



Universidad Mariano Gálvez de Guatemala
Facultad de Ingeniería en Sistemas de Información
Arquitectura de Computadoras II | Sección A | Grupo 2
Ing. Alex García

Proyecto Final

Mantenimiento predictivo industrial con K3s
Simulación de temperatura y vibración en una Raspberry Pi y dos máquinas virtuales

Integrantes

Nombre completo	Carné
Estuardo Alexander López Salvatierra	0900-23-15304
Otto Abraham Paz Letona	0900-23-15070
Elmer Eduardo Istupe Ruiz	0900-21-10969

Guatemala, 16 de octubre de 2026

Introducción

Este proyecto construye un sistema local para observar el comportamiento de una máquina industrial mediante lecturas simuladas de temperatura y vibración. La Raspberry Pi genera los datos, ejecuta un modelo que identifica valores fuera del comportamiento esperado y envía cada resultado al resto del clúster. Un panel en Grafana permite ver cómo cambia la temperatura y reconocer picos que merecen revisión.

En mantenimiento industrial, esperar a que una máquina deje de funcionar puede detener el trabajo sin aviso. Revisarla solamente por calendario tampoco muestra con precisión qué ocurre entre una inspección y otra. Observar variables como temperatura y vibración permite detectar cambios y decidir cuándo conviene investigar un equipo. En esta práctica no se emite una orden automática de mantenimiento: se demuestra la captura, el procesamiento y la consulta de datos que podrían apoyar esa decisión.

El procesamiento se mantiene en una red local. Para hacerlo posible, el equipo conectó una Raspberry Pi física y dos máquinas virtuales Ubuntu mediante K3s. El ingeniero aprobó esta combinación: la máquina virtual master-node coordina el clúster, worker1-node aloja los servicios de datos y visualización, y worker2-pi ejecuta el simulador y clasificador. Esta separación reproduce la idea de procesar cerca de la fuente de datos y presentar el resultado en otro nodo.

El informe sigue el mismo recorrido que una lectura: se genera en la Raspberry, se clasifica, se publica en MQTT, se guarda en InfluxDB y se consulta desde Grafana. Después se documentan la persistencia local del panel y la prueba de reemplazo del Pod de Mosquitto. Las figuras corresponden a las capturas recibidas; las direcciones que no figuran en ellas y el ejercicio de tiempos se identifican expresamente como datos ilustrativos.

Objetivos

Objetivo general. Implementar un clúster K3s local para generar, clasificar, guardar y visualizar datos simulados de temperatura y vibración, y comprobar la recuperación de un servicio ante la eliminación de su Pod.

- Integrar las dos máquinas virtuales y la Raspberry Pi en un mismo clúster.
- Simular lecturas de sensores y clasificar comportamientos normales o anómalos en el nodo de borde.
- Transportar eventos por MQTT, almacenarlos en InfluxDB y consultarlos desde Grafana.
- Verificar la ubicación de las cargas de trabajo y el reemplazo de un Pod eliminado.

Alcance de la demostración

El equipo utiliza datos generados por software; las capturas no muestran un sensor físico conectado al motor. El panel entregado visualiza temperatura, mientras que el registro enviado y guardado incluye también vibración y el estado de clasificación. La prueba de recuperación presentada consiste en eliminar el Pod del broker MQTT y observar que K3s crea otro. La caída de un worker completo y la redundancia del maestro requieren pruebas distintas.

Términos utilizados

Término	Significado en este montaje
Nodo	Equipo físico o virtual que participa en el clúster.
Pod	Unidad que ejecuta uno o más contenedores dentro de K3s.
Deployment	Recurso que declara cuántas réplicas de un Pod deben existir.
Service	Nombre y dirección estable que usan los Pods para comunicarse.
Edge	Procesamiento realizado en worker2-pi, cerca del origen simulado.

Metodología del montaje

El trabajo se desarrolló por etapas para comprobar cada parte antes de depender de la siguiente. Primero se prepararon los equipos y su comunicación en la LAN. Después se validó el clúster y la asignación de trabajo a cada worker. Con esa base se desplegó la aplicación, se revisó el recorrido de los mensajes y finalmente se provocó la caída de un Pod. Este orden permite ubicar un error sin tener que atribuirlo de inmediato a Grafana o al modelo de clasificación.

Etapa	Acción realizada	Evidencia que la respalda
Infraestructura	Integrar las dos VMs Ubuntu y la Raspberry mediante K3s.	Lista de tres nodos en estado Ready.
Programación	Comprobar la ubicación de Pods de prueba en ambos workers.	Columna NODE en las capturas de terminal.
Aplicación	Crear el namespace y desplegar sensor, broker, persistidor, base y panel.	Manifiestos y lista final de cinco Pods.
Datos	Publicar MQTT, almacenar los mensajes y consultar la serie.	Logs, suscripción, fuente de datos y dashboard.
Resiliencia	Eliminar el Pod del broker y observar su reemplazo.	Transición de estados y nuevo Pod Running.

Las capturas proceden de la ejecución del equipo y se conservan como evidencia de lo que se observó, no como sustituto de una medición que todavía no se realizó. El informe distingue la captura del código, la ejecución visible y las conclusiones que pueden extraerse de cada prueba. Los ejemplos numéricos añadidos para explicar la red y los tiempos se presentan separados de esos resultados.

Se eligió una Raspberry como nodo de borde porque permite ejecutar el simulador cerca del origen que se representaría con sensores reales. Las VMs facilitan alojar el servidor K3s y los servicios de datos sin exigir tres equipos físicos. La elección ya fue aprobada por el docente y forma parte de la implementación presentada. En este montaje, el resultado visible es un prototipo educativo, no una instalación conectada a una máquina industrial.

1 Arquitectura del clúster

K3s es una distribución ligera de Kubernetes que permite coordinar contenedores en equipos con recursos diferentes. En este montaje, el master es el plano de control. Los dos workers reciben las aplicaciones. Usar máquinas virtuales para el control y los servicios, y una Raspberry Pi como nodo de borde, permitió integrar arquitecturas x86_64 y ARM sin depender de servicios de cómputo en la nube.

El master mantiene la información del clúster y recibe los comandos de administración; no es el punto por el que cada lectura debe pasar obligatoriamente. Cuando ya están desplegados los componentes, el simulador se comunica con el Service de Mosquito y el persistidor se conecta a los Services de Mosquito e InfluxDB. Grafana consulta InfluxDB. Esa separación evita confundir el plano de control de K3s con el camino de los datos de sensores.

Equipo	Papel en el proyecto	Componentes observados
master-node (VM)	Servidor K3s y administración del clúster	API de Kubernetes y comandos kubectl
worker1-node (VM)	Servicios de datos y visualización	Mosquitto, InfluxDB, persistidor MQTT y Grafana
worker2-pi (Raspberry Pi)	Generación y análisis cerca del origen	edge-sensor-classifier

Los nodos se comunican mediante direcciones de una red local privada. La dirección mostrada en algunas capturas corresponde al entorno donde se hizo el montaje; al mover las VMs a otro equipo se debe verificar de nuevo la IP interna y confirmar que la Raspberry alcance al master.

