UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Infraestructura de Microservicios bajo Computing Continuum para Sistemas Inteligentes Autoadaptativos

Autor

José Manuel Casanova Martínez

Directores

José Luis Garrido Bullejos (tutor)

María Bermúdez Edo (co-tutora)

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2025

Infraestructura de Microservicios para el desarrollo de Sistemas Inteligentes Autoadaptativos bajo el paradigma Computing Continuum

Autor
José Manuel Casanova Martínez

Directores
José Luis Garrido Bullejos (tutor)
María Bermúdez Edo (co-tutora)

Infraestructura de Microservicios para el desarrollo de Sistemas Inteligentes Autoadaptativos bajo el paradigma Computing Continuum

José Manuel Casanova Martínez

Palabras clave: Continuidad Computacional, Microservicios, Sistemas Distribuidos, Internet de las Cosas, Computación en el borde, Computación en la niebla, Computación en la nube, Sistemas Inteligentes Autoadaptativos.

Resumen

Este trabajo presenta el diseño e implementación de una infraestructura de microservicios bajo el paradigma de la Continuidad Computacional (Computing Continuum), orientada a la creación de sistemas inteligentes autoadaptativos. En este trabajo, se integran dispositivos IoT, nodos en el borde, niebla y nube para distribuir dinámicamente las cargas de procesamiento, optimizando así la eficiencia energética, la latencia y la escalabilidad del sistema.

El sistema que se desarrolla permite a los usuarios enviar datos fisiológicos desde dispositivos inteligentes, como pulseras o relojes, que son procesados para predecir su estado o actividad (por ejemplo, "durmiendo", "sedentario" o "activo"). El sistema se apoya en algoritmos de aprendizaje automático para realizar inferencias personalizadas para cada usuario en los nodos de borde, mientras que los modelos globales se entrenan y actualizan en la nube con datos procedentes de todos los usuarios.

La infraestructura se gestiona de forma autoadaptativa mediante el ciclo MAPE-K, aplicado sobre los recursos del sistema. Cada nodo exporta métricas de su estado (carga, disponibilidad, rendimiento). Posteriormente, a partir de los datos de todos los nodos, un microservicio analiza posibles desviaciones, planifica ajustes y los ejecuta en tiempo real, permitiendo una orquestación dinámica y eficiente de los microservicios desplegados.

Este trabajo intenta contribuir al avance en la gestión inteligente de sistemas distribuidos, y pretende explorar las bases para futuras aplicaciones en ámbitos como la salud conectada.

Microservices Infrastructure for the Development of Self-Adaptive Intelligent Systems under the Computing Continuum Paradigm

José Manuel, Casanova Martínez

Keywords: Computing Continuum, Microservices, Distributed Systems, Internet of Things, Edge Computing, Fog Computing, Cloud Computing, Self-Adaptive Intelligent Systems.

Abstract

This work presents the design and implementation of a microservices infrastructure based on the Computing Continuum paradigm, aimed at developing self-adaptive intelligent systems. The proposed system integrates IoT devices, edge, fog, and cloud nodes to dynamically distribute processing loads, thereby optimizing energy efficiency, latency, and overall scalability.

The developed system allows users to transmit physiological data from smart devices, such as wristbands or watches, which are processed to predict their state or activity (e.g., "sleeping," "sedentary," or "active"). The system relies on machine learning algorithms to perform personalized inference for each user at the edge nodes, while global models are trained and updated in the cloud using data collected from all users.

The infrastructure is managed in a self-adaptive manner through the MAPE-K loop, applied to system resources. Each node exports metrics about its current state (load, availability, performance). These metrics are then analyzed by a dedicated microservice, which detects deviations, plans corrective actions, and executes them in real time—enabling dynamic and efficient orchestration of the deployed microservices.

This work aims to contribute to the advancement of intelligent distributed system management and to lay the groundwork for future applications in domains such as connected healthcare.

Yo, **José Manuel Casanova Martínez**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45923102Z, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Manuel Casanova Martínez

Granada a 5 de septiembre de 2025.

D. **José Luis Garrido Bullejos (tutor)**, Profesor del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Dña. **María Bermúdez Edo (co-tutora)**, Profesora del Área de Lenguajes y Sistemas Informáticos del Departamento Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Infraestructura de Microservicios para el desarrollo de Sistemas Inteligentes Autoadaptativos bajo el paradigma Computing Continuum***, ha sido realizado bajo su supervisión por **José Manuel Casanova Martínez**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 5 de septiembre de 2025.

Los directores:

José Luis Garrido Bullejos **María Bermúdez Edo**

Agradecimientos

Este trabajo ha representado mucho más que una simple tarea académica. Ha sido un proceso exigente, marcado por desafíos técnicos complejos y decisiones críticas que han requerido esfuerzo, constancia y capacidad de adaptación. En distintos momentos surgieron dificultades que pusieron a prueba tanto mis conocimientos como mi resiliencia. Sin embargo, el resultado alcanzado es fruto del compromiso sostenido y del acompañamiento de quienes me han brindado su apoyo a lo largo de este recorrido.

Quiero agradecer profundamente a mis tutores. Su orientación, paciencia y consejos han sido clave para dar forma al proyecto y para superar los obstáculos que fueron surgiendo. Su implicación ha sido un verdadero pilar en este proceso.

Y, sobre todo, gracias a mi familia y amigos. En los momentos más difíciles, vuestra confianza en mí y vuestro ánimo incondicional fueron el motor que me empujó a seguir adelante.

Índice general

1. Introducción **13**
1.1. Motivación 13
1.2. Objetivos 15
1.2.1. Objetivo General 15
1.2.2. Objetivos Específicos 16
1.2.3. Objetivos Formativos 18
1.3. Planificación 18
1.3.1. Planificación Inicial 18
1.3.2. Diagrama de Gantt 20
1.3.3. Costes del Proyecto 21
1.4. Estructura del Documento y Repositorio 22
1.4.1. Localización y Licencia del Repositorio 23

2. Estado del Arte y Trabajos Relacionados **25**
2.1. Antecedentes 25
2.1.1. Cloud Computing, Edge Computing y Fog Computing 25
2.1.2. Internet of Things (IoT) 29
2.2. Continuidad Computacional 32
2.2.1. Arquitectura Monolítica y Arquitectura de Microservicios 32
2.2.2. El Avance a la Continuidad Computacional 35
2.2.3. Desafíos de la Continuidad Computacional 36
2.2.4. Metodologías de Gestión de Sistemas en la Continuidad Computacional 37
2.3. Los Sistemas Autoadaptativos 42
2.3.1. Modelos Conceptuales de Auto-Adaptación 43
2.3.2. Conciencia del Contexto y Modelado del Entorno . . . 45
2.3.3. Mecanismos de la Auto-Adaptación 46
2.4. Estrategias para Abordar la Heterogeneidad de la Continuidad Computacional 47
2.5. Trabajos Relacionados 48
2.5.1. Ejemplo de Aplicación en Control de Tráfico 48

2.5.2. Aplicación IoT para la Monitorización de Seguridad de Edificios: Un Enfoque Auto-Adaptativo 49
2.5.3. Sistema Nautilus Adaptado al Caso SWAMP 50

3. Metodologías, Técnicas y Tecnologías para el Desarrollo 53
3.1. Metodologías de Desarrollo 53
3.1.1. Modelo en Cascada 54
3.1.2. Modelo Iterativo 54
3.1.3. Prototipado Evolutivo 55
3.1.4. Metodologías Ágiles 56
3.2. Tecnologías de Simulación 57
3.3. Contenerización de Aplicaciones 58
3.4. Despliegue y Orquestación de Aplicaciones: Kubernetes (K8s) 59
3.4.1. KubeEdge vs K3S 60
3.5. Herramientas para la Monitorización y Visualización del Sistema 61
3.5.1. Herramientas de Monitorización 61
3.5.2. Herramientas de Visualización 62
3.6. Tecnologías de Desarrollo y Programación del Sistema (Frontend y Backend) 63
3.7. Bibliotecas de Machine Learning 66
3.8. Bases de Datos 67
3.9. Cliente para APIs 69
3.10. Dispositivos Wearables y su Integración en la Continuidad Computacional 69
3.10.1. Dispositivos Utilizados 70
3.11. Conclusiones 71

4. Propuesta y Desarrollo del Sistema 73
4.1. Planteamiento General de la Propuesta 73
4.1.1. Escenarios Representativos de Uso del Sistema 75
4.1.2. Manual vs Auto-Adaptativo 75
4.2. Diseño de la Arquitectura Distribuida 78
4.2.1. Capas y Roles 78
4.2.2. Diagrama deL Sistema 79
4.2.3. Diagrama de Secuencia 80
4.3. Desarrollo del Sistema 80
4.3.1. Sprint 0: Infraestructura, Backlog y Prototipado Inicial 81
4.3.2. Sprint 1: Puesta en Marcha de la Conexión con las Pulseras 90
4.3.3. Sprint 2: Infraestructura Básica de Recolección y Nodo de Borde Simplificado 97
4.3.4. Sprint 3: Implementación del Gestor Inicial 103

4.3.5. Sprint 4: Entrenamiento Personalizado y Persistencia del Modelo 108
4.3.6. Sprint 5: Ciclo MAPE-K Básico 122
4.3.7. Sprint 6: Orquestación Local Básica, Monitoreo Central y MAPE-K 125
4.4. Problemas Durante el Desarrollo 136
4.5. Evaluación de la solución 138

5. Conclusiones, Trabajos Futuros y Potencial Emprendedor 139
5.1. Conclusiones 139
5.2. Trabajo Futuro 141
5.3. Potencial Emprendedor 142
5.4. Reflexión Final 143

A. Exploración de Escenarios de Despliegue 151
A.1. Implementación Simulada VS Implementación Física 151
A.1.1. Implementación Simulada 151
A.1.2. Implementación Física 152
A.1.3. Conclusión 152
A.2. Enfoque del Funcionamiento del Sistema 153
A.2.1. Conclusión 153

B. Guía de Despliegue del Sistema 154
B.1. Despliegue de la Aplicación de Recolección de Datos 154
B.2. Despliegue de la Infraestructura en K3s 154
B.2.1. Instalación de K3s 155
B.2.2. Verificación del Clúster 155
B.2.3. Despliegue de la Arquitectura de Microservicios 155

Índice de figuras

1.1. Representación de Ciudad Inteligente [17] 15
1.2. Diagrama de Gantt del Proyecto. Elaboración propia 20

2.1. Esquema de computación en la nube [16] 26
2.2. Esquema de computación en el borde [8] 27
2.3. Esquema de conexión entre computación en la nube y en la niebla [4] 29
2.4. Ejemplos de IoT [14] 30
2.5. Esquema de Cloud Computing, Fog Computing y Edge Computing [15] 31
2.6. Representación gráfica de una arquitectura monolítica y una arquitectura basada en microservicios. Elaboración propia. . . 35
2.7. Manta cartesiana [21] 39
2.8. Carguero de contenedores [13] 40
2.9. Representación del Ciclo MAPE-K [49] 41
2.10. Representación del grafo acíclico dirigido del sistema [21] . . 49
2.11. Esquema del proceso de despliegue [7] 51

3.1. Código para identificar nodos sobrecargados [6] 57
3.2. Código para búsqueda de nodos para su despliegue [6] 58
3.3. Código para la gestión de fallos de conexión [6] 58

4.1. Diagrama conceptual de la comunicación del sistema. Elaboración propia. 79
4.2. Diagrama de Secuencia inicial del sistema ejecutando el envío de datos fisiológicos y la actualización del modelo de predicción. Elaboración propia. 80
4.3. Captura del entorno de trabajo de Jira creado para el proyecto 82
4.4. HU descompuestas en tareas 87
4.5. Pantalla de inicio de la aplicación propia sin detectar la E4 . 91
4.6. Pantalla de inicio de la aplicación propia detectando la E4 . . 91
4.7. Pantalla de conexión de la aplicación propia con la E4 92
4.8. Pantalla de desconexión de la aplicación propia con la E4 . . 92
4.9. Organización de tareas reales del Sprint 1 95

4.10. Acumulado de horas reales del Sprint 1 95
4.11. Captura de las peticiones que recibía la API Gateway en el puerto 8080 98
4.12. Organización de tareas reales del Sprint 2 101
4.13. Acumulado de horas reales del Sprint 2 102
4.14. Esquema del flujo del sistema con el nodo manager. Elaboración propia. 104
4.15. Ejecución de test sobre las estrategias de asignación de nodo planteadas 105
4.16. Organización de tareas reales del Sprint 3 106
4.17. Acumulado de horas reales del Sprint 3 106
4.18. Esquema de las tablas de la base de datos 110
4.19. Captura de Postman enviando datos a un nodo de borde . . . 116
4.20. Captura de la ejecución del servicio de cloud_trainer 117
4.21. Captura del servicio fog_tester insertando datos de prueba . . 117
4.22. Captura del servicio de RabbitMQ 117
4.23. Captura del servicio de cloud_api para la generación de un modelo personalizado 118
4.24. Captura del servicio de fog_trainer 118
4.25. Organización de tareas reales del Sprint 4 119
4.26. Acumulado de horas reales del Sprint 4 120
4.27. Captura del Dashboard de Grafana 122
4.28. Ejecución sobre el Prometheus para consultar la memoria disponible 123
4.29. Ejecución del comando free -h sobre el terminal del equipo anfitrión 123
4.30. Organización de tareas reales del Sprint 5 124
4.31. Acumulado de horas reales del Sprint 5 124
4.32. Migración de versiones del servicio de base de datos 128
4.33. Comprobación del despliegue de RabbitMQ 128
4.34. Comprobación de la creación del volumen 129
4.35. Consulta del nombre del pod 129
4.36. Revisión de los logs del pod. Parte 1. 129
4.37. Revisión de los logs del pod. Parte 2. 129
4.38. Diagrama de arquitectura. Elaboración Propia. 132
4.39. Diagrama de arquitectura de monitoreo. Elaboración Propia. 133
4.40. Organización de tareas reales del Sprint 6 134
4.41. Acumulado de horas reales del Sprint 6 134

B.1. Verificación de los nodos del clúster 155
B.2. Verificación de los namespaces 156
B.3. Verificación de Prometheus 157
B.4. Verificación de Grafana 157
B.5. Dashboard en Grafana 158

B.6. Verificación del despliegue de db 159

Índice de cuadros

1.1. Resumen de costes del proyecto 22

2.1. Esquema comparativo de arquitecturas monolíticas y basadas en microservicios, según AWS [2] 34

3.1. Tabla comparativa entre Docker y RKT 59
3.2. Comparativa de características entre Zabbix y Prometheus [53] 62
3.3. Comparativa de interfaces entre Grafana y Zabbix [54] 63

4.3. Sprint backlog del Sprint 1 90
4.4. Sprint backlog del Sprint 2 97
4.5. Sprint backlog del Sprint 3 103
4.6. Sprint backlog del Sprint 4 108
4.7. Estructura de campos en el conjunto de datos `Ticwatchdata` 111
4.8. Estructura de campos en la tabla `user_model_mapping` 111
4.9. Listado de endpoints clave de Cloud API 112
4.10. Sprint backlog del Sprint 5 122
4.11. Sprint backlog del Sprint 6 125
4.12. Problemas encontrados durante el desarrollo del sistema . . . 137

Capítulo 1

Introducción

La computación moderna ha evolucionado hacia **arquitecturas distribuidas** más descentralizadas que integran dispositivos **IoT (Internet of Things)**, nodos en el **borde (edge)**, **niebla (fog)** y **nube (cloud)** en lo que se conoce como **Continuidad Computacional (Computing Continuum)**[1]. Este paradigma conecta recursos computacionales diversos, permitiendo el desarrollo de aplicaciones con requisitos de **eficiencia computacional y energética** más estrictos. Sin embargo, la heterogeneidad y la distribución geográfica de dichos recursos presentan retos significativos en su asignación dinámica y balanceo de carga para conseguir una computación más eficiente.

1.1. Motivación

La creciente presencia de dispositivos inteligentes en la vida cotidiana (como smartphones, relojes inteligentes, tabletas y sensores IoT) ha dado lugar a un ecosistema digital con el que interactuamos de forma constante y en tiempo real. En la actualidad, es poco común encontrar a alguien que no disponga de al menos un móvil o un smartwatch, dispositivos que, además de facilitar la comunicación, generan datos de manera continua. Este entorno, junto con el avance de otras tecnologías emergentes (vehículos inteligentes, ciudades inteligentes, etc.), ha impulsado el surgimiento del paradigma de **Continuidad Computacional**, que integra recursos de procesamiento en diferentes niveles: dispositivos IoT, nodos en el borde (edge), niebla (fog) y nube (cloud).

Uno de los principios fundamentales de este enfoque es realizar el **procesamiento lo más cerca posible del origen de los datos** (en el borde) con el objetivo de **reducir el consumo energético, disminuir la latencia, mejorar la privacidad y aumentar la seguridad**. Sin embargo, ciertas tareas complejas, como el entrenamiento de modelos de aprendizaje

profundo o el análisis de grandes volúmenes de datos históricos, requieren capacidades que solo la nube puede ofrecer. Por ello, la Continuidad Computacional no propone reemplazar la nube, sino **distribuir inteligentemente las cargas según las necesidades de procesamiento, conectividad y contexto**, favoreciendo una arquitectura ágil, escalable y eficiente.

No obstante, esta distribución heterogénea y geográficamente dispersa de recursos presenta desafíos significativos. Las capacidades computacionales varían ampliamente en términos de potencia, arquitectura y consumo energético, lo que complica su gestión óptima. Además, factores como la eficiencia computacional, la latencia y el uso racional de energía se han convertido en aspectos críticos para el desarrollo de aplicaciones sensibles al tiempo, como sistemas de salud conectada, movilidad autónoma y análisis de datos en tiempo real.

La relevancia de abordar este tema es doble. Por un lado, la necesidad de **optimizar la gestión de recursos en la Continuidad Computacional** es esencial para construir soluciones críticas en ámbitos como las ciudades inteligentes, la automatización industrial y los servicios sanitarios, con garantías de eficiencia, resiliencia y escalabilidad. En la Figura 1.1 se muestra un ejemplo visual que refleja cómo las ciudades inteligentes incorporan infraestructuras conectadas y sistemas distribuidos para mejorar la gestión urbana. Por otro lado, el desafío técnico de **coordinar y orquestar recursos tan diversos** requiere la intervención de perfiles especializados: ingenieros informáticos, expertos en simulación, programación paralela, integración de sistemas heterogéneos y diseño de algoritmos de aprendizaje. Este contexto multidisciplinar subraya la importancia de diseñar e implementar soluciones innovadoras para la gestión inteligente de la Continuidad Computacional, uno de los retos más distintivos en la ingeniería informática contemporánea.

Figura 1.1: Representación de Ciudad Inteligente [17]

Una de mis mayores motivaciones al emprender este proyecto es la convicción de que el poder de los datos reside en su capacidad para transformar la realidad. En un mundo saturado de información, donde la cantidad de datos supera nuestra capacidad de procesarlos manualmente, resulta de gran interés desarrollar sistemas que actúen de forma inteligente y proactiva para extraer valor y facilitar la toma de decisiones. Esta necesidad se vuelve especialmente relevante en áreas críticas como la salud, donde la monitorización continua de parámetros y actividades puede marcar una diferencia significativa en el bienestar y la calidad de vida de las personas. La capacidad de desarrollar soluciones que se adapten en tiempo real representa un avance significativo en el diseño de sistemas distribuidos inteligentes. Este enfoque permite construir arquitecturas capaces de responder dinámicamente a las condiciones operativas del entorno, optimizando el uso de recursos y garantizando la continuidad del servicio.

1.2. Objetivos

1.2.1. Objetivo General

El objetivo general de este trabajo es **diseñar e implementar una infraestructura basada en microservicios que, bajo el paradigma del Computing Continuum, permita el desarrollo de sistemas inteligentes autoadaptativos para la gestión eficiente de recursos en entornos heterogéneos**. Concretamente, sobre dicha infraestructura, se desarrollará un prototipo de sistema que, utilizando datos recogidos por dispositivos IoT (pulseras, relojes inteligentes, etc), permite monitorizar el estado de los usuarios basado en datos de constantes fisiológicos (ritmo cardíaco, actividad electrodérmica de la piel, etc) y asignar dinámicamente tareas de

análisis de esos datos a servicios desplegados en nodos del borde, niebla y nube, maximizando así la eficiencia energética y el rendimiento del sistema en función de las condiciones operativas y necesidades específicas.

1.2.2. Objetivos Específicos

Para alcanzar el objetivo general, adicionalmente se plantean los siguientes objetivos específicos:

1. **Estudio e investigación sobre el paradigma de Continuidad Computacional:** Investigar la evolución, características y componentes del paradigma de Continuidad Computacional, abarcando las capas de la computación en la nube, niebla y borde, así como los retos y desafíos inherentes a su gestión y asignación de recursos.

2. **Identificar estrategias de gestión de recursos de computación en sistemas distribuidos:** Analizar las estrategias y políticas de gestión de recursos utilizadas en la Continuidad Computacional, con especial énfasis en la asignación eficiente de recursos, balanceo de carga y optimización de la latencia y eficiencia energética.

3. **Propuesta de infraestructura para la gestión de la Continuidad Computacional:** Proponer y desarrollar una infraestructura que integre de forma óptima nodos de la nube, niebla y borde, permitiendo una asignación dinámica y autoadaptativa de tareas. Se buscará que el diseño aproveche las fortalezas de cada capa para garantizar baja latencia, eficiencia energética y escalabilidad.

4. **Uso de algoritmos para el análisis de datos (Machine Learning):** Se hará uso de algoritmos de Machine Learning para hacer predicciones de estado de los usuarios.

5. **Implementación de un caso de uso específico:** Desarrollar un sistema basado en la infraestructura propuesta bajo el paradigma de Continuidad Computacional en un escenario concreto orientado a la predicción de actividad física.

6. **Visualización del comportamiento del sistema:** Integrar herramientas de monitorización que permitan observar en tiempo real el estado de la infraestructura desplegada.

7. **Recomendaciones y conclusiones:** Proponer recomendaciones para mejorar las políticas de gestión de recursos en la Continuidad Computacional, basadas en los resultados obtenidos, y sugerir posibles direcciones para futuras investigaciones en este ámbito.

8. **Llevar a cabo un desarrollo del proyecto de forma ágil y evolutiva con herramientas actuales de gestión de proyectos:** Uso de metodologías ágiles como SCRUM, junto con herramientas colaborativas como GitHub y Jira, con el fin de planificar, documentar y dar seguimiento a las distintas fases del proyecto.

1.2.3. Objetivos Formativos

Además de los objetivos específicos anteriores, a continuación se plantean otros objetivos claramente orientados y relacionados con la complementación y ampliación de la formación que el estudiante ha recibido hasta el momento:

1. **Aprendizaje básico sobre Técnicas de Machine Learning:** Explorar el uso de algoritmos de aprendizaje automático para analizar e interpretar datos obtenidos desde dispositivos wearables.

2. **Estudiar y poner en práctica los principios sobre Metodologías Ágiles:** Familiarizarse y aplicar metodologías ágiles en la planificación y gestión del proyecto, facilitando la coordinación de tareas, la revisión continua y la adaptación del trabajo a lo largo del desarrollo.

3. **Exploración de tecnologías y herramientas que ayuden a dar soporte a Continuidad Computacional:** Adquirir conocimientos prácticos en tecnologías clave, como la computación en el borde, niebla y nube, así como en herramientas de simulación y evaluación de sistemas distribuidos.

4. **Aplicación de técnica para la integración e interoperabilidad de Sistemas Heterogéneos:** Mejorar las habilidades en la integración de diferentes dispositivos y arquitecturas, mediante la aplicación de técnicas de programación paralela, gestión de recursos y orquestación de servicios en entornos distribuidos.

1.3. Planificación

El proyecto comenzó el día 20 de Noviembre de 2024, tras una reunión inicial con los tutores donde se describieron y definieron los objetivos del proyecto, se comentaron las primeras ideas sobre el enfoque que tendrá el trabajo. Además, se definieron las primeras bibliografías para la elaboración del TFG y se acordaron sesiones de planificación y control.

1.3.1. Planificación Inicial

El proyecto se estructura en dos grandes fases: la fases de estudio e investigación, y las fases de desarrollo. Dado que, desde un primer momento sólo se estuvo profundizando en el conocimiento del paradigma Continuidad Computacional y sus implicaciones, la planificación de la fase de desarrollo se ha definido de manera independiente y detallada conforme se adquirieron mayores conocimientos sobre la temática.

