

Clúster de Edge Computing para Mantenimiento Predictivo con IA Local

Diseño, montaje y validación de un clúster K3s de tres nodos para inferencia de IA local autónoma

Integrante	Carné	Correo institucional
Kleiver Jedidias García Guarcas	0900-21-1971	kgarciag26@miumg.edu.gt
Ricardo Ismael Pérez Ajanel	0900-23-22081	rpereza22@miumg.edu.gt
Allan David Alessandro Racancoj Lopez	0900-22-3187	aracancojl@miumg.edu.gt
Pablo Javier Sandoval	0900-21-4333	psandoval@miumg.edu.gt

Datos del trabajo	
Universidad	Universidad Mariano Gálvez de Guatemala
Curso	Arquitectura de Computadoras 2
Docente	Alex García
Fecha	16 de octubre de 2026

Contenido

Resumen ejecutivo	3
1. Introducción	4
1.1 Contexto	4
1.2 Planteamiento del problema	4
1.3 Objetivos.....	4
1.4 Alcance y restricciones	4
1.5 Organización del documento.....	5
2. Marco teórico.....	5
2.1 Computación en el borde	5
2.2 Mantenimiento predictivo y análisis de vibraciones	5
2.3 Clasificación supervisada y métricas.....	5
2.4 Kubernetes y K3s	6
2.5 Alta disponibilidad y balanceo de carga	6
3. Diseño de la solución	7
3.1 Requisitos	7
3.2 Arquitectura lógica y flujo de datos.....	7
3.3 Selección de hardware.....	7
3.4 Plan de red.....	8
3.5 Diseño del servicio de inferencia	8
3.6 Diseño del simulador de datos	9
3.7 Diseño de la tolerancia a fallos	9
3.8 Costos del proyecto	10
4. Implementación.....	11
4.1 Preparación de los nodos	11
4.2 Entrenamiento del modelo.....	11
4.3 Instalación de K3s	11
4.4 Contenedorización y distribución de la imagen.....	12
4.5 Manifiestos de Kubernetes.....	12
4.6 Monitoreo.....	13
4.7 Seguridad y operación	14
5. Pruebas y resultados	15
5.1 Plan de pruebas	15
5.2 Evaluación del modelo (P01)	15
5.3 Clasificación en vivo (P03)	16
5.4 Prueba de caída de un trabajador (P04)	17
5.5 Reemplazo del pod, operación sin Internet y reincorporación (P05, P06, P09)	20
5.6 Serie comparativa A/B (P07).....	20
5.7 Consumo y recursos (P08)	21
5.8 Cumplimiento de los requisitos	22
6. Problemas encontrados y soluciones	22
7. Limitaciones y trabajo futuro	23
7.1 Limitaciones.....	23
7.2 Trabajo futuro.....	23
8. Conclusiones	24
9. Organización del equipo y herramientas	24
Referencias	24
Anexo A. Estructura del código	25
Anexo B. Comandos útiles	25
Anexo C. Glosario.....	26

Anexo D. Evidencias del entorno 26

Resumen ejecutivo

Este informe documenta el diseño, el montaje y la validación de un clúster de Edge Computing construido con tres mini PC de bajo consumo (Intel Atom x5-Z8350, 2 GB de RAM cada una). El clúster ejecuta K3s, una distribución ligera de Kubernetes, y sirve un modelo de Inteligencia Artificial que clasifica el estado de una máquina industrial en tres categorías: **normal**, **desgaste** y **falla inminente**. El caso de uso elegido es el mantenimiento predictivo a partir de lecturas de temperatura y vibración.

La arquitectura tiene un nodo master (plano de control) y dos nodos trabajadores (worker1 y worker2) que ejecutan una réplica del servicio de inferencia cada uno. Las direcciones IP son estáticas dentro de la subred privada 192.168.1.0/24 y los nodos se interconectan con un switch Gigabit. La inferencia se hace por completo dentro del clúster, sin servicios en la nube. Como los mini PC, el monitor, el cable HDMI y la memoria USB ya estaban disponibles, el gasto real en compras fue de Q262,00 (un switch Gigabit y cuatro cables de red).

Se compararon cuatro algoritmos de clasificación con datos **sintéticos** generados por un simulador de señales de vibración. Se eligió la regresión logística, con 95,33 % de exactitud sobre los datos de prueba, un tamaño de 1,3 KB y una latencia de 0,236 ms por lectura en la computadora de desarrollo.

La prueba central consistió en desconectar físicamente el cable de red de un trabajador mientras se enviaban peticiones de forma continua. Con la configuración por defecto (`externalTrafficPolicy: Cluster`) el servicio siguió respondiendo con el otro trabajador, pero perdió entre 5 % y 11 % de las peticiones aun con reintentos y mostró fallas durante unos 40 s tras el corte, el tiempo que Kubernetes tarda en marcar el nodo como no disponible. **Una serie de tres corridas por política mostró que con Local no se perdió ninguna petición** (frente a 11 a 13 sin respuesta por corrida con `Cluster`), aunque cerca de la mitad de las peticiones tardó unos 815 ms porque el cliente prueba primero el nodo apagado. La inferencia siguió funcionando con el cable al router desconectado, el reemplazo del pod caído quedó Pending por la anti-afinidad estricta, y la memoria usada fue de unos 580 MB por trabajador (de 1 571 MB) y cerca de 1 GB en el master.

1. Introducción

1.1 Contexto

Cada vez más sensores y equipos industriales generan datos que conviene procesar cerca de donde se producen. Este enfoque, llamado Edge Computing o computación en el borde, reduce la latencia, ahorra ancho de banda y permite que una instalación siga funcionando cuando la conexión a Internet falla [9]. Para que un sistema de este tipo sea útil en una planta, no basta con que funcione: debe seguir respondiendo aunque uno de sus equipos se averíe.

El mantenimiento predictivo es un caso natural para el borde. Consiste en estimar el estado de una máquina a partir de señales como la temperatura y la vibración, para intervenir antes de que ocurra la falla en lugar de esperar a que ocurra o de cambiar piezas según un calendario fijo. Las decisiones deben tomarse en fracciones de segundo y cerca de la máquina, lo que justifica ejecutar el modelo en un clúster local.

1.2 Planteamiento del problema

Se necesita un sistema local, autónomo y de bajo costo que clasifique lecturas de sensores con un modelo de IA, y que **signa prestando servicio cuando uno de sus nodos trabajadores deja de funcionar**, con un reparto de carga y una recuperación documentados.

1.3 Objetivos

Objetivo general. Diseñar, montar y validar un clúster de Edge Computing sobre hardware de bajo consumo que ejecute inferencia de IA local para mantenimiento predictivo y tolere la pérdida de un nodo trabajador.

Objetivos específicos:

- Montar un clúster K3s de tres nodos (un master y dos trabajadores) con direcciones IP estáticas en una subred privada de clase C y un switch Gigabit.
- Entrenar y comparar modelos de clasificación para tres estados de la máquina, y elegir uno adecuado para equipos con poca memoria.
- Empaquetar el modelo en un servicio HTTP dentro de un contenedor y desplegarlo con dos réplicas repartidas entre los trabajadores.
- Medir el comportamiento del servicio cuando se desconecta físicamente un trabajador, y documentar el reparto de carga y la recuperación.
- Documentar los problemas encontrados, las limitaciones y el trabajo futuro con honestidad sobre lo que se midió y lo que no.

1.4 Alcance y restricciones

El proyecto se rige por las siguientes restricciones del enunciado y por las decisiones del equipo:

Restricción	Cómo se aplicó en este proyecto
Hardware permitido	Mini PC x86 de bajo consumo con gráficos integrados (Intel Atom x5-Z8350). No se usaron servidores, computadoras de escritorio ni inferencia en la nube pública.
Red	IP estáticas en una subred privada de clase C (192.168.1.0/24) y un switch Gigabit Ethernet.
Orquestación	K3s (Kubernetes ligero) con un master y dos trabajadores.

Restricción	Cómo se aplicó en este proyecto
Tolerancia a fallos	Debe sobrevivir a la pérdida simulada de un trabajador, con reparto de carga y redistribución documentados.
Caso de uso	Mantenimiento predictivo industrial con datos simulados de temperatura y vibración.
Datos	Sintéticos. Las métricas del modelo no representan el desempeño sobre una máquina real.
Equipo	Grupo de hasta 4 integrantes.

1.5 Organización del documento

El capítulo 2 resume el marco teórico. El capítulo 3 describe el diseño de la solución y el capítulo 4 su implementación. El capítulo 5 presenta las pruebas y los resultados, el capítulo 6 los problemas encontrados, el capítulo 7 las limitaciones y el trabajo futuro, y el capítulo 8 las conclusiones. Los anexos reúnen la estructura del código, los comandos útiles, un glosario y las evidencias del entorno.

2. Marco teórico

2.1 Computación en el borde

En la computación en la nube, los datos viajan a centros de datos remotos para procesarse. En la computación en el borde, el procesamiento se hace en equipos cercanos a la fuente de los datos [9]. Las ventajas principales son una menor latencia, un menor consumo de ancho de banda hacia Internet, una mayor privacidad (los datos crudos no salen de la instalación) y la posibilidad de operar sin conexión. Los costos son un hardware con menos recursos y la necesidad de administrar uno mismo la disponibilidad del servicio.

2.2 Mantenimiento predictivo y análisis de vibraciones

Un rodamiento con un defecto en su pista externa produce impactos periódicos cada vez que un elemento rodante pasa por la falla. La frecuencia de esos impactos, llamada BPFO (del inglés ball pass frequency, outer race), depende de la geometría del rodamiento y de la velocidad de giro [6]. Cada impacto excita las resonancias de la estructura, así que la señal medida por un acelerómetro muestra ráfagas de vibración repetidas a esa frecuencia. Conforme el defecto crece, aumentan la amplitud de las ráfagas, el ruido y, a menudo, la temperatura.

Para convertir una ventana de vibración en datos útiles para un modelo se calculan indicadores estadísticos y espectrales:

- **Valor RMS:** raíz cuadrada del promedio de los cuadrados de la señal; mide su energía total.
- **Valor pico:** máximo valor absoluto de la señal.
- **Factor de cresta:** cociente entre el pico y el RMS; crece cuando aparecen impactos aislados fuertes.
- **Curtosis:** mide cuán "picuda" es la distribución de la señal; en una señal gaussiana vale 3 y aumenta cuando hay impulsos.
- **Energía por bandas de frecuencia:** se calcula la transformada rápida de Fourier (FFT) con una ventana de Hann y se suma la energía en rangos de frecuencia, lo que permite ver si crece la energía cerca de la resonancia o de la frecuencia de falla.