La Raspberry y las VMs deben poder alcanzarse por la red local. Para unir un agente a K3s se usa la dirección del servidor y un token de registro; para usar Grafana desde un navegador se emplea la dirección del nodo y el puerto que expone NodePort. Dentro del clúster, los Pods usan nombres DNS de Services. Por eso un cambio de IP de la LAN obliga a revisar el acceso al master y al panel, aunque los nombres internos de los Services sigan representando los mismos componentes.

Nodo	IP ilustrativa de LAN	Comprobación documentada
master-node	192.168.1.100	Administración del clúster y API K3s
worker1-node	192.168.1.101	Ready y servicios de datos
worker2-pi	192.168.1.102	Ready y publicación MQTT

Estas tres direcciones se usan como ejemplo coherente de una misma subred; las capturas no permiten atribuir una IP concreta a cada nodo. La barra del navegador sí muestra 192.168.1.100:30300 como dirección de acceso a Grafana en una prueba. Un NodePort puede atender desde la IP de un nodo alcanzable, por lo que ese dato no identifica por sí solo qué equipo aloja el Pod de Grafana.

```
master@master-node:~$ sudo kubectl get nodes
[sudo: authenticate] Password:
NAME                STATUS    ROLES    AGE     VERSION
master-node         Ready    control-plane   4d21h   v1.36.5+k3s1
worker1-node        Ready    <none>         4d21h   v1.36.5+k3s1
worker2-pi          Ready    <none>         103s    v1.36.5+k3s1
master@master-node:~$
```

Figura 1. Los tres nodos, incluido worker2-pi, figuran en estado Ready.

```

master@master-node:~$ sudo kubectl label node worker1-node node-role.kubernetes.io/worker=worker
node/worker1-node labeled
master@master-node:~$ sudo kubectl label node worker2-pi node-role.kubernetes.io/worker=worker
node/worker2-pi labeled
master@master-node:~$ sudo kubectl get nodes
NAME                STATUS    ROLES    AGE      VERSION
master-node         Ready     control-plane  4d21h    v1.36.5+k3s1
worker1-node        Ready     worker      4d21h    v1.36.5+k3s1
worker2-pi          Ready     worker      3m53s    v1.36.5+k3s1
master@master-node:~$

```

Figura 2. Etiquetas de rol asignadas a los workers.

2 Preparación y validación de nodos

Primero se instaló K3s en el master y se incorporaron los agentes usando la dirección del servidor y un token de unión. Las capturas del equipo muestran los tres nodos activos. El token sirve para registrar agentes y no debe incluirse en el informe; por esa razón se presenta la verificación del clúster sin la captura que lo expone.

Se etiquetaron los workers para identificar dónde debe ejecutarse cada componente. El equipo lanzó un Pod de prueba en la Raspberry y verificó la columna NODE de kubectl. Luego hizo la misma comprobación en worker1 y retiró los Pods temporales. Esto valida que ambos trabajadores pueden recibir cargas de trabajo.

La ubicación importa porque las imágenes y los procesos deben ejecutarse en el nodo adecuado. En particular, el Deployment edge-sensor-classifier usa un selector para worker2-pi, mientras que la captura final sitúa los servicios de almacenamiento y visualización en worker1-node. La prueba de Pods temporales se hizo antes de desplegar todo el flujo para comprobar que K3s podía asignar trabajo a ambos workers.

```

master@master-node:~$ sudo kubectl get nodes
NAME                STATUS    ROLES    AGE      VERSION
master-node         Ready     control-plane  4d21h    v1.36.5+k3s1
worker1-node        Ready     worker      4d21h    v1.36.5+k3s1
worker2-pi          Ready     worker      6m59s    v1.36.5+k3s1
master@master-node:~$ sudo kubectl run test-pi --image=nginx --restart=Never --overrides='{ "spec": { "nodeName": "worker2-pi" } }'
pod/test-pi created
master@master-node:~$ sudo kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
test-pi     1/1    Running   0           17s   10.42.1.3   worker2-pi   <none>           <none>
master@master-node:~$

```

Figura 3. Pod de prueba situado en la Raspberry Pi.

```

master@master-node:~$ sudo kubectl run test-pi --image=nginx --restart=Never --overrides='{ "spec": { "nodeName": "worker2-pi" } }'
pod/test-pi created
master@master-node:~$ sudo kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
test-pi     1/1    Running   0           17s   10.42.1.3   worker2-pi   <none>           <none>
master@master-node:~$ sudo kubectl delete pod test-pi
pod "test-pi" deleted from default namespace
master@master-node:~$ sudo kubectl get pods -o wide
No resources found in default namespace.
master@master-node:~$ sudo kubectl run test-w1 --image=nginx --restart=Never --overrides='{ "spec": { "nodeName": "worker1-node" } }'
pod/test-w1 created
master@master-node:~$ sudo kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
test-w1     0/1    ContainerCreating   0           3s    <none>      worker1-node   <none>           <none>
master@master-node:~$ sudo kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
test-w1     0/1    ContainerCreating   0           18s    <none>      worker1-node   <none>           <none>
master@master-node:~$ sudo kubectl get pods -o wide
NAME        READY   STATUS    RESTARTS   AGE   IP          NODE        NOMINATED NODE   READINESS GATES
test-w1     1/1    Running   0           32s   10.42.2.5   worker1-node   <none>           <none>
master@master-node:~$ sudo kubectl delete pod test-w1
pod "test-w1" deleted from default namespace
master@master-node:~$ sudo kubectl get pods -o wide
No resources found in default namespace.
master@master-node:~$

```

Figura 4. Prueba de asignación y limpieza de Pods temporales en los dos workers.

También se creó el namespace predictivo. Este espacio agrupa los recursos del proyecto y permite distinguirlos de los servicios internos de Kubernetes. No crea una red física separada; ayuda a organizar Deployments, Pods y Services.

```

master@master-node:~$ sudo kubectl create namespace predictivo
namespace/predictivo created
master@master-node:~$ sudo kubectl get ns
NAME                STATUS   AGE
default             Active   4d22h
kube-node-lease     Active   4d22h
kube-public         Active   4d22h
kube-system         Active   4d22h
predictivo          Active   9s
master@master-node:~$

```

Figura 5. Namespace predictivo creado y visible en el clúster.

3 Simulación de sensores y clasificación local

En worker2-pi se desplegó edge-sensor-classifier. El contenedor usa Python, NumPy, scikit-learn y paho-mqtt. Cada dos segundos genera una lectura de temperatura y vibración alrededor de valores normales. En algunas iteraciones añade un salto grande a ambas variables para representar una posible falla mecánica. Así se puede probar el flujo completo sin esperar una avería real ni conectar un sensor físico.

El clasificador utiliza Isolation Forest, entrenado localmente con muestras sintéticas alrededor de 45 °C y 1.2 unidades de vibración. El parámetro contamination se fijó en 0.1, y la inyección de fallas en el generador tiene probabilidad de 0.05. El modelo marca cada lectura como NORMAL o FALLO_MECANICO. Es una demostración de detección de anomalías: esas etiquetas no equivalen a un diagnóstico industrial validado con datos de una máquina real.

Cómo se obtiene una lectura

El código crea una temperatura base con una distribución normal de media 45 y desviación 2, y una vibración base con media 1.2 y desviación 0.1. En aproximadamente el 5 % de las iteraciones se suma un incremento aleatorio de 20 a 35 a la temperatura y de 2.5 a 4.0 a la vibración. La semilla fija del modelo facilita repetir su entrenamiento, pero las lecturas generadas en ejecución cambian porque el simulador utiliza valores aleatorios.