Para desarrollar el proyecto se adoptará una metodología ágil, en particular la metodología Scrum, que permite una planificación flexible y adaptativa a través de iteraciones (sprints) de dos semanas, sesiones de revisión y planificación al final de cada sprint. Esta metodología facilita la identificación temprana de posibles desviaciones y la re-priorización del backlog según los avances y descubrimientos obtenidos durante la fase de investigación.

Actividades de la Fase de Investigación

- **Revisión del Estado del Arte:** Recopilación y análisis de artículos y documentos para comprender el dominio de la Continuidad Computacional, identificar desafíos, tecnologías y metodologías existentes.
- **Resumen y Sistematización de la Bibliografía:** Elaboración de resúmenes y esquemas comparativos que permitan clasificar y evaluar las diferentes propuestas y soluciones en el ámbito de la Continuidad Computacional.
- **Identificación de Retos:** Definición de los principales desafíos en la gestión de recursos distribuidos.
- **Diseño Conceptual y Metodológico:** Desarrollo de una base teórica y conceptual que sustente la propuesta de solución, incluyendo la integración de enfoques como la Manta de Markov y el ciclo MAPE-K para la gestión adaptativa.

1.3.2. Diagrama de Gantt

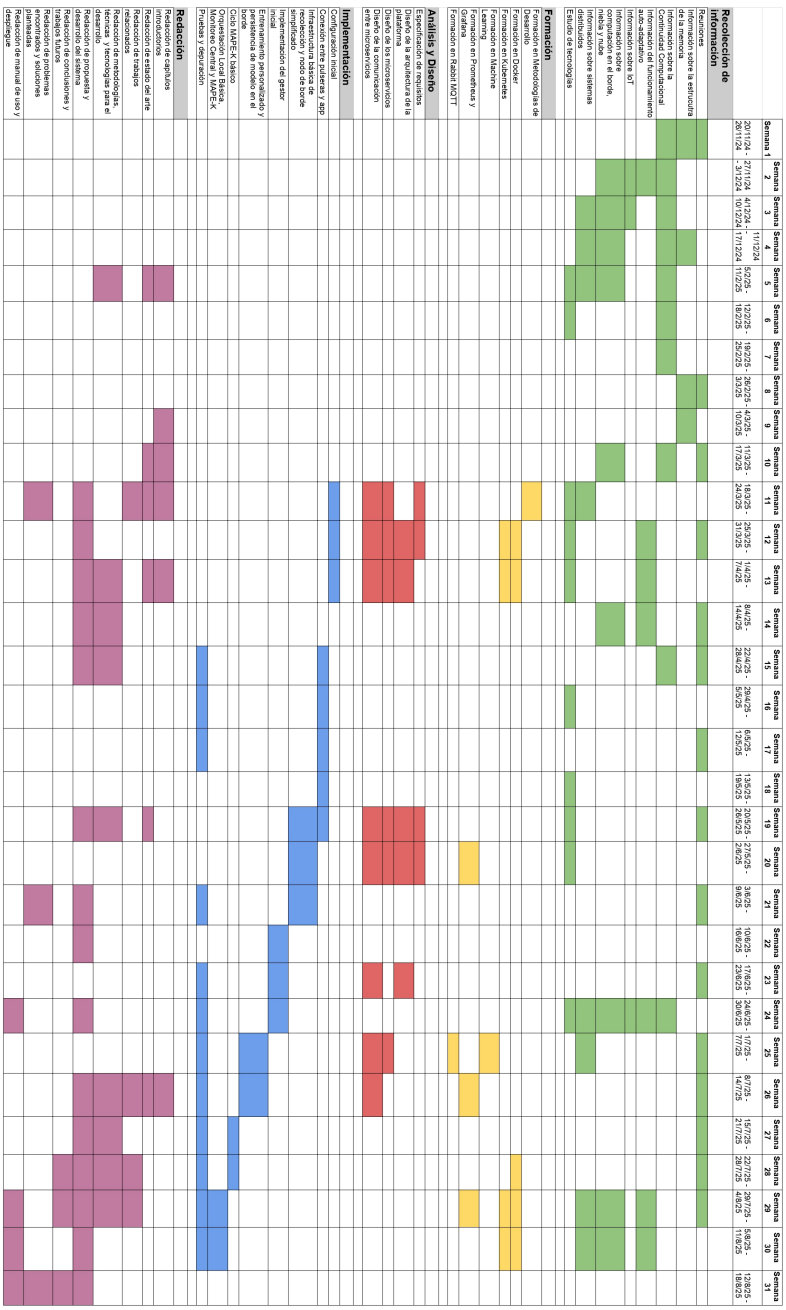


Figura 1.2: Diagrama de Gantt del Proyecto. Elaboración propia

1.3.3. Costes del Proyecto

A continuación se describe el cálculo de los costes para llevar a cabo la propuesta planteada.

Costes de Hardware

Para el desarrollo del proyecto se ha utilizado un portátil de gama media-alta: ASUS TUF Gaming F17 FX707, con las siguientes características:

- Procesador: Intel Core i7-13620H
- Memoria RAM: 32 GB
- Almacenamiento físico: 1TB SSD
- Pantalla: 17.3 pulgadas FullHD 144Hz
- Tarjeta gráfica: NVIDIA GeForce RTX 4060

Este equipo tiene un precio aproximado de **1200 €**. Considerando una vida útil estimada de **4 años**, se establece una amortización anual de **300 €**. Como el proyecto se desarrolla en un periodo de **5 meses**, el coste imputado al proyecto por el uso del equipo es de **125 €.**

Costes de Software

Todo el software utilizado en el proyecto es **open-source y gratuito**. Por lo tanto, el coste total destinado a software es de **0 €**.

Costes Humanos

El desarrollo ha sido realizado por un ingeniero informático con conocimientos en administración de sistemas, CI/CD, contenedores y orquestación con Kubernetes. Según datos de Glassdoor [25], el salario medio anual de un administrador de sistemas en España es de 27.000 € a jornada completa, es decir, 14,06 €/hora.

Dado que el proyecto tiene una duración de 320 horas, el coste humano estimado es de: 14,06 x 320 = **4780,40 €**

Costes en Infraestructura

Para el desarrollo se ha requerido conexión a internet y consumo eléctrico:

- **Internet:** Se ha contratado una tarifa de fibra óptica de 1 Gbps simétricos por 30 €/mes según Digi [18]. Para 5 meses, el coste total es de 150€.

- **Electricidad:** El consumo estimado del portátil representa un 10 de una factura doméstica media. Con un precio medio mensual de 60€, el coste es de 6€/mes, lo que supone 30€ en total.

Coste total en infraestructura: 180€

Coste Total

Descripción	**Coste Total**
Software	0€
Hardware	125€
Personal	4780,40€
Infraestructura	180€
Total	**5085,40€**

Cuadro 1.1: Resumen de costes del proyecto

El coste total del proyecto ha supuesto alrededor de **5085,40€**. El periodo de tiempo contemplado incluye tanto la primera parte de documentación como la segunda parte de implementación.

1.4. Estructura del Documento y Repositorio

El presente trabajo se organiza en dos grandes bloques interrelacionados. La primera parte se centra en la investigación del paradigma de la Continuidad Computacional, abarcando su evolución, las características inherentes y los desafíos asociados a la gestión de recursos en entornos distribuidos que integran la computación en la nube, la niebla y el borde. La segunda parte está orientada al desarrollo práctico de esta filosofía a través del diseño e implementación de un sistema autoadaptativo.

La memoria se estructura en capítulos tal como se describe a continuación:

- **Capítulo 1 – Introducción y Motivación:**

  Se presenta el contexto global en el que se inscribe el proyecto, destacando la importancia de la Continuidad Computacional en la actualidad. Se exponen los motivos que impulsan este trabajo, los problemas identificados en la gestión de recursos distribuidos y la relevancia de desarrollar soluciones autoadaptativas. Además, se definen los objetivos generales y específicos del proyecto.

- **Capítulo 2 – Estado del Arte y Trabajos Relacionados:**

Este capítulo presenta una revisión de los antecedentes y estudios previos sobre Continuidad Computacional, analizando su evolución, principales características y desafíos. Se estudian las soluciones propuestas en la bibliografía para la gestión de recursos en entornos distribuidos, incluyendo metodologías y herramientas utilizadas para optimizar el rendimiento de infraestructuras que abarcan la computación de la nube, la niebla y el borde.

- **Capítulo 3 – Metodologías, Técnicas y Tecnologías:**

  Este capítulo aborda las estrategias y herramientas para el desarrollo del proyecto. Se detallan las metodologías de desarrollo, las técnicas para la gestión de sistemas en la Continuidad Computacional y se realiza un análisis comparativo de las principales tecnologías de simulación y herramientas de programación y ejecución. También se presenta una tabla comparativa que sintetiza las características de las distintas tecnologías relevantes para la Continuidad Computacional.

- **Capítulo 4 – Propuesta y Desarrollo del Sistema:**

  Aquí se describe de manera detallada la propuesta de solución y el proceso de desarrollo del sistema autoadaptativo. Se explica cómo se ha aplicado la filosofía de la Continuidad Computacional en un caso práctico, abarcando el diseño, la implementación y la evaluación del sistema. Se presentan diagramas, esquemas y resultados de pruebas que evidencian la efectividad de la solución.

- **Capítulo 5 – Conclusiones y Trabajos Futuros:**

  Se resumen los principales hallazgos del proyecto, evaluando el grado de cumplimiento de los objetivos planteados. Además, se discuten las limitaciones encontradas y se proponen futuras líneas de investigación que permitan mejorar la eficiencia, la escalabilidad y la seguridad de los sistemas distribuidos en la Continuidad Computacional.

- **Apéndice A - Exploración de Escenarios de Despliegue:**

  Este anexo presenta los distintos escenarios evaluados para un posible despliegue futuro, analizando las alternativas contempladas y concluyendo con la selección justificada de la opción más adecuada.

- **Apéndice B - Guía de Despliegue del Sistema:**

  Este anexo proporcionará una guía práctica para el uso y la implementación del sistema desarrollado en el proyecto.

1.4.1. Localización y Licencia del Repositorio

El repositorio que contiene el desarrollo de este Trabajo de Fin de Grado se encuentra públicamente disponible en GitHub: `https://github.com/`

`jmcasanovam/computing_continuum.git`

Este proyecto utiliza la licencia Creative Commons CC0 1.0 Universal (CC0 1.0). Los beneficios de aplicar esta licencia son múltiples:

- **Reutilización sin barreras:** cualquier persona puede emplear el proyecto libremente, sin necesidad de atribución ni restricciones, incluso para fines comerciales o educativos.
- **Fomento de la innovación:** permite que otros investigadores, estudiantes o profesionales puedan adaptar y mejorar este trabajo sin condicionantes legales.
- **Difusión y alcance:** la ausencia de condiciones facilita una distribución más amplia, aumentando las probabilidades de que el trabajo sirva como base para futuras investigaciones.
- **Promoción de cultura libre:** se alinea con los valores de apertura y colaboración en entornos tecnológicos, científicos y académicos.

Esta decisión busca que el proyecto no solo tenga un impacto inmediato, sino que también se convierta en una herramienta útil para el futuro, sirviendo como ejemplo, inspiración o recurso en investigaciones relacionadas con la Continuidad Computacional, sistemas distribuidos y estrategias de auto-adaptación en entornos computacionales.

Capítulo 2

Estado del Arte y Trabajos Relacionados

2.1. Antecedentes

La computación ha experimentado una evolución significativa desde los **modelos centralizados** hacia **arquitecturas distribuidas** que buscan responder a los crecientes desafíos de **escalabilidad, latencia y heterogeneidad** en el manejo de datos. El paradigma de la computación en la nube (Cloud Computing) surgió como una solución para ofrecer **escalabilidad masiva y una infraestructura eficiente**. Sin embargo, a medida que proliferaron los dispositivos conectados y las aplicaciones con requisitos de latencia estrictos, se identificaron limitaciones en la centralización de los recursos. Esto marcó el inicio de paradigmas más distribuidos como la computación de borde **(Edge Computing)** y niebla **(Fog Computing)**, que acercan el procesamiento de datos a los dispositivos finales y a los usuarios, reduciendo la latencia y optimizando el uso de recursos (Dustdar, Pujol, & Donta, 2023) [21].

2.1.1. Cloud Computing, Edge Computing y Fog Computing

Cloud Computing

Según Wang et al. (2008) [64], la **computación en la nube** es un modelo tecnológico que permite a los usuarios acceder y utilizar recursos informáticos de manera remota a través de Internet. En lugar de depender de infraestructura local, los usuarios pueden acceder a servicios de hardware, software y almacenamiento de datos según sus necesidades, pagando solo por lo que utilizan.

Este paradigma se basa en distintos modelos de servicio:

- **HaaS (Hardware como Servicio):** Permite a los usuarios adquirir y utilizar hardware de TI (Tecnologías de la Información), incluidos servidores y centros de datos, como un servicio de suscripción flexible y escalable.
- **SaaS (Software como Servicio):** Ofrece aplicaciones y software alojados en la nube, eliminando la necesidad de instalación local y facilitando el mantenimiento y actualización.
- **DaaS (Datos como Servicio):** Brinda acceso a datos almacenados en la nube, permitiendo a los usuarios gestionar información remota de manera similar a como lo harían con almacenamiento local.

Al integrar estos modelos, la computación en la nube proporciona **Plataforma como Servicio** (PaaS), permitiendo a los usuarios personalizar su entorno de computación con configuraciones específicas de hardware, software y acceso a datos. Gracias a su flexibilidad, escalabilidad y optimización de costos, la computación en la nube se ha convertido en un pilar fundamental para empresas, instituciones y usuarios individuales en la era digital. Un ejemplo de la misma se puede ver en la siguiente imagen 2.1, donde se plantea un esquema de la computación en la nube.

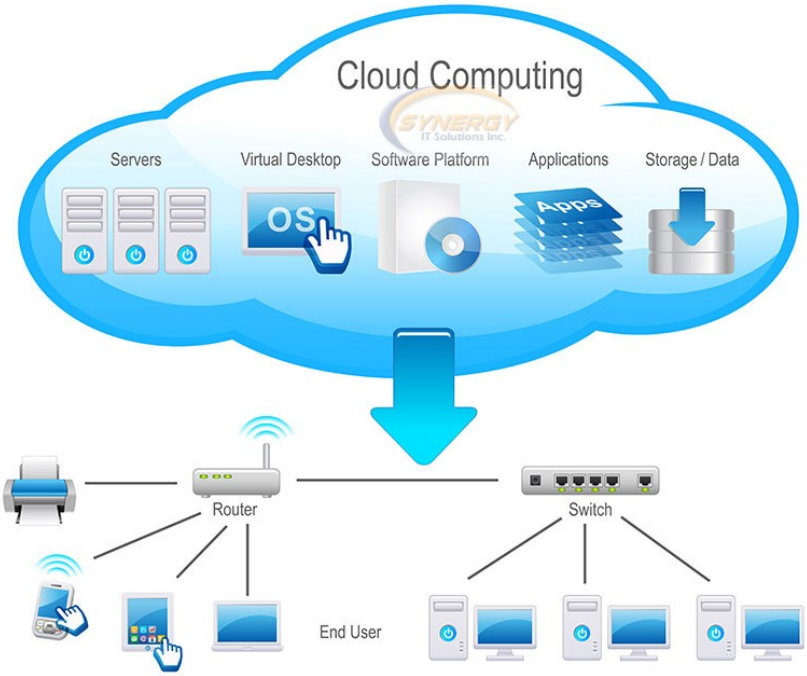


Figura 2.1: Esquema de computación en la nube [16]

Edge Computing

La **computación en el borde (Edge Computing)** es un enfoque que acerca los recursos computacionales a los dispositivos finales, en lugar de depender exclusivamente de servidores en la nube. Su objetivo es reducir la **latencia**, mejorar la **escalabilidad** y reforzar la **privacidad** en el procesamiento de datos. Se logra mediante una red distribuida de nodos de computación ubicados estratégicamente cerca de las fuentes de datos, como estaciones base celulares, puertas de enlace IoT o redes locales (Chen, Ran, 2019) [8].

Este paradigma permite el procesamiento en **tiempo real**, evitando retrasos asociados al envío de datos a la nube. Además, facilita una arquitectura escalable en la que los dispositivos finales, los nodos de borde y la nube trabajan juntos de manera eficiente, evitando cuellos de botella en la red. También mejora la privacidad al mantener los datos más cerca de su origen y reducir su exposición en la red pública.

A pesar de sus ventajas, la computación de borde enfrenta desafíos, como la necesidad de ejecutar modelos avanzados de **deep learning** (aprendizaje profundo) en dispositivos con **capacidades limitadas** y la coordinación eficiente entre múltiples nodos y la nube en entornos dinámicos.

En la siguiente imagen 2.2 podemos ver una representación de distintos dispositivos involucrados en la computación de borde como servidores de borde, cámaras de vigilancia, teléfonos móviles,... La diversidad y distribución de estos elementos reflejan la complejidad de coordinar el procesamiento local y la conectividad con la nube en escenarios reales.

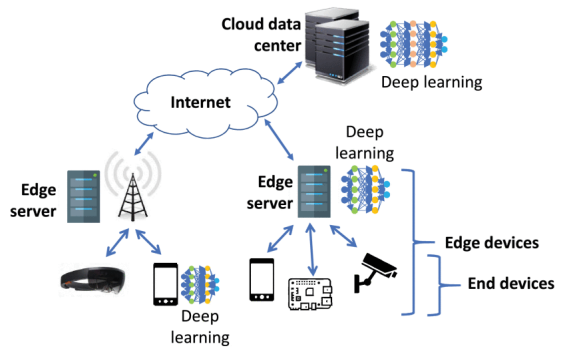


Figura 2.2: Esquema de computación en el borde [8]

Fog Computing

Según Atlam, Walters et al (2018) [4], la **computación en la niebla (fog computing)** es un modelo de computación distribuida que extiende los servicios de la nube hacia los dispositivos periféricos de una red, situándose más cerca de los usuarios y de los dispositivos IoT. Esta representación la podemos ver en la siguiente imagen 2.3. A diferencia de la computación en la nube, que centraliza el procesamiento y almacenamiento de datos en servidores remotos, la computación en la niebla permite procesar y almacenar información en dispositivos más cercanos, llamados **nodos de niebla**, reduciendo así la latencia y optimizando el uso del ancho de banda. Sus principales características son:

1. **Baja latencia y conciencia de la ubicación:** Al estar más cerca de los dispositivos finales, permite un procesamiento de datos más rápido y eficiente.

2. **Distribución geográfica:** A diferencia de la nube centralizada, los nodos de niebla pueden desplegarse en múltiples ubicaciones para optimizar el rendimiento.

3. **Escalabilidad:** Facilita la conexión y gestión de un gran número de dispositivos y sensores en redes IoT.

4. **Soporte para movilidad:** Puede interactuar directamente con dispositivos móviles, mejorando la conectividad en movimiento.

5. **Interoperabilidad:** Los nodos de niebla pueden funcionar en distintas plataformas y colaborar entre sí, permitiendo la integración de múltiples dispositivos y servicios.

6. **Análisis local y comunicación con la nube:** Filtra y procesa datos en el borde de la red antes de enviarlos a la nube, optimizando los recursos.

De esta manera obtiene beneficios como **mayor agilidad** a la hora de desplegar aplicaciones, menor latencia (llegando incluso servicios en tiempo real como juegos), **reducción de costos** al poder procesar datos de forma local, **flexibilidad** y **escalabilidad**.

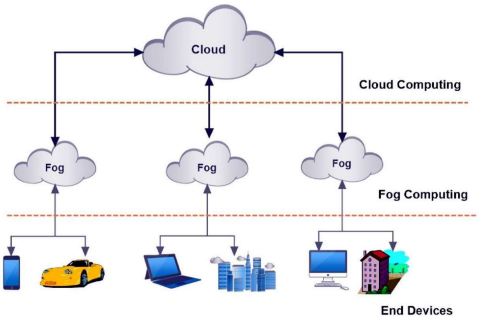


Figura 2.3: Esquema de conexión entre computación en la nube y en la niebla [4]

2.1.2. Internet of Things (IoT)

No existe una única definición aceptada del IoT, aunque la mayoría coincide en que se trata de una red abierta y comprensiva de objetos inteligentes capaces de organizarse, intercambiar información y responder a cambios en el entorno sin intervención humana. El IoT posibilita la comunicación entre personas, objetos y entre los propios objetos, dándoles a cada uno una identidad única. Así, cualquier cosa, desde dispositivos electrónicos y electrodomésticos hasta elementos cotidianos como ropa o mobiliario, puede formar parte de esta red. (Madakam, Ramaswamy & Tripathi, 2015) [40].

Para que el IoT funcione de manera efectiva se requieren:

- Recursos dinámicos y escalables, capaces de gestionar el crecimiento exponencial de datos.
- Procesamiento en tiempo real para atender aplicaciones sensibles al tiempo.
- Protección de datos y privacidad del usuario.
- Eficiencia energética y la capacidad de ejecutar aplicaciones cerca del usuario final mediante sistemas en la nube abiertos e interoperables.
- Estos elementos se integran en tres componentes principales: hardware (sensores, actuadores, cámaras, etc.), middleware (almacenamiento, herramientas de análisis de datos en la nube) y herramientas de presentación (visualización e interpretación de la información).

Madakam, Ramaswamy & Tripathi (2015)[40] cita que las principales tecnologías sobre las que se apoya el IoT son:

- **RFID** (Identificación por Radiofrecuencia): Permite identificar objetos de forma inalámbrica mediante etiquetas electrónicas.

- **IP** (Protocolo de Internet): Base para la comunicación entre dispositivos, con versiones como IPv4 e IPv6.
- **EPC** (Código Electrónico de Producto): Etiquetas que almacenan información única sobre productos.
- **Código de Barras y Códigos QR**: Métodos ópticos para codificar datos que identifican productos.
- **Wi-Fi**: Conecta dispositivos mediante redes inalámbricas de alta velocidad.
- **Bluetooth**: Facilita la comunicación de corto alcance entre dispositivos.
- **ZigBee**: Protocolo para redes de sensores inalámbricos de baja potencia y costo.
- **NFC** (Comunicación de Campo Cercano): Permite intercambios de datos a muy corta distancia, útil en transacciones y conexiones rápidas.
- **Actuadores**: Dispositivos que convierten energía en movimiento, esenciales en aplicaciones industriales y de automatización.
- **Redes de Sensores Inalámbricos** (WSN): Conjuntos de sensores distribuidos que monitorizan condiciones ambientales y transmiten datos de forma colaborativa.
- **Inteligencia Artificial** (IA): Proporciona capacidad de respuesta y adaptación en entornos inteligentes, haciendo que los dispositivos sean sensibles al contexto y las necesidades de los usuarios.

Figura 2.4: Ejemplos de IoT [14]

En la imagen 2.4 se puede ver algunos ejemplos de dispositivos IoT, muchos de uso cotidiano en la actualidad como electrodomésticos conectados a una red local, cámaras de seguridad, altavoces inteligentes, etc.

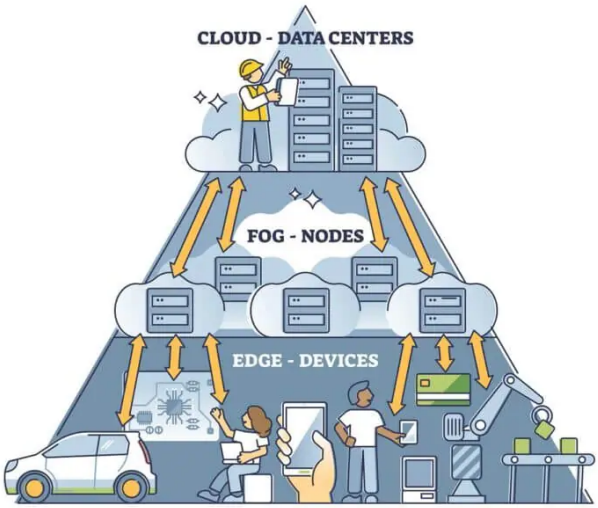


Figura 2.5: Esquema de Cloud Computing, Fog Computing y Edge Computing [15]

Con este esquema piramidal 2.5 se puede comprender más fácilmente la diferencia entre los distintos paradigmas computacionales descritos anteriormente, al igual que podemos ver una comparativa sobre la cantidad de centros y dispositivos de este tipo, notando como la cantidad de dispositivos de borde es mucho mayor que los centros de datos y procesamiento en la nube.

