temps real, amb latències de l'ordre de segons, en lloc d'un model estrictament de temps real, amb latències de mil·lisegons, és la decisió arquitectònica més determinant del sistema. Permet usar serveis *serverless* com Kinesis i Lambda amb un cost molt inferior, i és suficient per al cas d'ús considerat, on les alteracions fisiològiques rellevants es produeixen en escales temporals de minuts [1][11]. Aquesta decisió seguiria sent vàlida en un escenari de producció en àrees no crítiques.

5.4.2 Serverless vs infraestructura gestionada

L'arquitectura completament *serverless* maximitza la simplicitat operativa i minimitza el cost, però implica

menys control sobre paràmetres de baix nivell: el paral·lelisme intern de Lambda, la distribució de càrrega entre shards, la gestió de la memòria per invocació o el comportament en situacions de *throttling*. En un escenari de producció amb centenars de pacients simultanis, caldria reconsiderar el component de processament cap a Apache Flink, que ofereix control explícit sobre el paral·lelisme, *checkpoints* i finestres temporals, a canvi d'una complexitat operativa i un cost base superiors.

5.4.3 Simplicitat vs resiliència en la gestió d'errors

El prototip no implementa reprocessament automàtic dels events de la DLQ per simplicitat. Els events fallits requereixen intervenció manual, un compromís acceptable per al prototip però inacceptable en producció. Com a evolució futura, el reprocessament automàtic es podria implementar mitjançant una Lambda dedicada o Step Functions.

5.4.4 Single-regió vs multi-regió

El prototip es desplega en una única regió per cost i agilitat operativa. En un entorn de producció biomèdic real, l'alta disponibilitat i la resiliència justificarien una arquitectura multi-regió, especialment considerant la naturalesa sensible de les dades gestionades. Amb infraestructura com a codi, el sistema podria ser relançat en una segona regió en minuts.

6 IMPLEMENTACIÓ

La implementació s'ha desenvolupat seguint el paradigma Infrastructure as Code (IaC) mitjançant Terraform, definint la infraestructura de forma declarativa i reproduïble [23]. Tots els recursos es gestionen a través de fitxers de configuració versionats al repositori públic del projecte: https://github.com/luciarevaliente/serverless-biomedical-nearrealtime-pipeline-aws, on el `README.md` inclou el detall complet de la implementació i les instruccions d'execució.

Atès que el sistema no utilitza dispositius físics reals, s'ha desenvolupat un simulador que reprodueix el comportament de dispositius *wearable* generant fluxos continus d'esdeveniments cap a Kinesis Data Streams. S'implementa com un *script* Python executat localment, fora de l'entorn AWS. Per a cada sensor, emet esdeveniments a la freqüència de mostreig corresponent mitjançant un mecanisme de `next_emit` que evita l'acumulació de deriva temporal, garantint que la taxa d'ingestió es mantingui estable al llarg de tota la durada de cada escenari. A més, registra localment els esdeveniments enviats, facilitant la posterior validació de la integritat de les dades.

Per a cada *batch*, la funció Lambda escriu les dades crues a S3 com un únic objecte JSON, reduint el nombre de `PUT requests` en un factor de 500 respecte a l'escriptura per event. Processa cada event individualment afegint els *timestamps* de latència, escriu els registres processats a DynamoDB i envia els events fallits a la DLQ de SQS. Al final de cada invocació publica mètriques personalitzades a CloudWatch sota el *namespace* `BiomedicalPipeline`.

7 METODOLOGIA EXPERIMENTAL

Aquest capítol descriu el marc experimental dissenyat per avaluar el comportament de l'arquitectura sota diferents nivells de càrrega, incloent les mètriques, els escenaris, la configuració i les limitacions de l'estudi.

7.1 Mètriques d'avaluació

Per a cada esdeveniment processat es registren tres *timestamps* dins del domini temporal d'AWS: `kinesis_arrival_ts` és el moment en què Kinesis rep l'event, generat pel propi servei; `processing_start_ts` és el moment en què Lambda inicia el processament individual de l'event; i `processing_end_ts` és el moment en què finalitza la validació i l'enriquiment. A partir d'aquests tres *timestamps* es calculen les mètriques recollides a la Taula 2.

Taula 2. Mètriques d'avaluació del sistema.

Mètrica	Definició	Finalitat
`kinesis_to_lambda_ms`	`= processing_start_ts − kinesis_arrival_ts`	Temps d'espera al *buffer* de Kinesis
`processing_latency_ms`	`= processing_end_ts − processing_start_ts`	Temps de processament de Lambda
`pipeline_latency_ms`	`= processing_end_ts − kinesis_arrival_ts`	Latència end-to-end dins del domini AWS. Mètrica principal SLO

S'utilitza el P95 com a indicador de referència per al SLO [12], complementat amb el P50 i el P99 per caracteritzar la distribució completa de les latències i avaluar la variabilitat entre repeticions.

7.2 Escenaris experimentals

Es defineixen tres escenaris de càrrega creixent basats en les freqüències de mostreig reals del *dataset* WESAD [13], tal com es recull a la Taula 3.

Taula 3. Escenaris experimentals de càrrega creixent.

Escenari	Taxa	Durada	SLO
Base (~8 ev/s)	1 subjecte; E4 (EDA + TEMP, 4 Hz)	5 min	P95 < 10s; pèrdua < 1%
Sostingut (~208 ev/s)	2 subjectes; E4 (BVP 64Hz, ACC 32Hz, EDA+TEMP 4Hz)	5 min	P95 < 10s; pèrdua < 1%
Pic (~4.304 ev/s)	1 subjecte; E4 (4 sensors, 4-64 Hz) + RespiBAN (6 sensors, 700Hz); ràtio 175:1	30 s	P95 < 30s; pèrdua >1%; *backlog* < 60s

Nota: els criteris de l'escenari pic són orientatius. L'objectiu primari és la quantificació de la degradació i la identificació dels factors limitants.

La comparació entre escenaris permet aïllar dos efectes independents: l'efecte del nombre de subjectes simultanis (base vs. sostingut) i l'efecte de la heterogeneïtat extrema de freqüències (sostingut vs. pic).

Cada escenari s'executa tres vegades consecutives amb un interval de 60 segons entre repeticions. Aquest interval és deliberat: en un entorn hospitalari real, pics de càrrega consecutius poden produir-se sense temps de recuperació garantit. Això permet observar l'acumulació de *backlog* entre repeticions com a resultat experimental en si mateix.

7.3 Configuració dels serveis AWS

La configuració dels serveis s'ha definit a partir dels escenaris experimentals i dels límits de cada servei AWS. La Taula 4 recull els paràmetres principals. La configuració completa i la justicació es presenten a l'Apèndix 3.

Taula 4. Configuració dels serveis principal AWS.