El modelo se entrena con 100 pares de valores normales sintéticos y luego evalúa el par de la lectura actual. Isolation Forest devuelve una clasificación de valor usual o atípico con respecto a los datos de entrenamiento. El parámetro contamination=0.1 controla la proporción esperada de valores atípicos en esa referencia; no significa que exactamente una de cada diez lecturas de la simulación sea una falla. La inyección de picos y la decisión del modelo son dos operaciones distintas.

El resultado se redondea a dos decimales y se forma un mensaje con temperatura, vibración y estado_clasificado. En la evidencia hay lecturas con picos altos etiquetadas FALLO_MECANICO, y también una lectura cercana a la zona normal que recibe esa etiqueta. Esto muestra que la clasificación depende del modelo, no de un umbral fijo como “más de 65 °C”. La unidad de vibración no quedó definida en el código; antes de trabajar con un sensor real habría que fijar esa unidad y calibrarlo.

```
master@master-node:~/proyecto-predictivo$ nano edge-sensor-model.yaml
master@master-node:~/proyecto-predictivo$ cat edge-sensor-model.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: edge-sensor-classifier
  namespace: predictivo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: edge-classifier
  template:
    metadata:
      labels:
        app: edge-classifier
    spec:
      nodeSelector:
        kubernetes.io/hostname: worker2-pi
      containers:
      - name: sensor-simulator
        image: python:3.10-slim
        command: ["/bin/sh", "-c"]
        args:
        - |
          pip install paho-mqtt numpy scikit-learn
          cat << 'EOF' > app.py
          import time, random, json
          import paho.mqtt.client as mqtt
          from sklearn.ensemble import IsolationForest
          import numpy as np

          # Entrenamiento rápido de modelo local de detección de anomalías
          X_train = np.random.normal(loc=[45.0, 1.2], scale=[2.0, 0.1], size=(100, 2))
          model = IsolationForest(contamination=0.1, random_state=42)
          model.fit(X_train)

          client = mqtt.Client()
          client.connect("mosquitto-service.predictivo.svc.cluster.local", 1883, 60)

          while True:
            # Simulación de lectura de sensores
            temp = random.normalvariate(45.0, 2.0)
            vib = random.normalvariate(1.2, 0.1)

            # 5% de probabilidad de inyectar fallo mecánico en caliente
            if random.random() < 0.05:
              temp += random.uniform(20.0, 35.0)
              vib += random.uniform(2.5, 4.0)

            # Clasificación en tiempo real (Edge AI)
            pred = model.predict([[temp, vib]])[0] # 1: Normal, -1: Fallo/Anomalia
            estado = "NORMAL" if pred == 1 else "FALLO_MECANICO"

            payload = {
              "temperatura": round(temp, 2),
              "vibracion": round(vib, 2),
              "estado_clasificado": estado
            }

            client.publish("industrial/sensores/pi", json.dumps(payload))
            print(f"[EDGE API] Publicado: {payload}")
            time.sleep(2)

          EOF
          python app.py
master@master-node:~/proyecto-predictivo$ sudo kubectl apply -f edge-sensor-model.yaml
deployment.apps/edge-sensor-classifier created
```

Figura 6. Deployment que fija el simulador y clasificador en worker2-pi.

```
import paho.mqtt.client as mqtt
from sklearn.ensemble import IsolationForest
import numpy as np

# Entrenamiento rápido de modelo local de detección de anomalías
X_train = np.random.normal(loc=[45.0, 1.2], scale=[2.0, 0.1], size=(100, 2))
model = IsolationForest(contamination=0.1, random_state=42)
model.fit(X_train)

client = mqtt.Client()
client.connect("mosquitto-service.predictivo.svc.cluster.local", 1883, 60)

while True:
    # Simulación de lectura de sensores
    temp = random.normalvariate(45.0, 2.0)
    vib = random.normalvariate(1.2, 0.1)

    # 5% de probabilidad de inyectar fallo mecánico en caliente
    if random.random() < 0.05:
        temp += random.uniform(20.0, 35.0)
        vib += random.uniform(2.5, 4.0)

    # Clasificación en tiempo real (Edge AI)
    pred = model.predict([[temp, vib]])[0] # 1: Normal, -1: Fallo/Anomalia
    estado = "NORMAL" if pred == 1 else "FALLO_MECANICO"

    payload = {
      "temperatura": round(temp, 2),
      "vibracion": round(vib, 2),
      "estado_clasificado": estado
    }

    client.publish("industrial/sensores/pi", json.dumps(payload))
    print(f"[EDGE API] Publicado: {payload}")
    time.sleep(2)

EOF
python app.py
master@master-node:~/proyecto-predictivo$ sudo kubectl apply -f edge-sensor-model.yaml
deployment.apps/edge-sensor-classifier created
```

Figura 7. Generación de lecturas, predicción local y publicación periódica.

4 Transporte y almacenamiento de datos

El resultado de cada lectura se publica en el tema MQTT industrial/sensores/pi. El mensaje contiene temperatura, vibración y estado_clasificado. Mosquitto recibe estos mensajes dentro del namespace predictivo; la comunicación entre aplicaciones usa el nombre del Service del clúster y el puerto 1883. La captura de la suscripción confirma que los mensajes publicados llegan al broker.

Recorrido de una lectura

Paso	Componente	Qué ocurre
1	worker2-pi	Genera temperatura y vibración y obtiene NORMAL o FALLO_MECANICO.
2	MQTT / Mosquitto	Publica un JSON en industrial/sensores/pi y entrega el mensaje al suscriptor.
3	mqtt-persist	Lee el JSON, extrae los tres valores y prepara el punto de serie temporal.

4	InfluxDB	Guarda la medición sensores_edge para consultas por intervalo.
5	Grafana	Consulta InfluxDB y dibuja la temperatura en el panel.

MQTT desacopla al emisor de quien guarda los datos: la Raspberry publica en un tema y el persistidor se suscribe a ese tema. Ambos necesitan conocer el broker, pero el generador no escribe directamente en la base de datos. El mensaje que se observa en la terminal incluye, por ejemplo, una lectura de temperatura 69.82 y vibración 4.49 con estado FALLO_MECANICO; esa es una muestra capturada, no un valor permanente del sistema.

```
master@master-node:~/proyecto-predictivo$ cat mqtt-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mosquitto-broker
  namespace: predictivo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mosquitto
  template:
    metadata:
      labels:
        app: mosquitto
    spec:
      nodeSelector:
        kubernetes.io/hostname: worker1-node
      containers:
        - name: mosquitto
          image: eclipse-mosquitto:latest
          ports:
            - containerPort: 1883
---
apiVersion: v1
kind: Service
metadata:
  name: mosquitto-service
  namespace: predictivo
spec:
  type: ClusterIP
  ports:
    - port: 1883
  targetPort: 1883
  selector:
    app: mosquitto
master@master-node:~/proyecto-predictivo$
```

Figura 8. Manifiesto del Deployment y Service interno de Mosquitto.

```
master@master-node:~/proyecto-predictivo$ nano mqtt-deployment.yaml
master@master-node:~/proyecto-predictivo$ sudo kubectl apply -f mqtt-deployment.yaml
deployment.apps/mosquitto-broker unchanged
service/mosquitto-service created
master@master-node:~/proyecto-predictivo$
```