Dispositivos wearables como componentes IoT

Dentro del ecosistema del Internet de las Cosas, los **dispositivos wearables**, como pulseras inteligentes, relojes, gafas o ropa sensorizada, representan una categoría clave de **dispositivos de borde**. Estos elementos están diseñados para **recoger datos fisiológicos y de actividad** en tiempo real a través de sensores como acelerómetros, giroscopios, termómetros o monitores cardíacos.

Según Lee et al. (2016) [38], sus aplicaciones abarcan desde el monitoreo físico y fisiológico hasta la interacción ambiental y la generación de alarmas. Además, al estar conectados mediante tecnologías como Bluetooth o Wi-Fi, forman parte activa de redes inteligentes y entornos IoT más amplios, permitiendo la comunicación entre dispositivos y la toma de decisiones distribuidas.

La creciente adopción de estos dispositivos se ha vinculado con tendencias como el quantified self, donde los usuarios analizan datos personales para mejorar su salud, hábitos y bienestar. Esta capacidad de auto-cuantificación amplía el valor de los wearables como nodos sensoriales en arquitecturas IoT, proporcionando datos contextuales útiles para sistemas más complejos, como hogares inteligentes, medicina conectada o ciudades inteligentes.

2.2. Continuidad Computacional

El crecimiento exponencial del **Internet de las Cosas** (IoT) ha generado un flujo masivo de datos y una proliferación de dispositivos conectados, que se estima que alcance los 42 mil millones para el año 2025, según predicciones recientes (Bendechache, Svorobej, Endo, & Lynn, 2020) [5]. Este panorama plantea retos considerables para la computación tradicional en la nube, que originalmente fue diseñada para entornos centralizados y homogéneos. La alta latencia, el consumo energético y la falta de adaptabilidad a escenarios geográficamente dispersos se presentan como limitaciones claves de este modelo. Como respuesta a estas limitaciones, surge la **Continuidad Computacional**. Este paradigma integra dispositivos IoT, nodos de borde (edge), infraestructuras de niebla (fog) y centros de datos en la nube (cloud), estableciendo un ecosistema interconectado y dinámico. Este modelo no solo permite procesar datos de manera más eficiente y reducir la dependencia de conexiones centralizadas, sino que también posibilita el desarrollo de aplicaciones críticas en sectores como la salud, la movilidad autónoma y el análisis de grandes volúmenes de datos en tiempo real (Al-Dulaimy et al. ,2024) [1].

2.2.1. Arquitectura Monolítica y Arquitectura de Microservicios

En el paradigma de la **Continuidad Computacional**, es fundamental entender cómo el diseño arquitectónico de las aplicaciones **influye en su adaptabilidad, escalabilidad y eficiencia**. Tradicionalmente, las aplicaciones se han desarrollado con una arquitectura monolítica, en la que todas las funcionalidades se integran en un único bloque de código y se despliegan como una unidad indivisible. Este enfoque ofrece una implementación sencilla y una comunicación interna eficiente, pero presenta limitaciones considerables en entornos distribuidos y altamente dinámicos, como es el caso de la Continuidad Computacional, debido a la rigidez en la escalabilidad y a la dificultad para implementar cambios o actualizaciones sin afectar a todo el sistema.

Por otro lado, la arquitectura de microservicios propone **descomponer** la aplicación en **servicios pequeños, autónomos y especializados**. Ca-

da microservicio se encarga de una función concreta y se puede desplegar, escalar y actualizar de forma independiente. Esta metodología resulta especialmente ventajosa en el contexto de la Continuidad Computacional, ya que permite asignar y gestionar recursos de forma dinámica a lo largo de las diferentes capas (nube, niebla y borde). La flexibilidad inherente de los microservicios facilita la adaptación del sistema ante cambios en la demanda, optimiza la utilización de recursos y mejora la resiliencia general: la falla en un microservicio no compromete el funcionamiento del sistema completo. En este sentido, el diseño basado en microservicios es compatible con enfoques de orquestación automatizada y con la implementación de políticas de gestión de recursos en tiempo real. Estas capacidades son cruciales para abordar la heterogeneidad y la distribución geográfica de los recursos que caracterizan a la Continuidad Computacional, permitiendo que cada componente opere con la máxima eficiencia en su entorno específico.

Categoría	**Arquitectura Monolítica**	**Arquitectura de Microservicios**
Integración	Todos los componentes se integran en un único bloque de código y se despliegan de forma conjunta.	Cada funcionalidad se separa en servicios independientes que se comunican a través de APIs.
Escalabilidad	Escalado vertical (mejor hardware); poco flexible ante cambios en demanda en secciones específicas.	Escalado horizontal; cada microservicio se puede escalar de forma independiente según la demanda.
Flexibilidad y Mantenibilidad	Las actualizaciones o cambios requieren la modificación del sistema completo, lo que puede complicar el mantenimiento.	Permite cambios y actualizaciones de forma aislada, lo que facilita la evolución continua y el mantenimiento.
Tolerancia a fallos	Una falla en un componente puede afectar a toda la aplicación, dificultando la resiliencia.	Fallas aisladas en un microservicio no comprometen la totalidad del sistema, mejorando la resiliencia.
Despliegue y Actualización	El despliegue es único; actualizar una parte implica desplegar la aplicación completa de nuevo.	Los microservicios se despliegan y actualizan de forma independiente, permitiendo una integración y entrega continuas.
Complejidad de Desarrollo Inicial	Es más sencillo de desarrollar inicialmente por tener un único código base, pero se complica a medida que el sistema crece.	Requiere un diseño más complejo desde el inicio para definir las interfaces y la comunicación entre servicios, pero es más adecuado para sistemas distribuidos y heterogéneos.
Adaptabilidad a la Continuidad Computacional	Menor flexibilidad para adaptarse a entornos distribuidos y heterogéneos; difícil de orquestar en escenarios dinámicos.	Alta adaptabilidad; facilita la integración y orquestación de recursos a lo largo de la nube, la niebla y el borde, maximizando la eficiencia y respuesta en entornos distribuidos.

Cuadro 2.1: Esquema comparativo de arquitecturas monolíticas y basadas en microservicios, según AWS [2]

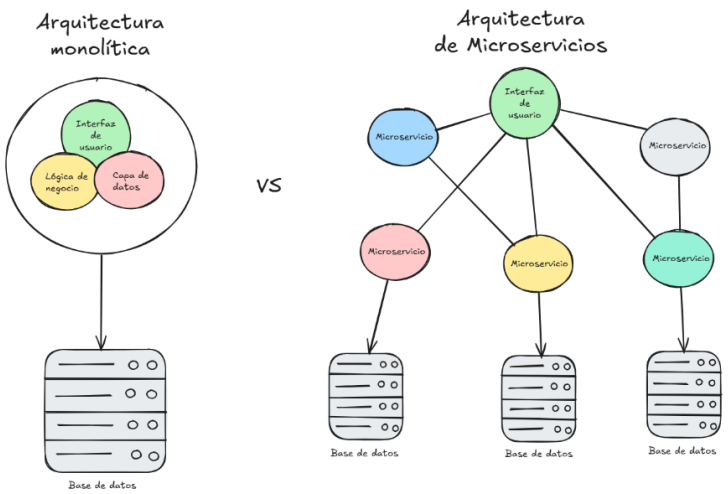


Figura 2.6: Representación gráfica de una arquitectura monolítica y una arquitectura basada en microservicios. Elaboración propia.

Con esta representación gráfica 2.6 se pretende ejemplificar la distribución de los servicios en la arquitectura monolítica y una arquitectura basada en microservicios. Se puede ver claramente donde la primera opción de la izquierda concentra todo en un único bloque mientras que la segunda opta por separar cada funcionalidad.

2.2.2. El Avance a la Continuidad Computacional

La **Continuidad Computacional** representa la **siguiente etapa en la evolución** de la infraestructura computacional. Al combinar los recursos de la nube, el borde y la niebla en un ecosistema, este modelo busca **maximizar la eficiencia y la interoperabilidad** a lo largo de esta continuidad de recursos entre plataformas heterogéneas (Morichetta, Pujol, & Dustdar, 2021)[46].

Sin embargo, la integración de estos recursos no está exenta de desafíos. Las **diferencias** en **arquitectura**, **protocolos** de comunicación y **capacidades** computacionales, han planteado **nuevos problemas** en la interoperabilidad, la programación de tareas y la seguridad de los sistemas distribuidos. Estos factores motivan la necesidad de investigaciones como las aquí presentes, que exploran cómo diseñar y gestionar estos sistemas de manera eficiente (Bendechache, Svorobej, Endo, & Lynn, 2020) [5].

2.2.3. Desafíos de la Continuidad Computacional

A pesar de su potencial, la Continuidad Computacional presenta una serie de **desafíos técnicos, operativos y de diseño** que requieren atención urgente para garantizar su implementación efectiva en entornos reales. Estos retos pueden agruparse en varias categorías clave:

1. **Heterogeneidad de los recursos**

   La Continuidad Computacional combina dispositivos y sistemas con capacidades computacionales, arquitecturas y protocolos de comunicación extremadamente variados. Por ejemplo, dispositivos IoT con recursos limitados en términos de CPU, memoria y almacenamiento deben interactuar con nodos de borde más potentes, infraestructura de niebla intermedia y centros de datos en la nube. Este entorno altamente heterogéneo complica la interoperabilidad y la gestión eficiente de los recursos distribuidos (Donta, Murturi, Pujol, Sedlak, & Dustdar, 2023) [19].

   Por otro lado, la integración de diferentes arquitecturas, como ARM o x86, requiere soluciones como **binarios universales** que puedan ejecutarse en múltiples plataformas sin necesidad de recompilación, un enfoque que todavía está en desarrollo (Danelutto, Dazzi & Torquati, 2024) [11].

2. **Distribución geográfica de los recursos**

   La gestión de recursos en la Continuidad Computacional es un proceso complejo que abarca dos pasos clave: (1) la provisión, que incluye la detección y selección de recursos adecuados, y (2) la programación, que se centra en la asignación de tareas, el monitoreo de recursos y el balanceo dinámico de la carga (Bendechache, Svorobej, Endo & Lynn, 2020) [5].

   En el contexto de la Continuidad Computacional, estos procesos enfrentan retos adicionales debido a la dispersión, incertidumbre y heterogeneidad de los recursos, dificultando garantizar una asignación eficiente y confiable de recursos en tiempo real. (Danelutto, Dazzi & Torquati, 2024) [11].

3. **Gestión de la energía y sostenibilidad**

   La eficiencia energética es un desafío fundamental, especialmente en dispositivos IoT y nodos de borde con fuentes de energía limitadas. Debemos asegurar que estos dispositivos no se queden sin energía, relacionando recursos y costos operativos (Dustdar, Pujol, & Donta, 2023) [21].

4. **Seguridad y privacidad de los datos**

   La naturaleza distribuida de la Continuidad Computacional aumenta la superficie de ataque, exponiendo datos sensibles a riesgos como intercepciones y manipulaciones. Los desafíos incluyen la protección de los datos, la implementación de controles de acceso y el monitoreo continuo para garantizar la seguridad en un entorno distribuido (Donta, Murturi, Pujol, Sedlak, & Dustdar, 2023) [19].

   Además, según Al-Dulaimy et al. (2024) [1], tecnologías como blockchain son útiles para garantizar la confianza y el no repudio en la cadena de suministro de datos, pero su implementación requiere un alto consumo de recursos y plantea nuevos retos de escalabilidad.

5. **Modelos de programación y protocolos**

   El diseño de aplicaciones en la Continuidad Computacional requiere nuevos modelos de programación que permitan gestionar la dinámica de recursos y la heterogeneidad del entorno. La falta de protocolos adaptativos dificulta la escalabilidad y optimización de recursos (Donta, Murturi, Pujol, Sedlak, & Dustdar, 2023)[19]. Siguientemente, la incorporación de técnicas de aprendizaje automático en la gestión de la Continuidad Computacional plantea interrogantes sobre la cantidad y calidad de datos necesarios para entrenar estos sistemas, así como sobre su capacidad de adaptación en tiempo real (Morichetta et al., 2021) [46].

6. **Coordinación y orquestación**

   La asignación de tareas y la coordinación eficiente entre múltiples capas de la Continuidad Computacional (nube, niebla y borde) requiere políticas de orquestación inteligentes. Estrategias como el balanceo de carga y la asignación dinámica de recursos son prometedoras, pero deben ser validadas en escenarios complejos y realistas para garantizar su efectividad (Danelutto, Dazzi & Torquati, 2024)[11].

Estos retos reflejan la necesidad de enfoques interdisciplinarios que combinen ingeniería de software, optimización de redes, ciberseguridad y sostenibilidad. Cada desafío ofrece oportunidades para explorar soluciones innovadoras que impulsen el desarrollo de la Continuidad Computacional como un paradigma viable y escalable para la computación moderna.

2.2.4. Metodologías de Gestión de Sistemas en la Continuidad Computacional

En el manejo efectivo de recursos en la Continuidad Computacional y para enfrentar su complejidad inherente, es fundamental adoptar técnicas

avanzadas que permitan una **gestión adaptativa y proactiva** de los recursos. Entre los enfoques clave de diseño destacan el uso de la **Manta de Markov (Markov Blanket)**, el **ciclo MAPE-K (Monitorizar, Analizar, Planificar y Ejecutar sobre conocimiento compartido)** y el **Principio de Energía Libre (Free Energy Principle, FEP)**.

Manta de Markov

En el contexto de sistemas distribuidos dentro del paradigma de Continuidad Computacional, Dustdar, Pujol y Donta (2023) [21] proponen un enfoque basado en espacios markovianos para modelar el comportamiento del sistema. Este modelo se estructura en torno a tres variables de alto nivel: **Recursos, Calidad y Costo**, lo que permite abstraer la complejidad de los entornos heterogéneos y facilitar su gestión. Sin embargo, esta abstracción exige adaptar las métricas observables según el dominio específico en el que se aplique.

Una de las herramientas clave en este enfoque es la Manta de Markov, que permite representar el sistema de forma modular, deducir estados no observables y seleccionar métricas críticas para la toma de decisiones. Actúa como un filtro de causalidad, eliminando relaciones redundantes y destacando aquellas métricas y acciones que son esenciales para mantener el sistema en equilibrio.

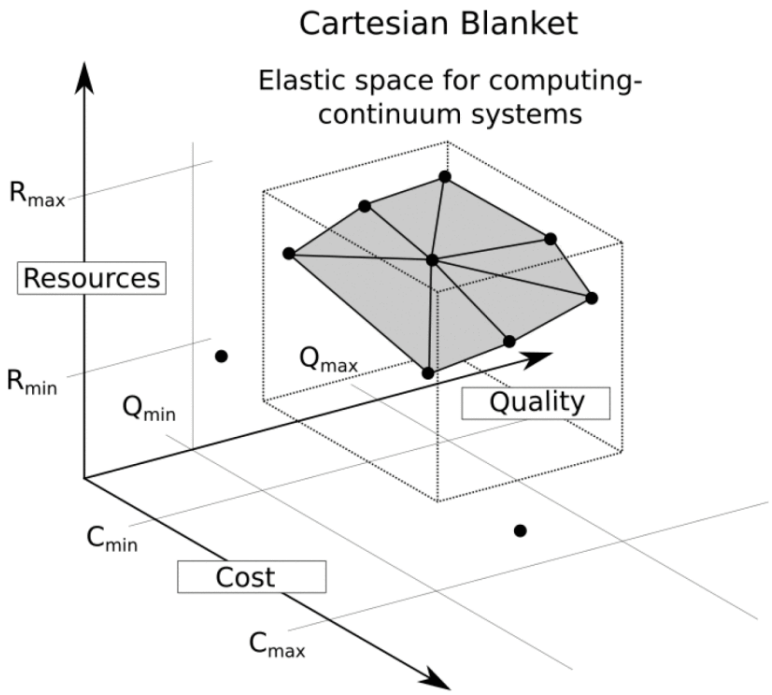


Figura 2.7: Manta cartesiana [21]

Complementariamente, el concepto de **manta cartesiana** (véase Figura 2.7) ofrece una perspectiva estructural que permite visualizar el espacio de configuraciones posibles del sistema en función de sus dimensiones clave. Mientras la manta de Markov se enfoca en la inferencia causal y la reducción de complejidad, la manta cartesiana facilita la exploración sistemática de combinaciones entre recursos, calidad y costo, ayudando a identificar regiones del espacio donde el sistema puede operar de forma óptima. Esta dualidad entre causalidad e interpretación estructural resulta especialmente útil en entornos dinámicos, donde la adaptabilidad y la eficiencia son prioritarias.

El equilibrio en un sistema continuo se define como un estado en el que la configuración de infraestructura satisface los requisitos de las aplicaciones. Cuando se producen cambios en la infraestructura o perturbaciones externas, el sistema puede perder dicho equilibrio, lo que requiere ajustes proactivos para restablecerlo. Para ello, el sistema debe ser capaz de identificar todas las configuraciones posibles y seleccionar la acción más adecuada.

Dado el elevado número de configuraciones potenciales, se hace imprescindible el uso de técnicas de **aprendizaje activo y adaptativo**, tanto en la fase de diseño como durante la ejecución. Este enfoque no solo permite preservar la estabilidad del sistema, sino que también reduce la dependencia

de respuestas exclusivamente correctivas, habilitando una gestión más eficiente, resiliente y dinámica en entornos de Continuidad Computacional.

Ejemplo ilustrativo, basado en el ejemplo citado de Dustdar et al., 2023 [21]:

Imaginemos un barco que transporta contenedores, similar al que se puede ver en la siguiente imagen (Figura 2.8). Mantener el equilibrio del barco es crucial para evitar que los contenedores caigan. Esto simboliza la necesidad de ajustar dinámicamente la infraestructura para mantener el rendimiento de la aplicación.

Figura 2.8: Carguero de contenedores [13]

Por otro lado, el barco está dirigido por un capitán. Su experiencia para interpretar las condiciones del mar es similar a la Manta de Markov. Se encargará de filtrar la información relevante para tomar decisiones efectivas.

Ciclo MAPE-K

Siguiendo el enfoque propuesto por Pujol, Donta, Morichetta, Murturi y Dustdar (2023) [49], este ciclo proporciona una arquitectura modular para la gestión de sistemas auto-adaptativos, estructurada en cinco bloques funcionales:

- **Monitorización**: Captura datos del sistema mediante sensores distribuidos.
- **Análisis**: Evalúa las condiciones del sistema, apoyándose en herramientas como el Principio de Energía Libre (FEP, por sus siglas en

inglés: Free Energy Principle) y la Manta de Markov para inferir estados no observables y reducir la incertidumbre.

- **Planificación**: Desarrolla estrategias de ajuste en función del estado analizado, anticipando posibles escenarios futuros.
- **Ejecución**: Aplica las estrategias seleccionadas mediante actuadores que modifican el comportamiento del sistema.
- **Conocimiento**: Proporciona una base compartida de información, enriquecida por el FEP, que orienta la toma de decisiones y permite la retroalimentación continua.

Este marco modular separa claramente el sistema de gestión del sistema gestionado, tal y como se puede ver en la imagen 2.9, lo que permite una mayor flexibilidad, escalabilidad y capacidad de adaptación en entornos distribuidos y dinámicos.

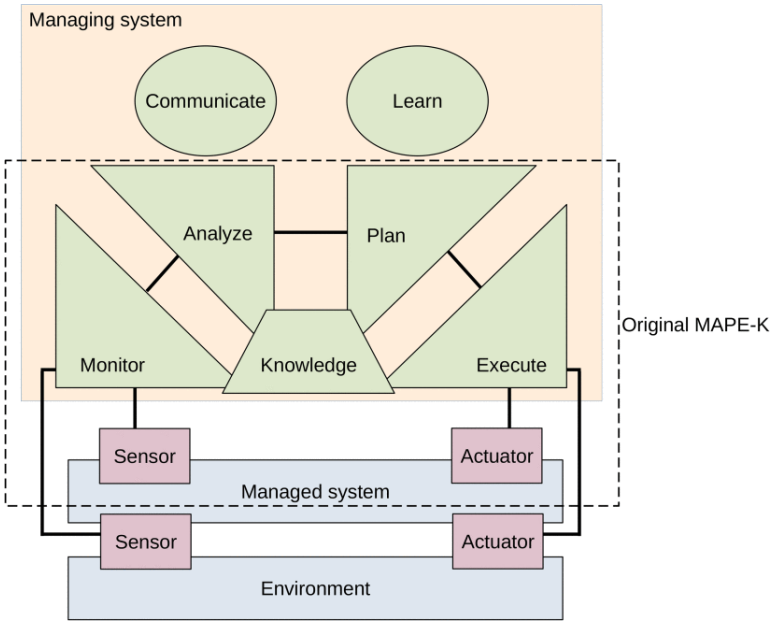


Figura 2.9: Representación del Ciclo MAPE-K [49]

Principio de Energía Libre

El Principio de Energía Libre (FEP, por sus siglas en inglés: Free Energy Principle), proveniente de la ciencia cognitiva, sostiene que un sistema adaptativo busca minimizar la diferencia entre las observaciones esperadas y las reales del entorno. Según Pujol, Donta, Morichetta, Murturi y Dustdar (2023)[49], esto implica que el sistema debe contar con un modelo generativo

del entorno que le permita anticipar y ajustar su comportamiento de forma dinámica. Este enfoque aporta varias ventajas clave:

- **Reducción de incertidumbre**: Facilita la adaptación continua a entornos dinámicos al identificar y minimizar discrepancias entre lo observado y lo esperado.
- **Interacción eficiente**: Promueve la cooperación entre componentes del sistema al buscar un equilibrio global, en lugar de optimizar cada recurso de forma aislada.
- **Escalabilidad**: Puede aplicarse eficazmente a diferentes niveles del sistema, lo que resulta especialmente útil en arquitecturas distribuidas de gran escala como las que plantea la Continuidad Computacional.

En el contexto del ciclo MAPE-K, el FEP complementa el bloque de **Conocimiento**, proporcionando información clave sobre el comportamiento esperado tanto del sistema como del entorno. Asimismo, contribuye al bloque de **Análisis**, ayudando a identificar patrones de discrepancia que orientan la planificación y la toma de decisiones adaptativas.

2.3. Los Sistemas Autoadaptativos

Los sistemas autoadaptativos se definen por su capacidad para ajustarse automáticamente a los cambios tanto en su entorno como en su propio estado. Esto se logra mediante un proceso iterativo que se articula en torno al ciclo MAPE-K, previamente mencionado, el cual comprende las fases de Monitorización, Análisis, Planificación y Ejecución. Este ciclo permite mantener el rendimiento y la calidad del servicio sin necesidad de intervención humana directa.

En primer lugar, el sistema debe **monitorizar** de forma constante y exhaustiva tanto su funcionamiento interno (métricas de rendimiento, consumo de recursos, estado de sus componentes, etc.) como las condiciones del entorno en el que opera (carga de trabajo, disponibilidad de recursos externos, interacciones con otros sistemas, etc.). Esta monitorización continua es crucial para capturar las variaciones relevantes que puedan afectar al sistema.

Una vez que los datos de la monitorización están disponibles, el sistema debe proceder a la **detección de cambios relevantes**. No todas las fluctuaciones en el entorno o en el estado interno requieren una adaptación. Es crucial que el sistema pueda distinguir qué variaciones son significativas y potencialmente perjudiciales para sus objetivos, o que presentan oportunidades para una optimización. Esta detección puede basarse en la comparación

con umbrales predefinidos, la identificación de tendencias anómalas o la aplicación de modelos predictivos.

Tras la detección de un cambio relevante, debemos **aplicar estrategias de adaptación**. Esto implica la toma de decisiones sobre qué modificaciones realizar en el sistema para responder al cambio detectado. Estas estrategias pueden variar ampliamente, desde ajustes en la configuración (por ejemplo, la asignación de memoria, la prioridad de los procesos, la política de gestión de energía) hasta reconfiguraciones más complejas de la arquitectura del sistema (por ejemplo, la adición o eliminación de componentes o la modificación de las interconexiones). La elección de la estrategia de la adaptación adecuada depende de múltiples factores, incluyendo la naturaleza del cambio detectado, los objetivos del sistema y las capacidades de adaptación que han sido diseñadas en él.

Finalmente, la estrategia de adaptación seleccionada debe ser **ejecutada**, lo que implica la implementación de los cambios decididos en el sistema en funcionamiento. Este proceso de ejecución debe ser llevado a cabo de manera controlada y segura para evitar introducir inestabilidad o interrumpir el servicio de forma no deseada.