2.3 Clasificación supervisada y métricas

En clasificación supervisada, un modelo aprende a asignar una etiqueta (aquí, normal, desgaste o falla inminente) a partir de ejemplos etiquetados. Se compararon cuatro familias de modelos [5]:

- **Regresión logística:** modelo lineal que estima la probabilidad de cada clase. Es pequeño, rápido y fácil de explicar.
- **Árbol de decisión:** conjunto de preguntas sí/no sobre las características.
- **Random Forest:** muchos árboles que votan; suele ser más robusto, pero más pesado.
- **Gradient Boosting:** árboles que se entrenan en secuencia corrigiendo los errores de los anteriores.

Las métricas usadas son la **exactitud** (proporción de lecturas bien clasificadas), la **precisión** y la **exhaustividad** de cada clase, y la **F1 macro**, que promedia la F1 de las clases sin favorecer a las más frecuentes. La **validación cruzada** de 5 particiones entrena y evalúa el modelo cinco veces con partes distintas de los datos de entrenamiento, para estimar qué tan estable es el resultado. La **matriz de confusión** muestra cuántas lecturas de cada clase real se asignaron a cada clase predicha.

2.4 Kubernetes y K3s

Kubernetes es un orquestador que decide en qué equipo se ejecuta cada contenedor y lo mantiene funcionando [2]. K3s es una distribución certificada de Kubernetes pensada para equipos pequeños: se instala como un único binario y omite componentes poco necesarios en el borde [1]. Los conceptos clave para este proyecto son:

Concepto	Qué hace en este proyecto
Nodo	Cada mini PC. El master ejecuta el plano de control; los trabajadores ejecutan los contenedores.
Pod	Unidad mínima de ejecución; aquí contiene un contenedor con el servicio de inferencia.
Deployment y ReplicaSet	Mantienen el número deseado de pods (2 réplicas) y crean uno nuevo si alguno desaparece.
Service (NodePort)	Da una dirección estable al conjunto de pods y abre el puerto 30080 en la IP de cada nodo.
kube-proxy	En cada nodo, reparte las peticiones del Service entre los pods que figuran como disponibles.
Etiquetas y nodeSelector	Se etiquetó a los trabajadores con <code>role=worker</code> y el despliegue exige esa etiqueta.
Anti-afinidad de pods	Regla del planificador que impide poner dos réplicas en el mismo nodo.
Sondas (readiness y liveness)	Permiten que Kubernetes sepa si un pod está listo para recibir tráfico o debe reiniciarse.
Tolerancias (tolerations)	Definen cuánto tiempo permanece un pod en un nodo marcado como no disponible antes de ser desalojado [3].

2.5 Alta disponibilidad y balanceo de carga

La alta disponibilidad busca que un servicio siga respondiendo cuando falla uno de sus componentes. Para ello se combinan tres piezas: **redundancia** (más de una réplica del servicio), **detección** de la falla y **redistribución** del trabajo hacia lo que sigue vivo. El balanceo de carga reparte las peticiones entre las réplicas; puede hacerlo el cliente (rotando entre las direcciones que conoce), un componente de red como kube-proxy o un balanceador externo dedicado.

En este proyecto el reparto lo hacen dos mecanismos: el cliente de prueba, que rota entre las IP de los trabajadores y reintenta en otro nodo si no obtiene respuesta, y kube-proxy, que reparte entre los pods del Service. No se usó un balanceador externo dedicado.

3. Diseño de la solución

3.1 Requisitos

ID	Requisito
RF1	Clasificar una lectura (temperatura y ventana de vibración) en normal, desgaste o falla inminente.
RF2	Exponer el servicio por HTTP en la IP de cada trabajador, puerto 30080.
RF3	Indicar en cada respuesta qué pod y qué nodo la atendieron.
RF4	Rechazar con un error claro las lecturas con formato inválido.
RNF1	Funcionar sin servicios en la nube y sin depender de Internet una vez instalado.
RNF2	Seguir respondiendo cuando se desconecta un trabajador.
RNF3	Caber en 2 GB de RAM por nodo; modelo pequeño y de baja latencia.
RNF4	Usar direcciones IP estáticas en una subred privada de clase C.

3.2 Arquitectura lógica y flujo de datos

El flujo de una lectura es el siguiente: (1) el cliente (la laptop con el simulador de sensores) elige un trabajador y envía una petición HTTP POST a /predict en el puerto 30080; (2) kube-proxy de ese nodo la dirige a un pod disponible; (3) el pod valida los datos, calcula las 9 características, aplica el modelo y responde con la clase, las probabilidades y el pod y el nodo que atendieron; (4) si el cliente no obtiene respuesta en 0,8 s, reintenta en otro nodo, hasta 3 intentos.

El master no ejecuta inferencia: solo mantiene el estado del clúster, decide dónde se programan los pods y vigila a los nodos. Por eso la caída de un trabajador se detecta a través del master, que es también un punto único de falla de la administración (sección 7).

3.3 Selección de hardware

El equipo dispone de tres mini PC Wintel W8 Pro, con procesador Intel Atom x5-Z8350 (4 núcleos, 1,44 GHz), 2 GB de RAM nominales (Ubuntu reporta 1 571 MB totales en los trabajadores), 32 GB de almacenamiento eMMC y un puerto Ethernet. Cumplen la restricción del enunciado (x86 de bajo consumo con gráficos integrados) y no representaron costo de compra. La tabla compara esta opción con las alternativas más usadas en este tipo de proyectos:

Opción	Costo de referencia	RAM	Red	Comentario
Mini PC Wintel W8 Pro (usada)	Equipo ya disponible	2 GB	1 000 Mb/s (medido en los trabajadores)	x86_64: instalación estándar de Ubuntu y K3s. Poca RAM para el master.
Raspberry Pi 4 (2 GB)	US\$ 67,50	2 GB	Gigabit	ARM64; requiere fuente, tarjeta microSD y carcasa aparte.

Opción	Costo de referencia	RAM	Red	Comentario
Raspberry Pi 5 (2 GB)	US\$ 77,50	2 GB	Gigabit	ARM64; más rápida, mayor costo.
Raspberry Pi 4 (1 GB)	US\$ 35,00	1 GB	Gigabit	Muy justa para ejecutar el plano de control de K3s.

Precios de referencia consultados el 1 de octubre de 2026 en tiendas en línea; pueden variar. Con tres unidades, el costo de hardware de la alternativa Raspberry Pi 4 de 2 GB habría sido de unos US\$ 202,50 sin contar fuentes, tarjetas ni cables.

La principal desventaja de los mini PC es la memoria: de los 2 GB nominales, Ubuntu reporta 1 571 MB totales en worker1 y worker2 (el resto lo reserva el hardware). La ficha técnica de estos equipos hacía pensar en una red de 10/100 Mbps, pero `ethtool enp1s0` reporta Speed: 1000Mb/s en el master y en los dos trabajadores, de modo que el enlace opera a Gigabit y encaja con el switch.

3.4 Plan de red

Se usó la subred privada 192.168.1.0/24 (clase C), con IP fija en cada nodo y la laptop como cliente. El switch Gigabit (Intellinet 530378, de 5 puertos) interconecta a los tres nodos y a la laptop con cuatro cables UTP categoría 6 de 1,5 m.

Equipo	Rol	Dirección IP	Observaciones
master	Plano de control (servidor K3s)	192.168.1.26	No ejecuta inferencia
worker1	Trabajador	192.168.1.27	Ejecuta una réplica del servicio
worker2	Trabajador	192.168.1.28	Ejecuta una réplica del servicio
Laptop	Cliente, simulador y pruebas	192.168.1.50	Máscara 255.255.255.0; el adaptador Ethernet no muestra puerta de enlace

Servicios y puertos

Servicio	Dónde corre	Puerto	Función
SSH	Todos los nodos	22/TCP	Administración remota
API de Kubernetes (K3s)	master	6443/TCP	Los trabajadores se registran y reciben órdenes
Kubelet	Todos los nodos	10250/TCP	El plano de control gestiona los pods
Flannel (VXLAN)	Todos los nodos	8472/UDP	Red de pods entre nodos
edge-infer (NodePort)	Cada nodo	30080/TCP	Entrada al servicio de inferencia (8000/TCP dentro del pod)

3.5 Diseño del servicio de inferencia

El servicio está escrito en Python con la biblioteca estándar (`http.server`) más NumPy y scikit-learn. Se evitó un framework web para reducir el tamaño de la imagen y el uso de memoria en nodos de 2 GB. Carga el modelo al arrancar y atiende varias peticiones en paralelo con hilos.

Ruta	Método	Función
/predict	POST	Recibe <code>temperature_c</code> (número) y <code>vibration</code> (lista de 256 números) y devuelve la clase, la bandera de alerta, las probabilidades, la latencia del modelo y el pod y el nodo que respondieron.

Ruta	Método	Función
/health	GET	Sonda de vida: el proceso responde.
/ready	GET	Sonda de preparación: responde 200 cuando el modelo ya está cargado, 503 mientras carga.
/info	GET	Pod, nodo, modelo, clases, versión y tiempo en marcha.
/metrics	GET	Métricas en formato de texto de Prometheus.

Validación de entrada. El servicio responde con HTTP 400 y un mensaje en JSON cuando el cuerpo está vacío o supera 64 KB, no es JSON válido, la temperatura no es un número finito entre -50 y 250 °C, o la vibración no es una lista de exactamente 256 números finitos. Los errores del cliente (4xx) no se reintentan.

Formato de la petición y la respuesta (valores ilustrativos)

```
POST /predict
{ "temperature_c": 62.5, "vibration": [0.12, -0.31, 0.05, ... 256 valores] }

200 OK
{ "label": "desgaste", "alerta": true,
  "probabilities": { "normal": 0.08, "desgaste": 0.85, "falla_inminente": 0.07 },
  "latency_ms": 0.9, "pod": "edge-infer-654c5b467-wpfs4", "node": "worker1" }
```

Métricas. /metrics publica edge_up, edge_start_time_seconds, edge_requests_total (por ruta y código HTTP), edge_predictions_total (por clase) y el histograma edge_request_duration_seconds, todos con las etiquetas pod y node. Esas etiquetas permiten ver en un panel cuál pod deja de recibir tráfico.

3.6 Diseño del simulador de datos

Como no se dispone de una máquina real instrumentada, las lecturas se generan con un modelo físico simple. Es la fuente de **todos** los datos de entrenamiento y de prueba. La señal es la suma de una componente del eje, ruido, y, cuando hay defecto, impulsos periódicos a la frecuencia BPFO que excitan una resonancia estructural. A mayor severidad hay más amplitud de impulsos, más ruido y más temperatura.

Parámetro	Valor
Frecuencia de muestreo	1 024 Hz
Ventana por lectura	256 muestras (0,25 s); resolución de frecuencia de 4 Hz
Velocidad de giro del eje	25 Hz (1 500 rpm)
Frecuencia de falla BPFO	3,58 veces la de giro: unos 89,5 Hz
Resonancia estructural excitada	300 Hz
Temperatura	Base de unos 45 °C más un aumento que crece con la severidad (hasta unos 35 °C)

Las clases dependen solo de la **severidad** (0 = sano, 1 = falla total): normal de 0 a 0,25; desgaste de 0,25 a 0,65; falla inminente de 0,65 a 1. La severidad solo se usa para generar y etiquetar la señal; **no es una entrada del modelo**. El modelo recibe únicamente la temperatura y la ventana de vibración, de la que se extraen 9 características: temperatura, RMS, pico, factor de cresta, curtosis y la energía en las bandas de 0–60, 60–150, 150–300 y 300–512 Hz.