Component	Configuració
Kinesis Data Stream	5 shards (PROVISIONED), retenció 24h
Lambda	Python 3.11, 128 MB
Lambda ESM	`batch_size=500`, `parallelization_factor=10`
DynamoDB	PAY_PER_REQUEST, `batch_writer`

El cost total de la sessió experimental completa és de **~0,533$,** cobrint els tres escenaris i les nou repeticions; el desglossament per servei i una estimació orientativa del cost en producció es presenten a l'Apèndix 4.

7.4 Limitacions de l'estudi experimental

Cal tenir en compte diverses limitacions en la interpretació dels resultats.

Quant a les mètriques, el tram generador-Kinesis queda exclòs de la mètrica principal a causa d'un *clock skew* sistemàtic entre la màquina local i el domini AWS. Per tant, `pipeline_latency_ms` representa la latència màxima mesurable dins del domini AWS, no la latència total absoluta. A més, `processing_latency_ms` exclou les escriptures a S3 i DynamoDB, ja que `processing_end_ts` es registra abans que aquestes s'executin. Incorporar-les al model temporal requeriria una operació addicional d'escriptura a DynamoDB per event, incompatible amb el disseny del *pipeline*. Les dades de durada d'invocació de CloudWatch presentades a l'Apèndix 5 permeten estimar-ne l'impacte.

Quant a l'entorn, els experiments s'han executat en un compte AWS personal sense càrrega concurrent, cosa que pot sobreestimar el rendiment respecte a un entorn de producció compartit. A més, el simulador genera dades sintètiques que reprodueixen les freqüències del *dataset* WESAD però no la variabilitat temporal ni els artefactes propis de dispositius físics reals.

Quant a la cobertura experimental, l'avaluació cobreix una única configuració d'infraestructura final (5 *shards*, 128 MB de Lambda), sense estudiar l'impacte individual de cada paràmetre. El procés iteratiu d'optimització de la configuració dels serveis AWS i de la implementació de l'arquitectura es detalla a l'Apèndix 5. Finalment, els escenaris no inclouen configuracions intermèdies que permetrien identificar amb més precisió el punt de ruptura de la configuració actual.

8 AVALUACIÓ EXPERIMENTAL

8.1 Resultats

La Taula 5 recull la distribució de `pipeline_latency_ms` per escenari i repetició, permetent avaluar el compliment del SLO i la reproductibilitat entre *runs*. El desglossament de la latència per component del pipeline es presenta a l'Apèndix 6, i les dades d'ingestió i integritat a la Taula 6.

Taula 5. Distribució de pipeline_latency_ms per escenari i repetció.

Escenari	*Run*	P50 (ms)	P95 (ms)	P99 (ms)
Base (~8 ev/s)	1	564	1.037	1.063
	2	564	975	1.071
	3	565	1.038	1.115
Sostingut (~208 ev/s)	1	608	1.074	1.698
	2	598	1.054	1.629
	3	597	1.060	1.632
Pic (~4.304 ev/s)	1	49.911	134.184	150.538
	2	75.394	156.835	165.952
	3	97.475	197.602	205.903

Taula 6. Dades d'ingestió per escenari i repetició.

Escenari	*Run*	Taxa real (ev/s)	Events generats	Events acceptats Kinesis	Events emmatgz.	Pèrdua total (%)
Base (~8 ev/s)	1-3	8,0	7.200	7.200	7.200	0,00%
Sostingut (~208 ev/s)	1-3	208,0	187.198	187.198	187.198	0,00%
Pic (~4.304 ev/s)	1	2.782,5	124.302	113.931	112.132	9,79%
	2	3.533,5	108.535	91.654	90.054	17,03%
	3	3.884,0	120.470	104.046	100.970	16,18%

8.1.1 Escenari base

L'escenari base estableix la línia de base del rendiment del sistema en condicions de càrrega mínima. A la Taula 5, les tres repeticions presenten valors de P95 en el rang 975–1.038 ms, molt per sota del SLO de 10.000 ms. Els experiments tenen una reproductibilitat alta: mitjana de 1.016,7 ms, desviació estàndard de 36,1 ms i IC95% de [927,0; 1.106,3] ms (t-distribució, n=3). A la Taula 6, els 7.200 events generats coincideixen exactament amb els 7.200 emmagatzemats a DynamoDB (pèrdua 0,000%).

La Figura 3 mostra la CDF de `pipeline_latency_ms` per a les tres repeticions. Les tres corbes es presenten quasi solapades, indicant una distribució consistent entre *runs*.

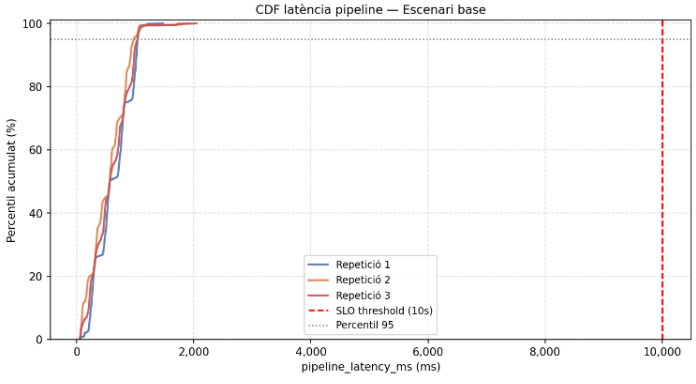

Figura 3. CDF de pipeline_latency_ms a l'escenari base (~8 ev/s).

Quant als indicadors d'AWS, l'`IteratorAge` de Kinesis, que mesura el temps que porten els events al *buffer* sense ser consumits, es manté a 0 durant tota l'execució, confirmant que el *pipeline* consumeix els events sense cap acumulació. Lambda opera amb un màxim de 3 invocacions simultànies, coherent amb la baixa taxa d'ingestió. La DLQ registra 0 missatges i la taxa d'errors és del 0%.

8.1.2 Escenari de càrrega sostinguda

L'escenari sostingut avalua l'efecte d'incrementar el nombre de subjectes i la heterogeneïtat de freqüències entre sensors. A la Taula 5, els valors de P95 (1.074, 1.054 i 1.060 ms) no presenten cap creixement monòton entre *runs*. Els experiments tenen la reproductibilitat més alta dels tres escenaris: mitjana de 1.062,7 ms, desviació estàndard de 10,3 ms i IC95% de [1.037,2; 1.088,2] ms (t-distribució, n=3). A la Taula 6, els 187.198 events generats coincideixen exactament amb els 187.198 emmagatzemats (pèrdua 0,000%).

La Figura 4 presenta les barres de P50, P95 i P99 per repetició, mostrant valors estables entre *runs* sense cap tendència creixent.