El principal objetivo de los sistemas autoadaptativos radica en su capacidad para **mantener el rendimiento y la calidad del servicio sin intervención humana directa**. Esto es especialmente valioso en sistemas complejos y distribuidos, donde la gestión manual de las adaptaciones sería laboriosa, lenta y propensa a errores. Al automatizar el proceso de adaptación, estos sistemas pueden responder de manera más ágil y eficiente a las dinámicas cambiantes, asegurando una operación continua y optimizada.

2.3.1. Modelos Conceptuales de Auto-Adaptación

Según Weyns, 2020 [65], la auto-adaptación puede concebirse como:

1. **Perspectiva Externa (Caja Negra)**

   En este enfoque, un sistema autoadaptativo se concibe como una çaja negra"que ajusta automáticamente su funcionamiento basándose únicamente en la relación entre las entradas y las salidas. Desde esta postura, el sistema reacciona a las variaciones en su entorno y en sus propias métricas sin exponer o depender de detalles internos. La ventaja de este enfoque es su simplicidad, ya que permite centrarse en los resultados observables de la adaptación sin profundizar en la complejidad de su mecanismo interno.

2. **Perspectiva Interna (Bucle de Retroalimentación)**

Contrariamente, la perspectiva interna enfatiza la importancia de un bucle de retroalimentación cerrado que monitoriza continuamente tanto el entorno como el estado interno del sistema. Este enfoque distingue entre:

a. Preocupaciones del Dominio: Relacionadas con los objetivos y necesidades del usuario.

b. Preocupaciones de la Adaptación: Referentes a cómo el sistema, a nivel interno, ajusta sus parámetros para cumplir con dichos objetivos bajo condiciones cambiantes.

En este sentido, el sistema se divide en dos componentes: uno encargado de interactuar con el entorno y satisfacer las demandas del dominio, y otro que, mediante un ciclo de retroalimentación, analiza, planifica y ejecuta las adaptaciones necesarias para mantener el equilibrio y la calidad del servicio.

Otro aspecto importante que abordan los modelos conceptuales es la representación del conocimiento. Para tomar decisiones informadas sobre cómo adaptarse, el modelo conceptual se estructura en cuatro componentes principales:

1. **Entorno**: Representa la parte externa con la que interactúa el sistema. Incluye elementos como sensores, dispositivos de entrada y actores externos (por ejemplo, pacientes, usuarios o proveedores de servicios) cuyos datos y condiciones afectan el comportamiento del sistema.

2. **Sistema Gestionado**: Comprende el software de aplicación que realiza las funciones destinadas a los usuarios. Este sistema se encarga de detectar cambios en el entorno mediante sensores y responder a ellos, aplicando estrategias de adaptación.

3. **Objetivos de Adaptación**: Representan las metas o requisitos de calidad que el sistema de gestión debe garantizar en el sistema gestionado. Estos objetivos pueden incluir la auto-configuración, auto-optimización, auto-sanación y autoprotección. Cada uno de ellos establece criterios específicos, como mantener un rendimiento óptimo, minimizar el consumo de recursos o asegurar la protección contra ataques.

4. **Bucle de Retroalimentación**: Es el mecanismo central que permite la auto-adaptación. Se encarga de:

   a. Monitorizar: Recoger datos continuamente tanto del entorno como del sistema gestionado.

   b. Analizar: Evaluar las condiciones actuales comparándolas con los objetivos de adaptación, identificando desviaciones o oportunidades de mejora.

c. Planificar: Diseñar estrategias o configuraciones alternativas para corregir o anticipar las variaciones.

d. Ejecutar: Aplicar las adaptaciones necesarias de manera controlada y segura.

Este ciclo puede adoptar políticas reactivas, respondiendo a cambios una vez que se detectan, o proactivas, anticipando posibles desviaciones antes de que se manifiesten.

El ciclo MAPE-K puede considerarse una implementación concreta y estructurada del concepto más general de bucle de retroalimentación. Mientras que este último describe el flujo de información y ajustes en cualquier sistema adaptativo, MAPE-K lo formaliza mediante componentes funcionales (Monitor, Analyze, Plan, Execute, y Knowledge) que permiten diseñar arquitecturas auto-gestionadas con mayor claridad, modularidad y trazabilidad.

La integración de estos componentes permite que el sistema se mantenga adaptativo y resiliente en entornos dinámicos y complejos, como los que se encuentran en la Continuidad Computacional.

2.3.2. Conciencia del Contexto y Modelado del Entorno

La necesidad de la auto-adaptación surge de la naturaleza dinámica e impredecible de los entornos en los que operan los sistemas.

Una parte esencial de la auto-adaptación es la **conciencia del contexto**, es decir, la capacidad del sistema para percibir y comprender su situación actual. Esto implica recopilar y analizar información del entorno, como recursos disponibles, condiciones de red, ubicación geográfica, entre otros. Para lograr esta comprensión, se utilizan **modelos del entorno**, que son representaciones abstractas del mundo exterior que son relevantes para el sistema y sus objetivos. Estos modelos se actualizan constantemente y son fundamentales para la toma de decisiones informadas dentro del ciclo MAPE-K.

Además de modelar el entorno externo, un sistema autoadaptativo también debe ser capaz de **modelar su propio estado interno**. Esto implica mantener información actualizada sobre la configuración de sus componentes, su rendimiento actual (por ejemplo: latencia y tasa de error), su consumo de recursos y su estado de salud general. La conciencia del propio estado es esencial para determinar si el sistema está funcionando dentro de los parámetros aceptables y si necesita realizar adaptaciones para optimizar su rendimiento o recuperarse de fallos.

2.3.3. Mecanismos de la Auto-Adaptación

Para mantener la estabilidad y eficiencia, los sistemas autoadaptativos incorporan mecanismos como la auto-optimización, auto-recuperación, auto-protección y auto-configuración. Cada uno de estos procesos contribuye a que el sistema pueda ajustarse de manera autónoma ante fallas o cambios en el entorno.

Auto-Optimización

La auto-optimización se refiere a la habilidad de un sistema para buscar continuamente maneras de utilizar sus recursos de la forma más eficiente posible, sin comprometer los objetivos de calidad para los que fue diseñado. En sistemas con recursos limitados como memoria, potencia de cálculo, ancho de banda o energía, esta capacidad es vital. El sistema monitoriza su rendimiento y el consumo de recursos, evalúa configuraciones alternativas y elige aquellas que permiten alcanzar los mismos niveles de calidad con un menor coste de recursos. Por ejemplo, un sistema podría ajustar la asignación de memoria a diferentes procesos basándose en su uso actual para evitar cuellos de botella o reducir el consumo total.

Auto-Recuperación

La auto-recuperación es la capacidad de un sistema para detectar automáticamente la aparición de fallos, ya sean errores de software o hardware, y para iniciar procesos de recuperación sin intervención manual. El objetivo es mantener la operatividad del sistema y la calidad de sus servicios a pesar de estos incidentes. Un sistema auto-recuperable podría, por ejemplo, reiniciar un componente que ha fallado, redirigir el tráfico a una instancia de respaldo o intentar reparar una estructura de datos corrupta. La rapidez y efectividad de estos mecanismos son cruciales para minimizar el impacto de los fallos en los usuarios.

Auto-Protección

La auto-protección implica la capacidad de un sistema para defenderse activamente contra ataques maliciosos y para anticipar posibles problemas de seguridad. A diferencia de las herramientas de seguridad tradicionales, un sistema auto-protector puede analizar patrones de tráfico, detectar comportamientos anómalos y tomar medidas para mitigar amenazas en tiempo real. Esto podría incluir el bloqueo de direcciones IP sospechosas, el aislamiento de componentes comprometidos o la modificación de las reglas de acceso para prevenir intrusiones futuras. La meta es mantener la integridad y la disponibilidad del sistema frente a las amenazas de seguridad.

Auto-Configuración

La auto-configuración es la habilidad de un sistema para integrar au-

tomáticamente nuevos elementos o componentes sin interrumpir su funcionamiento normal y con una mínima o nula intervención manual. Cuando se añade un nuevo componente, el sistema puede descubrirlo, asignar los recursos necesarios, establecer las comunicaciones adecuadas y configurarlo para que funcione de manera coherente con el resto del sistema. Esto facilita la escalabilidad y la flexibilidad, permitiendo que el sistema crezca y se adapte a nuevas necesidades sin la complejidad de la configuración manual.

2.4. Estrategias para Abordar la Heterogeneidad de la Continuidad Computacional

Para abordar la heterogeneidad de la Continuidad Computacional, se destacan varias estrategias técnicas implementadas en simuladores recientes (Danelutto, Dazzi & Torquati, 2024)[11]:

- **Programación paralela estructurada**: Esta técnica simplifica el desarrollo y gestión de aplicaciones paralelas mediante patrones predefinidos (e.g., farm, map, divide-and-conquer). Gracias a estos patrones, los desarrolladores pueden mejorar la legibilidad y mantenibilidad del código, facilitando su integración en arquitecturas distribuidas de la Continuidad Computacional.

- **Binario universal**: Permite empaquetar diferentes formatos ejecutables en un único despliegue, de manera que una aplicación puede ejecutarse en arquitecturas heterogéneas (x86, ARM, PowerPC) sin necesidad de recompilación.

- **Capa de abstracción de comunicación**: Herramientas como la biblioteca MTCL proporcionan una interfaz uniforme para la transmisión y recepción de datos. Esto permite soportar protocolos como TCP, MPI y MQTT, adaptándose dinámicamente a las condiciones de la red y mejorando la eficiencia en entornos distribuidos.

Para abordar la heterogeneidad del hardware en la Continuidad Computacional, diversas herramientas y técnicas han cobrado relevancia:

- **WebAssembly (Wasm)**: Permite ejecutar código en entornos aislados, garantizando portabilidad y seguridad. Es una opción clave para aplicaciones que requieren flexibilidad en despliegues distribuidos (Tinto & Vardanega, 2024)[60].

- **Rust**: Proporciona garantías de seguridad en memoria, reduciendo vulnerabilidades críticas y mejorando la eficiencia en la programación para entornos distribuidos (Tinto & Vardanega, 2024)[60].

- **Programación paralela estructurada**: Simplifica el desarrollo de aplicaciones paralelas mediante patrones predefinidos, como farm, map o divide-and-conquer, mejorando la legibilidad y mantenibilidad del código (Danelutto, Dazzi & Torquati, 2024)[11].
- **Binario universal**: Permite empaquetar diferentes formatos ejecutables en un único despliegue, posibilitando la ejecución de una aplicación en arquitecturas heterogéneas (x86, ARM, PowerPC) sin necesidad de recompilación (Danelutto, Dazzi & Torquati, 2024)[11].
- **MTCL**: Biblioteca que facilita la abstracción de la comunicación entre componentes distribuidos, proporcionando una interfaz uniforme para la transmisión y recepción de datos. Soporta protocolos como TCP, MPI y MQTT, adaptándose dinámicamente a las condiciones de red (Danelutto, Dazzi & Torquati, 2024)[11].

Estas tecnologías fortalecen la capacidad de programación, ejecución y simulación en Continuidad Computacional, facilitando la validación de soluciones adaptativas y eficientes.

2.5. Trabajos Relacionados

Esta sección tiene como objetivo proporcionar un contexto amplio y detallado de las contribuciones previas que guardan relación con la Continuidad Computacional. A través de la revisión y el análisis de diversos proyectos, se busca identificar las tendencias, metodologías y resultados clave que han influido en el desarrollo de la investigación propuesta.

2.5.1. Ejemplo de Aplicación en Control de Tráfico

El caso de uso mencionado en Dustdar et al (2023)[21] se plantea una aplicación orientada al control de tráfico, cuyo objetivo es detectar en tiempo real si los conductores utilizan el teléfono mientras conducen. Para ello, se implementa un sistema distribuido que integra diversos sensores (cámaras, radares, sensores de suelo) y unidades computacionales de borde con capacidades de inferencia de inteligencia artificial. La propuesta destaca por su enfoque en la Continuidad Computacional, mediante el cual se mapean los requerimientos de nivel de servicio (SLOs) a métricas medibles a través de un grafo acíclico dirigido (DAG) como el de la siguiente figura 2.10. Este enfoque permite no solo monitorizar el desempeño del sistema en términos de calidad, recursos y coste, sino también realizar un análisis causal que facilite la identificación de problemas y la ejecución de acciones correctivas.

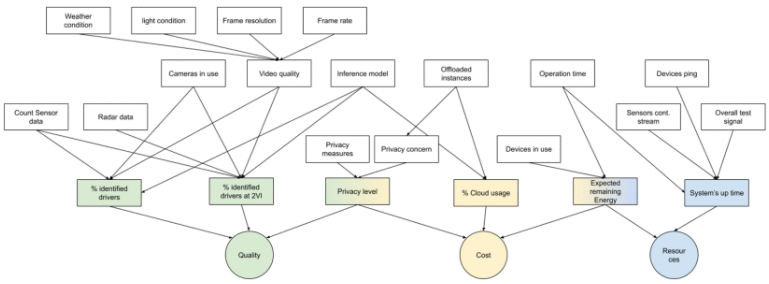


Figura 2.10: Representación del grafo acíclico dirigido del sistema [21]

Aunque la temática difiere del enfoque principal de este trabajo, este ejemplo ilustra de manera efectiva la aplicación práctica del concepto de Çomputing Continuum". La implementación de un DAG para el mapeo de SLOs y el análisis causal son técnicas directamente aplicables a nuestra investigación, ya que permiten una gestión más eficiente y adaptativa de los recursos computacionales en entornos distribuidos.

2.5.2. Aplicación IoT para la Monitorización de Seguridad de Edificios: Un Enfoque Auto-Adaptativo

Weyns (2020)[65] ofrece una visión comprensiva de los sistemas auto-adaptativos con ejemplos mediante DeltaIoT.

DeltaIoT es una aplicación para la monitorización de seguridad de edificios. Es un sistema integral de seguridad que utiliza dispositivos IoT interconectados para monitorizar y gestionar la seguridad de un edificio de manera inteligente y auto-adaptativa. El objetivo de la aplicación es proporcionar una solución que no solo detecte incidentes de seguridad, sino que también se adapte dinámicamente a las condiciones cambiantes y a las necesidades específicas del edificio.

El sistema se basa en una red de sensores distribuidos por todo el edificio, incluyendo sensores de movimiento y sensores de temperatura, entre otros. Estos sensores recopilan datos en tiempo real sobre diversas condiciones de seguridad y ambientales.

Una vez que los datos son recopilados, se transmiten a una plataforma donde se procesan para encontrar amenazas de seguridad y anomalías.

La aplicación está diseñada para auto-adaptarse a las condiciones cambiantes. Por ejemplo, en casos donde la pérdida de paquetes supera el umbral establecido como requisito de calidad, evalúa el contexto, selecciona las correcciones que debe hacer y las ejecuta. Éstas podrían ser desde generar

alertas, ejecutar un diagnóstico automático para identificar las causas y una reconfiguración de la red, ajustando parámetros de transmisión, como la potencia de señal o la frecuencia, para mitigar la pérdida de paquetes. También podría redirigir el tráfico a través de rutas alternativas para evitar nodos problemáticos.

2.5.3. Sistema Nautilus Adaptado al Caso SWAMP

La gestión eficiente de servicios en entornos distribuidos representa un desafío central en el despliegue de aplicaciones inteligentes de IoT. Un ejemplo ilustrativo lo ofrece el desarrollo de la plataforma SWAMP para riego inteligente (Kamienski et al., 2019)[30], que enfrentó dificultades significativas debido a las configuraciones heterogéneas entre distintas granjas a lo largo de la Continuidad Computacional. Las herramientas DevOps (Desarrollo y Operaciones para ofrecer valor a los clientes de forma constante) disponibles no permitían implementar automáticamente el mismo servicio en distintas etapas de la Continuidad Computacional, lo que exigía configuraciones manuales y carecía de soporte para la redistribución dinámica de servicios. En este contexto, el Sistema Nautilus (Borges Oliveira, Di Felice & Kamienski, 2024)[7] emerge como una solución inspiradora para abordar estas limitaciones. Su capacidad para monitorizar recursos y realizar migraciones inteligentes de servicios entre el borde y la nube bajo condiciones de carga lo convierte en un referente adaptativo.

Entre las funcionalidades de Nautilus que se alinean con las necesidades de la plataforma SWAMP destacan:

- Monitorización proactiva del estado operativo de los nodos, facilitando decisiones orquestadas.
- Migración dinámica de servicios en respuesta a cambios en la carga o condiciones del entorno.
- Compatibilidad con herramientas DevOps (Jenkins, Kubernetes, Docker) para la entrega continua en arquitecturas distribuidas.

Sobre la base de estos desafíos, surge la necesidad de definir un flujo de despliegue robusto y automatizado que responda a las exigencias del IoT moderno. Este es el flujo que se describe en el trabajo de Borges Oliveira, Di Felice y Kamienski (2024)[7]:

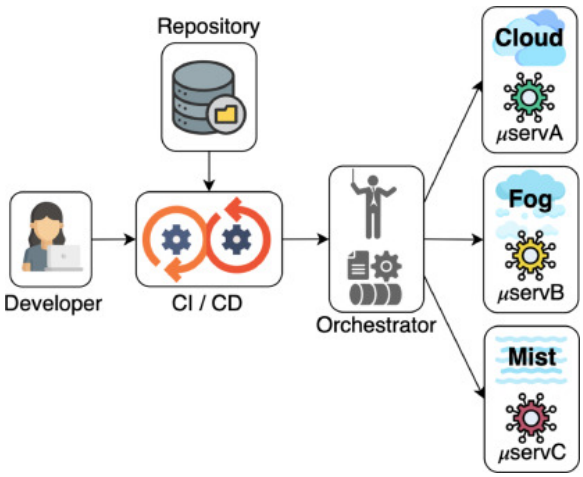


Figura 2.11: Esquema del proceso de despliegue [7]

En la figura 2.11 podemos ver que el proceso de despliegue consiste en:

1. El Desarrollador u Operador inicia el proceso de despliegue desde una herramienta CI/CD (Integración Continua / Entrega Continua), como Jenkins.

2. La herramienta CI/CD obtiene del repositorio (por ejemplo, GitHub) el código fuente y los manifiestos de configuración que definen la infraestructura y los microservicios a desplegar.

3. La herramienta CI/CD invoca al orquestador (Kubernetes o KubeEdge) para gestionar el despliegue.

4. El orquestador interpreta las configuraciones del manifiesto y despliega los microservicios en los distintos entornos definidos: Mist, Fog y Cloud.

5. Los agentes locales del orquestador, instalados en cada dispositivo, reciben las órdenes, descargan las imágenes Docker y activan los contenedores correspondientes en el entorno adecuado.

El desarrollo de IoT incorpora y extiende estas capacidades mediante un cauce de DevOps, que permite migraciones estáticas y dinámicas de microservicios en el IoT de la Continuidad Computacional, adaptándose a la infraestructura disponible en cada instalación. En consecuencia, introduce plugins personalizados en la herramienta CI/CD que comunican decisiones de despliegue al orquestador, incluso en configuraciones predefinidas sin intervención manual.

Esta aproximación consolida un enfoque adaptable y escalable para el despliegue de servicios inteligentes, marcando un paso decisivo hacia plataformas IoT más autónomas y resilientes.

Capítulo 3

Metodologías, Técnicas y Tecnologías para el Desarrollo

El presente capítulo se centra en las metodologías, técnicas y tecnologías que sustentan el desarrollo en el contexto de la Continuidad Computacional. En un entorno caracterizado por la heterogeneidad y la distribución geográfica de recursos, que abarca dispositivos IoT, nodos de borde, infraestructuras de niebla y centros de datos en la nube, es esencial adoptar enfoques de desarrollo que garanticen la adaptabilidad y eficiencia de las aplicaciones distribuidas.

Este capítulo proporciona una visión integral de los métodos y tecnologías que permiten desarrollar, simular y desplegar aplicaciones en la Continuidad Computacional, fundamentando el enfoque adoptado en este trabajo para abordar los desafíos de un entorno de computación distribuido y dinámico.

3.1. Metodologías de Desarrollo

En el contexto de la Continuidad Computacional, caracterizado por entornos altamente dinámicos y heterogéneos, la elección de una metodología de desarrollo adecuada es crucial para garantizar la evolución continua del software y su adaptación a los cambios en la infraestructura y los requisitos del sistema. Los sistemas diseñados para la Continuidad Computacional requieren **enfoques iterativos** que promuevan la **evolución constante** del software y la **capacidad** de **respuesta** a las **variaciones** del entorno. En este sentido, destacan metodologías como el **prototipado evolutivo** y los enfoques basados en **experimentación controlada**, los cuales permiten validar tempranamente configuraciones de recursos y estrategias de asigna-

ción antes de su implementación a gran escala. (Pujol, Raith, & Dustdar, 2021)[50].

Para implementar sistemas robustos y adaptables en la Continuidad Computacional, se requiere la aplicación de metodologías de desarrollo que permitan una evolución continua y una respuesta ágil a los cambios. A continuación, se describen las metodologías más relevantes.

3.1.1. Modelo en Cascada

Este enfoque se caracteriza por un **proceso lineal y secuencial en el que cada fase** (análisis y recopilación de requisitos, diseño, implementación, pruebas, despliegue y mantenimiento) se **ejecuta de forma ordenada y sin solapamientos**, de forma que cada fase se debe de completar antes de pasar a la siguiente.

Características principales:

- Estructura clara con pasos bien definidos que se ejecutan de forma secuencial.
- Fases bien definidas y documentadas.

Ventajas:

- Claridad en la planificación y documentación.
- Fácil de entender y gestionar en requisitos fijos.

Desventajas:

- Rigidez ante cambios imprevistos, ya que cualquier modificación puede requerir rehacer fases anteriores.
- Difícil retroalimentación, lo que puede retrasar la corrección de errores.
- Retrasa las pruebas hasta después de la finalización.

Relación con la Continuidad Computacional: Este enfoque puede ser útil en escenarios con requisitos muy estables. Su falta de flexibilidad la hace menos adecuada para entornos donde la adaptabilidad y la respuesta rápida a cambios son esenciales.

3.1.2. Modelo Iterativo

Este modelo permite el **desarrollo del software en ciclos repetitivos**, en los cuales cada iteración involucra las fases de análisis, diseño, implementación y evaluación.

Características principales:

- Desarrollo en ciclos o iteraciones, donde cada ciclo abarca actividades de análisis, diseño, implementación y evaluación.
- Permite mejorar el producto de manera progresiva.

Ventajas:

- Flexibilidad para incorporar cambios y mejoras en cada iteración.
- Facilidad de validación y el ajuste continuo del producto.

Desventajas:

- Requiere un control riguroso para evitar desviaciones y asegurar la coherencia entre iteraciones.
- Puede aumentar los costos y tiempos si las iteraciones se extienden sin un control adecuado.

Relación con la Continuidad Computacional: Es un enfoque adecuado en entornos donde los requisitos evolucionan, ya que permite ajustar el diseño del sistema y la asignación de recursos conforme se obtienen nuevos datos y se valida el desempeño.

3.1.3. Prototipado Evolutivo

Este enfoque se centra en el desarrollo rápido de **prototipos que se refinan de manera progresiva** con base en la retroalimentación de los usuarios y pruebas experimentales.

Características principales:

- Desarrollo rápido de prototipos funcionales que se refinan a partir de la retroalimentación de los usuarios y pruebas empíricas.
- Enfoque en la validación temprana de hipótesis y funcionalidades.

Ventajas:

- Permite detectar y corregir errores en fases tempranas del proyecto.
- Facilita la adaptación del sistema a las necesidades reales del usuario, reduciendo el riesgo de desalineación con los objetivos.

Desventajas:

- Puede generar una planificación menos estructurada, si no se establecen límites y criterios claros para la evolución de los prototipos.
- La integración final de múltiples prototipos puede ser compleja si no se lleva un seguimiento adecuado.

Relación con la Continuidad Computacional: Permite experimentar con diferentes configuraciones de recursos y estrategias de asignación, siendo especialmente útil para validar enfoques adaptativos en escenarios heterogéneos.

3.1.4. Metodologías Ágiles

Estas metodologías se centran en el **desarrollo incremental y colaborativo,** con ciclos cortos de iteración y una constante comunicación entre los miembros del equipo.

Características Principales:

- Desarrollo incremental basado en iteraciones cortas (sprints) y reuniones regulares de revisión (daily stand-ups, sprint reviews y retrospectives).
- Alta comunicación y colaboración entre los miembros del equipo, con roles y responsabilidades definidos.
- Enfoque en la entrega continua de valor y la adaptabilidad a cambios en los requisitos.