3.7 Diseño de la tolerancia a fallos

La estrategia combina tres decisiones: dos réplicas, una por trabajador (anti-afinidad estricta); un cliente que rota entre nodos y reintenta; y la detección de nodos caídos de Kubernetes. Con los valores por defecto de Kubernetes [3][4], lo que ocurre al desconectar un trabajador es:

Momento	Evento	Componente
0 s	Se desconecta el cable. El nodo deja de informar, pero sus pods todavía figuran como disponibles.	—
Hasta unos 40 s	kube-proxy sigue incluyendo al pod del nodo caído entre los destinos del Service. Parte de las peticiones se pierde o espera el tiempo máximo del cliente.	kube-proxy, cliente
Unos 40 s	El nodo pasa a NotReady (parámetro node-monitor-grace-period).	Controlador de nodos
Poco después	El pod deja de contar como disponible y se retira de los destinos del Service. Desde aquí, el tráfico va solo al pod vivo.	Controlador de endpoints
Unos 300 s	Se desaloja el pod del nodo caído (tolerationSeconds por defecto) y el ReplicaSet crea un reemplazo.	Controlador de nodos, ReplicaSet
Después	El planificador busca dónde ubicar el reemplazo. Con anti-afinidad estricta no puede ir al trabajador que ya tiene una réplica y queda en Pending hasta que el otro nodo vuelva.	Planificador

Esta tabla describe el comportamiento esperado con valores por defecto, no una medición. Los tiempos reales de este clúster se registran en la sección 5.6.

Hay mejoras previstas que reducen la ventana de pérdida y que **no estaban aplicadas** en las pruebas iniciales: sondas de readiness y liveness en el despliegue, tolerancias de 15 s en lugar de 300 s, y externalTrafficPolicy: Local en el Service, que hace que cada nodo responda solo con su pod local y evita enviar tráfico a un pod caído en otro nodo. Con Local se debe usar solo las IP de los trabajadores como destinos y, al volver un nodo, reiniciar el despliegue para que recupere su pod. El efecto de Local se midió en la sección 5.6.

3.8 Costos del proyecto

El proyecto se hizo reutilizando equipos que los integrantes ya tenían, de modo que el gasto real fue bajo. La tabla separa lo que ya estaba disponible de lo que hubo que comprar. Moneda: quetzales (GTQ), según la factura de compra.

Tabla de costos de hardware y materiales

Elemento	Cantidad	Origen	Costo de compra
Mini PC Wintel W8 Pro (nodos master, worker1 y worker2)	3	Ya disponibles (equipos sin uso de un integrante)	0
Monitor (instalación y configuración inicial)	1	Ya disponible (de un integrante)	0
Cable HDMI	1	Ya disponible	0
Memoria USB (instalador de Ubuntu Server)	1	Ya disponible	0
Switch Intellinet 530378, 5 puertos Gigabit 10/100/1000	1	Comprado	Q178,00 (precio Q198,00 menos descuento de Q20,00)

Elemento	Cantidad	Origen	Costo de compra
Cable UTP categoría 6, Intellinet 342056, 1,5 m	4	Comprados (Q21,00 cada uno)	Q84,00
Total gastado en compras			Q262,00 (subtotal Q282,00 menos descuento de Q20,00)

El gasto total en compras fue de **Q262,00**, alrededor de US\$ 34 al cambio aproximado de Q7,7 por dólar. Reutilizar equipos no significa que sean gratuitos: tienen un valor propio. Si hubiera que comprarlos, su costo se sumaría al presupuesto. Como referencia, la alternativa con tres Raspberry Pi 4 de 2 GB costaría unos US\$ 202,50 solo en las placas (sección 3.3), sin fuentes de poder, tarjetas de memoria ni carcasas.

4. Implementación

4.1 Preparación de los nodos

Cada mini PC se instaló con Ubuntu Server 26.04.1 LTS (edición mínima, kernel 7.0.0-38-generic, x86_64) y se le configuró una dirección IP fija con Netplan en `/etc/netplan/01-netcfg.yaml` (en cada nodo se cambia X por 26, 27 o 28):

```
network:
  version: 2
  renderer: networkd
  ethernets:
    enp1s0:
      dhcp4: no
      addresses:
        - 192.168.1.X/24
      routes:
        - to: default
          via: 192.168.1.1
      nameservers:
        addresses: [8.8.8.8, 1.1.1.1]
```

Se ajustaron los permisos con `chmod 600 /etc/netplan/01-netcfg.yaml` y se aplicó con `sudo netplan apply`. Como la edición mínima de Ubuntu no incluye editor de texto, los archivos se escribieron con `cat << 'EOF' | sudo tee`.

Sobre el aislamiento de Internet. La ruta por defecto y los DNS apuntan al router y se usaron para instalar K3s, que descarga archivos de Internet. Una vez instalado el clúster y cargada la imagen, la inferencia no sale a Internet, porque el cliente habla directamente con las IP de los nodos. La prueba P06 (sección 5.5) lo respalda.

4.2 Entrenamiento del modelo

En la computadora de desarrollo se ejecuta `python train.py`, que genera 6 000 lecturas sintéticas (2 000 por clase), las divide en 75 % para entrenamiento (4 500) y 25 % para prueba (1 500) con estratificación, evalúa cuatro modelos con validación cruzada de 5 particiones y guarda el elegido en `artifacts/model.joblib`, junto con las métricas, la comparación y la matriz de confusión. La semilla es fija (42), por lo que el resultado es reproducible.

4.3 Instalación de K3s

1. En el master:

```
curl -sfl https://get.k3s.io | sh -
```

2. En cada trabajador, con el token del master (/var/lib/rancher/k3s/server/node-token):

```
curl -sfl https://get.k3s.io | K3S_URL=https://192.168.1.26:6443 K3S_TOKEN=<TOKEN> sh -
```

3. Etiquetar los trabajadores y crear el espacio de nombres del proyecto:

```
sudo kubectl label node worker1 role=worker
sudo kubectl label node worker2 role=worker
sudo kubectl create namespace edge
```

Versión instalada (salida de `k3s --version`): **K3s v1.36.5+k3s1** (Go 1.26.8), con containerd 2.3.4-k3s1.36. Los tres nodos figuran como Ready con las IP 192.168.1.26, .27 y .28, y los dos pods de edge-infer están en Running, uno en cada trabajador (anexo D).

4.4 Contenedorización y distribución de la imagen

La imagen se basa en `python:3.12-slim`, instala solo NumPy, scikit-learn y joblib, copia el código y el modelo, y se ejecuta con un usuario sin privilegios. Como el despliegue usa `imagePullPolicy: Never`, el clúster no descarga imágenes: se compilan en la PC de desarrollo y se importan a mano en cada trabajador.

```
docker build -t edge-infer:1.0 -f service/Dockerfile .
docker save -o edge-infer.tar edge-infer:1.0
scp edge-infer.tar edgeuser@192.168.1.27:/home/edgeuser/
scp edge-infer.tar edgeuser@192.168.1.28:/home/edgeuser/
# en cada trabajador:
sudo k3s ctr images import /home/edgeuser/edge-infer.tar
```

Esta precarga es la que permite arrancar pods sin acceso a un registro de imágenes, y es una de las razones por las que el servicio puede operar sin Internet.

4.5 Manifiestos de Kubernetes

Esta es la versión usada en las pruebas iniciales.

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edge-infer
  namespace: edge
spec:
  replicas: 2
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 0
      maxUnavailable: 1
  selector:
    matchLabels:
      app: edge-infer
  template:
    metadata:
      labels:
        app: edge-infer
```