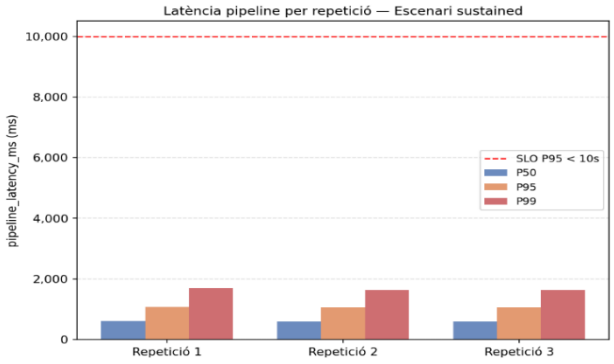

Figura 4. P50, P95 i P99 de pipeline_latency_ms per repetició de l'escenari sostingut (~208 ev/s).

Quant als indicadors d'AWS, l'`IteratorAge` de Kinesis es manté a 0 durant tota l'execució, confirmant l'absència d'acumulació al *buffer*. Lambda escala automàticament de ≤3 a ≤11 invocacions simultànies. La DLQ registra 0 missatges i la taxa d'errors és del 0%.

8.1.3 Escenari de pic

L'escenari de pic avalua la resiliència del sistema sota càrrega extrema, superant la capacitat nominal de la configuració actual. A la Taula 5, el P95 creix de forma monòtona entre repeticions (134.184 ms, 156.835 ms i 197.602 ms), amb una variabilitat elevada: mitjana de 162.873,7 ms, desviació estàndard de 32.137,4 ms i IC95% de [83.040; 242.707] ms (t-distribució, n=3). A la Taula 6, dels 353.307 events generats, 309.631 han estat acceptats per Kinesis i 303.156 emmagatzemats a DynamoDB, amb una pèrdua total del 14,195%. La DLQ registra 0 missatges i la taxa d'errors de Lambda és del 0%.

La Figura 5 mostra l'evolució temporal del P95 durant el burst de 30 s. La latència creix de forma quasi lineal des del segon 0, sense cap fase d'estabilització.

La Figura 6 presenta la CDF completa de les tres repeticions, on les tres distribucions presenten un fort desplaçament horitzontal: *Run 1* creua el P95 a ~134.184 ms, *Run 2* a ~156.835 ms i *Run 3* a ~197.602 ms.

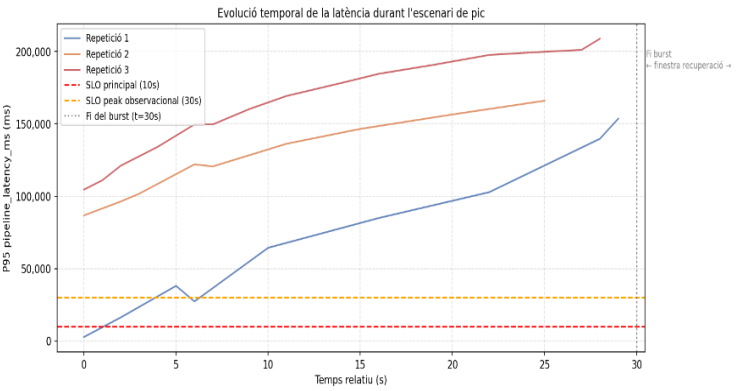

Figura 5. Evolució temporal del P95 de pipeline_latency_ms durant l'escenari de pic (~4.304 ev/s).

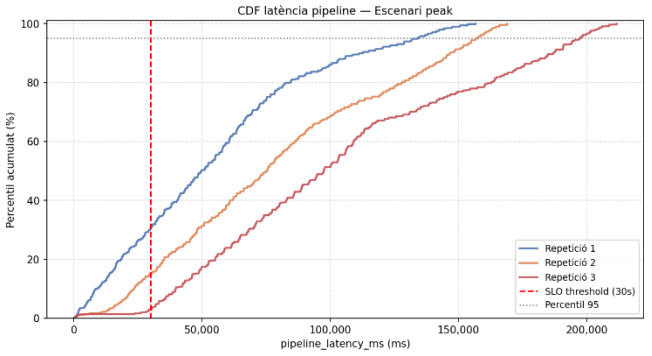

Figura 6. CDF de pipeline_latency_ms de l'escenari de pic (~4.304 ev/s).

Quant als indicadors d'AWS, l'IteratorAge de Kinesis creix de forma progressiva durant el *burst*, reflectint l'acumulació de *backlog* al *buffer*. El *backlog* postburst triga ~3,4 min a buidar-se. Lambda opera amb un màxim de 11 invocacions simultànies, una taxa d'errors del 0% i la DLQ registra 0 missatges.

8.2 Anàlisi

L'anàlisi es presenta organitzat al voltant de tres arguments transversals que sintetitzen els resultats experimentalment, independentment de l'escenari concret en què s'observen.

8.2.1 L'arquitectura serverless escala sense degradació de latència (escenari base-sostingut)

L'increment de càrrega de 26× entre l'escenari base i el sostingut (de ~8 a ~208 ev/s) comporta un augment del P95 mitjà de tan sols 46 ms, és a dir, un ~4,5% de la latència base. Atesa la variabilitat intrínseca de cada escenari (std de 36,1 ms al base i 10,3 ms al sostingut) aquest increment no és estadísticament significatiu. La comparació visual entre la Figura 3 i la Figura 4 confirma aquesta estabilitat: les corbes CDF del base estan solapades (rang 975–1.038 ms) i les barres del sostingut no presenten cap tendència creixent entre repeticions (P95 rang 1.054–1.074 ms).

Des del punt de vista operatiu, l'escalat automàtic de Lambda (de ≤3 invocacions simultànies al base a ≤11 al sostingut) absorbeix el volum addicional d'events sense cap intervenció operativa ni degradació perceptible del temps de resposta. L'`IteratorAge` de Kinesis es manté a 0 durant

tot el sostingut, confirmant que el *pipeline* consumeix els events a mesura que arriben sense cap acumulació al *buffer*. En conjunt, l'envolupant de funcionament correcte de l'arquitectura s'estén, almenys, fins a 208 ev/s en condicions de càrrega sostinguda.

8.2.2 L'escenari pic revela el punt de ruptura de la configuració actual

L'escenari de pic posa de manifest dos mecanismes de degradació que operen en cascada i que la configuració actual no és capaç d'absorbir.

El primer mecanisme és el *throttling* a la capa d'ingestió de Kinesis. Els 5 *shards* configurats proporcionen una capacitat teòrica de 5.000 records/s. Tanmateix, la distribució no uniforme dels events per *partition key* i les variacions instantànies del generador provoquen que alguns *shards* superin el límit individual de 1.000 records/s, generant errors `put_records`. Els events rebutjats mai no arriben a Lambda: això explica per què la DLQ registra 0 missatges i la taxa d'errors de Lambda és del 0% malgrat una pèrdua global del 14,195%. La pèrdua es descompon en un 12,6% per *throttling* a la capa d'ingestió de Kinesis i un 2,09% addicional atribuïble a events que quedaven al *buffer* o en processament en el moment d'execució de l'anàlisi.