Ventajas:

- Gran flexibilidad para adaptarse a cambios en los requerimientos y el entorno
- Permite una respuesta rápida ante imprevistos y facilita la integración de nuevas ideas durante el desarrollo.
- Fomenta la transparencia y la colaboración, lo que mejora la calidad del producto final.

Desventajas:

- Requiere compromiso y coordinación constante del equipo, lo que puede ser desafiante en equipos grandes o distribuidos.
- La planificación a largo plazo puede ser más difícil de prever debido a la naturaleza iterativa y adaptativa del proceso.

Relación con la Continuidad Computacional: Las metodologías ágiles son especialmente adecuadas para entornos dinámicos, ya que facilitan la iteración rápida y la adaptación continua a los cambios en la infraestructura y en los requerimientos de la aplicación.

3.2. Tecnologías de Simulación

Dada la dificultad y el costo asociados a la experimentación en infraestructuras reales, las herramientas de simulación han demostrado ser fundamentales para el estudio de la gestión de recursos en la Continuidad Computacional. Tecnologías como **iFogSim, CloudSim, OMNeT++, GreenCloud y FogNetSim++** permiten modelar escenarios complejos, evaluar políticas de asignación de recursos y simular redes de nodos niebla y de borde (Bendechache et al., 2020; Qayyum et al., 2018)[5][51].

Simuladores avanzados como **SCORE y CloudLightning Simulator** ofrecen enfoques híbridos que combinan eventos discretos con técnicas multiagente, permitiendo evaluar dinámicas complejas en ecosistemas altamente distribuidos (Fernández-Cerero et al., 2018; Filelis-Papadopoulos et al., 2019)[23][24].

Un enfoque complementario es el uso de **ontologías semánticas**, es decir, representaciones formales del conocimiento que definen conceptos y relaciones en un dominio específico, para modelar y gestionar eficientemente las relaciones y capacidades de los elementos del continuo nube-borde. La ontología **ECO (Edge-Cloud Ontology)** permite representar semánticamente nodos de computación, software desplegado y conexiones entre servicios (Bernabé et al., 2022)[6].

El uso de ECO y consultas SPARQL en simulaciones permite escenarios clave como:

- **Detección de nodos sobrecargados:** Identificando aquellos con uso crítico de CPU, memoria o espacio en disco, para evitar fallos.Un ejemplo puede ser el código que se puede ver en la siguiente figura 3.1.

```
SELECT ?computingNode
WHERE {
   ?computingNode eco:cpuLoad ?loadCPU .
   ?computingNode eco:ramMemoryFree ?ramFree .
   ?computingNode eco:diskSpaceFree ?diskFree .
   FILTER (?loadCPU > 90 || ?ramFree <100 || ?diskFree < 700)
}
```

Figura 3.1: Código para identificar nodos sobrecargados [6]

- **Búsqueda de nodos para despliegue:** Localizando recursos disponibles con capacidad suficiente para instalar software y mantener tiempos de respuesta óptimos. Ejemplo de código en la figura 3.2

```
SELECT ?computingNode
WHERE {
     ?connectionPoint eco:id "routerNetwork01"   .
     ?connectionPoint seas:connectsSystemThrough ?connection .
     ?connection seas:connectsSystem ?system .
     ?system eco:contains ?computingNode .
     ?computingNode eco:ramMemoryFree ?ramMemoryFree .
     ?service rdf:type eco:Service .
     ?service eco:id "S4" .
     ?service eco:ramRequirement ?ramRequirement .
     FILTER (?ramMemoryFree > ?ramRequirement) .
     ?computingNode eco:diskSpaceFree  ?diskSpaceFree .
     ?service eco:diskSpaceRequirement   ?diskSpaceRequirement .
     FILTER (?diskSpaceFree > ?diskSpaceRequirement) .
     ?computingNode eco:cpuLoad ?cpuLoad .
     FILTER (?cpuLoad < 30 ) .
}
```

Figura 3.2: Código para búsqueda de nodos para su despliegue [6]

- **Gestión de fallos de conexión:** Detectando aplicaciones afectadas por caídas de conexiones críticas.Ejemplo de código en la figura 3.3

```
SELECT  ?application
WHERE {
     ?application eco:isComposedOf ?serviceApp .
     {
        ?serviceApp eco:deployedIn ?middleware  .
        ?serviceApp eco:remoteConnectionTo ?anyRemoteService .
     }
     UNION
     {
        ?serviceApp eco:remoteConnectionTo+  ?remoteService  .
        ?remoteService eco:deployedIn ?middleware  .
     }
     ?middleware eco:deployedIn ?computingNode .
     ?system eco:contains ?computingNode .
     ?connection seas:connectsSystem ?system .
     ?connection eco:id "C2"  .
}
```

Figura 3.3: Código para la gestión de fallos de conexión [6]

La incorporación de estas técnicas semánticas permite mejorar la capacidad de simulación, facilitando la toma de decisiones adaptativas y asegurando la resiliencia del sistema.

3.3. Contenerización de Aplicaciones

Un contenedor es una forma de virtualización ligera que encapsula una aplicación junto con todas sus bibliotecas, dependencias y configuraciones necesarias para ejecutarse, sin necesidad de un sistema operativo completo. Esto lo convierte en una unidad portátil, eficiente y aislada de ejecución. En este contexto, existen varias tecnologías clave:

- Docker: plataforma para empaquetado y despliegue de contenedores,

surgida en 2013. Aporta herramientas robustas como Docker Engine, Docker Compose y Docker Hub, y utiliza un demonio central (dockerd) que gestiona los contenedores a través de API REST (Docker-vs-coreos, 2025)[62].

- RKT(Rocket): lanzado en 2014 por CoreOS, pensado como alternativa segura. Prescinde del demonio central, no necesita privilegios root para ejecutar contenedores (Docker-vs-coreos, 2025)[62].

A continuación, podemos ver una tabla comparativa 3.1 entre Docker y RKT basada en (Docker-vs-coreos, 2025)[62] y (Docker-vs-rkt Technology-Hub, 2025) [44] , (Docker-vs-rkt Stackups, 2025) [55]:

Característica	**Docker**	**RKT**
Arquitectura	Modelo cliente-servidor, demonio Docker monolítico	Sin demonio, integración con systemd, enfoque minimalista con arquitectura modular desde el inicio
Ecosistema y comunidad	Muy activo, con Docker Hub, Compose, CI/CD	Ecosistema más pequeño, menos soporte comunitario
Integraciones de terceros	Docker Hub ofrece más de 100.000 aplicaciones gratuitas, registros públicos y privados, con repositorios oficiales de proveedores líderes (Nginx, MongoDB, etc.)	Sistema tipo DIY (Do It Yourself), aunque todos sus proyectos están disponibles en GitHub
Usabilidad	CLI intuitiva, con interfaz gráfica como Docker Desktop	Menos intuitivo, sin interfaz gráfica oficial, curva de aprendizaje superior

Cuadro 3.1: Tabla comparativa entre Docker y RKT

3.4. Despliegue y Orquestación de Aplicaciones: Kubernetes (K8s)

Kubernetes es una plataforma de orquestación de contenedores diseñada para automatizar tareas de despliegue, escalado y gestión de aplicaciones

empaquetadas en contenedor. Sus componentes principales son (Kubernetes, 2025)[33]:

- **Pods**: Es la unidad más pequeña desplegable en Kubernetes. Representa una instancia de un proceso en ejecución dentro del cluster. Puede contener uno o más contenedores que comparten almacenamiento, red y especificaciones de ejecución. Todos los contenedores en un Pod comparten la misma IP, espacio de puertos y volúmenes (Pods)[35].
- **Servicio**: Expone una aplicación que se ejecuta en uno o más Pods, proporcionando una dirección IP estable y un nombre DNS. Permite la comunicación interna o externa con los Pods, incluso si estos cambian o se reinician (Service)[36].
- **Volumen**: Es un directorio accesible por los contenedores de un Pod, usado para almacenar datos persistentes. A diferencia del almacenamiento interno del contenedor, los volúmenes persisten entre reinicios (Volumes)[37].
- **Namespace**: Proporciona un espacio lógico para agrupar y aislar recursos dentro de un cluster. Es útil para multi-tenencia, control de acceso y organización por equipos o entornos (Namespaces)[34].

Las principales ventajas de su uso son:

- Escalado automático de aplicaciones según demanda de recursos
- Autoreparación: reinicia Pods caídos y mantiene la disponibilidad
- Balanceo de carga automático
- Despliegues continuos
- Configuración declarativa mediante archivos YAML/JSON, facilitando la reproducibilidad

Pese a tener grandes ventajas, plantea desafíos como un consumo elevado, ya que requiere de unos 2GB de RAM y uso considerable de CPU y disco por nodo, lo que lo hace poco práctico en dispositivos limitados como Raspberry PI, surgiendo distribuciones más ligeras como K3s o KubeEdge (LinuxFoundation) [59].

3.4.1. KubeEdge vs K3S

K3s: Es una distribución ligera de Kubernetes adaptada a entornos con dispositivos de bajos recursos[32].

Principales características:

- Tamaño pequeño (50 MB en disco, 300 MB de RAM por nodo).
- Está diseñado para ser sencillo de instalar y mantener, ideal en dispositivos con pocas prestaciones.

KubeEdge: Añade una capa de extensión a Kubernetes para ambientes de borde, con sincronización entre nube y nodos de borde (CNCF)[9]. Está compuesto por:

- CloudCore (coordina con K8s)
- EdgeCore y EdgeHub (gestionan contenedores localmente)
- EventBus basado en MQTT para comunicación confiable y tolerante a desconexiones

3.5. Herramientas para la Monitorización y Visualización del Sistema

3.5.1. Herramientas de Monitorización

Zabbix

Es una solución de monitorización para entornos empresariales grandes y complejos que utiliza una **arquitectura cliente-servidor**, con agentes desplegados en los sistemas a vigilar y un servidor central que recopila métricas, genera alertas y almacena datos en bases relacionales. Sus principales ventajas son:

- Vigilancia integral de hardware, red y servicios, con capacidad para procesar grandes entornos
- Alertas configurables y sistema de plantillas para facilitar la gestión

Entre sus desventajas destaca el hecho de que puede ser complejo de configurar y administrar en entornos de rápida evolución o contenerizados (Techrepublic)[57](Zabbix)[66].

Prometheus

Es una herramienta de monitorización y alertas centradas en series temporales. Sus características principales son (MetricFire - Prometheus vs Zabbix)[43] (Prometheus)[48]:

- Utiliza una base de datos propia optimizada para métricas en tiempo real
- Recoge datos mediante un modelo pull desde exporters

- Se integra con Grafana para visualización avanzada
- Ideal para entornos cloud-native y Kubernetes

Como se muestra en la Tabla 3.2, ambas herramientas presentan enfoques distintos:

Característica	**Zabbix**	**Prometheus**
Modelo de datos	Métricas, logs, eventos	Series temporales
Almacenamiento	Bases SQL externas	Base TSDB interna (retención por defecto de 15 días)
Visualización	Dashboards nativos, mapas, informes	Grafana (requiere configuración adicional)
Alertas	Integradas, con escalado y múltiples canales	Requiere Alertmanager para gestión avanzada
Facilidad de uso	Curva de aprendizaje elevada	Más sencillo de configurar, especialmente con Grafana

Cuadro 3.2: Comparativa de características entre Zabbix y Prometheus [53]

3.5.2. Herramientas de Visualización

Grafana

Es una herramienta de código abierto ampliamente utilizada para visualización de datos, dashboards y alertas. Está especializada en crear dashboard interactivos y personalizados. Aunque no recolecta datos por sí misma, se conecta a múltiples fuentes como Prometheus, InfluxDB,... para mostrar métricas de forma visual. Es ideal para entornos cloud-native, DevOps y observabilidad. Su interfaz es moderna, intuitiva y altamente personalizable, permitiendo incluso crear paneles con gráficos, mapas, alertas y más (Grafana)[27].

Zabbix

La Interfaz de Zabbix es una plataforma web robusta y orientada a la administración técnica, que permite visualizar datos, gestionar alertas y configurar elementos de monitoreo. Si bien ofrece paneles personalizables, su diseño es más funcional que visual. Permite explorar métricas detalladas por host, definir triggers y gestionar notificaciones. Está especialmente

pensada para entornos empresariales con alta granularidad de configuración (Zabbix)[66] (Squadcast - Grafana vs Zabbix)[54].

Como se muestra en la Tabla 3.3, las interfaces de Grafana y Zabbix presentan diferencias significativas en estilo visual, personalización y facilidad de uso.

Característica	**Grafana UI**	**Zabbix UI**
Estilo visual	Moderna, atractiva, con diseño tipo dashboard interactivo	Tradicional, funcional, orientada a tareas administrativas
Personalización	Altamente personalizable con paneles, temas, plugins y variables	Limitada a widgets y plantillas predefinidas
Facilidad de uso	Intuitiva, ideal para usuarios técnicos y no técnicos	Requiere de experiencia previa para aprovecharla al máximo
Interacción	Arrastrar y soltar, filtros dinámicos, anotaciones en tiempo real	Navegación basada en menús y formularios estáticos
Visualización de datos	Gráficos avanzados: líneas, barras, mapas, histogramas, etc.	Gráficos básicos: líneas, barras, mapas de red
Plugins	Ecosistema amplio con cientos de paneles y fuentes de datos	No admite plugins externos para visualización

Cuadro 3.3: Comparativa de interfaces entre Grafana y Zabbix [54]

3.6. Tecnologías de Desarrollo y Programación del Sistema (Frontend y Backend)

El **backend** es el corazón del sistema: **procesa datos, aplica lógica de negocio y expone APIs que utilizan las capas frontend o dispositivos**. En sistemas distribuidos y microservicios, elegir el lenguaje adecuado para el backend es crucial debido a sus implicaciones en rendimiento, escalabilidad, mantenibilidad y ecosistemas (logging, pruebas, seguridad, etc.).

Dos de los lenguajes más utilizados en este ámbito son Java y Python, cada uno con fortalezas distintas.

Java

Es un lenguaje compilado, orientado a objetos y multiplataforma, conocido por su robustez y rendimiento. Es ampliamente adoptado en entornos empresariales para construir aplicaciones backend escalables y seguras (Technoscriber - Java vs Python)[56] (MediumMultimedia - Beneficios de Java para el backend)[42].

Pros:

- Portabilidad: la aplicación se desarrolla una sola vez y puede funcionar en múltiples sistemas operativos sin modificaciones, gracias a la máquina virtual de Java (JVM), que actúa como una capa de compatibilidad entre el código y el entorno donde se ejecuta.
- Alto rendimiento y escalabilidad, ideal para sistemas con gran carga de tráfico
- Frameworks potentes como Spring Boot
- Seguridad avanzada, con gestión automática de memoria y control de acceso
- Amplia comunidad y ecosistema

Contras:

- Mayor complejidad y verbosidad: implica más líneas de código y configuración inicial
- Consumo de memoria mayor

Python

Es un lenguaje de programación interpretado, de alto nivel y con una sintaxis clara y legible. Es ampliamente utilizado en el backend por su rapidez en el desarrollo y su ecosistema de frameworks como Django, Flask y FastAPI. (DataCamp - Guía para el desarrollo backend en Python)[12] (Arsys - Python)[3] (Technoscriber - Java vs Python)[56].

Pros:

- Alta productividad: sintaxis clara, gran comunidad con frameworks ampliamente usados.
- Ecosistema de CI/CD y ML: fácil integración con bibliotecas de datos y aprendizaje automático, como Flask + Pandas, TensorFlow, Pythorch.
- Flexibilidad: tipado dinámico permite cambios rápidos, ideal para MVP y prototipos.

Contras:

- Menor rendimiento en tareas intensivas: la interpretación es más lenta que Java, requiere trabajo extra para optimización
- Tipo dinámico: puede causar errores en tiempo de ejecución sin un buen test que verifique los tipos de datos
- Uso de memoria más alto para cargas pesadas

El **frontend**, por el contrario, representa la **capa visual e interactiva del sistema, aquella con la que el usuario final se comunica directamente.** En sistemas distribuidos, esta capa debe ser eficiente, reactiva y capaz de integrarse con APIs externas, ofreciendo una experiencia fluida y dinámica.

Entre las tecnologías más utilizadas para el desarrollo front-end destacan **React** y **Vue.js**, ambas basadas en JavaScript y con arquitecturas orientadas a componentes reutilizables.

React React es una biblioteca de JavaScript desarrollada por Meta (Facebook) para la construcción de interfaces de usuario. Su enfoque basado en componentes y su sistema de renderizado eficiente mediante el **DOM (Document Objetct Model) virtual** lo convierten en una opción ideal para aplicaciones dinámicas y escalables (Why React Continues to Lead Front-End Development in 2025, 2025)[58] (Reactjs)[52].

Pros:

- Arquitectura basada en componentes reutilizables
- Gran rendimiento gracias al Virtual DOM
- Ecosistema robusto: React Router, Redux, Next.js, etc.
- Amplia comunidad y documentación
- Compatible con frameworks full-stack como Next.js

Contras:

- Curva de aprendizaje moderada (hooks, estado, etc.)
- Requiere configuración adicional para proyectos complejos
- No incluye funcionalidades como enrutamiento o gestión de estado por defecto

Vue.js Vue.js es un framework progresivo de JavaScript que permite construir interfaces reactivas de forma sencilla. Su sintaxis clara y su enfoque gradual lo hacen especialmente atractivo para desarrolladores que buscan rapidez y flexibilidad (Vue.js)[63] (Vue.js: A powerful Fronted Technology for modern web development) [41].

Pros:

- Curva de aprendizaje suave y sintaxis intuitiva
- Componentes encapsulados con HTML, CSS y JS en un solo archivo (.vue)
- Ecosistema completo: Vue Router, Vuex, Nuxt.js
- Ideal para proyectos de tamaño medio o para integrar en sistemas existentes

Contras:

- Menor adopción en grandes corporaciones frente a React
- Menor disponibilidad de desarrolladores especializados en algunos mercados

3.7. Bibliotecas de Machine Learning

El ecosistema de **Machine Learning** ha evolucionado rápidamente gracias al **desarrollo de bibliotecas especializadas** que permiten implementar modelos de forma eficiente, escalable y accesible. A continuación se presentan algunas de las bibliotecas más destacadas.

Scikit-learn Scikit-learn (Scikit-learn)[28] es una biblioteca de código abierto para Machine Learning en Python, construida sobre NumPy, SciPy y Matplotlib. Es ideal para tareas tradicionales como clasificación, regresión, clustering y reducción de dimensionalidad.

Ventajas:

- Interfaz simple y coherente.
- Gran variedad de algoritmos supervisados y no supervisados.
- Herramientas para preprocesamiento, validación cruzada y selección de modelos.
- Excelente documentación y comunidad activa.

TensorFlow Desarrollada por Google, TensorFlow (Codemotion - Herramientas Machine Learning)[10] es una biblioteca de Deep Learning que permite construir modelos complejos mediante grafos computacionales. Es altamente escalable y compatible con CPUs, GPUs y TPUs.

Ventajas:

- Soporte para redes neuronales profundas.
- Ecosistema amplio (TensorBoard, TensorFlow Lite).
- Ideal para producción y despliegue en múltiples plataformas.

PyTorch Creada por Facebook AI Research, PyTorch (Unite.AI)[61] es conocida por su enfoque dinámico en la construcción de modelos, lo que la hace especialmente popular en investigación y desarrollo académico.

Ventajas:

- Gráficos computacionales dinámicos.
- Sintaxis intuitiva y estilo Pythonic.
- Excelente para prototipado rápido.

3.8. Bases de Datos

Elegir el tipo de base de datos adecuada es fundamental para garantizar un rendimiento eficiente, escalabilidad, integridad y flexibilidad de esquema. Las dos grandes categorías son:

- **Base de datos relacionales**, como PostgreSQL, MySQL o MariaDB:

- Almacenan datos en tablas con filas y columnas fuertemente tipadas.
- Implementan relaciones entre tablas mediante claves primarias y foráneas
- Ofrecen transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), lo que asegura integridad y seguridad de los datos.

- **Bases de datos NoSQL**, como MongoDB o Redis:

- Almacenan datos en documentos, pares clave-valor, grafos o series temporales.
- No requieren de esquemas estrictos, lo que permite una mayor adaptabilidad al cambio de requisitos.

A continuación se describen las principales ventajas y desventajas de cada tipo de base de datos (SQL vs NoSQL)[29].

Entornos Relacionales
Ventajas:

- Alta fiabilidad gracias a transacciones ACID.
- Potente lenguaje de consulta SQL, con capacidades de agregación y joins complejos.
- Madurez y herramientas como pgAdmin con capacidades avanzadas de backup y replicación.
- Modelo estructurado, mejora para datos bien definidos y relaciones complejas.

Desventajas:

- Mayor rigidez en cambios de esquema, ya que requiere modificaciones que pueden implicar migraciones cuando evolucionan los requisitos.
- Escalabilidad vertical limitada, ya que crecer suele significar más CPU/-RAM/disco.

Entornos No Relacionales
Ventajas:

- Esquemas flexibles: los documentos JSON pueden tener campos dinámicos, permitiendo evolución sin interrupciones
- Escalabilidad horizontal
- Diseñadas para casos de uso específicos:
  - Documentales (MongoDB): ideal cuando los objetos son naturales en JSON y admiten consultas complejas sin joins.
  - Clave-valor o caché (Redis): extremadamente rápidas para acceso directo; útiles para estado sesión.

Desventajas:

- La mayoría no ofrecen transacciones ACID, dificultando las garantías de integridad.
- Menor soporte para relaciones complejas: unir documentos entre sí es menos natural que en SQL.

3.9. Cliente para APIs

Postman

Es una plataforma integral para el ciclo de vida de las APIs. Sus funcionalidades clave son:

- Petición y prueba de endpoints: Permite ejecutar llamadas HTTP, establecer encabezados, body, y analizar respuestas
- Colecciones reutilizables: Agrupa un conjunto de peticiones relacionadas

Esta herramienta destaca por ofrecer una **usabilidad extrema**, gracias a una interfaz gráfica muy intuitiva que facilita el trabajo de desarrolladores y testers. Además, cuenta con una versión orientada a la colaboración, lo que permite organizar equipos, gestionar el acceso y mantener repositorios de colecciones de manera eficiente. Incorpora también un **validador de calidad embebido**, capaz de verificar el funcionamiento de las APIs sin intervención humana, optimizando el control de calidad desde etapas tempranas. Todo esto la convierte en una solución ideal para procesos de integración entre aplicaciones móviles y arquitecturas distribuidas como nodos de borde, niebla o nube, así como para realizar pruebas durante la migración de estos entornos (Postman)[47].

3.10. Dispositivos Wearables y su Integración en la Continuidad Computacional

Los **dispositivos wearables**, entre los que se encuentran **pulseras inteligentes, relojes inteligentes** y otros accesorios tecnológicos, se caracterizan por incorporar microprocesadores, sensores avanzados y conectividad inalámbrica. Estas tecnologías permiten la **monitorización, preprocesamiento y transmisión de datos** fisiológicos y ambientales hacia entornos de computación distribuidos, como la computación de la niebla o la nube, integrándose eficazmente en el paradigma de la Continuidad Computacional.

Funcionando como nodos periféricos, los wearables capturan información en tiempo real y posibilitan respuestas rápidas gracias a su baja latencia, optimización energética y capacidad de adaptación contextual. Técnicas como el reconocimiento de patrones, el procesamiento de señales fisiológicas y el aprendizaje automático potencian su funcionalidad, permitiendo servicios personalizados y adaptativos para el usuario.

Fabricantes como Apple, Samsung, Xiaomi, Fitbit, Empatica o Mobvoi han impulsado un amplio abanico de wearables con funcionalidades cada vez

más sofisticadas, abarcando desde el seguimiento del estado físico hasta la investigación médica especializada. No obstante, aún persisten retos como la variabilidad en la precisión de los sensores, la interoperabilidad entre dispositivos y la protección de los datos personales, especialmente en aplicaciones críticas como salud, deporte o atención clínica.

Por tanto, el diseño de soluciones basadas en wearables debe contemplar tanto sus capacidades y limitaciones tecnológicas, como su integración eficiente en infraestructuras distribuidas, garantizando una gestión segura de los datos y una operación resiliente en contextos dinámicos.