```
spec:
  nodeSelector:
    role: worker
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchExpressions:
              - key: app
                operator: In
                values:
                  - edge-infer
          topologyKey: "kubernetes.io/hostname"
  containers:
    - name: edge-infer
      image: edge-infer:1.0
      imagePullPolicy: Never
      ports:
        - containerPort: 8000
      env:
        - name: NODE_NAME
          valueFrom:
            fieldRef:
              fieldPath: spec.nodeName
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: edge-infer
  namespace: edge
spec:
  type: NodePort
  selector:
    app: edge-infer
  ports:
    - port: 8000
      targetPort: 8000
      nodePort: 30080
```

Qué hace cada decisión

- **Anti-afinidad estricta:** obliga a que cada réplica viva en un trabajador distinto, para que la caída de un nodo no deje el servicio sin pods. Consecuencia: si un trabajador cae, el reemplazo no puede programarse en el otro y queda en Pending hasta que el nodo vuelva; la capacidad baja a una réplica durante la falla. Confirmado en la prueba P05 (sección 5.5).
- **maxSurge: 0 y maxUnavailable: 1:** necesarios para que una actualización progresiva no se bloquee con la anti-afinidad estricta.
- **NODE_NAME desde la Downward API:** cada respuesta indica en qué nodo físico se ejecutó.
- **Tiempos por defecto:** estos manifiestos no cambian los tiempos de detección de Kubernetes descritos en la sección 3.7.

4.6 Monitoreo

El kit incluye una configuración de Prometheus y Grafana que corre en la laptop y recoge métricas de los nodos (con node_exporter, puerto 9100) y del servicio de inferencia (puerto 30080), cada 5 s. Los paneles propuestos (no desplegados en el clúster físico) son:

Panel	Consulta (PromQL)	Qué muestra
Nodos activos	<code>up{job="nodos"}</code>	1 si el nodo responde, 0 si cayó
Peticiones por pod	<code>sum by (pod, node) (rate(edge_requests_total{endpoint="/predict",status="200"}[15s]))</code>	Cómo un pod deja de recibir y el otro sube
Latencia p95	<code>histogram_quantile(0.95, sum by (le) (rate(edge_request_duration_seconds_bucket[30s])))</code>	Latencia del modelo
Alertas por clase	<code>sum by (label) (rate(edge_predictions_total[30s]))</code>	Lecturas por clase
Errores de validación	<code>sum(rate(edge_requests_total{endpoint="/predict",status="400"}[30s]))</code>	Peticiones rechazadas

En el clúster físico no se desplegó Prometheus ni Grafana, para no consumir los 2 GB de RAM por nodo ni ciclos del procesador Intel Atom, y priorizar el servicio de inferencia y la orquestación de K3s (decisión del equipo). La configuración queda en el kit como referencia y el monitoreo se hizo con las salidas de `failover_test.py` y de `kubect1`. Ver sección 7.1.

4.7 Seguridad y operación

Las siguientes prácticas son recomendables para un clúster de este tipo; se listan como guía y no se verificó cuáles estaban aplicadas en el montaje.

- Acceso por SSH con llaves y sin contraseña, y un usuario no administrador para el día a día.
- Cortafuegos (por ejemplo `ufw`) que permita solo los puertos de la sección 3.4 dentro de la subred del clúster.
- Resguardo del token del clúster (`node-token`), que permite unir nodos.
- Contenedor sin privilegios (el `Dockerfile` usa un usuario no root) y validación estricta de la entrada del servicio.

5. Pruebas y resultados

5.1 Plan de pruebas

ID	Objetivo	Procedimiento y criterio	Estado
P01	Comparar y elegir el modelo	<code>train.py</code> con validación cruzada; se elige la mayor F1 macro.	Hecha
P02	Pruebas unitarias del kit	<code>python -m unittest discover -s tests</code> (8 pruebas: señales, características, modelo, validación y servicio).	Hecha en el entorno de preparación (8 pruebas aprobadas)
P03	Clasificación en vivo	<code>simulator.py</code> : severidad de 0 a 1; el modelo debe pasar de normal a falla inminente.	Hecha
P04	Caída de un trabajador	<code>failover_test.py</code> : 5 peticiones/s, cable de un trabajador desconectado; medir éxito final, primer intento, latencia y ventana sin servicio.	Hecha (1 corrida por trabajador, más corridas adicionales)
P05	Reemplazo del pod bajo anti-afinidad estricta	Observar si el pod nuevo queda Pending o se ubica en el otro nodo.	Hecha
P06	Operación sin Internet	Desconectar el cable al router y repetir una corrida corta del simulador.	Hecha
P07	Serie comparativa A/B	3 corridas con Cluster y 3 con Local, 150 s, corte a los ~30 s.	Hecha; cifras leídas de las gráficas
P08	Consumo y recursos	<code>free -m</code> y medidor de potencia en reposo y bajo carga.	Hecha para memoria; potencia no medida
P09	Reincorporación del nodo	Reconectar el trabajador y observar que vuelve a Ready.	Hecha; tiempo no medido

5.2 Evaluación del modelo (P01)

Tabla 1. Comparación de modelos (datos sintéticos; latencia medida en la computadora de desarrollo)

Modelo	F1 macro (CV)	Exactitud (prueba)	Latencia por lectura	Tamaño	¿Elegido?
Regresión logística	0,9458	95,33 %	0,236 ms	1,3 KB	Sí
Árbol de decisión	0,9265	92,67 %	0,086 ms	5,7 KB	No
Random Forest	0,9390	94,67 %	5,875 ms	634,5 KB	No
Gradient Boosting	0,9357	95,27 %	4,714 ms	669,8 KB	No

Se eligió la regresión logística por tener la mayor F1 macro en validación cruzada y ser, a la vez, el modelo más pequeño y uno de los más rápidos, lo que encaja con nodos de 2 GB de RAM. Las diferencias entre los modelos son pequeñas frente a la variación de la validación cruzada (alrededor de 0,5 puntos), así que la elección se apoya sobre todo en el tamaño y la velocidad. La latencia por lectura de la tabla se midió en la computadora de desarrollo; en el clúster, la latencia de extremo a extremo medida por el simulador fue de unos 12 a 41 ms (secciones 5.3 y 5.5).

Tabla 2. Métricas por clase del modelo elegido (1 500 lecturas de prueba, 500 por clase)

Clase	Precisión	Exhaustividad	F1
normal	0,964	0,978	0,971
desgaste	0,944	0,914	0,929
falla_inminente	0,951	0,968	0,959

Matriz de confusión (Regresión logística, datos sintéticos)

	normal	desgaste	falla_inminente
True label			
normal	489	11	0
desgaste	18	457	25
falla_inminente	0	16	484
	normal	desgaste	falla_inminente
Predicted label			

Figura 1. Matriz de confusión del modelo elegido sobre los datos de prueba (datos sintéticos).

Los errores se concentran en la clase intermedia: 18 lecturas de desgaste se clasificaron como normales y 25 como falla inminente. De las lecturas normales, 11 se tomaron por desgaste, y de las de falla inminente, 16 por desgaste. No hubo ninguna confusión directa entre normal y falla inminente. Es lo esperable: la frontera entre estados contiguos es donde las señales se parecen más.

5.3 Clasificación en vivo (P03)

El simulador envía una lectura por segundo con una severidad que sube de forma gradual y muestra la clase predicha, el nodo que respondió y la latencia de extremo a extremo:

- **Fase normal (0–16 s):** temperaturas por debajo de 55 °C; el modelo responde normal (latencias de unos 23 a 38 ms).
- **Fase de desgaste (17–38 s):** la temperatura sube de 60 °C; el modelo responde desgaste y marca alerta.
- **Fase de falla inminente (39–60 s):** picos de más de 70 °C y severidad mayor a 0,65; el modelo responde falla_inminente y marca alerta.

```

PS C:\Users\lguar\Documents> .\simulador.py http://192.168.1.27:8080 http://192.168.1.28:8080
t(s) sev. real temp(C) prediccion node ms
0.0 0.50 50.6 normal worker1 171.2
1.0 0.62 48.5 normal worker1 30.6
2.0 0.63 46.3 normal worker1 21.5
3.0 0.65 44.4 normal worker2 33.0
4.0 0.67 42.1 normal worker1 13.1
5.0 0.68 40.9 normal worker1 24.5
6.0 0.10 54.5 normal worker2 14.4
7.0 0.12 48.9 normal worker1 25.5
8.0 0.13 49.1 normal worker2 32.2
9.0 0.15 48.3 normal worker2 13.3
10.0 0.17 49.0 normal worker2 15.1
11.0 0.18 55.8 normal worker1 35.0
12.0 0.20 53.2 normal worker1 32.8
13.0 0.22 49.5 normal worker2 26.5
14.0 0.23 44.9 normal worker1 31.3
15.0 0.25 41.8 normal worker1 16.8
16.0 0.27 54.6 normal worker2 17.8
17.0 0.28 48.7 desgaste worker2 1625.8 <-- ALERTA
18.0 0.31 48.8 desgaste worker2 826.6 <-- ALERTA
19.0 0.33 48.9 desgaste worker2 27.5 <-- ALERTA
20.0 0.34 56.8 SIN RESPUESTA (URLError)
21.0 0.38 64.3 desgaste worker2 24.7 <-- ALERTA
22.0 0.40 53.4 SIN RESPUESTA (URLError)
23.0 0.40 53.9 SIN RESPUESTA (URLError)
24.0 0.40 53.2 SIN RESPUESTA (URLError)
25.0 0.40 53.2 desgaste worker2 13.5 <-- ALERTA
26.0 0.54 64.5 desgaste worker2 826.6 <-- ALERTA
27.0 0.56 57.6 desgaste worker2 25.8 <-- ALERTA
28.0 0.57 62.1 SIN RESPUESTA (URLError)
29.0 0.61 61.0 desgaste worker2 15.4 <-- ALERTA
30.0 0.61 61.8 desgaste worker2 821.9 <-- ALERTA
31.0 0.65 67.1 desgaste worker2 22.8 <-- ALERTA
32.0 0.66 70.2 falla_inminente worker2 827.6 <-- ALERTA
33.0 0.68 74.4 falla_inminente worker2 26.1 <-- ALERTA
34.0 0.70 65.1 SIN RESPUESTA (URLError)
35.0 0.74 69.7 falla_inminente worker2 23.9 <-- ALERTA
36.0 0.75 72.2 SIN RESPUESTA (URLError)
37.0 0.79 73.6 falla_inminente worker2 1637.2 <-- ALERTA
38.0 0.82 68.8 falla_inminente worker2 826.2 <-- ALERTA
39.0 0.84 74.4 falla_inminente worker2 24.8 <-- ALERTA
40.0 0.85 77.7 falla_inminente worker2 822.2 <-- ALERTA
41.0 0.87 76.1 falla_inminente worker2 36.2 <-- ALERTA
42.0 0.89 73.1 SIN RESPUESTA (URLError)
43.0 0.93 73.8 falla_inminente worker2 1636.8 <-- ALERTA
44.0 0.96 70.9 SIN RESPUESTA (URLError)
45.0 1.00 79.4 SIN RESPUESTA (URLError)
66 lecturas enviadas.

```

Figura 2. Salida de `simulador.py` durante la primera prueba: la clasificación avanza de normal a desgaste a falla inminente.

5.4 Prueba de caída de un trabajador (P04)

Método

Desde la laptop se envían 5 peticiones por segundo a las IP de los dos trabajadores (<http://192.168.1.27:30080> y <http://192.168.1.28:30080>). El cliente rota entre nodos, espera como máximo 0,8 s por intento y reintenta hasta 3 veces. Durante la prueba se desconecta físicamente el cable de red de un trabajador. Se registran, por petición, el resultado, el número de intentos, la latencia y el nodo y el pod que respondieron.

Definiciones: **éxito final** es la proporción de peticiones que recibieron respuesta tras todos sus intentos; **éxito al primer intento** es la proporción que respondió sin reintentar; la **ventana más larga sin servicio** es el mayor intervalo en que todas las peticiones fallaron.

Resultados registrados

Tabla 3. Corridas registradas

Corrida	Política de tráfico	Peticiones	Éxito final	Primer intento	Mediana / p95	Mayor ventana
R1: prueba inicial (worker1 apagado)	Cluster (por defecto)	265	252 (95,09 %)	no registrado	30,7 / 834,3 ms	2,6 s
R2: prueba inicial, tabla (worker1)	Cluster (por defecto)	115	102 (88,70 %)	88 (76,52 %)	27,7 ms / no registrado	5,1 s
R3: prueba inicial, tabla (worker2)	Cluster (por defecto)	112	101 (90,18 %)	87 (77,68 %)	26,4 ms / no registrado	7,5 s
R4: captura (worker1, corte a ~16 s)	Cluster (por defecto)	118	106 (89,83 %)	90 (76,27 %)	28,0 / 833,6 ms	2,6 s
R5: captura (worker2, corte a ~16 s)	Cluster (por defecto)	118	106 (89,83 %)	92 (77,97 %)	28,8 / 843,9 ms	5,1 s
L1: segunda serie (worker1, corte a 30 s)	Cluster (comprobado después)	175	166 (94,86 %)	157 (89,71 %)	16,2 / 805,2 ms	2,6 s
L2: segunda serie (worker2, corte a 30 s)	Cluster (comprobado después)	172	162 (94,19 %)	156 (90,70 %)	16,4 / 38,5 ms	7,5 s

Notas: (1) las divisiones se verificaron y coinciden con los porcentajes. (2) La prueba inicial citaba 90,52 % en un resumen, pero la división de la tabla da 90,18 %. (3) R4 y R5 se ejecutaron con `--duration 60`; el total de 118 peticiones es menor que las 300 esperables a 5 por segundo porque, tras el corte, cada petición fallida bloquea al cliente hasta 2,4 s (3 intentos de 0,8 s). (4) R1 a R3 provienen de la prueba inicial y R4 y R5 de capturas posteriores; el manifiesto, el momento del corte y la duración no se registraron de forma uniforme, por lo que no se comparan entre sí. (5) L1 y L2 se anotaron como Local, pero la revisión posterior mostró que el Service estaba en Cluster; por eso se tratan como corridas Cluster (sección 5.6).