Figura 9. Creación del broker MQTT y de su Service.

```
master@master-node:~/proyecto-predictivo$ sudo kubectl logs -n predictivo -l app=edge-classifier -f
[EDGE RPi] Publicado: {'temperatura': 49.3, 'vibracion': 1.25, 'estado_clasificado': 'FALLO_MECANICO'}
[EDGE RPi] Publicado: {'temperatura': 42.5, 'vibracion': 1.19, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 44.6, 'vibracion': 1.26, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 45.45, 'vibracion': 1.17, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 47.09, 'vibracion': 1.23, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 75.64, 'vibracion': 3.91, 'estado_clasificado': 'FALLO_MECANICO'}
[EDGE RPi] Publicado: {'temperatura': 40.54, 'vibracion': 1.21, 'estado_clasificado': 'FALLO_MECANICO'}
[EDGE RPi] Publicado: {'temperatura': 45.95, 'vibracion': 1.23, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 42.89, 'vibracion': 1.26, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 48.46, 'vibracion': 1.29, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 50.38, 'vibracion': 1.03, 'estado_clasificado': 'FALLO_MECANICO'}
[EDGE RPi] Publicado: {'temperatura': 43.85, 'vibracion': 1.36, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 41.39, 'vibracion': 1.23, 'estado_clasificado': 'NORMAL'}
[EDGE RPi] Publicado: {'temperatura': 46.71, 'vibracion': 1.15, 'estado_clasificado': 'NORMAL'}
master@master-node:~/proyecto-predictivo$ sudo kubectl exec -it -n predictivo deployment/mosquitto-broker -- mosquitto_sub -t "industrial/sensores/p
i"
{"temperatura": 45.19, "vibracion": 1.1, "estado_clasificado": "NORMAL"}
{"temperatura": 44.27, "vibracion": 1.29, "estado_clasificado": "NORMAL"}
{"temperatura": 44.07, "vibracion": 1.27, "estado_clasificado": "NORMAL"}
{"temperatura": 69.82, "vibracion": 4.49, "estado_clasificado": "FALLO_MECANICO"}
{"temperatura": 41.86, "vibracion": 1.24, "estado_clasificado": "NORMAL"}
{"temperatura": 43.74, "vibracion": 1.1, "estado_clasificado": "NORMAL"}
{"temperatura": 38.33, "vibracion": 1.17, "estado_clasificado": "FALLO_MECANICO"}
{"temperatura": 46.24, "vibracion": 1.13, "estado_clasificado": "NORMAL"}
{"temperatura": 42.95, "vibracion": 1.24, "estado_clasificado": "NORMAL"}
{"temperatura": 80.06, "vibracion": 4.69, "estado_clasificado": "FALLO_MECANICO"}
master@master-node:~/proyecto-predictivo$
```

Figura 10. Publicación desde la Raspberry y recepción de mensajes MQTT.

5 Persistidor e InfluxDB

El proceso mqtt-persister se suscribe al mismo tema MQTT, convierte el JSON recibido y escribe cada lectura en InfluxDB. Usa la base mantenimiento, la medición sensores_edge, una etiqueta de nodo y el estado de clasificación. Los campos guardados incluyen temperatura, vibración y una señal numérica de anomalía. InfluxDB resulta adecuado aquí porque las lecturas forman una secuencia temporal que Grafana puede consultar por intervalos.

En el código visible, la etiqueta nodo toma el valor worker2-pi y la etiqueta estado conserva NORMAL o FALLO_MECANICO. La señal anomalía guarda 1 para FALLO_MECANICO y 0 para NORMAL. Esto permitiría filtrar las lecturas por estado, contar anomalías o construir más paneles. La captura demuestra la lógica de escritura y el despliegue del persistidor; la consulta satisfactoria de Grafana y el gráfico aportan la evidencia de que la ruta de almacenamiento y lectura funcionó durante la prueba.

```
master@master-node: ~/proj x master@worker1-node: ~ x ubuntu@worker2-pi: ~ x + v
import json
import paho.mqtt.client as mqtt
from influxdb import InfluxDBClient

db = InfluxDBClient(host='influxdb-service.predictivo.svc.cluster.local', port=8086, database='mantenimiento')

def on_message(client, userdata, msg):
    payload = json.loads(msg.payload.decode())
    temp = payload['temperatura']
    vib = payload['vibracion']
    estado = payload['estado_clasificado']
    anomalia_num = 1 if estado == "FALLO_MECANICO" else 0

    json_body = [
        {
            "measurement": "sensores_edge",
            "tags": {"nodo": "worker2-pi", "estado": estado},
            "fields": {
                "temperatura": float(temp),
                "vibracion": float(vib),
                "anomalía": int(anomalia_num)
            }
        }
    ]
    db.write_points(json_body)
    print(f"[PERSISTIR] Guardado en InfluxDB: {payload}", flush=True)

client = mqtt.Client()
client.on_message = on_message
client.connect("mosquitto-service.predictivo.svc.cluster.local", 1883, 60)
client.subscribe("industrial/sensores/pi")
client.loop_forever()
PYEOF
python persister.py

EOF

sudo kubectl apply -f mqtt-to-influx.yaml
deployment.apps/mqtt-persister created
master@master-node: ~/proyecto-predictivo$
```

Figura 11. Código del persistidor que recibe MQTT y escribe puntos en InfluxDB.

```
master@master-node: ~/proyecto-predictivo$ sudo kubectl get pods -n predictivo -o wide
NAME                                READY    STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
edge-sensor-classifier-6bc77f79d-snkzt 1/1      Running   0           23m   10.42.1.4       worker2-pi       <none>            <none>
grafana-58c7fd6bbc-sdh9g              1/1      Running   0           3m33s 10.42.2.7       worker1-node     <none>            <none>
influxdb-7ccb49b556-vq2xf              1/1      Running   0           3m33s 10.42.2.8       worker1-node     <none>            <none>
mosquitto-broker-6c597c6cf8-r9zqg      1/1      Running   0           28m   10.42.2.6       worker1-node     <none>            <none>
mqtt-persister-5b7df86d47-qgzw4         1/1      Running   0           2m9s  10.42.2.9       worker1-node     <none>            <none>
```

Figura 12. Distribución de los cinco Pods: el clasificador en worker2-pi y los demás en worker1-node.

La distribución observada separa el trabajo de borde del almacenamiento y la visualización. Los componentes aparecen con una réplica cada uno y ubicaciones fijadas por nodo. Esta organización de tareas es la distribución de carga implementada; las capturas no muestran balanceo de solicitudes entre réplicas equivalentes.

Distribución de carga y balanceo

El reparto de funciones reduce el trabajo que hace la Raspberry: allí se ejecutan la simulación y la clasificación, mientras worker1 asume el broker, la base de datos, el persistidor y Grafana. K3s permite programar estos Pods y el Service mantiene un punto de acceso estable para los clientes. Sin embargo, con una sola réplica del broker y de los demás servicios no hay varias instancias entre las cuales repartir

simultáneamente las solicitudes. Para demostrar un balanceo real de tráfico habría que aumentar réplicas donde la aplicación lo permita y medir el envío a cada una.