3.10.1. Dispositivos Utilizados

Para el adecuado desarrollo del presente proyecto, resulta imprescindible realizar un estudio detallado de los dispositivos proporcionados, con el propósito de identificar y comprender tanto sus potencialidades funcionales como las limitaciones técnicas que puedan incidir en su implementación.

Pulsera Empatica E4 Band

La pulsera **E4** (Empatica E4 Band Manual[22]) es un dispositivo wearable diseñado para registrar parámetros fisiológicos en tiempo real. Además de su capacidad de monitorización continua, cuenta con una memoria interna que permite almacenar los datos recopilados para su sincronización y análisis posterior.

Su finalidad principal es proporcionar una herramienta de recopilación fiable para investigadores y profesionales del ámbito sanitario, facilitando el estudio y evaluación de las señales fisiológicas obtenidas durante el periodo de uso.

La E4 está equipada con varios sensores diseñados para la recogida continua de datos fisiológicos en tiempo real. Estos son los principales:

- **PPG (Photoplethysmography)**: mide el volumen de pulso sanguíneo, a partir del cual se derivan señales como la frecuencia cardíaca y la variabilidad del ritmo cardíaco.
- **EDA (Electrodermal Activity)**: registra la actividad electrodermal de la piel, útil para evaluar la activación del sistema nervioso simpático y detectar niveles de estrés o excitación emocional.
- **Temperatura de la piel**: sensor infrarrojo que mide la temperatura superficial del cuerpo, útil para estudios de regulación térmica y estados fisiológicos.

- **Acelerómetro de 3 ejes**: captura el movimiento del usuario en los ejes X, Y y Z, permitiendo analizar patrones de actividad física y postura.

SmartWatch TicWatch Pro

El **TicWatch Pro** (TicWatchPro3)[45] es un reloj inteligente desarrollado por Mobvoi que forma parte del ecosistema de dispositivos wearables. Está basado en el sistema operativo **Wear OS** (WearOS)[26] de Google, lo que permite la instalación de aplicaciones compatibles y facilita la interacción directa con servicios digitales desde el propio reloj.

Su diseño y funcionalidades están orientados al uso cotidiano, combinando aspectos como conectividad, seguimiento de actividad física, notificaciones y control básico del entorno digital del usuario. Además, debido al uso de un sistema operativo estándar, el dispositivo resulta compatible con entornos de desarrollo como Android Studio, lo que habilita la creación e instalación de aplicaciones personalizadas, ampliando su utilidad en contextos de experimentación o proyectos específicos.

Los principales sensores integrados en el TicWatch Pro son:

- **PPG (Photoplethysmography)**: mide la frecuencia cardíaca y permite estimar la variabilidad del ritmo cardíaco.
- **SpO2 (oxígeno en sangre)**: estima la saturación de oxígeno, útil para evaluar la función respiratoria.
- **Temperatura de la piel**: registra la temperatura superficial del cuerpo.
- **Acelerómetro y giroscopio**: detecta movimiento y orientación, útiles para el seguimiento de actividad física.
- **Barómetro y altímetro**: miden presión atmosférica y altitud, aplicables en actividades al aire libre.
- **Brújula digital**: proporciona orientación espacial.
- **GPS multibanda**: permite posicionamiento preciso sin depender del teléfono.

3.11. Conclusiones

En el transcurso de este apartado se han explorado diversas metodologías y herramientas que permiten abordar el desarrollo de sistemas distribuidos

dentro del marco de la Continuidad Computacional. Tras analizar distintos enfoques de desarrollo, se ha optado por emplear **metodologías ágiles**, por su capacidad para adaptarse a entornos evolutivos, facilitar la iteración continua y responder de forma dinámica a cambios en los requisitos.

Asimismo, se ha revisado el papel de las **tecnologías de simulación**, que ofrecen un entorno controlado para validar hipótesis, reducir costes y minimizar riesgos durante fases tempranas del diseño. No obstante, dado que estas soluciones superan el alcance práctico de este proyecto y presentan requerimientos específicos no alineados con el caso de uso planteado, se ha decidido no integrarlas en esta implementación.

Respecto a las tecnologías empleadas, se ha seleccionado un conjunto orientado a la eficiencia, escalabilidad y claridad en la gestión de servicios:

- **Docker**, como plataforma de contenerización, por su ligereza y amplio ecosistema.
- **Kubernetes**, como orquestador principal, para gestionar despliegues distribuidos de forma automatizada. Concretamente la versión de K3s.
- **Prometheus y Grafana**, para la recolección y visualización de métricas operativas en tiempo real, facilitando el análisis infraestructural.
- **Python**, como lenguaje de programación central, por su versatilidad y disponibilidad de bibliotecas científicas.
- **Scikit-learn**, como biblioteca de aprendizaje automático, ideal para modelos ligeros y reentrenables en entornos distribuidos.
- **PostgreSQL**, como base de datos relacional, por su robustez, escalabilidad y compatibilidad con sistemas externos.
- **Postman**, como herramienta cliente para APIs REST, facilitando pruebas, documentación y verificación de flujos de datos.

Este conjunto tecnológico permite abordar de manera eficiente las necesidades funcionales del sistema, favorecer su evolución futura y mantener un equilibrio entre complejidad técnica y viabilidad práctica.

Capítulo 4

Propuesta y Desarrollo del Sistema

4.1. Planteamiento General de la Propuesta

La propuesta central de este proyecto es el diseño e implementación de una **infraestructura de microservicios autoadaptativa para sistemas inteligentes, operando bajo el paradigma de Continuidad Computacional**. Esta infraestructura es una solución genérica, diseñada para autogestionarse dinámicamente mediante el ciclo MAPE-K, adaptando el comportamiento de cada nodo (borde, niebla y nube) en función de métricas como el consumo energético, la carga de trabajo o la latencia.

Para validar y demostrar la viabilidad de esta infraestructura, se ha desarrollado un caso de uso concreto: un sistema de monitoreo y predicción de la actividad física. A través de este escenario representativo, se demuestra cómo la infraestructura genérica es capaz de capturar, procesar y analizar datos en un entorno distribuido, garantizando la continuidad del servicio y la resiliencia del sistema.

El flujo del sistema, aplicado a este caso de uso, sería el siguiente:

1. **Captura de datos:** Los dispositivos wearables recogen información fisiológica del usuario (como ritmo cardíaco, movimiento, temperatura), la transmiten a una aplicación móvil, y esta a su vez envía los datos a los nodos de borde para su procesamiento inicial.

2. **Preprocesamiento en el borde:** Los nodos de borde ejecutan un primer procesamiento: filtrado de ruido, limpieza de datos inconsistentes y normalización. Esta etapa reduce la latencia y minimiza el volumen de datos que asciende a capas superiores, optimizando el ancho de banda y el rendimiento energético.

3. **Entrenamiento personalizado en la capa de niebla:** La capa de niebla se encarga de **entrenar los modelos adaptados para cada usuario**, utilizando los datos preprocesados en el borde. Cada modelo se ajusta de forma autónoma a las características individuales, permitiendo que las decisiones locales reflejen patrones personalizados sin necesidad de que la nube intervenga directamente en la adaptación continua.

4. **Coordinación y aprendizaje en la nube:** La nube actúa como **repositorio central de datos históricos y agregados**, manteniendo un modelo de predicción genérico basado en la experiencia colectiva de todos los usuarios. Además, almacena información estructurada que permite mejorar la calidad de los modelos locales y retroalimentar decisiones futuras.

5. **Ciclo MAPE-K aplicado a cada nodo**: El sistema de gestión utiliza el ciclo MAPE-K, aplicado infraestructuralmente a los nodos:

- **Monitor**: Cada nodo registra métricas operativas como carga, consumo energético, latencia, disponibilidad, entre otras.
- **Analyze**: Se detectan desviaciones de rendimiento y condiciones anómalas (sobrecarga, inactividad, fallos latentes).
- **Plan**: Se generan planes de acción como migración de tareas, activación de nodos pasivos o reconfiguración de parámetros técnicos.
- **Execute**: Se aplican las modificaciones pertinentes de manera autónoma y en tiempo real.
- **Knowledge**: Las decisiones se almacenan para enriquecer el conocimiento compartido y mejorar la toma de decisiones futuras.

6. **Actualización opcional del modelo por parte del usuario**: Cuando un usuario se registra en el sistema, se le asigna inicialmente un **modelo personalizado derivado del modelo genérico global** disponible en ese momento. Con el tiempo, dicho modelo genérico evoluciona, incorporando nuevos patrones, mejoras técnicas y mayor precisión gracias al aprendizaje colectivo.

El sistema ofrece al usuario la **posibilidad de sustituir su modelo personalizado local por la versión más reciente del modelo genérico.** Esta opción implica sacrificar temporalmente la adaptación individual a cambio de beneficiarse de los avances técnicos globales. Si el usuario elige este camino, su nuevo modelo comenzará a **personalizar automáticamente en las siguientes iteraciones**, basándose en sus datos recientes. Este mecanismo permite al usuario decidir entre **continuidad personalizada o actualización tecnológica** con adaptación progresiva.

4.1.1. Escenarios Representativos de Uso del Sistema

Los siguientes escenarios ilustran la capacidad de la infraestructura para autogestionarse en diferentes condiciones, independientemente de la carga de trabajo específica que esté ejecutando.

Escenario Genérico

Durante el funcionamiento del sistema, la infraestructura autoadaptativa detecta que un **nodo de borde experimenta una carga crítica**, elevando su temperatura más allá del umbral de seguridad configurado.

El sistema detecta esta condición y ejecuta una serie de acciones: analiza el entorno, **migra el servicio activo a un nodo vecino con menor carga**, y **desactiva temporalmente el nodo afectado** para evitar daños o degradación del rendimiento.

Todo este proceso ocurre de manera **automática y transparente para el usuario final**, garantizando la continuidad del servicio, la eficiencia energética y la resiliencia de la infraestructura distribuida.

Escenario Clínico

Un hospital utiliza la misma infraestructura para monitorear pacientes en tiempo real. Durante una emergencia, **varios nodos** de borde **aumentan su carga** al recibir múltiples señales simultáneas. El sistema MAPE-K **detecta la saturación** y **reconfigura automáticamente la arquitectura de microservicios**: algunos servicios se migran a la nube, y nodos secundarios se activan para distribuir el procesamiento. Esta auto-adaptación permite **mantener la estabilidad** y asegurar una respuesta clínica oportuna sin intervención humana directa.

4.1.2. Manual vs Auto-Adaptativo

Dentro del diseño de arquitecturas distribuidas, se contemplan distintas estrategias de gestión infraestructural que permiten adaptar los recursos y optimizar el sistema en tiempo de ejecución. Entre ellas, destacan el enfoque **manual** y el enfoque **auto-adaptativo**, cada uno con sus propias fortalezas y limitaciones en función del grado de autonomía deseado, la complejidad del entorno y los requisitos de escalabilidad. Aunque en este trabajo no se implementan ambos enfoques, se presentan aquí como referencia conceptual para contextualizar las decisiones adoptadas en el desarrollo del sistema.

Enfoque Manual

La gestión manual implica que los parámetros de operación y configuración de los nodos (frecuencia de actualización, asignación de tareas, activación de recursos) son definidos de forma estática o ajustados explícitamente por el administrador o usuario. Las características principales que proporciona este enfoque son:

- **Políticas fijas o predefinidas**: La infraestructura opera siguiendo reglas establecidas previamente, lo que limita su capacidad de respuesta ante cambios inesperados en el entorno. Esta rigidez puede afectar la adaptabilidad y el rendimiento ante fluctuaciones en la carga o en las condiciones de red.
- **Decisiones manuales**: La migración de tareas, la reconfiguración de nodos y la asignación de cargas se realizan de manera manual, lo cual requiere intervención humana constante y puede ralentizar el tiempo de respuesta ante situaciones críticas o emergencias.
- **Ausencia de retroalimentación autónoma**: No se incorporan mecanismos de aprendizaje automático ni bucles de realimentación que permitan al sistema ajustar sus comportamientos en función de experiencias pasadas o de datos en tiempo real. Esto impide que el sistema evolucione o se optimice de forma autónoma.
- **Necesidad de supervisión constante**: Para mantener niveles óptimos de rendimiento y eficiencia energética, el sistema requiere una vigilancia continua por parte de operadores humanos, lo que implica un mayor coste operativo y una menor escalabilidad en entornos distribuidos complejos.

La principal ventaja del enfoque manual radica en el **control absoluto** que ofrece al administrador o usuario sobre las decisiones operativas del sistema. Esta supervisión directa permite definir con precisión las condiciones de funcionamiento y mantener una gestión centralizada. No obstante, esta modalidad presenta varias **limitaciones significativas**:

- **Baja capacidad de adaptación**: Al depender de configuraciones estáticas y ajustes manuales, el sistema muestra escasa flexibilidad ante variaciones súbitas en la carga, disponibilidad de recursos o condiciones del entorno.
- **Elevado coste operativo**: La intervención humana constante exige dedicación de tiempo y recursos, lo que aumenta los costes de mantenimiento, especialmente en entornos que requieren supervisión continua o respuesta inmediata.

- **Limitada escalabilidad**: En arquitecturas distribuidas y sistemas heterogéneos, este enfoque resulta poco eficiente, ya que la coordinación manual entre nodos independientes dificulta la expansión del sistema y la integración de nuevas funcionalidades de manera ágil.

Enfoque Adaptativo

El enfoque adaptativo se fundamenta en la capacidad de la infraestructura para reconfigurarse dinámicamente a través de la integración del ciclo **MAPE-K** (Monitorizar, Analizar, Planificar, Ejecutar y Conocimiento) en cada nodo del sistema. Esta arquitectura habilita una gestión inteligente y distribuida que permite ajustar el comportamiento del sistema en tiempo real ante condiciones variables.

Las principales características y ventajas que ofrece este modelo incluyen:

- **Recopilación de métricas locales**: Cada nodo realiza un monitoreo continuo de parámetros como uso de CPU, temperatura, latencia y disponibilidad. Esta información constituye la base para la detección temprana de posibles ineficiencias.
- **Detección de condiciones subóptimas o anomalías**: A partir de los datos recopilados, el sistema identifica en tiempo real estados no deseados, por ejemplo, saturación de recursos o deterioro del rendimiento, lo que posibilita una respuesta proactiva.
- **Planificación y ejecución de ajustes**: El sistema adapta su configuración mediante acciones como redistribución de tareas, activación de nodos en reposo o modificación de parámetros operativos. Estas decisiones se toman sin intervención humana directa.
- **Base de conocimiento compartido**: Las acciones ejecutadas y sus resultados se integran en una base de conocimiento común, fortaleciendo la inteligencia colectiva del sistema y permitiendo mejoras evolutivas en la toma de decisiones futuras.
- **Personalización autónoma de los nodos**: Los nodos ajustan su comportamiento empleando modelos de aprendizaje automático entrenados localmente y sincronizados periódicamente desde la nube, lo que garantiza respuestas específicas y contextualizadas.

Si bien este enfoque implica una menor intervención directa del usuario, se obtienen beneficios clave:

- **Alta capacidad de adaptación** frente a entornos impredecibles, como los propios de la Continuidad Computacional, donde los eventos pueden variar de forma abrupta.

- **Optimización de recursos operativos**, incluyendo energía, rendimiento y latencia, gracias al ajuste continuo y autónomo del sistema.
- **Escalabilidad y flexibilidad**, especialmente en redes IoT heterogéneas, donde la diversidad de dispositivos requiere respuestas diferenciadas y coordinadas.

En resumen, si bien el enfoque manual puede ser adecuado en entornos controlados o con baja variabilidad, en el marco de la Continuidad Computacional, caracterizado por su dinamismo, heterogeneidad y necesidad de respuesta en tiempo real, resulta más apropiado optar por estrategias auto-adaptativas. Por ello, **este trabajo se orienta hacia la adopción de un enfoque auto-adaptativo**, que ofrece mayor capacidad de resiliencia, escalabilidad y autonomía operativa ante eventos imprevisibles.

4.2. Diseño de la Arquitectura Distribuida

4.2.1. Capas y Roles

- **Dispositivos IoT** (Pulseras Inteligentes): Su función es recoger datos psicofisiológicos en tiempo real (actividad física, sueño, etc.) enviados mediante protocolos ligeros. Cuentan con tecnologías como sensores integrados y Wifi para transmitir datos.
- **Aplicación móvil**: Su función es mostrar los datos al usuario. Se encarga de visualizar datos y es el medio de entrada de los datos hacia el sistema.
- **Nodos de borde**: Filtran, limpian y normalizan los datos recibidos. Por otro lado, también ejecutan un modelo de Machine Learning inicial para clasificar la actividad del usuario. Para la implementación se hará uso de una Raspberry Pi.
- **Nodos en la niebla**: Representa un nivel intermedio de procesamiento entre los nodos de borde y la nube. Su función principal es realizar el entrenamiento de los modelos personalizados de los usuarios, pero también se podrían usar para desplegar réplicas de servicios para evitar una saturación del sistema en capas inferiores.
- **Nodos en la nube**: Su función será el almacenamiento a largo plazo y el procesamiento intensivo de la información preprocesada, de forma que entrena y actualiza el modelo global de Machine Learning.

4.2.2. Diagrama deL Sistema

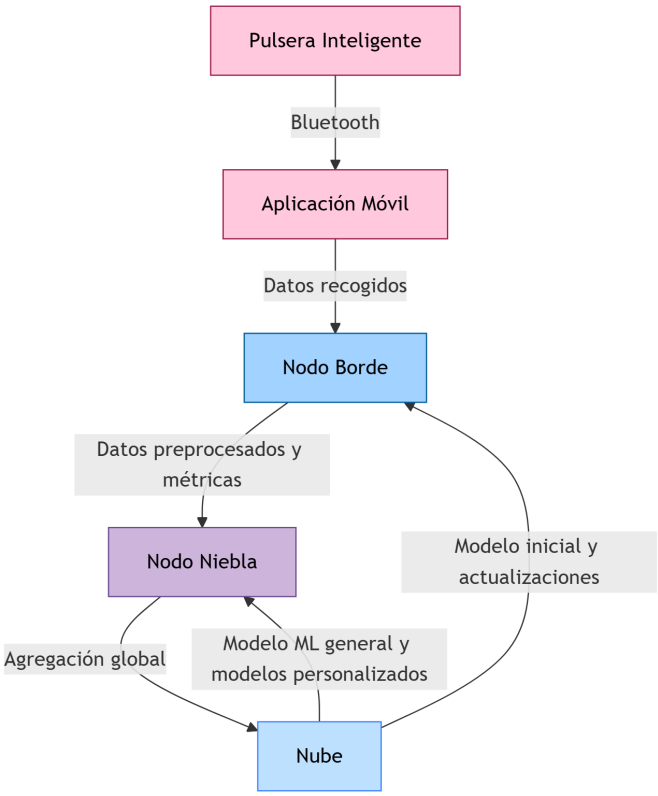


Figura 4.1: Diagrama conceptual de la comunicación del sistema. Elaboración propia.

Este diagrama 4.1 de arquitectura nos permite entender la comunicación que se realiza en el sistema. Podemos observar que la nube se conecta con el nodo de borde para mandar el modelo inicial o sus actualizaciones con modelos personalizados o modelos genéricos más recientes. Por otro lado, el nodo niebla recibe la comunicación del nodo de borde cuando se generan nuevos datos y se comunica con la nube para subir los datos y aquellos modelos personalizados con los que haya estado trabajando, al igual que recibe el modelo genérico de ese momento para ir personalizando a los datos de un usuario concreto.

4.2.3. Diagrama de Secuencia

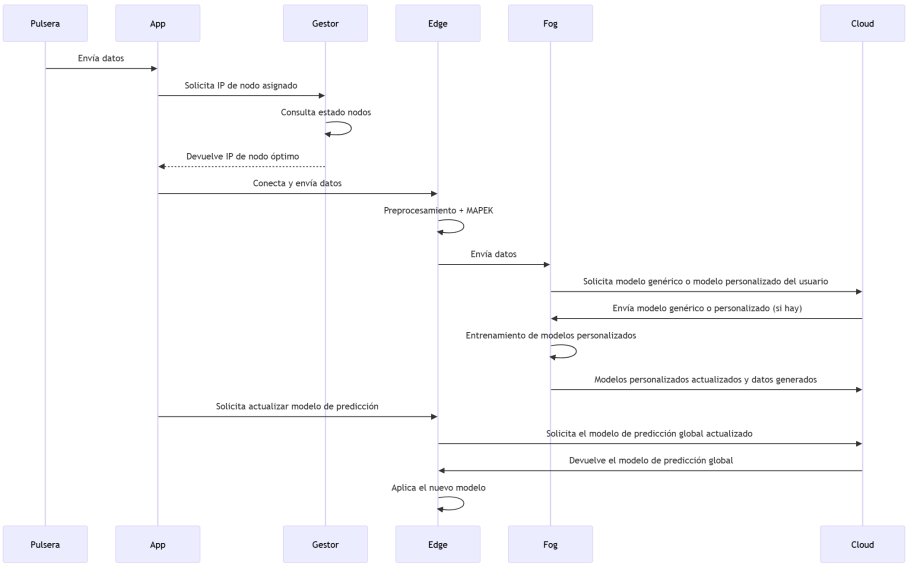


Figura 4.2: Diagrama de Secuencia inicial del sistema ejecutando el envío de datos fisiológicos y la actualización del modelo de predicción. Elaboración propia.

En este diagrama 4.2 se puede seguir los pasos que sigue el sistema. En los últimos pasos hay que destacar la interacción entre los nodos de borde y nube para actualizar el modelo del usuario, ya sea una versión mejorada del propio usuario o una versión actualizada del modelo genérico que posee el sistema, lo cual hace que se satisfaga el caso de uso planteado en este proyecto.

4.3. Desarrollo del Sistema

Para el desarrollo del sistema se ha seleccionado como marco de trabajo la metodología ágil **Scrum** debido a su idoneidad para gestionar la complejidad e incertidumbre inherentes al proyecto. Dado que el sistema implica la integración de múltiples componentes (dispositivos IoT, nodos de borde, niebla y nube) y la implementación de mecanismos de auto-adaptación que requieren un desarrollo iterativo y una estrecha retroalimentación, **Scrum ofrece un marco ágil que facilita la adaptación a los cambios en los**

requisitos y la evolución del diseño. La estructura de Sprints en Scrum permite dividir el desarrollo en incrementos funcionales, lo que facilita la entrega temprana de funcionalidades y la validación continua.

Adicionalmente, se emplea **Git como sistema de control de versiones y GitHub como plataforma de gestión del proyecto**. Git permitirá gestionar eficientemente las diferentes versiones del código fuente, facilitando la organización del trabajo y la integración de cambios. GitHub, por su parte, proporcionará un entorno centralizado para el almacenamiento del código, la gestión de tareas, el seguimiento de incidencias y la documentación del proyecto. Esta combinación de herramientas es esencial para mantener la trazabilidad de los cambios, fomentar la organización y asegurar la calidad del software desarrollado en un proyecto de esta envergadura.

4.3.1. Sprint 0: Infraestructura, Backlog y Prototipado Inicial

- **Objetivos clave:**
  - Configurar herramienta de gestión (Jira, repositorios Git)
    - Crear tablero de trabajo de Jira
    - Configurar repositorio GitHub
  - Definir y priorizar el Product Backlog Inicial, con historias de usuario y lista de requisitos
    - Identificar y redactar historias de usuario iniciales
    - Priorizar las historias según valor y viabilidad
    - Elaborar una lista de requisitos funcionales y no funcionales
  - Montar el entorno base
    - Crear estructura de carpetas en el repositorio
    - Configurar entorno Docker básico para el desarrollo
    - Redactar un script de despliegue inicial
- **Fecha de Inicio:** 24/3/25
- **Fecha de Fin:** 7/4/25

Herramienta de trabajo para SCRUM: Jira

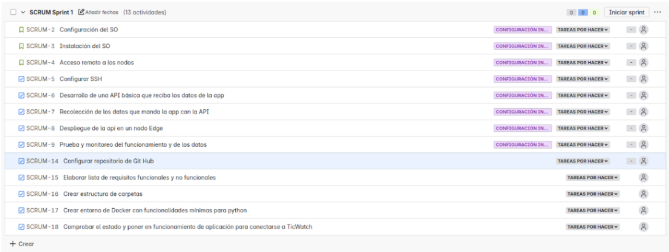


Figura 4.3: Captura del entorno de trabajo de Jira creado para el proyecto

Jira es una excelente herramienta para gestionar proyectos con metodología Scrum porque **permite organizar y visualizar el trabajo de forma clara y estructurada**. Facilita la creación de sprints, la asignación de tareas al equipo, el seguimiento del progreso mediante tableros Kanban o Scrum, y la generación de informes que ayudan a evaluar el rendimiento y mejorar continuamente. Además, su capacidad de integrarse con otras herramientas y su flexibilidad para adaptarse a distintos flujos de trabajo la convierten en una opción muy potente para equipos ágiles.