El segon mecanisme és el *backlog* hereditari entre repeticions. La Figura 5 mostra que el P95 creix de forma quasi lineal des del segon 0 sense assolir mai un règim estacionari: el sistema ingesta events més ràpidament del que els pot processar, generant una cua creixent al *buffer* de Kinesis. L'interval de 60 s entre repeticions és insuficient per buidar aquest *backlog* (~3,4 min de recuperació), de manera que *Run 2* i *Run 3* comencen amb un P95 inicial ja elevat. La Figura 6 confirma aquest efecte: les tres distribucions presenten un fort desplaçament horitzontal, a diferència del solapament observat a la Figura 3 (base). El P95 creix de 134.184 ms (*Run 1*) a 197.602 ms (*Run 3*), una signatura inequívoca d'acumulació hereditària.

En síntesi, la configuració actual (5 *shards* i `parallelization_factor=10`) no és capaç d'absorbir la càrrega de l'escenari de pic ni de recuperar-se en el temps requerit. Les causes no resideixen en la lògica de processament de Lambda, que opera sense errors, sinó en la capa de consum de Kinesis, tal com s'argumenta a la secció 8.2.3.

8.2.3 El coll d'ampolla és el buffer de Kinesis, no Lambda

Una observació transversal als tres escenaris revela el component que concentra la totalitat del retard mesurat: el *buffer* de Kinesis. En tots els escenaris i percentils analitzats, `processing_latency_ms` és 0 ms, mentre que `kinesis_to_lambda_ms` coincideix pràcticament amb `pipeline_latency_ms` (vegeu Apèndix 6). Això indica que la totalitat del retard mesurat s'acumula al *buffer* de Kinesis, durant l'espera del *batch* a ser consumit per Lambda, i no durant el processament intern.

Cal recordar que `processing_latency_ms` exclou les escriptures a S3 i DynamoDB; tanmateix, la durada total d'invocació de Lambda confirmada per CloudWatch (~277 ms al sostingut i ~2.594 ms al pic) és molt inferior al temps d'espera al *buffer* de Kinesis en ambdós escenaris (P50 de 608 ms i 75.394 ms respectivament), confirmant quantitativament que Lambda no és el coll d'ampolla fins i tot considerant les escriptures (vegeu Figura 7, Apèndix 5).

Finalment, cal destacar una observació rellevant per al *capacity planning*: al pic s'han observat ≤11 invocacions concurrents malgrat el màxim configurat de 50 (5 *shards* × `parallelization_factor=10`). La *partition key* actual (`subject_id#sensor_type`) genera una distribució desigual quan coexisteixen sensors amb freqüències molt heterogènies: els shards assignats als sensors d'alta freqüència (RespiBAN a 700 Hz) reben fins a 175 vegades més events que els dels sensors de baixa freqüència (Empatica EDA/TEMP a 4 Hz), deixant alguns *shards* infrautilitzats. Aquesta observació indica que el *capacity planning* de Kinesis ha de tenir en compte no només el volum total d'events sinó també la seva distribució entre *partition keys*, especialment en escenaris amb dispositius de freqüències molt heterogènies.

8.2.4 Avaluació dels requisits i trade-offs

La Taula 7 presenta l'avaluació dels requisits del sistema per escenari.

Taula 7. Avaluació dels requisits del sistema per escenari.

Requisit	Base (~8 ev/s)	Sostingut (~208 ev/s)	Pic (~4.304 ev/s)
Latència P95 < 10.000 ms	✓ (1.037 ms)	✓ (1.074 ms)	✗ (197.602 ms)
Pèrdua < 1%	✓ (0,000%)	✓ (0,000%)	✗ (14,195%)
Throughput real ≥ teòric	✓ (100%)	✓ (100%)	✗ (78,3%)
Tolerància a errors (DLQ = 0)	✓	✓	✓
Escalabilitat Lambda	✓ (≤3 inv.)	✓ (≤11 inv.)	✓ (≤11 inv.)
Recuperació *backlog* < 60 s	✓ (0 *backlog*)	✓ (0 *backlog*)	✗ (~3,4 min)

Els escenaris base i sostingut compleixen la totalitat dels requisits. En canvi, l'escenari de pic no compleix els requisits de latència, integritat ni *throughput*, si bé la tolerància a errors i l'escalabilitat de Lambda es mantenen satisfets fins i tot sota condicions extremes. Els fallaments al pic no són fallades de processament, sinó conseqüència del subdimensionament de la capa d'ingestió de Kinesis.

Pel que fa als *trade-offs* arquitectònics del capítol 5, els resultats els validen en la seva majoria: el model quasi en temps real presenta un marge superior a 9× sobre el SLO; el model *serverless* absorbeix un increment de 26× de càrrega sense intervenció operativa; la DLQ no s'ha activat en cap escenari; i DynamoDB ha respost sense degradació en tots els casos. El *trade-off single*-regió vs multi-regió no és avaluable en el context del prototip. Finalment, els experiments han revelat un *trade-off* no identificat inicialment: la *partition key* `subject_id#sensor_type` garanteix l'ordre temporal per sensor però genera una distribució no uniforme de càrrega entre *shards* en presència de freqüències molt heterogènies, un aspecte a considerar en futures extensions del sistema.

CONCLUSIONS

Aquest treball ha demostrat experimentalment la viabilitat del model *serverless* per al processament quasi en temps real de dades biomèdiques en àrees hospitalàries no crítiques. Els resultats obtinguts confirmen que l'arquitectura proposada compleix els objectius plantejats i proporcionen un diagnosi tècnic fonamentat en evidència experimental sobre el seu comportament i els seus límits sota càrrega creixent. La Taula 8 recull el grau d'assoliment de cadascun dels objectius específics plantejats.

Taula 8. Grau d'assoliment dels objectius específics.

Objectiu	Resultat clau	Estat
Identificar requisits	5 categories de requisits no funcionals amb SLOs quantificables derivats del context clínic	Assolit
Dissenyar arquitectura	Arquitectura distribuïda completament *serverless* sobre AWS, amb decisions justificades i alternatives avaluades	Assolit
Implementar el prototip	Prototip funcional desplegat sobre AWS via IaC amb Terraform; repositori públic documentat	Assolit
Definir metodologia experimental	3 escenaris de càrrega creixent, 3 repeticions cadascun, mètriques quantitatives de latència, integritat i *throughput*	Assolit
Analitzar experimentalment	*Buffer* de Kinesis identificat com a coll d'ampolla; distribució de *partition keys* com a variable crítica per al *capacity planning*	Assolit

Els resultats experimentals validen l'arquitectura per al cas d'ús considerat. Els escenaris base i sostingut confirmen que el model *serverless* absorbeix increments de càrrega de fins a 26× sense degradació perceptible de la latència ni pèrdua d'events. Lambda escala automàticament de 3 a 11 invocacions sense intervenció operativa, i l'IteratorAge de Kinesis es manté a zero en ambdós escenaris. L'escenari de pic, en canvi, revela el subdimensionament de la capa d'ingestió com a factor limitant: el *throttling* de Kinesis provoca una pèrdua del 14,195% i un *backlog* que triga ~3,4 minuts a buidar-se.