Comunicación entre componentes

Conexión	Destino	Uso
Agentes K3s → master	Dirección LAN del servidor	Unión de workers y administración del clúster.
Simulador → broker	mosquitto-service, puerto 1883	Publicación del JSON por MQTT.
Persistidor → InfluxDB	influxdb-service, puerto 8086	Escritura de cada lectura.
Grafana → InfluxDB	Service interno de InfluxDB	Consulta de la serie temporal.
Navegador → Grafana	IP LAN y NodePort	Acceso al panel desde otro equipo de la red.

El nombre de un Service evita escribir en el código la IP cambiante de un Pod. La publicación del puerto de Grafana permite que una persona abra el panel desde la LAN; los otros servicios pueden mantenerse como conexiones internas entre Pods. Al trasladar las VMs, se debe verificar la IP de los nodos y el acceso desde la Raspberry. La exposición fuera de la LAN no forma parte de la demostración.

Cada enlace puede verificarse con una evidencia diferente: estado Ready para la unión al clúster, logs para la publicación, suscripción MQTT para la recepción, prueba de la fuente en Grafana para la consulta y puntos recientes en el dashboard para el resultado. Una captura de un único Pod Running no demuestra por sí sola que todos los enlaces funcionen.

6 Panel de visualización

Grafana se expuso en la red local mediante un Service NodePort. El equipo abrió la interfaz desde el navegador, configuró InfluxDB como fuente de datos y probó la conexión a la base mantenimiento. El mensaje de éxito confirma que Grafana pudo consultar al menos una medición. La IP y el puerto que aparecen en la barra del navegador pertenecen al montaje de esa captura y deben comprobarse cuando las VMs funcionen en otra red.

Se distinguen dos direcciones en esta etapa. Grafana accede a InfluxDB con el nombre del Service interno y el puerto 8086, que resuelve dentro del clúster. En cambio, una persona entra a Grafana mediante una IP accesible en la LAN y el puerto publicado por NodePort. Si el navegador deja de abrir el panel después de trasladar las VMs, conviene revisar primero la conectividad y esa dirección externa; cambiar la IP visible del navegador no implica cambiar el nombre del Service interno.

```

replicas: 1
selector:
  matchLabels:
    app: grafana
template:
  metadata:
    labels:
      app: grafana
  spec:
    nodeSelector:
      kubernetes.io/hostname: worker1-node
    containers:
      - name: grafana
        image: grafana/grafana:latest
        ports:
          - containerPort: 3000
---
apiVersion: v1
kind: Service
metadata:
  name: grafana-service
  namespace: predictivo
spec:
  type: NodePort
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 30300
  selector:
    app: grafana
EOF

sudo kubectl apply -f metrics-stack.yaml
[oude: authenticate] Password:
deployment.apps/influxdb created
service/influxdb-service created
deployment.apps/grafana created
service/grafana-service created
master@master-node:~/proyecto-predictivo$

```

Figura 13. Manifiesto de Grafana y Service NodePort empleado para el acceso local.

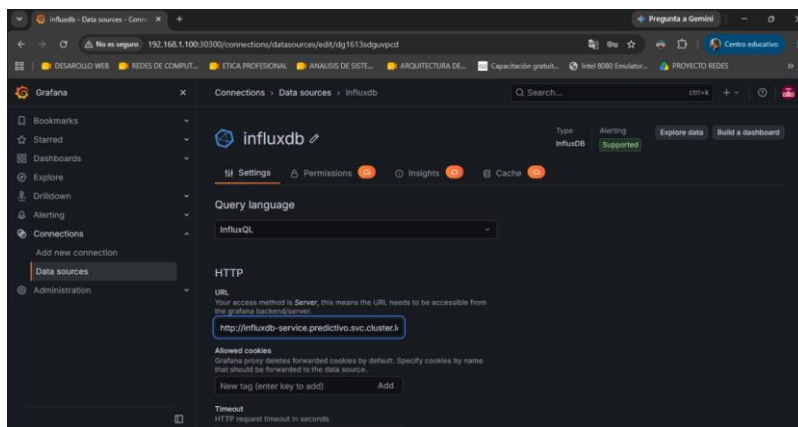


Figura 14. Fuente de datos InfluxDB configurada mediante el Service interno.

7 Resultado visible en el dashboard

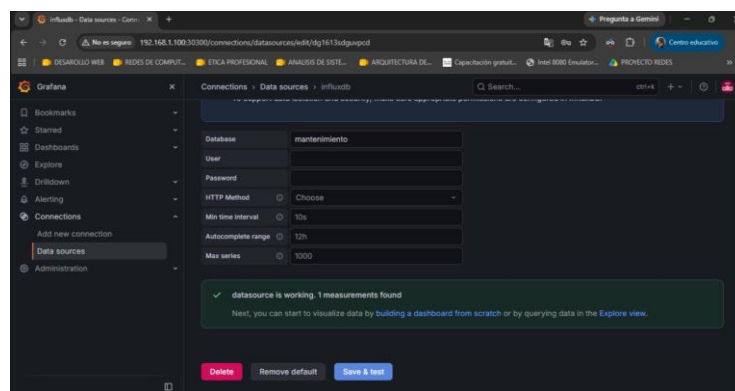


Figura 15. Prueba satisfactoria de la fuente de datos con una medición disponible.

El panel Monitoreo Predictivo Industrial muestra la temperatura del motor en °C. En la captura se aprecia un valor habitual cercano a 45 °C y varios picos por encima de 65 °C, algunos alrededor de 80 °C. Esos saltos provienen de la simulación de fallas; ayudan a explicar visualmente por qué un sistema de monitoreo debe revisar cambios bruscos y no solamente el promedio.

Las bandas de color del gráfico sirven como referencia visual para interpretar la serie; por sí solas no prueban que exista una alerta automática configurada ni que el modelo haya tomado la decisión usando esos límites.

La clasificación se produce antes, dentro del contenedor de la Raspberry. Un operador podría observar el momento del pico en Grafana, contrastarlo con vibración y estado_clasificado, y decidir si corresponde revisar la máquina.