```
PS C:\Users\lguar\Documents\UNIVERSIDAD\12 semestre\Arquitectura de Computadoras II\Proyecto\edge-cluster-kit> python fa
ilover_test.py --endpoints http://192.168.1.27:38080,http://192.168.1.28:38080 --rate 5 --duration 60 --plot failover.pn
g
Enviando 5 peticiones/s durante 60 s... (apaga un worker cuando quieras)
t= 4.8s ultimas 25 peticiones: 25 ok
t= 9.8s ultimas 25 peticiones: 25 ok
t= 14.8s ultimas 25 peticiones: 25 ok
t= 28.8s ultimas 25 peticiones: 23 ok

=== RESUMEN ===
Peticiones totales: 118
Exito final (con reintento): 106/118 = 89.83 %
Exito al primer intento: 98/118 = 76.27 %
Latencia (ok) mediana / p95: 28.8 ms / 833.6 ms
Ventana mas larga sin servicio: 2.6 s
Reparto por nodo: {'worker1': 51, 'worker2': 55}
Reparto por pod: {'edge-infer-654c5b467-wpfs4': 51, 'edge-infer-654c5b467-22zq6': 55}
CSV guardado en resultados.failover.csv
Gráfica guardada en failover.png
PS C:\Users\lguar\Documents\UNIVERSIDAD\12 semestre\Arquitectura de Computadoras II\Proyecto\edge-cluster-kit>

PS C:\Users\lguar\Documents\UNIVERSIDAD\12 semestre\Arquitectura de Computadoras II\Proyecto\edge-cluster-kit> python fa
ilover_test.py --endpoints http://192.168.1.27:38080,http://192.168.1.28:38080 --rate 5 --duration 60 --plot failover.pn
g
Enviando 5 peticiones/s durante 60 s... (apaga un worker cuando quieras)
t= 4.8s ultimas 25 peticiones: 25 ok
t= 9.8s ultimas 25 peticiones: 25 ok
t= 14.8s ultimas 25 peticiones: 25 ok
t= 38.2s ultimas 25 peticiones: 17 ok

=== RESUMEN ===
Peticiones totales: 118
Exito final (con reintento): 106/118 = 89.83 %
Exito al primer intento: 92/118 = 77.97 %
Latencia (ok) mediana / p95: 28.8 ms / 843.9 ms
Ventana mas larga sin servicio: 5.1 s
Reparto por nodo: {'worker1': 68, 'worker2': 38}
Reparto por pod: {'edge-infer-654c5b467-wpfs4': 68, 'edge-infer-654c5b467-22zq6': 38}
CSV guardado en resultados.failover.csv
Gráfica guardada en failover.png
```

Figura 3. Resumen impreso por `failover_test.py`: corrida R4 (arriba, worker1 desconectado) y R5 (abajo, worker2 desconectado).

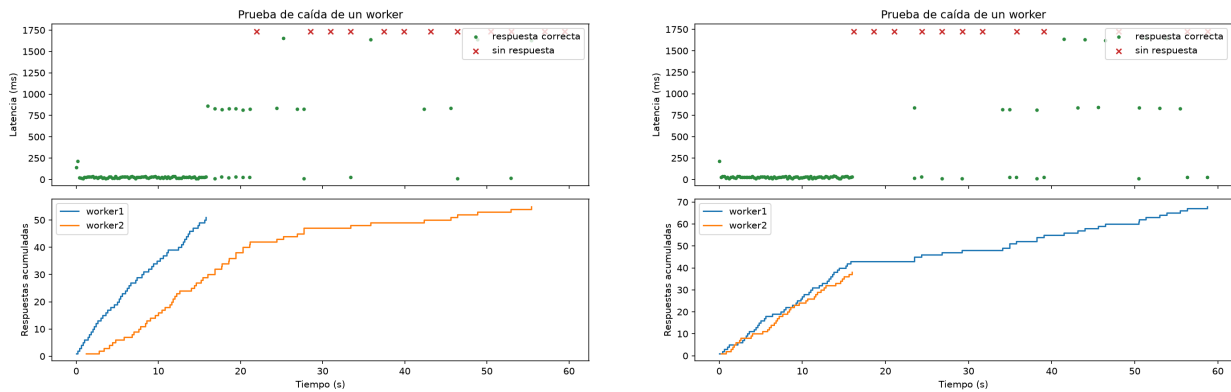


Figura 4. Gráficas de R4 (izquierda) y R5 (derecha): latencia por petición (arriba) y respuestas acumuladas por nodo (abajo).

Corridas L1 y L2

L1 y L2 se hicieron con el corte del cable a los 30 s y 60 s de duración. Se anotaron como Local, pero después se comprobó que el Service estaba en Cluster (sección 5.6); son, por tanto, corridas Cluster adicionales. En ambas, las primeras 150 peticiones (hasta los 29,8 s) fueron correctas, con una latencia mediana de unos 16 ms. Después del corte, el cliente dejó de completar peticiones al ritmo previsto.

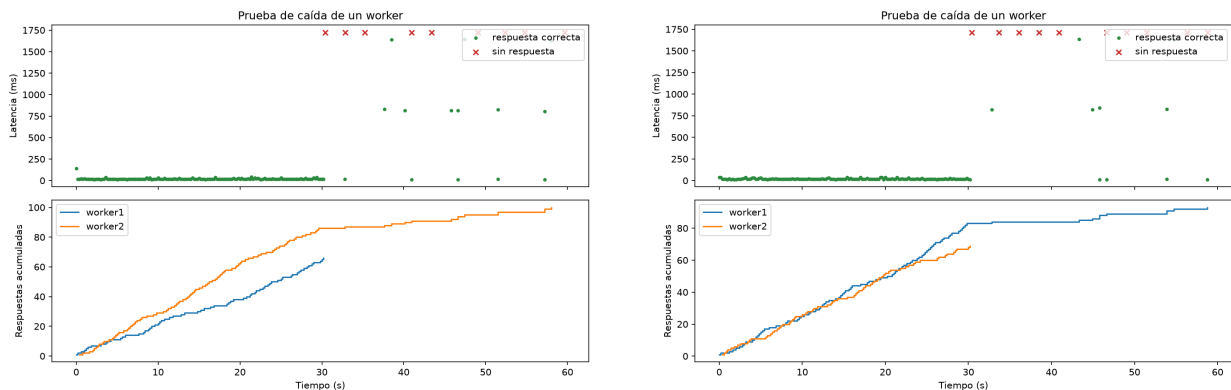


Figura 5. Gráficas de L1 (izquierda) y L2 (derecha). Tras los 30 s, la curva del nodo vivo casi se aplanan y aparecen peticiones sin respuesta (cruces rojas).

Solo después del corte

El resumen global promedia las peticiones previas al corte, todas correctas, con las posteriores, y por eso suaviza el efecto de la caída. Con los datos de las capturas se puede calcular el tramo posterior al corte (los 30 s siguientes a la desconexión):

Tabla 4. Tramo posterior al corte en L1 y L2

Corrida	Peticiones tras el corte	Éxito final	Primer intento (estimado)	Respuestas correctas por segundo
L1 (worker1 apagado)	25	16 (64,0 %)	7 (28,0 %)	0,53
L2 (worker2 apagado)	22	12 (54,5 %)	6 (27,3 %)	0,40

Cálculo: total menos las 150 peticiones previas, que los resúmenes por bloques muestran todas correctas. Antes del corte el ritmo era de 5 respuestas correctas por segundo; después, de entre 0,4 y 0,5. La cifra exacta se obtiene del CSV con `python analizar_csv.py resultados_failover.csv --cut-at 30`, y desde esta versión `failover_test.py` imprime también este tramo.

¿Qué esperaríamos con cada política?

Con un modelo simple se puede calcular lo esperable tras el corte. Supuestos: el cliente alterna el nodo con el que empieza cada petición, hace 3 intentos de 0,8 s, el nodo caído nunca responde y, con la política Cluster, kube-proxy envía la mitad de las peticiones del nodo vivo al pod del nodo caído, mientras que con Local las atiende siempre el pod local.

Escenario	Éxito final tras el corte	Éxito al primer intento
Esperado con política Cluster	62,5 %	25 %
Esperado con política Local	100 %	50 %
Observado en L1	64,0 %	28,0 %
Observado en L2	54,5 %	27,3 %

Los valores observados en L1 y L2 se parecen a lo esperado con la política **Cluster**. Tiene sentido: la revisión posterior mostró que el Service estaba en Cluster, y la serie A/B con Local verificado dio el comportamiento previsto para esa política (ninguna petición perdida y cerca de la mitad de las peticiones al primer intento; sección 5.6). Con 22 y 25 peticiones, la comparación de esta tabla es solo orientativa.

Sobre la diferencia de latencia mediana entre tandas (unos 28 ms en R4 y R5 y unos 16 ms en L1 y L2): no se midió su causa. El equipo propone dos explicaciones, el salto entre nodos por la red virtual (VXLAN) y el escalado de frecuencia del procesador, pero son hipótesis; en las gráficas de las series A (Cluster) y B (Local) la latencia anterior al corte se ve similar, de unos 10 a 30 ms.

Lectura de los resultados

- **El servicio no se detuvo, pero se degradó mucho.** El trabajador restante siguió respondiendo y el modelo siguió clasificando; sin embargo, tras el corte solo se obtuvieron entre 0,4 y 0,5 respuestas correctas por segundo, frente a 5 antes. Parte de esa caída es del cliente de prueba, que espera hasta 2,4 s por petición fallida.
- **La recuperación no es transparente.** En el total de cada corrida, entre el 5 % y el 11 % de las peticiones nunca recibió respuesta; mirando solo después del corte, fue más de un tercio (36 % en L1 y 45,5 % en L2). Entre el 22 % y el 24 % de las peticiones de R2 a R5 falló en el primer intento. Las peticiones que sí se reintentaron tardaron unos 800 ms, el tiempo máximo de espera por intento del cliente.
- **Las cifras globales subestiman el problema.** Dependen de cuándo se corta el cable: un corte tardío diluye las fallas en muchas peticiones previas correctas. Por eso las corridas con corte a los ~16 s (R4, R5) y a los 30 s (L1, L2) no son comparables por su porcentaje global.
- **Causa de la pérdida.** Mientras Kubernetes no marca como no disponible al nodo apagado (unos 40 s después del corte), kube-proxy sigue repartiendo parte del tráfico hacia el pod que ya no responde.

Con 3 intentos y probabilidad de fallo del 50 % por intento, se pierde cerca de 12,5 % de las peticiones, cifra cercana a lo medido en las corridas globales. Con Local ese reparto no ocurre y la serie B no perdió ninguna petición (sección 5.6); en L1 y L2 no hubo mejora porque el Service seguía en Cluster.

- **Dispersión entre corridas.** El éxito global varía entre 88,70 % y 95,09 % y la mayor ventana entre 2,6 s y 7,5 s. Con una sola corrida por condición no se puede decir cuánto de eso es azar.

5.5 Reemplazo del pod, operación sin Internet y reincorporación (P05, P06, P09)

Reemplazo bajo anti-afinidad estricta (P05)

Con worker2 en NotReady, el pod de worker2 pasó a Terminating y el ReplicaSet creó un reemplazo que quedó en Pending, sin nodo asignado (76 s de antigüedad en la captura), mientras el pod de worker1 siguió Running con 1/1. Se confirma lo previsto: con anti-afinidad estricta el reemplazo no se ubica en el trabajador que ya tiene una réplica, y la capacidad baja a una réplica mientras dure la falla.