Una de les aportacions més rellevants d'aquest treball és el diagnosi experimental del coll d'ampolla: la totalitat del retard mesurat s'acumula al *buffer* de Kinesis i no al processament intern de Lambda. Aquesta distinció és crítica per orientar correctament les optimitzacions futures, ja que implica que augmentar la memòria de Lambda o el paral·lelisme intern no resoldria el problema.

Els experiments han posat de manifest, a més, que la coexistència de sensors amb freqüències molt heterogènies (de 4 Hz a 700 Hz) afecta la distribució de càrrega entre *shards* i limita el paral·lelisme real de Lambda. És per aquest motiu que el *capacity planning* ha de tenir en compte no només el volum total d'events sinó també la seva distribució entre *partition keys*.

Des d'una perspectiva econòmica, el cost operatiu és consistent amb la sostenibilitat del paradigma *serverless* adoptat: la sessió experimental ha tingut un cost de ~0,533 $. En producció, el cost dominant seria Amazon Kinesis amb un cost fix de ~2,04 $/dia en mode PROVISIONED, al qual caldria afegir el cost variable de DynamoDB i S3 en funció del volum de dades del dispositiu clínic adoptat.

Les principals limitacions de l'estudi s'han documentat i quantificat al llarg de la memòria: el *clock skew* entre el simulador i el domini AWS, l'ús de dades sintètiques i l'avaluació d'una única configuració d'infraestructura.

A curt termini, les optimitzacions prioritàries actuen sobre la capa de consum de Kinesis: incrementar el `shard_count` de 5 a 10, augmentar la memòria de Lambda per aprofitar millor el `parallelization_factor`, i revisar el disseny de la *partition key* per equilibrar la distribució de càrrega en presència de freqüències heterogènies. Aquestes millores haurien de validar-se mitjançant l'exploració sistemàtica de configuracions intermèdies d'infraestructura, per identificar amb precisió el punt de ruptura de la configuració actual. A mig termini, el sistema podria estendre's amb validació de rangs fisiològics al *pipeline* Lambda, reprocessament automàtic de la DLQ i avaluació d'Apache Flink per a desplegs amb centenars de pacients simultanis. A llarg termini, la projecció cap a producció clínica real requereix abordar el compliment de normatives de privacitat (GDPR), la sincronització NTP als dispositius *wearable* i el desplegament multi-regió per a alta disponibilitat (assolible en minuts gràcies a la infraestructura com a codi amb Terraform).

Aquest treball posa de manifest que en sistemes distribuïts el diagnosi experimental és tan important com el disseny inicial: detectar col·lisions silencioses a DynamoDB, identificar que el coll d'ampolla era el *buffer* de Kinesis i no Lambda, o descobrir que la heterogeneïtat de freqüències entre sensors afecta la distribució de càrrega són aprenentatges que només emergen executant experiments reals. D'altra banda, treballar amb dades biomèdiques, encara que sintètiques, m'ha fet reflexionar sobre la dimensió ètica de l'enginyeria de dades: decisions aparentment tècniques com el disseny de les claus de partició, la traçabilitat dels events o la gestió dels errors tenen implicacions directes sobre la integritat de la informació clínica i, en última instància, sobre la seguretat del pacient.

En conjunt, aquest treball contribueix a fonamentar experimentalment la viabilitat del model *serverless* per a aplicacions de monitorització biomèdica contínua en àrees no crítiques, un àmbit on l'evidència empírica sobre arquitectures d'aquest tipus és encara limitada. El paradigma *serverless* assumeix compromisos inherents, com ara el menor control sobre el paral·lelisme de baix nivell, la dependència del proveïdor *cloud* i *cold starts* ocasionals. A canvi, ofereix simplicitat operativa, escalabilitat automàtica i cost reduït, tal com confirmen els escenaris base i sostingut.

AGRAÏMENTS

Vull agrair especialment al meu tutor, Esteve Alquézar, la seva dedicació i orientació al llarg de tot el projecte. Les seves aportacions tècniques i el seu seguiment constant han estat fonamentals per a la qualitat d'aquest treball. Vull agrair també a la meva família el suport incondicional durant tot el procés.

BIBLIOGRAFIA

[1] C. Downey, W. Tahir, R. Randell, J. Brown, and D. Jayne, "Strengths and limitations of early warning scores: A systematic review and narrative synthesis," *Int. J. Nursing Studies*, vol. 76, pp. 106–119, 2017.

[2] G. J. Williams et al., "*Wearable* technology and the cardiovascular system: The future of patient assessment," *The Lancet Digital Health*, 2023.

[3] A. Keogh et al., "Six-month pilot testing of a digital health tool to support effective self-care in people with heart failure: Mixed methods study," *JMIR Formative Research*, vol. 8, p. e52442, 2023.

[4] H. Xie, L. Yang, B. Jiang, Z. Huang, and Y. Lin, "State-of-the-art *wearable* sensors for cardiovascular health: A review," *npj Cardiovascular Health*, vol. 2, no. 53, 2025.

[5] D. S. Temple et al., "Modular open-core system for collection and near real-time processing of high-resolution data from *wearable* sensors," *Applied System Innovation*, vol. 6, no. 5, p. 79, 2023.

[6] A. Bahmani et al., "A scalable, secure, and interoperable platform for deep data-driven health management," *Nature Communications*, vol. 12, p. 5757, 2021.

[7] A. Orro et al., "A cloud infrastructure for health monitoring in emergency response scenarios," *Sensors*, vol. 24, p. 6992, 2024.

[8] M. Shafiei, H. Khazaei, and M. Litoiu, "Serverless computing: A survey of opportunities, challenges, and applications," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, 2022.

[9] H. B. Hassan, S. A. Barakat, and Q. I. Sarhan, "Survey on serverless computing," *J. Cloud Computing*, vol. 10, no. 1, p. 39, 2021.

[10] E. Shojaee Rad, M. Babaei, M. Esmaeili, and H. Khazaei, "Serverless data processing and analytics: A systematic review," *J. Systems and Software*, vol. 217, p. 112177, 2024.

[11] G. Wilnai, "Building near-real-time data pipelines," *Datagize*, Feb. 2025. [Online]. Available: https://datagize.us/building-near-real-time-data-pipelines/

[12] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, *Site Reliability Engineering: How Google Runs Production Systems*. Sebastopol, CA: O'Reilly Media, 2016.