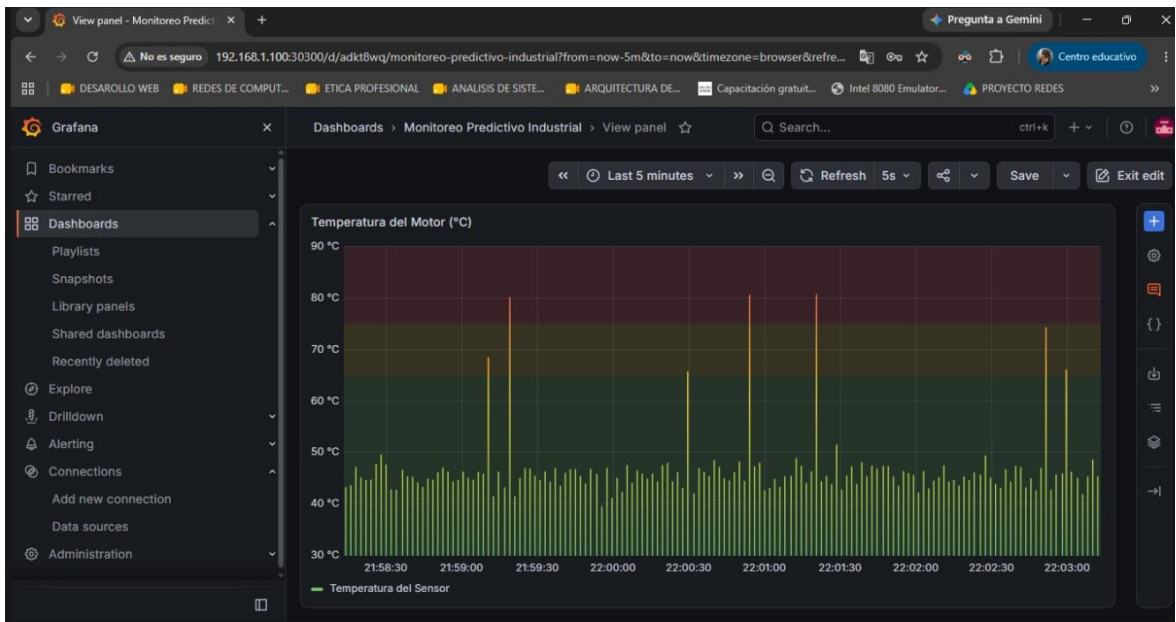


Figura 16. Dashboard de temperatura con lecturas normales y picos simulados.

El gráfico visible está configurado para mostrar los últimos cinco minutos y refrescarse cada cinco segundos. La vibración y el estado también viajan por MQTT y se guardan en InfluxDB, pero no aparecen como paneles separados en esta evidencia. Su consulta es técnicamente posible a partir de los campos que escribe el persistidor; el panel mostrado en la demostración representa solo temperatura.

Como el simulador publica cada dos segundos, en cinco minutos podrían generarse alrededor de 150 lecturas si el proceso y la comunicación permanecen activos todo el intervalo. El refresco de cinco segundos indica cada cuánto se vuelve a dibujar el panel, no la frecuencia a la que se producen las lecturas. Esta diferencia ayuda a interpretar los huecos: un panel vacío puede deberse a la consulta, a la escritura o al transporte, y se debe verificar cada etapa antes de concluir que el sensor dejó de producir datos.

Variable	Evidencia en el material recibido	Presentación actual
Temperatura	Mensajes MQTT, código y serie temporal	Panel visible en Grafana.
Vibración	Mensajes MQTT y campo de InfluxDB	Guardada; no hay panel independiente en las capturas.
Estado	Etiqueta en mensajes y escritura del persistidor	Clasificado; no hay panel independiente en las capturas.

8 Persistencia de la configuración

El documento técnico del equipo describe un volumen hostPath apuntando al directorio /var/lib/grafana-data de worker1-node. Con ello, la configuración de Grafana permanece en el disco del nodo cuando se recrea el contenedor o se reinicia ese mismo equipo. En la prueba relatada por el equipo, el tablero volvió a cargar después del reinicio de la VM. Ese resultado se atribuye al relato técnico recibido; no se adjuntó una captura de antes y después del reinicio.

El motivo de guardar este directorio es que los archivos creados solamente dentro del contenedor pueden perderse cuando el contenedor se reemplaza. Al montar un directorio del host, Grafana vuelve a encontrar sus datos de configuración al iniciarse en el mismo worker. La prueba descrita en el documento técnico aporta una observación del equipo y delimita la persistencia de la configuración del tablero.

Este tipo de almacenamiento depende de worker1-node. Si el nodo completo se pierde y el Pod se mueve a otro worker, el directorio local no lo acompaña automáticamente. Por eso la prueba de reinicio sirve para hablar de persistencia local; no demuestra por sí misma una copia redundante ni recuperación de datos ante la pérdida del disco.

El documento fuente especifica la persistencia del tablero de Grafana, pero no muestra en esta evidencia un volumen equivalente ni una copia de seguridad de InfluxDB. Por ello, conservar el panel no debe confundirse con garantizar la conservación del historial de lecturas ante cualquier falla del disco. Para esa meta habría que documentar el volumen de la base y una estrategia de respaldo y restauración.

Resumen de la prueba de reinicio

Momento	Resultado según el documento técnico	Alcance de la evidencia
Antes del reinicio	Tablero de Grafana configurado y accesible.	La captura del panel muestra la configuración en uso.
Reinicio de la VM	El equipo reporta apagar y volver a iniciar worker1.	El documento técnico describe la acción; no incluye su hora exacta.
Después del reinicio	El tablero volvió a cargar desde el disco local.	Se informa como resultado relatado, sin afirmar tiempo medido.

9 Prueba de recuperación del broker MQTT

Para simular una interrupción de software, el equipo eliminó el Pod de Mosquitto mediante kubectl. Al seguir el estado de los Pods, se observan las etapas Terminating, Pending y ContainerCreating, y después un nuevo Pod en estado Running. La sustitución ocurre porque el Deployment mantiene declarada una réplica del broker. Esta es la función de autocuración que se verificó.

El comando de eliminación modifica el estado real del clúster. Como todavía existe un Deployment que declara una réplica, el controlador detecta la diferencia entre el estado deseado y el estado actual y crea otro Pod. El nuevo Pod puede recibir una dirección interna distinta; los clientes deben conectarse por el Service en lugar de depender de la IP efímera de un Pod. La captura final confirma el reemplazo, pero no muestra una prueba de pérdida física del worker.

```
master@master-node:~$ sudo kubectl delete pod -n predictivo -l app=mosquitto
[sudo: authenticate] Password:
pod "mosquitto-broker-6c597c6cf8-r9zqg" deleted from predictivo namespace
master@master-node:~$
```

Figura 17. Comando que elimina el Pod del broker para provocar la prueba.

```
master@master-node:~$ sudo kubectl get pods -n predictivo -w
NAME                                READY   STATUS             RESTARTS   AGE
edge-sensor-classifier-6575d6879-bvkz2 1/1     Running            1 (26m ago) 27m
grafana-7f8c7f898d-8fv4n              1/1     Running            0           10m
influxdb-7ccb49b556-vq2xf             1/1     Running            2 (26m ago) 90m
mosquitto-broker-6c597c6cf8-r9zqg     1/1     Running            2 (26m ago) 115m
mqtt-persister-85ff88ffd9-lfcvq       1/1     Running            1 (26m ago) 41m
mosquitto-broker-6c597c6cf8-r9zqg     1/1     Terminating       2 (27m ago) 115m
mosquitto-broker-6c597c6cf8-r9zqg     1/1     Terminating       2 (27m ago) 115m
mosquitto-broker-6c597c6cf8-bw7w5     0/1     Pending            0           1s
mosquitto-broker-6c597c6cf8-bw7w5     0/1     Pending            0           1s
mosquitto-broker-6c597c6cf8-bw7w5     0/1     ContainerCreating  0           1s
mosquitto-broker-6c597c6cf8-r9zqg     0/1     Completed          2 (27m ago) 115m
mosquitto-broker-6c597c6cf8-r9zqg     0/1     Completed          2 (27m ago) 115m
mosquitto-broker-6c597c6cf8-r9zqg     0/1     Completed          2 (27m ago) 116m
mosquitto-broker-6c597c6cf8-bw7w5     0/1     ContainerCreating  0           4s
mosquitto-broker-6c597c6cf8-bw7w5     1/1     Running            0           6s
```

Figura 18. Secuencia capturada del Pod anterior y el reemplazo que llega a Running.