En la imagen 4.3 se puede apreciar la creación de las primeras tareas para el sprint. Mediante Jira se podrá organizar las tareas, dar prioridad, establecer el estado de cada una y muchas otras opciones que facilitan el seguimiento del desarrollo del proyecto.

Listado de Requisitos Funcionales

Identificador	Título	Descripción
RF1	Registro y Autenticación	El sistema debe permitir a los usuarios registrarse e iniciar sesión.
RF2	Asignación de Nodo	La app debe consultar al Gestor y recibir la IP del nodo de borde o niebla óptima antes de enviar datos.
RF3	Captura de Datos	La pulsera inteligente debe transmitir datos a la app móvil por Bluetooth.
RF4	Envío de Datos al Nodo Asignado	La app debe enviar los datos fisiológicos al nodo indicado por el Gestor.
RF5	Preprocesamiento en el borde	Cada nodo de borde debe filtrar, limpiar y normalizar los datos recibidos para reducir volumen y ruido.
RF6	Inferencia Local	Los nodos de borde deben ejecutar un modelo de ML para clasificar la actividad del usuario en "durmiendo", "sedentario" o "activo".
RF7	Ciclo MAPE-K en el borde	Los nodos de borde deben implementar las fases Monitor, Analyze, Plan, Execute y Knowledge para auto-adaptarse.
RF8	Envío a la niebla y nube	Los nodos de borde/niebla deben transmitir datos preprocesados, métricas y modelos ajustados al siguiente nivel (niebla o nube).
RF9	Entrenamiento y Actualización de Modelo Global	La nube debe entrenar un modelo global con los datos agregados de todos los usuarios y generar políticas de ajuste.
RF10	Distribución de Modelos	La nube debe poder distribuir nuevas versiones del modelo de aprendizaje.
RF11	Actualización Manual de Usuario	La aplicación debe permitir al usuario descargar e instalar la última versión del modelo global en su nodo de borde.
RF12	Monitorización Centralizada	El gestor debe ofrecer un dashboard con el estado de los nodos, número de clientes y métricas clave del sistema.

Cuadro 4.3.1: Listado de requisitos funcionales del sistema

Listado de Requisitos No Funcionales

Identificador	Título	Descripción
RNF1	Seguridad	El acceso a las APIs debe protegerse y la comunicación debe ser cifrada.
RNF2	Rendimiento y Latencia	El preprocesamiento local en nodos de borde debe completarse en menos de 200ms por lote de datos, y la inferencia local en menos de 100ms.
RNF3	Escalabilidad	El sistema debe soportar el aumento progresivo de usuarios (mínimo 5 usuarios concurrentes) sin degradación significativa de rendimiento.
RNF4	Disponibilidad	Los servicios críticos (gestor, nodos de borde/niebla y APIs de la nube) deben tener un SLA de disponibilidad mayor o igual al 99 %.
RNF5	Tolerancia a Fallos	En caso de caída de un nodo de borde, el gestor debe redirigir clientes automáticamente a otro nodo disponible en menos de 30s.
RNF6	Mantenibilidad y Despliegue	Todas las componentes deben ejecutarse en contenedores Docker.
RNF7	Usabilidad	La interfaz móvil y web debe ser intuitiva, con tiempos de respuesta inferiores a 500ms en interacciones clave (login, vista de datos, actualización de modelo).
RNF8	Consistencia de Datos	El sistema debe garantizar la consistencia eventual entre nodos de borde, niebla y nube, y registrar timestamps sincronizados para todas las transacciones.

Cuadro 4.3.1: Lista de requisitos no funcionales del sistema

Listado de Historias de Usuario

Orden	Historia de Usuario	Prioridad	Estimación
1	Como usuario quiero registrarme e iniciar sesión en la aplicación móvil.	Must	3
2	Como aplicación quiero conectarme al gestor/orquestador para obtener la IP del nodo de borde o niebla más libre.	Must	5
3	Como usuario quiero que la app envíe mis datos fisiológicos (pulsera) al nodo asignado para iniciar el flujo.	Must	3
4	Como nodo de borde quiero recibir datos de la app, filtrarlos y normalizarlos para reducir el volumen enviado.	Must	5
5	Como nodo de borde quiero ejecutar la inferencia ML local para determinar el estado del usuario (dormir/-sedentario/activo).	Must	8
6	Como nodo de borde quiero monitorizar en tiempo real métricas de rendimiento y precisión (fase Monitor).	Must	5
7	Como nodo de borde quiero analizar anomalías y desviaciones en la calidad de la inferencia (fase Analyze).	Must	8
8	Como nodo de borde quiero planificar ajustes de configuración (p. ej. tasa de muestreo) basados en el análisis (fase Plan).	Must	5
9	Como nodo de borde quiero aplicar esos ajustes automáticamente en tiempo real (fase Execute).	Must	5
10	Como nodo de borde quiero enviar datos preprocesados y métricas al sistema (fase Knowledge).	Must	3
11	Como nodo niebla quiero insertar datos de múltiples nodos, balancear carga y re-enviar los datos a la nube.	Should	8
12	Como nube quiero entrenar y actualizar el modelo global de ML con los datos agregados de todos los usuarios.	Should	13
13	Como gestor quiero devolver la IP del nodo más libre.	Must	5
14	Como gestor quiero mostrar el estado de los nodos y clientes activos.	Could	8
15	Como usuario quiero poder actualizar mi modelo local con la última versión del modelo global desde la app.	Should	5

Cuadro 4.3.1: Listado de historias de usuario

Las historias de usuario 4.3.1 se han priorizado con **3 valores de prioridad**, siendo “Must” la mayor prioridad, ”Shouldün punto intermedio y

"Could" la menor teniendo en cuenta las necesidades para el caso de uso. La estimación de su valor en puntos de historia se ha hecho utilizando Planning Poker según los siguientes criterios:

- 3 puntos: baja complejidad, bien definida
- 5 puntos: lógica moderada, requiere de varios componentes
- 8 puntos: involucra lógica adaptativa, análisis o entrenamiento ML
- 13 puntos: altamente incierta o integración compleja

Con el objetivo de facilitar y organizar mejor el desarrollo del proyecto, se ha decidido descomponer **historias de usuario en un conjunto de tareas principales**. Esta descomposición 4.4 permite trabajar con una **mayor granularidad**, ayudando a distribuir el esfuerzo de forma más precisa entre los sprints y a identificar los entregables intermedios de cada funcionalidad clave.

ID HU	Historia de Usuario	Prioridad	ID Tarea	Descripción de Tarea
1	Como usuario quiero registrarme e iniciar sesión en la aplicación móvil.	Must		
			T1	Puesta inicial de la aplicación HDRescuer o similar
			T2	Configurar la conexión de la pulsera a la App
			T3	Configurar creación de usuarios e inicio de sesión en la app
2	Como aplicación quiero conectarme al Gestor/Orquestador para obtener la IP del nodo Edge o Fog más libre.	Must		
			T4	Creación de la consulta a la IP del gestor para pedir la IP de un nodo
3	Como usuario quiero que la app envíe mis datos biométricos al nodo asignado	Must		
			T5	Crear petición en App para enviar datos a un nodo fijo
			T6	Redirección de los datos al nodo asignado por el gesto
4	Como Nodo Edge quiero recibir datos de la app, filtrarlos y normalizarlos para reducir el volumen	Must		
			T7	Implementar API simple en el Nodo Edge que reciba datos desde la App
			T8	Implementar preprocesamiento: limpieza, normalización en el nodo Edge
5	Como Nodo Edge quiero ejecutar inferencia ML local para determinar el estado del usuario	Must		
6	Como Nodo Edge quiero monitorizar en tiempo real métricas de rendimiento y precisión	Must		
			T9	Implementar monitorización del estado de cada nodo
7	Como Nodo Edge quiero analizar desviaciones en la calidad de la inferencia	Must		
8	Como Nodo Edge quiero planificar ajustes basados en el análisis	Must		
9	Como Nodo Edge quiero aplicar ajustes automáticamente en tiempo real	Must		
10	Como Nodo Edge quiero enviar datos preprocesados y métricas al sistema	Must		
11	Como Nodo Fog quiero insertar datos de múltiples Edge y reenviarlos a la nube	Should		
			T10	Sistema de comunicación entre el Cloud y Fog para consultar modelos en versiones anteriores
			T11	Crear servicio de ingesta de datos
			T12	Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.
12	Como Nube quiero entrenar y actualizar el modelo global de ML con los datos agregados de todos los usuarios.	Should		
			T13	Implementar entrenamiento genérico
			T14	Crear api en el cloud para gestionar la demanda del resto de servicios que tratan de acceder a la base de datos
13	Como Gestor quiero devolver la IP del nodo más libre	Must		
			T15	Crear API del Gestor con endpoint /nodo-disponible
			T16	Implementar lógica de asignación
14	Como Gestor quiero mostrar el estado de los nodos y clientes activos	Could		
15	Como usuario quiero poder actualizar mi modelo local con la última versión del modelo global desde la app.	Should		

Figura 4.4: HU descompuestas en tareas

Junto a las tareas mencionadas en la imagen anterior, a lo largo de cada sprint se realizarán tareas recurrentes, principalmente de:

- **Documentación**: Aquellas tareas relacionadas con acudir a bibliografía, redactar apartados de esta memoria, reuniones genéricas, etc.
- **Correcciones**: Principalmente hace referencia a cambios de refactorización, adaptación y mejoras a nivel de código y a rectificaciones sobre

planteamientos.

- **Test y pruebas**: Tareas que se realicen para comprobar características de alguna parte implementada y, de esta forma, asegurarse de un correcto funcionamiento.

En el marco metodológico de Scrum, los spikes son tareas de investigación o exploración que se incorporan al sprint con el objetivo de reducir incertidumbre técnica o funcional. A diferencia de las historias de usuario, los spikes no generan directamente valor funcional para el usuario final, pero son esenciales para tomar decisiones informadas, validar enfoques y preparar el terreno para futuras implementaciones.

A continuación, se detallan los spikes realizados en el proyecto:

SPK1 – Investigación y puesta en marcha de aplicación que se conecta a las pulseras inteligentes

Este spike tiene como objetivo explorar las posibilidades de conexión entre la aplicación desarrollada y dispositivos portables como pulseras inteligentes (por ejemplo, TicWatch Pro). Se investiga el tipo de datos que pueden recolectarse (frecuencia cardíaca, sueño, actividad física), los protocolos de comunicación disponibles (Bluetooth, APIs propietarias) y como desplegar aplicaciones de conexión que nos permitan recoger los datos.

SPK2 – Evaluación de tecnologías para monitorización

Este spike se centra en comparar distintas tecnologías disponibles para la monitorización de nodos y servicios en tiempo real. Se analizan herramientas como Prometheus, Grafana, Node Exporter y alternativas más ligeras o específicas para entornos distribuidos. El objetivo es seleccionar una solución que permita recolectar métricas relevantes (CPU, RAM, red, etc.) de forma eficiente y escalable, especialmente en dispositivos con recursos limitados.

SPK3 – Exploración sobre arquitecturas MAPE-K aplicadas a sistemas IoT distribuidos

Este spike tiene como finalidad comprender cómo se puede aplicar el modelo MAPE-K en sistemas distribuidos basados en IoT. Se revisa bibliografía técnica, casos de uso y ejemplos de implementación para definir cómo cada fase del ciclo puede ser representada en la arquitectura del proyecto.

SPK4 – Investigación sobre tecnologías Kubernetes, K3s y KubeEdge

Este spike comienza con una fase de búsqueda y formación en la tecnología de Kubernetes, con el objetivo de comprender sus fundamentos, capacidades de orquestación y posibles limitaciones en entornos distribuidos.

A partir de esta base, se aborda el estudio comparativo entre Kubernetes (K3s) y KubeEdge, una extensión diseñada específicamente para entornos Edge Computing.

4.3.2. Sprint 1: Puesta en Marcha de la Conexión con las Pulseras

- **Objetivo**: Establecer una conexión entre las pulseras y relojes inteligentes y una aplicación que obtenga los datos fisiológicos.
- **Fecha de Inicio**: 26/4/25
- **Fecha de Fin**: 25/5/25

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
T1	Puesta inicial de la aplicación HDRescuer o similar	20h	1
T2	Configurar creación de usuarios e inicio de sesión en la app	2h	1
T3	Configurar la conexión de la pulsera a la App	3h	1

Cuadro 4.3: Sprint backlog del Sprint 1

Implementación

El desarrollo llevado a cabo durante este sprint se centró en poner en marcha la aplicación HDRescuer (o una alternativa similar), integrarla con **dispositivos inteligentes** (pulseras E4 y TicWatch Pro), entender y adaptar su backend y habilitar funcionalidades básicas como la gestión de usuarios. El proceso se enfrentó a múltiples retos técnicos, pero también generó aprendizajes clave para futuras iteraciones.

Para la implementación, se ha optado por el lenguaje de programación Python dado que, tal y como se describió en puntos anteriores, ofrece un desarrollo ágil con framework como FastAPI al igual que es el principal lenguaje usado para los modelos de aprendizaje. De igual modo, para agilizar el despliegue, empaquetar las dependencias y facilitar un futuro despliegue se ha usado como herramienta de contenerización Docker debido a su curva de aprendizaje más sencilla y a su amplia comunidad que ofrece soporte y ayuda en los primeros pasos.

Análisis y Preparación Inicial

El sprint comenzó con la revisión de la documentación técnica de HDRescuer[39], con el objetivo de entender su estructura y funcionalidad. La app estaba orientada a conectarse con pulseras inteligentes, pero al tratarse de un proyecto sin mantenimiento en los últimos años, el primer contacto reveló una

serie de problemas de compilación y dependencias obsoletas. Este avance corresponde al desarrollo del Spike 1.

Tras varios intentos fallidos de ejecución, se tomó la decisión de explorar un camino alternativo: crear una aplicación propia que sirviera como base para establecer conexión con los dispositivos mediante Bluetooth. Esta elección respondió no solo a la complejidad técnica que suponía adaptar el código existente, sino también a la necesidad de avanzar rápidamente con pruebas funcionales que permitieran validar la viabilidad del proyecto.

Desarrollo Alternativo y Experimentación con Bluetooth

Una vez comprobadas las dificultades para compilar y ejecutar HDRescuer, se optó por crear una aplicación propia que permitiera avanzar en las pruebas de conectividad con las pulseras inteligentes. El principal objetivo era conseguir una conexión Bluetooth funcional que validara el enfoque del proyecto y sirviera como base para futuras integraciones.

Figura 4.5: Pantalla de inicio de la aplicación propia sin detectar la E4

Figura 4.6: Pantalla de inicio de la aplicación propia detectando la E4

En estas dos imágenes 4.5 4.6 se muestra como la aplicación estaba capacitada para detectar los dispositivos Bluetooth cercanos. Tras hacer clic sobre nuestro dispositivo, trata de conectarse a sus servicios.

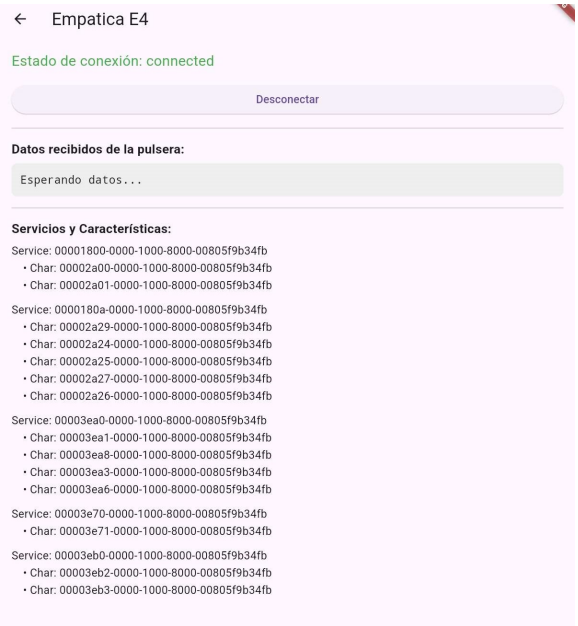


Figura 4.7: Pantalla de conexión de la aplicación propia con la E4

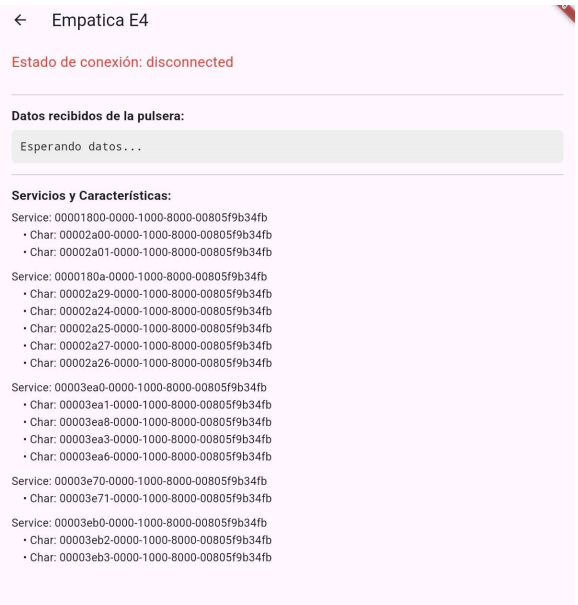


Figura 4.8: Pantalla de desconexión de la aplicación propia con la E4

La nueva app permitía visualizar los dispositivos Bluetooth cercanos, lo que ya suponía un primer logro técnico. Sin embargo, al intentar establecer una conexión 4.7, surgieron nuevas barreras: cada pulsera requería claves

específicas de autorización y paquetes particulares para intercambiar datos, lo que hacía que se desconectase en poco tiempo 4.8. Esta limitación hizo evidente que no bastaba con detectar los dispositivos; se necesitaba entender el protocolo de comunicación propio de cada uno.

Durante esta fase se realizó una investigación técnica más profunda sobre cómo funciona la conexión Bluetooth en dispositivos como la E4 y la TicWatch, cuáles son sus dependencias y qué acceso ofrece cada fabricante. Esto permitió dimensionar mejor los retos del proyecto y preparar las siguientes decisiones con mayor criterio.

Reactivación y Adaptación de HDRescuer

Tras los obstáculos encontrados con la app original y las limitaciones del desarrollo propio, se decidió volver a trabajar sobre HDRescuer. Aunque el proyecto presentaba deficiencias técnicas, se consideró que valía la pena intentar adaptarlo a un entorno más actual para aprovechar la base ya existente y las integraciones específicas que ofrecía, especialmente con las pulseras inteligentes.

Se inició un proceso de migración del código a versiones más recientes, ajustando y resolviendo conflictos que impedían su compilación. Tras varios cambios, finalmente se logró que la aplicación ejecutara de forma estable. Sin embargo, los problemas con las pulseras E4 persistían: a pesar de los intentos por conectar el dispositivo, la app no conseguía establecer la comunicación esperada.

En paralelo, se investigó el motivo del fallo de conexión y se descubrió que el proveedor del dispositivo había dejado de ofrecer soporte para su API, lo que impedía obtener claves de acceso nuevas. Esta falta de soporte externo bloqueaba completamente la integración con las E4 y dejaba a HDRescuer con funcionalidades incompletas para el proyecto.

Este punto marcó un momento clave en el sprint, en el que la solución técnica inicial tuvo que redirigirse hacia un nuevo enfoque.

Coordinación y Cambio de Dispositivo

Ante la imposibilidad de continuar trabajando con las pulseras E4 por la falta de soporte y las restricciones de acceso a su API, se trasladó el problema al equipo de trabajo en una reunión técnica. La exposición de los avances, los bloqueos encontrados y las alternativas exploradas permitió tomar una decisión conjunta: incorporar un nuevo dispositivo al proyecto, la pulsera TicWatch Pro.

La llegada de este nuevo hardware supuso un giro significativo. Aunque

la TicWatch también presentaba algunas complicaciones por tener versiones antiguas y requerir ajustes, al menos contaba con soporte activo y documentación más accesible. Tras realizar actualizaciones de firmware y revisar los procesos de sincronización, se consiguió establecer la conexión entre la pulsera y la aplicación.

Este momento marcó un avance claro en el sprint, ya que se logró recuperar la línea de trabajo prevista: la recolección de datos mediante dispositivos inteligentes. Además, el acompañamiento y las orientaciones de los tutores fueron clave para validar decisiones, priorizar esfuerzos y reforzar el enfoque técnico del proyecto.

Backend y Funcionalidades Complementarias

Una vez establecida la conexión con la pulsera TicWatch Pro, el siguiente paso dentro del sprint fue la puesta en marcha del backend de la aplicación. Esta parte del sistema es clave, ya que gestiona la persistencia de los datos recolectados y permite interactuar con funcionalidades como la creación de usuarios.

Durante esta etapa se configuró y depuró el entorno del backend, prestando especial atención a las rutas de conexión con la base de datos. Aunque algunos métodos aún presentaban fallos o respuestas inesperadas, ya se logró establecer comunicación con el sistema, lo que permitió validar que la base de datos estaba operativa y dispuesta a recibir datos.

Además, se puso en marcha la funcionalidad de creación de usuarios, lo cual constituyó un hito importante dentro del sprint. Esta acción confirmó que la app no solo estaba conectada con los dispositivos externos, sino que también podía registrar nuevas entidades dentro del sistema, lo que allanaba el camino para el procesamiento posterior de datos fisiológicos y la gestión de sesiones.

Este avance sentó las bases para una integración más robusta en los próximos sprints, en los que se podrá trabajar sobre capas más complejas del backend.

Resultados del Sprint

A continuación se presentan los resultados obtenidos durante este sprint, los cuales servirán como base para el análisis retrospectivo y contribuirán a la mejora continua del equipo. Los elementos completados, así como los hallazgos y métricas relevantes, permiten valorar el rendimiento y ajustar futuras planificaciones de manera más efectiva.

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Puesta inicial de la Aplicación HDRescuer o similar	26/4/25	20	3	3	Comienzo a leer la documentación inicial
Puesta inicial de la Aplicación HDRescuer o similar	27/4/25	20	7	4	Al ver la cantidad de errores a la hora de compilar, trato de crear una versión propia rapida que me permita la conexión con las pulseras inteligentes
Puesta inicial de la Aplicación HDRescuer o similar	29/4/25	20	8	1	Investigo sobre como funciona la conexión bluetooth con pulseras
Puesta inicial de la Aplicación HDRescuer o similar	30/4/25	20	9	1	Trato de implementar la conexión bluetooth
Puesta inicial de la Aplicación HDRescuer o similar	1/5/25	20	10	1	Corrijo errores y me doy cuenta de que necesito claves para poder conectarme al dispositivo
Puesta inicial de la Aplicación HDRescuer o similar	5/5/25	20	14	4	Trato de poner en funcionamiento la aplicación de Domingo.
Puesta inicial de la Aplicación HDRescuer o similar	6/5/25	20	17	3	Puesta en funcionamiento del backend y comprensión del mismo. No funcionan todos los metodos pero ya da señales la base de datos
Puesta inicial de la Aplicación HDRescuer o similar	8/5/25	20	18	1	Reunión con los tutores
Puesta inicial de la Aplicación HDRescuer o similar	12/5/25	20	22	4	Puesta en marcha la app de domingo y detecto el problema de la API KEY en la E4
Configurar la conexión de la pulsera a la App	14/5/25	3	4	4	Busqueda de soluciones para el problema de la E4
Configurar la conexión de la pulsera a la App	22/5/25	3	6	2	Reunión con los tutores
Configurar la conexión de la pulsera a la App	24/5/25	3	16	10	Puesta en marcha de la pulsera, app de Domingo y recolección de datos
Configurar creación de usuarios e inicio de sesión en la app	25/5/25	2	1	1	Creación de usuarios
Totales		25		39	

Figura 4.9: Organización de tareas reales del Sprint 1

Como se puede observar en la figura anterior 4.9, en total se han empleado 14 horas más de trabajo de las estimadas. Esto se ha debido principalmente a una puesta en funcionamiento inicial, algo muy común en este tipo de desarrollos. Además, los problemas que han surgido a lo largo del desarrollo, al no ser del todo previsibles, han causado que esta diferencia se incrementase.