```
edgeuser@master:~$ sudo kubectl -n edge get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
edge-infer-654c5b467-f8t16          1/1     Running   0           28h   10.42.1.28      worker2   <none>             <none>
edge-infer-654c5b467-scz74          1/1     Running   0           21m   10.42.2.33      worker1   <none>             <none>
edgeuser@master:~$ sudo kubectl get nodes
NAME                STATUS   ROLES    AGE   VERSION
master              Ready    control-plane   5d3h   v1.36.5+k3s1
worker1             Ready    <none>         5d3h   v1.36.5+k3s1
worker2             NotReady <none>         5d3h   v1.36.5+k3s1
edgeuser@master:~$ sudo kubectl -n edge get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
edge-infer-654c5b467-f8t16          1/1     Terminating 0       28h   10.42.1.28      worker2   <none>             <none>
edge-infer-654c5b467-scz74          1/1     Running      0       23m   10.42.2.33      worker1   <none>             <none>
edge-infer-654c5b467-w5c6g          0/1     Pending      0       76s   <none>          <none>    <none>             <none>
```

Figura 6. `kubectl get nodes` y `kubectl -n edge get pods -o wide` con worker2 caído: el pod de worker2 en Terminating y el reemplazo en Pending sin nodo.

Operación sin Internet (P06)

Con el cable hacia el router desconectado (según el equipo, la prueba se hizo 100 % sin conexión), `simulator.py` envió 30 lecturas a las IP de los dos trabajadores y todas recibieron respuesta, con latencias de 11,7 a 41,3 ms y alternando entre ambos nodos. La clasificación avanzó de normal (hasta los 8 s) a desgaste (9 a 18 s) y a falla_inminente (desde los 19 s, con una lectura que volvió a desgaste en t = 20 s), con alerta en todas las lecturas desde t = 9 s (figura D5).

Reincorporación del nodo (P09)

Al reconectar el cable, `kubectl get nodes -w` muestra a worker2 pasando de NotReady a Ready (figura D6). El tiempo hasta Ready no se registró.

5.6 Serie comparativa A/B (P07)

Para medir el efecto de `externalTrafficPolicy`: Local frente a Cluster se hicieron 3 corridas por configuración, de 150 s, cortando el cable a los ~30 s y alternando los trabajadores (worker1, worker2, worker1). La política se verificó con `kubectl -n edge get svc edge-infer -o jsonpath='{.spec.externalTrafficPolicy}'`: el Service estaba en Cluster y se cambió a Local aplicando el manifiesto del kit, tras lo cual el comando devolvió Local (figura D7).

Tabla 5. Series A (Cluster) y B (Local); cifras aproximadas leídas de las gráficas

Corrida	Política	Worker apagado	Respuestas correctas en 150 s	Peticiones sin respuesta	Primera y última falla	Éxito final (aprox.)
A1	Cluster	worker1	≈ 280	13	33,8 s a 75,3 s	≈ 95,6 %
A2	Cluster	worker2	≈ 267	13	31,2 s a 71,9 s	≈ 95,4 %
A3	Cluster	worker1	≈ 280	11	32,7 s a 75,1 s	≈ 96,2 %

Corrida	Política	Worker apagado	Respuestas correctas en 150 s	Peticiones sin respuesta	Primera y última falla	Éxito final (aprox.)
B1	Local	worker1	≈ 362	0	—	100 %
B2	Local	worker2	≈ 362	0	—	100 %
B3	Local	worker1	≈ 362	0	—	100 %

Las cifras se leyeron de las gráficas (cruces rojas contadas y valor final de la curva acumulada, con un error de unas pocas respuestas y de $\pm 0,5$ s); el éxito final se calculó como respuestas correctas entre correctas más fallidas. Las cifras exactas se obtienen con `python analizar_csv.py archivo.csv --cut-at 30` sobre el CSV de cada corrida; los tiempos hasta NotReady y hasta el pod nuevo Running no se registraron.

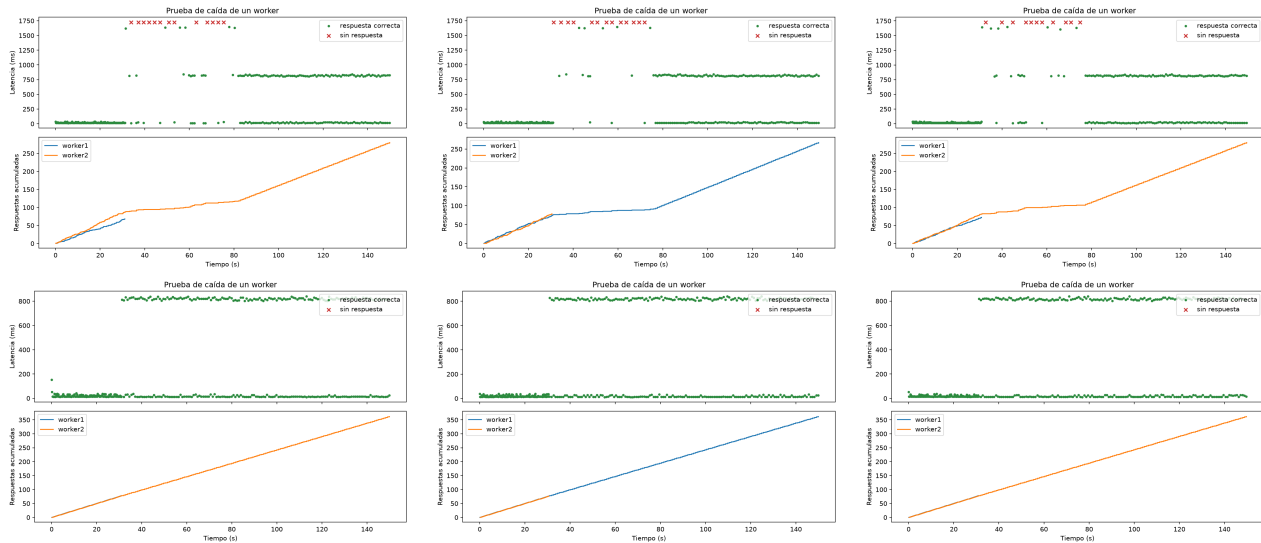


Figura 7. Gráficas de las corridas A1, A2 y A3 (Cluster, fila superior) y B1, B2 y B3 (Local, fila inferior). Con Cluster aparecen cruces rojas entre los ~31 s y los ~75 s; con Local no hay ninguna.

Lectura. Con Cluster hubo entre 11 y 13 peticiones sin respuesta por corrida, concentradas entre unos 31 y 75 s, es decir, alrededor de 40 s después del corte, coherente con el `node-monitor-grace-period` de 40 s; el total de respuestas en 150 s fue de unas 267 a 280. Con Local no se perdió ninguna petición en las tres corridas y el total subió a unas 362, porque el cliente no se bloquea con peticiones fallidas. En ambas series, cerca de la mitad de las peticiones posteriores al corte tarda unos 815 ms: es el intento fallido contra la IP del nodo apagado (0,8 s de espera) seguido del reintento al nodo vivo, y ocurre porque el cliente sigue rotando entre las dos IP, no por el Service. Es lo que predecía el modelo simple (100 % de éxito final y 50 % al primer intento con Local). El resultado exige que el cliente conozca las IP de ambos trabajadores: un cliente con una sola IP seguiría fallando si ese nodo cae.

En estas gráficas el ritmo anterior al corte es de unas 2,5 peticiones por segundo (unas 78 respuestas en los primeros 31 s), menor que en las pruebas anteriores. Son tres corridas por condición, suficientes para ver la diferencia pero no para estimar intervalos de confianza.

5.7 Consumo y recursos (P08)

Memoria medida con `free -m` en los tres nodos (figura D4), con el pod de inferencia en ejecución, en reposo y con el simulador corriendo.

Nodo	RAM en reposo (con pod)	RAM con el simulador corriendo	Potencia en reposo / con carga
master	982 MB usados de 1 571 (588 disponibles); 100 MB en swap	1 003 MB usados (567 disponibles); 100 MB en swap	no medida
worker1	577 MB usados (993 disponibles)	583 MB usados (987 disponibles)	no medida
worker2	579 MB usados (991 disponibles)	580 MB usados (990 disponibles)	no medida

El simulador casi no cambia el uso de memoria (de 0 a 21 MB más). Los trabajadores usan unos 580 MB de 1 571 MB (alrededor del 37 %); el master usa entre 982 y 1 003 MB (cerca del 63 %) y ya tiene 100 MB en swap, así que el plano de control es el nodo con menos margen. Una medición anterior de worker1 dio 409 MB usados.

5.8 Cumplimiento de los requisitos

Requisito	Estado	Evidencia
Hardware permitido (x86 de bajo consumo)	Cumple	Tres mini PC Intel Atom x5-Z8350 (sección 3.3).
IP estáticas en subred privada de clase C	Cumple	192.168.1.26, .27 y .28 con Netplan (sección 4.1).
Switch Gigabit Ethernet	Cumple	Switch Intellinet de 5 puertos Gigabit; ethtool reporta 1000 Mb/s en master (enp1s0), worker1 y worker2 (anexo D).
Orquestación ligera (K3s)	Cumple	Master y dos trabajadores (sección 4.3).
Inferencia de IA local, sin nube	Cumple	Modelo dentro de los pods; imagen precargada (sección 4.4). Una corrida de 30 lecturas se completó con el cable al router desconectado (sección 5.5).
Sobrevivir a la pérdida de un trabajador	Cumple	Con Local y un cliente que reintenta entre ambos nodos: 0 peticiones sin respuesta en 3 corridas; con Cluster, 11 a 13 por corrida (sección 5.6).
Reparto de carga y redistribución documentados	Cumple	Reparto entre ambos trabajadores (curvas acumuladas, 5.4) y comportamiento tras la caída documentado (5.5 y 5.6).
Caso de uso con datos simulados	Cumple	Simulador de vibración y temperatura (sección 3.6).

6. Problemas encontrados y soluciones

Durante el desarrollo se identificaron y resolvieron los siguientes inconvenientes:

#	Síntoma	Causa	Solución
1	Los pods quedaban en Pending.	El despliegue pedía role=worker y los nodos no tenían esa etiqueta.	Etiquetar los trabajadores: <code>kubectl label node worker1 role=worker (y worker2)</code> .
2	ErrImageNeverPull / ErrImagePull.	Se usó un comando inexistente (k3s image load) en lugar de importar al motor containerd.	<code>sudo k3s ctr images import edge-infer.tar</code> en cada trabajador.
3	Error de conexión con Docker Desktop (npope:///./pipe/dockerDesktopLinuxEngine).	El servicio de Docker Desktop no estaba iniciado en Windows.	Iniciar Docker Desktop antes de compilar.

#	Síntoma	Causa	Solución
4	Respuestas SIN RESPUESTA (HTTP 404) en simulator.py.	--endpoints agrega /predict por su cuenta; al pasar la URL ya con /predict se pedía /predict/predict.	Pasar solo la URL base, por ejemplo <code>http://192.168.1.27:30080</code> .
5	sudo: 'nano': command not found.	Ubuntu Server minimizado no trae editor.	Escribir los archivos con <code>cat << 'EOF' sudo tee /etc/netplan/01-netcfg.yaml</code> .
6	curl: (7) Failed to connect al NodePort.	service.yaml enviaba el tráfico al puerto 5000, pero el contenedor escucha en el 8000.	Alinear <code>containerPort: 8000</code> y <code>targetPort: 8000</code> .
7	ENDPOINTS: <none> en el servicio.	Tras recrear los pods, los selectores del servicio no coincidían con las etiquetas en ejecución.	Reaplicar el servicio con el selector <code>app: edge-infer</code> .
8	PermissionError: [Errno 13] al terminar failover_test.py.	El CSV de resultados estaba abierto en Excel, que bloquea la escritura.	Cerrar el archivo antes de iniciar la prueba.
9	Con un nodo apagado el servicio quedó sin pods (en una prueba, 26,8 s sin servicio).	El planificador dejó las dos réplicas en el mismo nodo tras un reinicio.	Primero <code>kubectl cordon</code> y reinicio controlado; después, anti-afinidad estricta con <code>maxSurge: 0</code> y <code>maxUnavailable: 1</code> .
10	El nodo aparecía como desconocido en las respuestas.	NODE_NAME no estaba definido en el manifiesto.	Injectar <code>spec.nodeName</code> con la Downward API.

Dos lecciones se repiten en esta lista: los errores más costosos fueron de **coherencia** (etiquetas, puertos y selectores que no coincidían) y de **distribución de réplicas**, no de código. Verificar con `kubectl get pods -o wide` y `kubectl get endpoints` después de cada cambio habría ahorrado la mayor parte del tiempo.

7. Limitaciones y trabajo futuro

7.1 Limitaciones

- **Datos sintéticos.** El modelo no se ha validado con mediciones de una máquina real. La exactitud de 95,33 % describe el desempeño sobre señales generadas por el mismo simulador que se usó para entrenar.
- **Punto único de falla en el control.** Hay un solo master; si cae, el clúster no puede programar ni reparar pods, aunque los pods en ejecución siguen sirviendo.
- **Pérdida durante la detección.** Hasta que Kubernetes detecta el nodo caído, una parte de las peticiones falla (sección 5.4). Con `externalTrafficPolicy: Local` no se observó pérdida en la serie B (sección 5.6).

- **Memoria limitada.** Cada nodo ve 1 571 MB de RAM; el servicio cabe con holgura, pero el master de K3s tiene poco margen.
- **Métrica global engañosa.** El resumen de `failover_test.py` promedia las peticiones anteriores al corte (todas correctas) con las posteriores, y tras la caída el cliente envía muchas menos peticiones por segundo. Por eso se reporta aparte lo ocurrido después del corte (sección 5.4).
- **Pocas repeticiones.** Cada condición tiene entre una y tres corridas; no se calcularon intervalos de confianza.
- **Equilibrio de carga sencillo.** Lo reparten el cliente y kube-proxy; no hay un balanceador externo dedicado. Con Local, el cliente debe conocer las IP de ambos trabajadores: si usara una sola y ese nodo cae, fallaría.
- **Sin monitoreo en el clúster.** No se desplegaron Prometheus ni Grafana para no sobrecargar la RAM y la CPU de los nodos; las mediciones salen de `failover_test.py`, `kubectl` y `free -m`.
- **Capacidad reducida durante la falla.** Con anti-afinidad estricta y dos trabajadores, durante una falla solo hay una réplica.

7.2 Trabajo futuro

- Repetir el entrenamiento y la evaluación con un conjunto público de vibraciones de rodamientos [7] y reportar esos resultados por separado.
- Aplicar y medir el resto de las mejoras de la sección 3.7 (sondas de readiness y liveness, tolerancias de 15 s); el efecto de `externalTrafficPolicy: Local` ya se midió en la sección 5.6.
- Añadir un tercer trabajador o ajustar la regla de anti-afinidad para que el reemplazo pueda ubicarse en el nodo que sigue vivo.
- Configurar más de un nodo de control para eliminar el punto único de falla.
- Implementar el dashboard de Grafana en un equipo aparte o con más memoria, y medir el consumo eléctrico.
- Poner una IP virtual o un balanceador externo (por ejemplo keepalived o MetalLB) delante de los trabajadores, para que el cliente no dependa de conocer cada nodo.
- Comparar con Raspberry Pi (ARM64) en rendimiento por vatio y por dólar.

8. Conclusiones

- **Viabilidad.** Es posible ejecutar un servicio de inferencia en un clúster K3s sobre mini PC de 2 GB de RAM, con un modelo pequeño (regresión logística de 1,3 KB) que responde en decenas de milisegundos de extremo a extremo.
- **Tolerancia a fallas.** Con la política Cluster (por defecto) el servicio sobrevive a la desconexión de un trabajador, pero pierde entre 11 y 13 peticiones por corrida durante unos 40 s (entre el 5 % y el 11 % en las corridas anteriores). Con `externalTrafficPolicy: Local` y un cliente que reintenta entre ambos nodos no se perdió ninguna petición en tres corridas, a costa de unos 815 ms en cerca de la mitad de las peticiones mientras el cliente siga probando el nodo apagado.
- **Operación local.** Con IP estáticas y la imagen precargada en cada nodo, la inferencia no depende de Internet una vez instalado el clúster: la prueba P06 completó 30 lecturas con el cable al router desconectado.

- **Elección del modelo.** Un modelo muy pequeño fue suficiente para este caso y dejó margen de memoria. La conclusión depende de que los datos sean sintéticos: con datos reales habría que validarlo de nuevo.
- **Aprendizajes.** Etiquetas de nodos, importación de imágenes a containerd, puertos coherentes y distribución de réplicas en nodos distintos fueron los puntos que más tiempo costaron (capítulo 6). Medir antes de afirmar cambió varias conclusiones del documento inicial.

9. Organización del equipo y herramientas

Rol	Responsabilidad	Integrante
Coordinación	Entrega del informe y el calendario	Pablo Javier Sandoval
Infraestructura y red	Montaje, IP estáticas, K3s, pruebas de caída	Kleiver Jedidias García Guarcas
IA y datos	Simulador, entrenamiento y evaluación del modelo	Allan David Alessandro Racancoj Lopez
Pruebas y documentación	Mediciones, capturas, redacción del informe	Ricardo Ismael Pérez Ajanel

Declaración sobre herramientas. El equipo contó con un kit de código de apoyo (simulador, entrenamiento, servicio, pruebas y manifiestos) preparado con ayuda de un asistente de inteligencia artificial (Claude, de Anthropic). El equipo desplegó el kit en su propio clúster, ejecutó las pruebas y midió los resultados.

Referencias

- [1] K3s Documentation. <https://docs.k3s.io>
- [2] Kubernetes Documentation: Services, Load Balancing, and Networking. <https://kubernetes.io/docs/concepts/services-networking/>
- [3] Kubernetes Documentation: Taints and Tolerations. <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>
- [4] Kubernetes Documentation: kube-controller-manager (parámetro node-monitor-grace-period). <https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/>
- [5] scikit-learn User Guide: Logistic regression. https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- [6] Randall, R. B., & Antoni, J. (2011). Rolling element bearing diagnostics: A tutorial. *Mechanical Systems and Signal Processing*, 25(2), 485–520.
- [7] Case Western Reserve University Bearing Data Center. <https://engineering.case.edu/bearingdatacenter>
- [8] Prometheus Documentation. <https://prometheus.io/docs/>
- [9] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5), 637–646.

Anexo A. Estructura del código