En la captura, el nuevo Pod aparece en Running con una antigüedad de 6 segundos. Ese dato no mide por sí solo los segundos transcurridos desde el comando de borrado. El equipo reportó además que el dibujo de la gráfica tardó aproximadamente cinco minutos en reanudarse por la reconexión de los clientes MQTT; para sustentar ese intervalo con precisión se necesitan marcas de tiempo o capturas antes y después.

El documento técnico atribuye la demora observada a la reconexión TCP/MQTT de los procesos Python. Esa explicación es plausible, pero las capturas entregadas no incluyen registros con marcas de tiempo que permitan aislar cuánto tardó cada cliente. Por eso se informa como observación del equipo y queda pendiente medirla. Un Pod en Running significa que el contenedor está iniciado; para confirmar que el servicio completo volvió también se debe ver un mensaje nuevo en MQTT, una escritura en InfluxDB y un punto nuevo en Grafana.

Comprobación	Resultado documentado
Recuperación del Pod	Nuevo Pod de Mosquitto en estado Running

Retorno de datos al dashboard	Aproximadamente 5 minutos, según registro del equipo
Caída completa de un worker	No forma parte de esta prueba de eliminación de Pod

9.1 Cómo medir la recuperación completa

La recuperación tiene al menos dos momentos distintos. El primero es la reposición de infraestructura: desde que se elimina el Pod hasta que el reemplazo está Running y listo para atender. El segundo es la recuperación del flujo de aplicación: desde la interrupción hasta que aparece una lectura nueva y actual en Grafana. El dato de “6 segundos” que aparece en la columna AGE de la captura representa la edad de ese Pod al momento de ejecutar kubectl; no es una resta confiable entre el instante de la falla y el instante de la recuperación.

Marca	Dónde observarla	Qué demuestra
T0	Comando y hora de eliminación del Pod	Inicio de la prueba.
T1	Estado Ready del nuevo broker	Tiempo de reposición del componente.
T2	Primer mensaje MQTT recibido tras la falla	Reconexión del productor y del broker.
T3	Primera escritura nueva en InfluxDB	Reconexión del persistidor y almacenamiento.
T4	Primer punto nuevo visible en Grafana	Recuperación percibida por el usuario.

Con estas marcas se calculan $T1 - T0$ para el broker y $T4 - T0$ para el servicio completo. Comparar $T2 - T1$ y $T3 - T2$ también ayuda a localizar la demora. La tabla siguiente es una simulación didáctica coherente con un Pod repuesto en pocos segundos y un retorno del gráfico cercano a cinco minutos; ninguna de sus horas se obtuvo de los logs. Los resultados comprobados siguen siendo el nuevo Pod Running y el intervalo aproximado comunicado por el equipo.

Marca	Hora simulada	Observación en el ejemplo
T0	22:05:00	Se elimina el Pod.
T1	22:05:06	El reemplazo queda listo.
T2	22:09:40	El broker recibe mensajes nuevos.
T3	22:09:42	El persistidor escribe lecturas nuevas.
T4	22:10:00	Grafana dibuja puntos nuevos.

En el ejemplo, $T1 - T0$ sería 6 segundos y $T4 - T0$ sería 5 minutos. Esa aritmética ilustra por qué la recuperación del contenedor y la recuperación visible para el usuario deben informarse por separado. Las capturas no permiten afirmar que esos fueron los tiempos exactos del ensayo real.

10 Resultados y evaluación

Se integraron tres nodos, se validó el despliegue de una carga de prueba en cada worker, se ejecutó el simulador y clasificador en la Raspberry, y los eventos llegaron al broker y a InfluxDB. Grafana conectó con la base de datos y mostró una serie de temperatura con picos simulados. Al eliminar el Pod del broker, Kubernetes creó uno nuevo. Las figuras anteriores relacionan cada resultado con su evidencia.

Resultado	Evidencia disponible	Lectura del resultado
Clúster de tres nodos	Estado Ready y Pods de prueba	Master y ambos workers recibieron sus funciones.
Datos de borde	Código y logs del clasificador	Temperatura, vibración y estado se publicaron.
Canal de datos	Suscripción MQTT, persistidor y fuente de Grafana	La ruta hasta la consulta quedó operativa.
Panel	Gráfica de temperatura	Se observan valores normales y picos simulados.
Autocuración	Borrado y nuevo Pod de Mosquitto	Se repuso un Pod; falta medir recuperación integral.

La arquitectura reparte funciones entre dos workers, pero las aplicaciones principales tienen una sola réplica y el plano de control dispone de un solo master. Por ello, la recuperación observada corresponde al fallo de un Pod, no a disponibilidad continua frente a la pérdida física de worker1 o del master. De igual manera, la captura de temperatura confirma la visualización de esa variable, mientras que el almacenamiento de vibración se aprecia en el código del persistidor y en los mensajes MQTT.

Alcance de las pruebas y de los datos ilustrativos

Las figuras prueban la integración de los tres nodos, la asignación de Pods, el flujo de mensajes, la consulta de temperatura y el reemplazo de Mosquitto. La tabla de IP y la cronología T0–T4 son ejemplos de presentación, identificados como tales en sus secciones. No se documentó la caída completa de un worker ni un ensayo de failover del master. Por ello, estos casos no se presentan como resultados obtenidos.

Afirmación	Sustento	Estado en el informe
Tres nodos Ready	Captura de kubectll	Resultado observado.
Temperatura graficada	Captura de Grafana	Resultado observado.
Pod de Mosquitto repuesto	Captura de eliminación y estado Running	Resultado observado.
IP de cada nodo	Topología de ejemplo; navegador muestra una IP	Dato ilustrativo por nodo.
Tiempos T0–T4	Ejercicio de cálculo	Ejemplo hipotético.
Caída completa de worker	No se adjuntó evidencia	No se afirma como prueba realizada.

11 Verificación de extremo a extremo

La verificación del prototipo sigue los mismos enlaces que recorre una lectura. Se comprueba el estado de los nodos y de los Pods del namespace predictivo, luego el mensaje que sale de la Raspberry, la recepción en el broker, la escritura del persistidor y los puntos de la gráfica. Un estado Ready o Running en cada componente es el punto de partida; la circulación de datos demuestra que los componentes trabajan juntos.

Orden	Comprobación realizada o descrita	Resultado documentado
1	kubectll get nodes	Tres nodos Ready en la captura.
2	Pods de prueba y lista final	Ubicación del edge en la Raspberry y servicios en worker1.
3	Logs de edge-sensor-classifier	Lecturas con temperatura, vibración y estado.
4	Suscripción a industrial/sensores/pi	Mensajes recibidos por Mosquitto.

5	Persistidor y fuente de Grafana	Código de escritura y prueba de fuente satisfactoria.
6	Dashboard de temperatura	Serie temporal con valores usuales y picos simulados.

Si Grafana abre, pero no aparecen puntos recientes, se sigue el recorrido hacia atrás: comprobar la consulta a InfluxDB, las escrituras del persistidor, los mensajes en MQTT y los logs del simulador. Esto identifica en qué tramo se interrumpió el flujo. Si la Raspberry no figura Ready, revisar su acceso al master y su unión al clúster antes de investigar el dashboard. En las capturas destinadas a la entrega se debe ocultar cualquier token de unión o contraseña.