[13] UCI Machine Learning Repository, "WESAD: *Wearable* Stress and Affect Detection," 2018. [Online]. Available: https://archive.ics.uci.edu/dataset/465/wesad+*wearable*+stress+and+affect+detection

[14] Amazon Web Services, "Amazon Kinesis Data Streams Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/streams/latest/dev/introduction.html

[15] Amazon Web Services, "AWS Lambda Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html

[16] Amazon Web Services, "Amazon Managed Service for Apache Flink," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/managed-flink/latest/java/whatis.html

[17] Amazon Web Services, "Amazon S3 User Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html

[18] Amazon Web Services, "Amazon DynamoDB Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[19] Amazon Web Services, "Amazon Timestream Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/timestream/latest/developerguide/what-is-timestream.html

[20] Amazon Web Services, "Amazon CloudWatch User Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html

[21] Amazon Web Services, "AWS X-Ray Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html

[22] Amazon Web Services, "Amazon OpenSearch Service Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/opensearch-service/latest/developerguide/whatis.html

[23] Amazon Web Services, "AWS Well-Architected Framework," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

[24] Australasian Health Facility Guidelines, "Part B: Health Facility Briefing & Design — Inpatient Unit General," 2012. [Online]. Available: https://www.healthfacilityguidelines.com/ViewPDF/ViewIndexPDF/iHFG_part_b_inpatient_unit_general

[25] Amazon Web Services, "Amazon SQS Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html

[26] Amazon Web Services, "Amazon Data Firehose Developer Guide," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/firehose/latest/dev/what-is-this-service.html

[27] Apache Software Foundation, "Apache Kafka Documentation," 2024. [Online]. Available: https://kafka.apache.org/

[28] Amazon Web Services, "AWS Glue Developer Guide — Streaming ETL," AWS Documentation. [Online]. Available: https://docs.aws.amazon.com/glue/latest/dg/add-job-streaming.html

[29] CockroachLabs, "Idempotency and ordering in event-driven systems," CockroachLabs Blog. [Online]. Available: https://www.cockroachlabs.com/blog/idempotency-and-ordering-in-event-driven-systems/

DECLARACIÓ D'ÚS D'EINES D'IA I RECURSOS

Per a la revisió lingüística i ortogràfica del text: Scribbr. Per a la cerca i gestió de referències bibliogràfiques: Anara. Les eines d'intel·ligència artificial generativa Claude (Anthropic) i ChatGPT (OpenAI) s'han utilitzat com a suport en la revisió i reformulació de fragments de redacció, la detecció de repeticions i la millora de l'estructura expositiva. En cap cas s'han utilitzat per generar contingut tècnic, resultats experimentals ni conclusions del treball.

APÈNDIX

Apèndix 1. *Dataset* i estructura de dades

El *dataset* WESAD [13] conté dades multimodals recollides mitjançant dos dispositius *wearable*: l'Empatica E4, orientat a la monitorització lleugera, i el RespiBAN, orientat a la captura d'alta freqüència. La Taula 9 recull les característiques dels sensors utilitzats com a referència per al disseny del simulador.

Taula 9. Característiques dels sensors del dataset WESAD.

Dispositiu	Sensor	Senyal fisiològica	Freqüència de mostreig	Tipus dada	Unitat	Eixos
RespiBAN	ECG	Activitat cardíaca	700 Hz	Float	mV	1
	EDA	Activitat electrodermal	700 Hz	Float	µS	1
	EMG	Activitat muscular	700 Hz	Float	mV	1
	TEMP	Temperatura corporal	700 Hz	Float	ºC	1
	RESP	Respiració	700 Hz	Float	%	1
	ACC	Acceleració	700 Hz	Float[3]	g	3 (X, Y, Z)
Empatica E4	BVP	Volum de puls (PPG)	64 Hz	Float	-	1
	ACC	Acceleració	32 Hz	Float[3]	1/64 g	3 (X,Y, Z)
	EDA	Activitat electrodermal	4 Hz	Float	µS	1
	TEMP	Temperatura corporal	4 Hz	Float	ºC	1

La Taula 10 recull els camps del registre *raw*, que representa l'esdeveniment tal com és generat pel simulador i ingestat al sistema. La Taula 11 recull els camps addicionals del registre processat, que incorpora la informació temporal i les mètriques de latència generades durant el processament.

Taula 10. Camps del registre raw.

Camp	Tipus	Descripció
`event_id`	String	Identificador únic universal de l'esdeveniment. Garanteix la unicitat i la traçabilitat independentment de la resolució temporal.
`subject_id`	String	Identificador del subjecte.
`device_id`	String	Identificador del dispositiu.
`sensor_type`	String	Tipus de sensor.
`sampling_rate_hz`	Integer	Freqüència de mostreig del sensor.
`sensor_timestamp`	Float	*Timestamp* de generació de la mostra (Unix UTC).
`value`	Float / Object	Valor de la mesura. Escalar per a sensors uniaxials i objecte per a sensors triaxials.
`unit`	String	Unitat de mesura.
`scenario`	String	Escenari experimental *(base, sostingut, pic*).
`schema_version`	String	Versió de l'esquema de dades.

Taula 11. Camps addicionals del registre processat.

Camp	Tipus	Descripció
`kinesis_arrival_ts`	Float	*Timestamp* de recepció per Kinesis (Unix UTC). T0 del model temporal
`ingest_timestamp`	Float	*Timestamp* d'inici de processament per Lambda (Unix UTC)
`processed_ timestamp`	Float	*Timestamp* de finalització del processament (Unix UTC)
`kinesis_to_lambda_ms`	Float	Temps d'espera al *buffer* de Kinesis (ms)
`processing_latency_ms`	Float	Temps de processament intern de Lambda (ms)
`pipeline_latency_ms`	Float	Latència *end-to-end* dins del domini AWS (ms). Mètrica principal del SLO

En la Taula 12, es mostren exemples complets dels dos tipus de registres: *raw* i processat respectivament.

Taula 12. Exemples de registres i processat.

Registre *raw*	Registre processat

`{`
`  "event_id": "a3f2c1d4-8b7e-4a9f-b2c3-1d4e5f6a7b8c",`
`  "subject_id": "S1",`
`  "device_id": "EmpaticaE4",`
`  "sensor_type": "EDA",`
`  "sampling_rate_hz": 4,`
`  "sensor_timestamp": 1710000.123,`
`  "value": 0.85,`
`  "unit": "µS",`
`  "scenario": "base",`
	`{`
	`  "event_id": "a3f2c1d4-8b7e-4a9f-b2c3-1d4e5f6a7b8c",`
	`  "subject_id": "S1",`
	`  "device_id": "EmpaticaE4",`
	`  "sensor_type": "EDA",`
	`  "sampling_rate_hz": 4,`
	`  "sensor_timestamp": 1710000.123,`
	`  "kinesis_arrival_ts": 1710000.587,`
	`  "ingest_timestamp": 1710001.201,`
	`  "processed_timestamp": 1710001.203,`

```
  "schema_version": "1.0"
}
```

```
  "value": 0.85,
  "unit": "µS",
  "scenario": "base",
  "schema_version": "1.0",
  "kinesis_to_lambda_ms": 614.0,
  "processing_latency_ms": 2.0,
  "pipeline_latency_ms": 616.0
}
```