```
edge-cluster-kit/
edgekit/          signals.py, features.py, client.py
```

service/	app.py, Dockerfile, requirements.txt
k8s/	deployment.yaml, service.yaml
monitoring/	prometheus.yml, docker-compose.yml
docs/	plan_de_red.md, dashboard.md
tests/	test_pipeline.py (8 pruebas unitarias)
train.py	entrenamiento y comparación de modelos
simulator.py	simulador de sensores en vivo
failover_test.py	prueba de caída: CSV, gráfica y resumen
artifacts/	model.joblib, metrics.json, confusion_matrix.png

Resumen: edgekit contiene el simulador de señales, la extracción de características y el cliente con reintentos; service el servicio y su imagen; k8s los manifiestos; monitoring la configuración de Prometheus y Grafana.

Anexo B. Comandos útiles

Comando	Para qué sirve
kubectl get nodes -o wide	Estado e IP de los nodos (Ready o NotReady).
kubectl -n edge get pods -o wide	Dónde corre cada pod y en qué estado está.
kubectl -n edge get endpoints edge-infer	A qué pods envía tráfico el Service.
kubectl get nodes -w	Observar en vivo los cambios de estado de los nodos.
kubectl -n edge describe pod <pod>	Eventos y motivos de un pod en Pending o con errores.
kubectl -n edge rollout restart deploy/edge-infer	Reiniciar el despliegue para repartir de nuevo los pods.
kubectl apply -f k8s/service.yaml	Aplicar un cambio en el Service.
sudo k3s ctr images ls	Ver las imágenes importadas en un nodo.
ethtool enp1s0	Velocidad negociada de la tarjeta de red.
free -m	Memoria usada y disponible.

Anexo C. Glosario

Término	Significado
Edge Computing	Procesar los datos cerca de donde se generan, en lugar de enviarlos a un centro de datos remoto.
Clúster	Conjunto de equipos que trabajan juntos como un solo sistema.
K3s	Distribución ligera de Kubernetes, instalada como un único binario.
Pod	Unidad mínima de ejecución de Kubernetes; contiene uno o más contenedores.
NodePort	Tipo de Service que abre un puerto fijo en todos los nodos.
kube-proxy	Componente que reparte el tráfico de un Service entre sus pods.
Failover	Paso automático del trabajo a otro componente cuando uno falla.
BPFO	Frecuencia con que los elementos rodantes pasan por un defecto de la pista externa de un rodamiento.
RMS	Raíz del promedio de los cuadrados; mide la energía de una señal.

Término	Significado
FFT	Transformada rápida de Fourier; descompone una señal en frecuencias.
F1 macro	Promedio de la F1 de cada clase; combina precisión y exhaustividad sin favorecer a las clases más numerosas.

Anexo D. Evidencias del entorno

Capturas tomadas en el clúster: versiones y estado de los nodos y pods (master), velocidad de red y memoria de los trabajadores, y dirección de la laptop.