11.1 Lectura integrada de los resultados

Al relacionar las figuras se ve el recorrido completo: el Pod de la Raspberry produce la lectura; los logs y la suscripción MQTT muestran que el mensaje salió y llegó; el persistidor tiene la lógica para convertirlo en un punto; Grafana valida la conexión con InfluxDB y dibuja la temperatura. La secuencia es más sólida que presentar una captura aislada del panel, porque relaciona generación, transporte, almacenamiento y consulta.

Verificación	Resultado incluido	Figura o fuente
Tres nodos Ready	Master y dos workers activos.	Figura 1.
Pods en los workers	Edge en Raspberry; otros servicios en worker1.	Figuras 3, 4 y 12.
Mensajes MQTT	JSON con temperatura, vibración y estado.	Figura 10.
Datos consultables	Fuente InfluxDB probada en Grafana.	Figuras 14 y 15.
Serie de temperatura	Lecturas normales y picos visibles.	Figura 16.
Reinicio de worker1	Tablero recuperado según relato del equipo.	Documento técnico recibido.

Conclusiones

- K3s permitió reunir dos máquinas virtuales y una Raspberry Pi en un mismo clúster local. La Raspberry asumió el trabajo de simular y clasificar lecturas, y worker1 alojó los servicios que reciben, guardan y presentan los datos.
- El recorrido Raspberry Pi → MQTT → InfluxDB → Grafana funcionó en la demostración. El dashboard permite distinguir una temperatura habitual de los picos que se introdujeron para simular condiciones anómalas.
- La eliminación del Pod de Mosquitto mostró que el Deployment lo reemplaza automáticamente. La continuidad de toda la aplicación depende también de que sus clientes vuelvan a conectarse y de la disponibilidad de los nodos y del almacenamiento.
- La distribución entre workers permitió mostrar cómo se separan las tareas de procesamiento y de datos. Con una sola réplica por servicio, el resultado demostrado es organización de cargas y reposición de un Pod, sin una medición de balanceo de tráfico entre instancias.

Recomendaciones

- Registrar hora de falla y de restablecimiento en logs del broker, del persistidor y del dashboard para medir la recuperación de extremo a extremo.
- Agregar paneles para vibración y estado clasificado, de modo que los tres datos enviados puedan consultarse sin entrar a la terminal.
- Para mayor tolerancia a fallos, estudiar réplicas de servicios y un almacenamiento que no dependa de un único nodo. Ajustar los reintentos MQTT de los clientes.
- Probar el modelo con datos reales etiquetados antes de usar sus alertas para decisiones de mantenimiento industrial.

Bibliografía y material del proyecto

El documento técnico del equipo aporta el contexto, la topología, la prueba de reinicio de Grafana y el relato de la recuperación. El archivo de capturas muestra los manifiestos, el código, el estado del clúster, los mensajes y el panel. Ambos se utilizaron para describir el montaje concreto. Las páginas oficiales siguientes respaldan la explicación general de K3s, la reposición de Pods y el alcance del almacenamiento local.

- Equipo del proyecto. Informe Técnico - Mantenimiento Predictivo K3s ACII G2. Documento interno proporcionado para este informe, 2026.
- Equipo del proyecto. raspberry pi. Capturas de instalación, código, flujo MQTT, Grafana y prueba de recuperación, 2026.
- K3s. Quick-Start Guide. <https://docs.k3s.io/quick-start>
- K3s. Architecture. <https://docs.k3s.io/architecture>
- Kubernetes. ReplicaSet. <https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/>
- Kubernetes. Volumes. <https://kubernetes.io/docs/concepts/storage/volumes/>

Los estados y valores visibles en las figuras corresponden al material recibido del grupo. Las IP asignadas por nodo y la cronología con horas se incluyen únicamente como ejemplos ilustrativos y se identifican en el cuerpo. Los nombres marcados con asterisco en la carátula también son ficticios, pues el material proporcionado no identifica a los otros dos autores.

Anexo A Lecturas de muestra

La tabla transcribe diez lecturas consecutivas visibles en los logs del simulador, mostrados en la figura 10. Se conservan los valores y etiquetas de la captura para que el lector pueda relacionar una salida del código con el mensaje enviado por MQTT. La captura no trae una marca de tiempo para cada fila, por lo que el número de fila representa únicamente el orden visible, no un minuto concreto.

Orden	Temperatura	Vibración	Clasificación
1	49.30	1.25	FALLO_MECANICO
2	42.80	1.19	NORMAL
3	44.60	1.26	NORMAL
4	45.05	1.17	NORMAL
5	47.09	1.23	NORMAL
6	75.64	3.91	FALLO_MECANICO
7	40.54	1.21	NORMAL
8	45.95	1.23	NORMAL
9	42.89	1.26	NORMAL
10	48.46	1.29	NORMAL

La fila 6 representa claramente una elevación simultánea de temperatura y vibración. La fila 1, en cambio, está relativamente cerca de la región habitual, pero aparece etiquetada como anomalía. El clasificador no usa un límite fijo de temperatura; compara cada par con el patrón aprendido por Isolation Forest. Este ejemplo también explica por qué conviene ver ambas variables y el estado del modelo cuando se analiza un pico en el panel.

Dos filas de esta selección llevan la etiqueta FALLO_MECANICO. No se debe convertir 2/10 en una tasa general de fallas del sistema: son diez líneas de una captura, elegidas por su disponibilidad visual. Para estimar una tasa y evaluar el modelo haría falta un conjunto mayor de datos etiquetados y separar las fallas inyectadas de las decisiones del clasificador.

Anexo B Relación de evidencias

Cada grupo de imágenes respalda una afirmación concreta. Esta relación permite revisar el informe sin asumir que una sola captura demuestra todos los aspectos del sistema. Los números se refieren a las figuras colocadas en los capítulos anteriores.

Figuras	Qué muestran	Alcance de la conclusión
1–5	Nodos Ready, etiquetas, Pods de prueba y namespace.	El clúster quedó integrado y aceptó cargas en ambos workers.
6–7	Manifiesto y código del edge.	El simulador genera y clasifica pares de valores.
8–10	Broker, Service y mensajes.	Hay publicación y recepción de JSON por MQTT.
11–12	Persistidor y ubicación de cinco Pods.	Se observa la lógica de escritura y el reparto por nodos.
13–16	Grafana, fuente de datos y serie.	La temperatura se pudo consultar y graficar.
17–18	Eliminación y reemplazo de Mosquitto.	El Deployment creó un nuevo Pod Running.

El documento técnico complementa las figuras con el relato del reinicio de worker1 y la recuperación de la configuración de Grafana. La continuidad de InfluxDB ante pérdida de disco, la caída completa de un worker y los tiempos exactos de reconexión no están demostrados por estas imágenes. Las IP por nodo y las horas T0–T4 son ejemplos explicativos, no capturas del entorno real.

Con esta distinción, el resultado central queda sustentado: el prototipo procesa lecturas simuladas en la Raspberry, las transmite y guarda para su consulta local, y K3s repone el Pod del broker cuando se elimina. Las pruebas adicionales descritas como posibilidades futuras no se incorporan como logros de la ejecución documentada.