Apèndix 2. Decisions tècniques i alternatives considerades

A continuació es detallen les decisions tècniques de disseny que, per raons d'extensió, no s'han exposat al cos de la memòria:

- **Disseny de claus de partició a DynamoDB:** usar `subject_id` com a única partition key concentraria totes les escriptures d'un subjecte en una sola partició, potencialment superant el límit de 1.000 WCU/s per partició en escenaris d'alta freqüència. Amb la concatenació `subject_id#sensor_type`, la càrrega màxima per partició en l'escenari de pic (RespiBAN a 700 Hz per sensor) és de ~700 ev/s, dins del límit de DynamoDB.
- **UUID com a *sort key*:** en escenaris amb múltiples repeticions o freqüències de mostreig elevades, dos events del mateix sensor poden generar *timestamps* idèntics si la resolució del rellotge no és suficient. L'ús d'un UUID garanteix la traçabilitat i la idempotència de les operacions d'escriptura al llarg de tot el *pipeline* [29].
- **CloudTrail en producció:** en el context del prototip s'activen únicament els `management events`, gratuïts per al primer *trail*. En un entorn de producció amb requisits d'auditoria estrictes, s'activarien els tres tipus i el *bucket* de *logs* es configuraria en mode `Vault` amb funcionalitat WORM (Write Once, Read Many), que impedeix la modificació o eliminació dels logs fins i tot per part de l'usuari root.

La Taula 13 recull els serveis d'AWS avaluats i descartats per a cada component de l'arquitectura.

Taula 13. Serveis AWS alternatius considerats i descartats, incloent el motiu principal de la seva exclusió.

Component	Servei	Motiu de descart
Ingestió	SQS FIFO	Absència de reprocessament natiu i escalabilitat insuficient per al pic de càrrega (RespiBAN a 4.200 ev/s per subjecte) [25]
	Amazon Firehose	Latència mínima de 60 s incompatible amb el processament quasi en temps real [26]
	Amazon MSK (Kafka)	Absència de Free Tier i complexitat operativa desproporcionada per a un prototip; tècnicament idoni per a producció amb centenars de pacients simultanis [27]
Processament	Apache Flink	Complexitat operativa alta i cost base sense Free Tier; tècnicament idoni per a producció amb alta càrrega
	AWS Glue	Orientació ETL i latència de segons a minuts inadequada per a operacions lleugeres per event [28]
Persistència	S3 + Athena	Latència i cost per escaneig en consultes freqüents; absència d'accés directe per clau
	Amazon Timestream	Absència de Free Tier significatiu; tècnicament superior per a sèries temporals biomèdiques [19]
Monitorització	AWS X-Ray	Absència de sistema d'alarmes natiu i incapacitat per comparar recomptes entre blocs
	Amazon OpenSearch	Cost de clúster actiu per node/hora i complexitat excessiva per a un prototip [22]

Apèndix 3. Configuració dels serveis AWS

La Taula 14 recull la configuració completa dels serveis AWS del prototip.

Taula 14. Configuració dels serveis AWS del prototip.

Component	Paràmetre	Configuració
Kinesis Data Streams	*Shards*	5 (PROVISIONED)
	Retenció	24 hores
Lambda	*Runtime*	Python 3.11, 128MB
Lambda Event Source Mapping	`Batch_size`	500
	`Parallelization_factor`	10
	`Bisect_batch_on_function_error`	activat
S3	Clau objectes *raw*	`raw/{timestamp}_{request_id}.json`
DynamoDB	Mode de facturació	PAY_PER_REQUEST
	Escriptura	`batch_writer`
CloudWatch	Freqüència	60 segons
	Retenció	15 mesos
	Namespace	`BiomedicalPipeline`
AWS Budgets	Alertes	80% i 100% pressupost mensual de 10$

El prototip es desplega a la regió `eu-west-1` (Europe, Ireland), prioritzant la disponibilitat de serveis i la maduresa de la infraestructura AWS. Els rols IAM segueixen el principi de mínim privilegi: cada component del sistema disposa d'un rol específic amb els permisos estrictament necessaris per a la seva funció, sense accés a recursos que no li corresponen. Pel que fa a Kinesis, els 5 *shards* en mode PROVISIONED proporcionen una capacitat d'ingestió de fins a 5.000 records/s i la retenció de 24 hores garanteix el reprocessament en cas de fallada. El `batch_size` de 500 correspon al límit màxim de l'API de Kinesis per crida `put_records`; el `parallelization_factor` de 10 permet fins a 50 invocacions concurrents (5 *shards* × 10); i `bisect_batch_on_function_error` aïlla events defectuosos dividint el *batch* per la meitat. Quant a S3, la clau dels objectes inclou el `request_id` de la invocació per garantir la unicitat sota alt paral·lelisme. Quant a DynamoDB, el mode PAY_PER_REQUEST s'adapta a la càrrega variable i `batch_writer` agrupa les 500 escriptures en ~20 crides *batch*. Quant a CloudWatch, les mètriques custom es publiquen cada 60 segons sota el *namespace* `BiomedicalPipeline`, amb una retenció de 15 mesos. Finalment, AWS Budgets s'ha configurat amb alertes al 80 i 100% d'un pressupost de 10$/mes per garantir el control preventiu del cost operatiu.

Apèndix 4. Estimació de costos de la sessió experimental

L'estimació de la sessió experimental s'ha elaborat manualment a partir de la llista de preus pública d'AWS per a la regió `eu-west-1` (juny 2026), ja que AWS Pricing Calculator assumeix càrrega constant mensual i no reflecteix la realitat d'una sessió ~2h amb càrregues molt variables. Els volums utilitzats corresponen als events teòrics, atès que la facturació de Kinesis i DynamoDB es basa en les operacions intentades, independentment del *throttling*. El desglossament per servei es presenta a la Taula 15.

Taula 15.Estimació de costos de la sessió experimental (eu-west-1, juny 2026).

Servei	Unitats consumides	Càlcul	Cost estimat
AWS Kinesis Data Streams	5 *shard*-hora + 581.760 events	(5 shard × 1h × $0,017/shard-hora) + (581.760 PUT Payload Units / 1.000.000 PUT Payload Units × $0,0165)	$0,095
AWS Lambda	581.760 events / 500 *batch* = ~1.164 invocacions; ~1.164 invocacions x 1s de durada (conservador) * (128MB de Lambda / 1.024MB/GB) = 0,125 GB	Cobert pel Free Tier: 1M invocacions/mes + 400.000 GB-s/mes	$0,00
Amazon DynamoDB	581.760 WRU + 58.176 RRU	(581.760 WRU / 1.000.000 WRU × $0,705) + (58.176 RRU / 1.000.000 RRU × $0,1415)	$0,418
Amazon S3	581.760 events * 1.024 bytes + 1.164 `PUT requests`	(0,596 GB × $0,023/GB) + (1.164 `PUT requests` / 1.000 `PUT requests` × $0,005)	$ 0,020
Amazon CloudWatch	4 mètriques	Cobert pel Free Tier: primeres 10 mètriques/mes	$0,00
Total sessió experimental			**~$0,533**

A títol orientatiu, el cost dominant d'un desplegament continu en producció és Amazon Kinesis, que factura per capacitat aprovisionada independentment del volum. En mode PROVISIONED (5 *shards*), el cost fix seria de ~2,04 $/dia (5 *shards* × 24h × 0,017 $/*shard*-hora); en mode ON-DEMAND, la facturació es realitza per unitat processada, resultant més econòmic per a càrregues variables o inferiors a la capacitat nominal. Lambda i CloudWatch quedarien coberts pel Free Tier fins a escales molt superiors a les avaluades. El cost variable de DynamoDB i S3 dependrà del volum de dades del dispositiu clínic adoptat; extrapolant la sessió experimental, el cost total s'estimaria en el rang **de ~2–7 $/dia** per a una planta hospitalària d'àrees no crítiques.