```

edgeuser@master: ~
Welcome to Ubuntu 26.04.1 LTS (GNU/Linux 7.0.0-38-generic x86_64)

 * Documentation:  https://docs.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Oct 11 04:07:59 2026 from 192.168.1.50
edgeuser@master:~$ k3s --version
k3s version v1.36.5+k3s1 (3dd98cc5)
go version go1.26.8
edgeuser@master:~$ sudo kubectl get nodes -o wide
[sudo: authenticate] Password:
NAME          STATUS    ROLES          AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE             KERNEL-VERSI
master        Ready     control-plane   3d22h v1.36.5+k3s1 192.168.1.26   <none>          Ubuntu 26.04.1 LTS   7.0.0-38-gen
eric (amd64)  Ready     <none>          3d22h v1.36.5+k3s1 192.168.1.27   <none>          Ubuntu 26.04.1 LTS   7.0.0-38-gen
eric (amd64)  Ready     <none>          3d22h v1.36.5+k3s1 192.168.1.28   <none>          Ubuntu 26.04.1 LTS   7.0.0-38-gen
worker2       Ready     <none>          3d22h v1.36.5+k3s1 192.168.1.28   <none>          Ubuntu 26.04.1 LTS   7.0.0-38-gen
eric (amd64)  Ready     <none>          3d22h v1.36.5+k3s1 192.168.1.28   <none>          Ubuntu 26.04.1 LTS   7.0.0-38-gen
edgeuser@master:~$ sudo kubectl -n edge get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE          NOMINATED NODE   READINESS GA
tes
edge-infer-654c5b467-22zq6  1/1     Running   3 (170m ago)  28h   10.42.1.26   worker2       <none>            <none>
edge-infer-654c5b467-wpfs4  1/1     Running   3 (170m ago)  28h   10.42.2.30   worker1       <none>            <none>
edgeuser@master:~$

```

Figura D1. Master: `k3s --version`, `kubectl get nodes -o wide` (tres nodos Ready, Ubuntu 26.04.1 LTS, containerd 2.3.4) y `kubectl -n edge get pods -o wide` (un pod por trabajador).

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Wlaver> ssh edgeuser@192.168.1.27
edgeuser@192.168.1.27:~$ pwd
Welcome to Ubuntu 26.04.1 LTS (GNU/Linux 7.0.0-38-generic x86_64)

 * Documentation:  https://docs.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Oct 11 04:08:05 2026 from 192.168.1.50
edgeuser@worker1:~$ ethtool enp1s0 | grep -i speed
netlink error: Operation not permitted
Speed: 1000Mb/s
edgeuser@worker1:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         489         548           2         638        1162
Swap:            0              0          2047
edgeuser@worker1:~$

```

```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Wlaver> ssh edgeuser@192.168.1.28
edgeuser@192.168.1.28:~$ pwd
Welcome to Ubuntu 26.04.1 LTS (GNU/Linux 7.0.0-38-generic x86_64)

 * Documentation:  https://docs.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.
Last login: Sun Oct 11 04:08:11 2026 from 192.168.1.50
edgeuser@worker2:~$ ethtool enp1s0 | grep -i speed
netlink error: Operation not permitted
Speed: 1000Mb/s
edgeuser@worker2:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         578         561           2         639        992
Swap:            0              0          2047
edgeuser@worker2:~$

```

Figura D2. worker1 (izquierda) y worker2 (derecha): `ethtool enp1s0` (Speed: 1000Mb/s) y `free -m`.

```

Microsoft Windows [Versión 10.0.26280.9457]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Wlaver> ipconfig

Configuración IP de Windows

Adaptador de LAN inalámbrica Conexión de área local* 1:
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de LAN inalámbrica Conexión de área local* 10:
Estado de los medios. . . . . : medios desconectados
Sufijo DNS específico para la conexión. . . :

Adaptador de Ethernet Ethernet:
Sufijo DNS específico para la conexión. . . :
Vínculo: dirección IPv6 local. . . : fe80::93b2:861d:1571:77ad%37
Dirección IPv4. . . . . : 192.168.1.50
Máscara de subred. . . . . : 255.255.255.0
Puerta de enlace predeterminada. . . . . :

```

Figura D3. Laptop: `ipconfig` con IPv4 192.168.1.50 y máscara 255.255.255.0 en el adaptador Ethernet.

```
edgeuser@master:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         982         132          2         604         588
Swap:          2047         100         1947

edgeuser@worker1:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         577         501          2         642         993
Swap:          2047           0         2047

edgeuser@worker2:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         579         497          2         644         991
Swap:          2047           0         2047

edgeuser@master:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         103         112          2         603         567
Swap:          2047         100         1947

edgeuser@worker1:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         583         498          2         640         987
Swap:          2047           0         2047

edgeuser@worker2:~$ free -m
              total        used        free      shared  buff/cache   available
Mem:           1571         580         496          2         644         990
Swap:          2047           0         2047

edgeuser@master:~$ sudo ethtool enp1s0 | grep -i speed
Speed: 1000Mb/s
```

Figura D4. `free -m` en master, worker1 y worker2 en reposo (tres primeras) y con el simulador corriendo (tres siguientes), y `ethtool enp1s0` del master (Speed: 1000Mb/s).

```
PS C:\Users\Wleiver\Desktop\edge-cluster-kit> python simulator.py --endpoint
s http://192.168.1.27:30808 http://192.168.1.28:30808 --duration 30
t(s) sev.real temp(C) prediction node ns
0.0 0.00 50.6 normal worker1 41.3
1.0 0.03 48.9 normal worker2 25.4
2.0 0.07 47.0 normal worker1 12.7
3.0 0.10 45.6 normal worker2 14.7
4.0 0.13 46.9 normal worker1 14.0
5.0 0.17 48.2 normal worker2 14.1
6.0 0.20 53.4 normal worker1 12.3
7.0 0.23 52.3 normal worker2 36.1
8.0 0.27 53.2 normal worker1 34.0
9.0 0.30 53.0 desgaste worker2 15.3 <-- ALERTA
10.0 0.33 54.3 desgaste worker1 13.0 <-- ALERTA
11.0 0.37 61.7 desgaste worker2 14.6 <-- ALERTA
12.0 0.40 59.9 desgaste worker1 13.3 <-- ALERTA
13.0 0.43 56.7 desgaste worker2 12.9 <-- ALERTA
14.0 0.47 52.8 desgaste worker1 11.7 <-- ALERTA
15.0 0.50 60.6 desgaste worker2 13.8 <-- ALERTA
16.0 0.53 63.9 desgaste worker1 25.0 <-- ALERTA
17.0 0.57 54.7 desgaste worker2 21.2 <-- ALERTA
18.0 0.60 59.2 desgaste worker1 13.7 <-- ALERTA
19.0 0.63 60.0 falla_inminente worker2 19.7 <-- ALERTA
20.0 0.67 67.8 desgaste worker1 14.6 <-- ALERTA
21.0 0.70 72.0 falla_inminente worker2 15.3 <-- ALERTA
22.0 0.73 67.8 falla_inminente worker1 12.9 <-- ALERTA
23.0 0.77 64.2 falla_inminente worker2 14.7 <-- ALERTA
24.0 0.80 71.6 falla_inminente worker1 17.6 <-- ALERTA
25.0 0.83 75.2 falla_inminente worker2 13.2 <-- ALERTA
26.0 0.87 77.3 falla_inminente worker1 13.5 <-- ALERTA
27.0 0.90 71.0 falla_inminente worker2 14.5 <-- ALERTA
28.0 0.93 76.4 falla_inminente worker1 23.8 <-- ALERTA
29.0 0.97 75.1 falla_inminente worker2 16.1 <-- ALERTA

30 lecturas enviadas.
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
PS C:\Users\Wleiver\Desktop\edge-cluster-kit>
```

```
edgeuser@master:~$ sudo kubectl get nodes -w
NAME        STATUS    ROLES    AGE   VERSION
master      Ready     control-plane 5d3h  v1.36.5+k3s1
worker1     Ready     <none>      5d3h  v1.36.5+k3s1
worker2     NotReady  <none>      5d3h  v1.36.5+k3s1
worker2     Ready     <none>      5d3h  v1.36.5+k3s1
worker2     Ready     <none>      5d3h  v1.36.5+k3s1
worker2     Ready     <none>      5d3h  v1.36.5+k3s1
master      Ready     control-plane 5d3h  v1.36.5+k3s1
```

Figura D5. Prueba sin Internet (P06): salida de `simulator.py` con 30 lecturas. Figura D6 (derecha): `kubectl get nodes -w` con worker2 pasando de NotReady a Ready (P09).

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Wleiver> cat edge-cluster-kit\service.yaml
apiVersion: v1
kind: Service
metadata:
  name: edge-infer
  namespace: edge
spec:
  type: NodePort
  selector:
    app: edge-infer
  ports:
    - port: 8000
      targetPort: 8000
      nodePort: 30808

edgeuser@master:~$ sudo kubectl -n edge get svc edge-infer -o jsonpath='{.spec.externalTrafficPolicy}'
[Empty authentication response]
Clusteredgeuser@master:~$ sudo kubectl -n edge get svc edge-infer -o jsonpath='{.spec.externalTrafficPolicy}'
service/edge-infer unchanged
Clusteredgeuser@master:~$ sudo kubectl -n edge get svc edge-infer -o jsonpath='{.spec.externalTrafficPolicy}'
Clusteredgeuser@master:~$
```

```
edgeuser@master:~$ cat << 'EOF' > service.yaml
apiVersion: v1
kind: Service
metadata:
  name: edge-infer
  namespace: edge
spec:
  type: NodePort
  selector:
    app: edge-infer
  ports:
    - port: 8000
      targetPort: 8000
      nodePort: 30808

edgeuser@master:~$ sudo kubectl apply -f service.yaml
service/edge-infer configured
Clusteredgeuser@master:~$ sudo kubectl -n edge get svc edge-infer -o jsonpath='{.spec.externalTrafficPolicy}'
Local-edgeuser@master:~$
```

Figura D7. Política del Service: `CLuster` al consultarla (izquierda) y `Local` tras aplicar el manifiesto con `externalTrafficPolicy: Local` (derecha).