Apèndix 5. Procés iteratiu d'optimització

Durant l'execució dels experiments previs a l'avaluació definitiva es van detectar quatre limitacions que van conduir a optimitzacions de la configuració del sistema.

En primer lloc, el *shard* únic inicial (1 shard, límit 1.000 writes/s) saturava completament el *stream* a l'escenari de pic (~4.304 ev/s), generant centenars de `'warnings records failed in put_records'`. La solució va ser incrementar progressivament el `shard_count` fins a 5. Aquesta configuració proporcionava una capacitat nominal de 5.000 writes/s. per sobre del màxim teòric dels escenaris experimentals (~4.303 ev/s).

En segon lloc, es va detectar una pèrdua del 3,944% de registres a l'escenari base malgrat DLQ buida i 0 errors de Lambda. La *sort key* original de DynamoDB era `sensor_timestamp`, el *timestamp* de generació de la mostra des del simulador, que semblava suficient per identificar de forma única cada registre. Tanmateix, l'anàlisi de la distribució per `run_id` i `sensor_type` va revelar una pèrdua simètrica entre EDA i TEMP i creixent entre repeticions (26; 49; 67 registres

perduts en cada *run*), consistent amb sobrescriptures silencioses: en repeticions consecutives, el simulador generava *timestamps* continus amb `time.time()` que podien coincidir amb els de la repetició anterior, sobreescrivint registres existents silenciosament. La solució va ser substituir `sensor_timestamp` per un `event_id` UUID com a *sort key*, garantint la unicitat absoluta de cada registre independentment de la resolució temporal.

En tercer lloc, es va detectar un *backlog* creixent a l'escenari sostingut sense una causa òbvia: la DLQ estava buida, no hi havia *throttling* de Kinesis i la configuració de *shards* era suficient. Atès que `processing_latency_ms` no inclou les escriptures a DynamoDB, la causa no era directament observable a les mètriques. L'anàlisi de les mètriques de CloudWatch va revelar una durada mitjana d'invocació de 7.526 ms, consistent amb 500 crides síncrones `put_item` de ~15 ms cadascuna. La substitució per `batch_write`, que agrupa les 500 escriptures en 20 crides *batch*, va reduir la durada a ~200 ms i va incrementar el *throughput* en un factor de ~37×, estabilitzant el P95 en el rang exposat en la secció 8.

La Figura 7, en la gràfica esquerra, mostra les mètriques de CloudWatch de Lambda amb `put_item` individual: durada mitjana de 7.526 ms i màxim de 19.546 ms, consistent amb 500 crides síncrones de ~15 ms cadascuna. En la gràfica de la dreta es mostra les mètriques amb `batch_write`: durada mitjana de ~277 ms, una reducció de ~27× respecte a la configuració anterior.

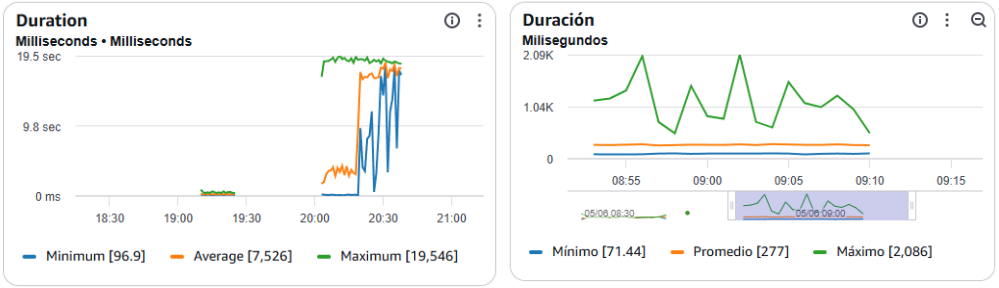

Figura 7. Mètriques de CloudWatch de Lambda. Esquerra: configuració put_item individual (durada mitjana 7.526 ms, màxim 19.546 ms). Dreta: configuració batch_write (durada mitjana 277 ms, màxim 2.086 ms).

Tot i no disposar d'una mètrica desagregada per S3 i DynamoDB per separat, la durada total d'invocació reflectida a CloudWatch inclou ambdues escriptures. Amb `batch_write`, la durada total de ~277 ms al sostingut i ~2.594 ms al pic confirma que les escriptures no representen un coll d'ampolla ja que, tal com s'exposa en la secció 8, el temps d'espera al *buffer* de Kinesis és molt superior en tots els escenaris.

Finalment, es va detectar que el 46% de registres presentaven `pipeline_latency_ms` negatius (~−74 ms) a causa de la desincronització entre el rellotge local (`sensor_timestamp`) i el rellotge AWS. La substitució de `time.time()` per `approximateArrivalTimestamp` de Kinesis no va funcionar per resolució insuficient (1 segon). La decisió final, va ser usar `kinesis_arrival_ts` com a T0 del model temporal, excloent el tram generador - Kinesis de la mètrica principal, tal com es documenta a la secció 7.1 i 7.5.

Apèndix 6. Desglossament de latències per component

Taula 16. Desglossament de kinesis_to_lambda_ms i processing_latency_ms per escenari i repetició.

Escenari	run_id	kinesis_to_lambda_ms_p50	kinesis_to_lambda_ms_p95	kinesis_to_lambda_ms_p99	kinesis_to_lambda_ms_mean	processing_latency_ms_p50	processing_latency_ms_p95	processing_latency_ms_p99	processing_latency_ms_mean
Base	1	564	1037	1063	624	0	0	0	0
	2	564	975	1071	537	0	0	0	0
	3	565	1038	1115	584	0	0	0	0
Sostingut	1	608	1074	1698	621	0	0	0	0
	2	597	1054	1629	608	0	0	0	0
	3	597	1060	1632	607	0	0	0	0
Pic	1	49911	134184	150538	54888	0	0	0	0
	2	75394	156835	165952	79679	0	0	0	0
	3	97475	197602	205903	104263	0	0	0	0