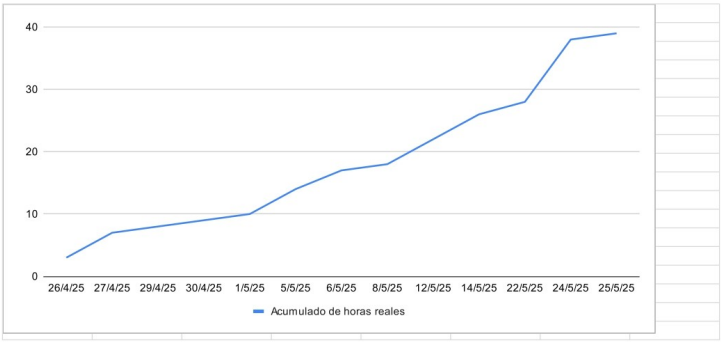


Figura 4.10: Acumulado de horas reales del Sprint 1

Como podemos ver en esta gráfica 4.10 de acumulado de horas reales, podemos ver picos que indican una mayor dedicación desarrollada durante

ese día.

Durante este sprint se lograron avances relevantes tanto en la puesta en marcha técnica como en la validación funcional de los componentes principales del proyecto. A pesar de los obstáculos encontrados con la aplicación HDRescuer y las limitaciones del dispositivo E4, se adoptaron soluciones prácticas que permitieron mantener el ritmo de trabajo y cumplir con los objetivos principales.

Los principales resultados obtenidos incluyen:

- Establecimiento de conexión con la pulsera TicWatchPro
- Adaptación y compilación de la aplicación HDRescuer
- Validación del backend, comprobando la conectividad con la base de datos
- Identificación de limitaciones técnicas de la pulsera E4
- Puesta en marcha las funcionalidades básicas como la creación de usuarios

4.3.3. Sprint 2: Infraestructura Básica de Recolección y Nodo de Borde Simplificado

- **Objetivo**: Establecer el flujo de datos básico desde la pulsera hasta un nodo de borde funcional para el preprocesamiento.
- **Fecha de Inicio**: 26/5/25
- **Fecha de Fin**: 9/6/25

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
–	Documentación	12h	–
T5	Crear petición en App para enviar datos a un nodo fijo	2h	3
T7	Implementar API simple en el nodo de borde que reciba datos desde la App	4h	4
T8	Implementar preprocesamiento: limpieza y normalización en el nodo de borde	5h	4

Cuadro 4.4: Sprint backlog del Sprint 2

Implementación

Durante el segundo sprint del proyecto, el foco principal estuvo en el despliegue de la infraestructura básica de procesamiento en los nodos de borde y la puesta en marcha de una API que permita la comunicación entre la aplicación móvil y dichos nodos. Además, se trabajó en la identificación de problemas técnicos mediante el diseño de una tabla de errores y en el desarrollo de funciones específicas para el tratamiento de los datos recibidos.

Implementación de API en el Nodo de Borde

Se desarrolló una API sencilla cuyo propósito es recibir los datos enviados desde la app instalada en los dispositivos móviles. Esta API representa el punto de entrada para los datos recolectados por los sensores y debe estar preparada para procesarlos en tiempo real desde los nodos de borde.

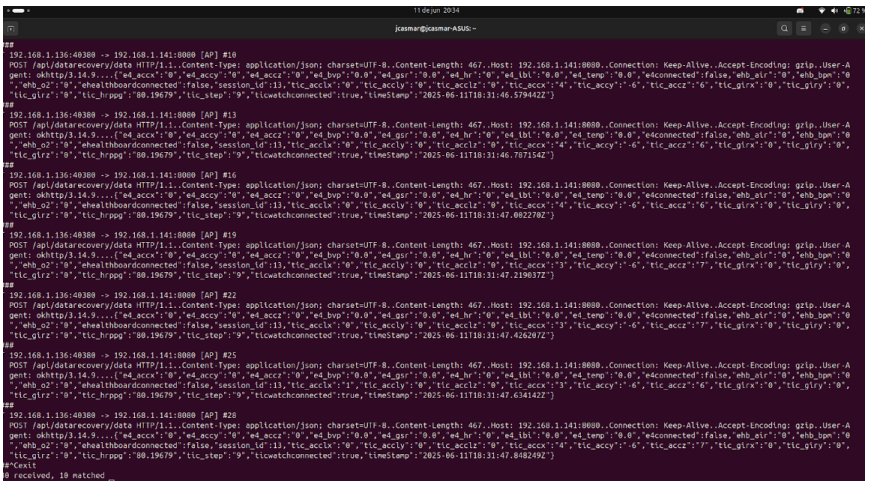


Figura 4.11: Captura de las peticiones que recibía la API Gateway en el puerto 8080

Mediante esta captura 4.11, se puede ver claramente la estructura de las peticiones que hace la aplicación a la API, tratando de subir los datos fisiológicos de un usuario. Gracias a la información que proporciona esta captura se pudo establecer el tipo de datos esperado, el método de recepción y el formato de almacenamiento provisional.

Los endpoints definidos hasta el momento para esta API con las funcionalidades mínimas del nodo de borde y procesamiento de datos son los siguientes:

GET /

- **Descripción**: Confirma que la API del nodo está activa y operativa.
- **Respuesta exitosa**: 200 OK
- **Campos de respuesta**:
  - message (string): Mensaje de confirmación

GET /health

- **Descripción**: Comprobación simple del estado de salud del sistema. Se utiliza para verificar que hay respuesta del nodo.
- **Respuesta exitosa**: 200 OK
- **Campos de respuesta**:

- status (string): Estado del servicio ("ok" si está funcionando correctamente)

GET /status

- **Descripción**: Consulta el estado actual del nodo de borde, incluyendo métricas de carga del sistema, memoria y usuarios activos.
- **Respuesta exitosa**: 200 OK
- **Campos de respuesta**:
  - edge_id (string): ID del nodo
  - status (string): Estado del nodo (p. ej., .ºnline").
  - current_load (objeto): Métricas de carga actuales:
  - `cpu_load` (float): Carga de CPU.
  - `memory_usage` (float): Uso de memoria.
  - `active_users_count` (int): Usuarios activos.
  - `last_ingestion_timestamp` (int): Última ingesta de datos (timestamp UNIX).
  - `mape_k_knowledge_snapshot` (objeto):
    - `sampling_rate_suggestions_count` (int): Nº sugerencias de muestreo.
    - `anomaly_history_count` (int): Nº total de anomalías.
  - message (string): Resultado de la operación.

POST /api/datarecovery/data

- **Descripción**: Procesa y almacena datos del dispositivo TicWatch. Verifica conexión, transforma los datos y guarda en la base de datos si todo es correcto.
- **Cuerpo de la petición** (application/json):
  - session_id (string): ID único de la sesión.
  - timeStamp (int): Marca temporal de los datos.
  - ticwatchconnected (bool): Indica si el dispositivo está conectado.
  - `tic_accx`, `tic_accy`, `tic_accz` (float): Acelerómetro sin corrección.
  - `tic_acclx`, `tic_accly`, `tic_acclz` (float): Acelerómetro corregido.
  - `tic_girx`, `tic_giry`, `tic_girz` (float): Giroscopio.

  - tic_hrppg (float): Ritmo cardíaco estimado.
  - tic_step (int): Contador de pasos.

- Respuesta exitosa: 200 OK

  "status": Estado del usuario (durmiendo, haciendo deporte, sedentario)

- Errores posibles:

  - 400 Bad Request: El dispositivo no está conectado o los datos son inválidos.
  - 500 internal Server Error: Fallo al almacenar datos en la base de datos.

Filtrado y Preprocesamiento de Datos

Como complemento a la recepción de datos, se comenzaron a desarrollar funciones de preprocesamiento que operan directamente en el nodo de borde. Estas funciones permiten realizar tareas como limpieza de datos, normalización de formatos, o segmentación de información relevante antes de ser enviada a sistemas de almacenamiento o utilizados para análisis posteriores. Este paso es crucial para garantizar la calidad y fiabilidad de las predicciones.

Actualmente se divide en las siguientes etapas:

1. Ritmo Cardiáco:

   a) Se convierte el valor a entero para asegurar la consistencia
   b) Se valida que esté dentro de un rango fisiológicamente razonable
   c) Si el valor es invalido, se asigna None

2. Número de Pasos:

   a) Se convierte a entero
   b) Se valida que el número de pasos sea mayor o igual a cero
   c) Si el valor es negativo o no convertible, se asigna None

El objetivo es eliminar valores atípicos o corruptos, asegurar la coherencia de tipos de datos y preparar las señales para su uso en modelos de aprendizaje y análisis estadístico.

Para tratar de probar el funcionamiento en un entorno distribuido, una vez implementada la API, se procedió a su despliegue en una Raspberry Pi 5, que funcionará como uno de los nodos de borde del sistema. Se configuró el repositorio en GitHub para gestionar el control de versiones, y se realizó la primera prueba de conexión entre la aplicación móvil y el nodo, verificando que los datos lleguen correctamente.

Resultados del Sprint

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Documentación	26/5/25	12	5	5	Redacción de actas, planteamiento de la propuesta con conexión a gestor y posible caída del servicio, puesta en funcionamiento de la raspberry pi 5
Documentación	27/5/25	12	8	3	Revisión de HU, requisitos, configuración inicial de Jira, primera versión de programa para nodos Edge
Documentación	28/5/25	12	11	3	Documentación de Grafana, Prometheus y Node Exporter para el monitoreo de los servicios
Documentación	30/5/25	12	13	2	Planteamiento sobre un sistema robusto tolerante a fallos
Documentación	4/6/25	12	15	2	Correciones de los cambios del tutor en la documentación y creación de tareas para Jira
Documentación	5/6/25	12	16	1	Reunión con los tutores
Documentación	6/6/25	12	17	1	Redacción de acta y planteamiento de tabla con errores
Crear petición en App para enviar datos a un nodo fijo	7/6/25	2	2	2	Implementar API simple en el Nodo Edge que reciba datos desde la App
Implementar API simple en el Nodo Edge que reciba datos desde la App	8/6/25	4	4	4	Puesta en funcionamiento api de los nodos edges. Despliegue de api en raspberry pi 5, configuración de github y primera prueba de conexión con la app
Implementar preprocesamiento: limpieza, normalización en el nodo Edge	9/6/25	5	3	3	Creación de las funciones de preprocesamiento para los nodos Edge
Totales		23		26	

Figura 4.12: Organización de tareas reales del Sprint 2

Observando la tabla de tareas reales de este sprint 4.12 podemos ver como las horas reales se han aproximado mucho más que en el sprint anterior a las horas estimadas, lo cual quiere decir que se ha mejorado en la planificación.

En la parte de documentación es la tarea en la que se produce el principal desfase de horas en relación a las reales y las estimadas, por lo que de cara a futuros sprines se pretende ajustar mejor esto.

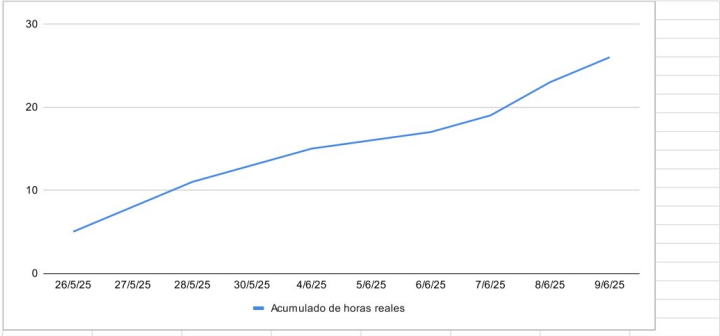


Figura 4.13: Acumulado de horas reales del Sprint 2

Si observamos la gráfica del acumulado de horas en este sprint 4.13, se puede observar como es más lineal que la del sprint anterior, lo cual es un indicativo de que el trabajo realizado durante este periodo ha sido más uniforme.

Durante este segundo sprint, se consolidaron avances importantes en la infraestructura del sistema y en la implementación técnica de los nodos de borde. Los resultados obtenidos permiten validar el funcionamiento básico de la arquitectura distribuida y preparan el terreno para el procesamiento de datos en tiempo real.

Entre los principales logros del sprint destacan:

- Implementación de una API básica con el nodo de borde
- Creación de función de preprocesamiento
- Despliegue exitoso en Raspberry Pi 5

4.3.4. Sprint 3: Implementación del Gestor Inicial

- **Objetivo**: Crear la funcionalidad básica del Gestor para la asignación de nodos.
- **Fecha de Inicio**: 16/6/25
- **Fecha de Fin**: 30/6/25

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
–	Documentación	12h	–
–	Correcciones	2h	–
T4	Creación de la consulta a la IP del gestor para pedir la IP de un nodo	2h	2
T15	Crear API del gestor con endpoint `/nodo-disponible`	4h	13
T16	Implementar lógica de asignación del gestor	4h	13
T6	Redirección de los datos al nodo asignado por el gestor	3h	3

Cuadro 4.5: Sprint backlog del Sprint 3

Implementación

El tercer sprint del proyecto se centró en reforzar la infraestructura de gestión de sesiones, optimizar el flujo de comunicación entre la aplicación móvil y los nodos de borde, y documentar las funcionalidades técnicas de las APIs disponibles. También se abordaron aspectos arquitectónicos clave, como la elección dinámica de nodos mediante el patrón estrategia, y se incluyeron referencias teóricas para reforzar el diseño del sistema.

Gestión de Sesiones y Revisión del Despliegue

Se documentó el funcionamiento del mecanismo de "nueva sesión", orientado a iniciar el flujo de recogida de datos desde la app. Esto implicó revisar los archivos de configuración, explorar las redes entre contenedores y validar el despliegue general de los nodos de borde y niebla dentro del entorno distribuido. Estos ajustes mejoraron la comprensión de la estructura del sistema y sentaron las bases para una conexión más ágil con la aplicación.

Desarrollo e Integración del Gestor de Nodos

Se creó el código correspondiente al componente manager, encargado de coordinar nodos disponibles. Esta pieza se integró dentro de la aplicación

móvil para que pudiera consultar su IP de destino al iniciar una sesión. Tras las primeras pruebas, se confirmó que la app podría enviar datos al gestor, aunque se detectó un error relacionado con el puerto de conexión en el nodo de borde.

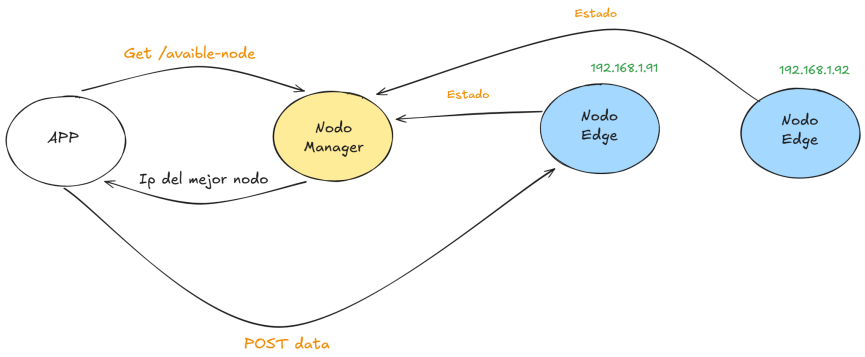


Figura 4.14: Esquema del flujo del sistema con el nodo manager. Elaboración propia.

Después de varias pruebas y correcciones, se logró establecer una comunicación fluida, tal y como se muestra en la imagen anterior 4.14: la app se conecta al gestor, este le responde con la IP del nodo de borde más adecuado, y los datos se envían correctamente al destino.

Como mejora del sistema, se añadió un comportamiento proactivo al nodo manager, de forma que no necesita que se le manden peticiones nuevas para comprobar el estado del resto de nodos, sino que constantemente se encuentra comprobando el estado, mejorando de esta forma los tiempos de detección de errores y teniendo actualizada la información sobre el estado de los nodos de cara a la asignación para un usuario.

Lógica de Asignación Inicial

El sistema implementa un componente centralizado de gestión (nodo manager) cuya responsabilidad es supervisar el estado operativo de un conjunto de nodos de borde y, en función de su disponibilidad y carga, asignar dinámicamente el nodo más adecuado cuando un cliente lo solicita.

Para ello, el manager ejecuta un proceso de monitorización en segundo plano que realiza peticiones periódicas, comportándose proactivamente, a cada nodo mediante el endpoint /status, recopilando información estructurada sobre su estado general (status), carga de CPU (cpu_load), uso de memoria (memory_usage) y número de usuarios activos (active_users_count). Esta información se almacena en un diccionario en memoria llamado nodes_status.

El proceso de asignación se basa en una arquitectura extensible utilizando el patrón Estrategia, lo que permite definir diferentes criterios de selección sin modificar el núcleo del sistema. Actualmente se han implementado dos estrategias principales:

- **Menor número de usuarios activos**: selecciona el nodo online con la menor cantidad de usuarios activos registrados. Esta estrategia favorece la distribución de carga por número de conexiones concurrentes.
- **Puntuación ponderada**: calcula un "score" por nodo combinando de forma configurable el número de usuarios activos, la carga de CPU y el uso de memoria. Esta estrategia busca un equilibrio más fino entre múltiples indicadores de carga, priorizando los recursos menos comprometidos.

Cada vez que un cliente realiza una petición al endpoint /api/available-node, el sistema consulta el estado de los nodos, aplica la estrategia de asignación configurada, y responde con la IP del nodo seleccionado. Si no hay ningún nodo operativo o disponible, se retorna un error 503.

Este diseño permite escalar fácilmente las políticas de asignación mediante nuevas estrategias, así como mantener un alto grado de separación de responsabilidades entre la monitorización, el almacenamiento de estado y la lógica de decisión.

```
🚀 Ejecutando tests con pytest en el contenedor Docker...
======================== test session starts ========================
platform linux -- Python 3.11.13, pytest-8.2.1, pluggy-1.6.0
rootdir: /app
configfile: pytest.ini
testpaths: tests
plugins: anyio-4.9.0
collected 3 items

tests/test_least_users.py .                                    [ 33%]
tests/test_selector_no_available.py .                          [ 66%]
tests/test_weighted_score.py .                                 [100%]

========================= 3 passed in 0.04s =========================
✅ Tests completados.
```

Figura 4.15: Ejecución de test sobre las estrategias de asignación de nodo planteadas

Tras realizar los test mostrados en la imagen anterior se comprueba que las estrategias tienen un correcto funcionamiento, proporcionando los resultados esperados.

Resultados del Sprint

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Documentación	16/6/25	12	4	4	Documentar como funciona la parte de nueva sesión, revisar archivos docker compose y ver despliegue y redes.
Crear API del Gestor con endpoint /nodo-disponible	17/6/25	4	2	2	Creo el codigo para el manager
Creación de la consulta a la IP del gestor para pedir la Ip de un nodo	18/6/25	2	2	2	Meto el codigo en la app para que consulte la ip. Pruebo la app y si funciona el envío de datos
Documentación	19/6/25	12	5	1	Reunión con los tutores
Corrección	20/6/25	2	2	2	Reviso y la implementación para el gestor
Creación de la consulta a la IP del gestor para pedir la Ip de un nodo y Redirección de los datos al nodo asignado por el gestor	21/6/25	3	3	3	Consigo conectar la app al gestor y este le envía la ip nueva del nodo edge
Corrección	22/6/25	2	3	1	Consigo que se envíe correctamente los datos (fallaba el puerto del nodo edge)
Documentación	23/6/25	12	7	2	Revisión de sprints y reorganización sobre las funciones de los nodos Edges y Fog.
Documentación	24/6/25	12	9,5	2,5	Documentación de los endpoints de las APIS que tengo funcionando, creación de la función de consulta de estado proactiva en nodo Gestor
Implementar lógica de asignación del Gestor	25/6/25	4	3,5	3,5	Aplico patrón estrategia para la elección de ips, lo documento y creo los tests. También rediseño los sprints.
Documentación	28/6/25	12	11	1,5	The vision of autonomic computing Bibliografia de los tutores
Documentación	29/6/25	12	14	3	IoTDeploy: Deployment of IoT Smart Applications over the Computing Continuum Bibliografía de los tutores. Incluye mucha info sobre como configurar todo con jenkins, CI/CD,...
Totales		27		27,5	

Figura 4.16: Organización de tareas reales del Sprint 3

Tal y como se puede ver en esta imagen de organización de tareas reales de este sprint 4.16 se puede decir que se ha mejorado la planificación, fallando únicamente por 0,5 horas.

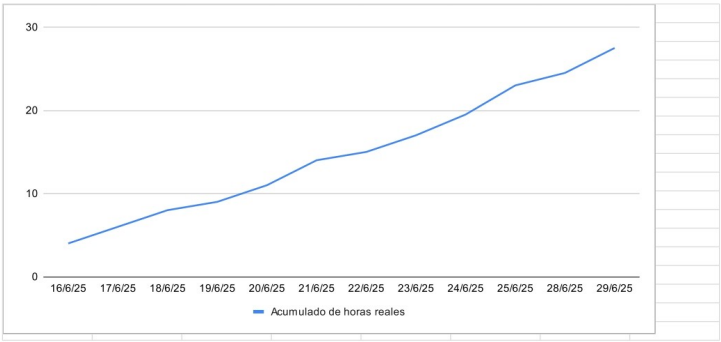


Figura 4.17: Acumulado de horas reales del Sprint 3

La evolución del desarrollo durante este sprint, tal y como se ve en la

gráfica 4.17, ha sido bastante homogénea, lo que indica un trabajo constante mayoritariamente uniforme.

Durante este tercer sprint se consolidó la arquitectura distribuida del sistema, centrando los esfuerzos en establecer conexiones dinámicas entre los distintos componentes, y aumentar la inteligencia operativa de los nodos. Gracias al trabajo realizado, se alcanzaron los siguientes resultados:

- Conexión estable entre la App, el Gestor y los nodos de borde
- Solución al fallo de comunicación con el nodo de borde
- Aplicación del patrón estrategia en el gestor, documentado y testeado

En cuanto a mejoras, de cara a futuros sprines, se podría mejorar la complejidad del entorno distribuido, especialmente en la gestión de redes y contenedores Docker. A pesar de los avances, aún se requiere una mayor automatización en el despliegue.

4.3.5. Sprint 4: Entrenamiento Personalizado y Persistencia del Modelo

- **Objetivo**: Permitir que el nodo de borde comience a entrenar modelos personalizados por usuario.
- **Fecha de inicio**: 30/6/205
- **Fecha de finalización**: 14/7/2025

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
–	Documentación	20h	–
–	Correcciones y test	5h	–
T13	Implementar entrenamiento genérico	8h	12
T14	Crear API en el cloud para gestionar la demanda del resto de servicios que tratan de acceder a la base de datos	4h	12
T10	Sistema de comunicación entre el Cloud y Fog para consultar modelos en versiones anteriores	4h	11
T11	Crear servicio de ingesta de datos	3h	11

Cuadro 4.6: Sprint backlog del Sprint 4

Implementación

Este sprint representó un salto significativo hacia la integración de capacidades de aprendizaje automático en la arquitectura del sistema, así como en la consolidación de mecanismos de comunicación distribuidos entre los nodos de borde, niebla y nube. Además, se avanzó en la reorganización del repositorio, la incorporación de herramientas de monitorización, y la mejora documental.

Para la implementación se opta por una base de datos relacional, ya que se establece una estructura fija definida para la misma, al igual que se necesita tratar de mantener la mayor cantidad de integridad para tratar de perder el menor número de datos, cosa que no es tan viable con bases de datos no relacionales.

Fine Tuning VS Re-Entrenamiento

- Re-entrenamiento (Nube):

Su propósito es crear un modelo **completamente nuevo desde cero o re-entrenar** un modelo existente con un conjunto de datos grande y posiblemente actualizado. En nuestro caso, este proceso se realiza en la nube, con un gran volumen de datos anónimos. En este caso, el modelo se "olvida" de su estado anterior (si lo tuviera) y aprende patrones globales del nuevo (o refrescado) conjunto de datos.

- Fine-tuning (Niebla):

  Su propósito es adaptar un modelo **pre-existente** (en nuestro caso, el modelo genérico o el modelo personalizado anterior del usuario) a un conjunto de datos más pequeño y específico (los datos etiquetados de un usuario individual). En nuestro caso, este proceso se realiza en la niebla tomando el modelo genérico (o el último modelo personalizado del usuario), y lo ajusta con los nuevos datos etiquetados de ese usuario. El objetivo es que el modelo se adapte mejor a las particularidades del usuario sin "olvidar" lo que aprendió a nivel genérico.

Actualización de los Componentes de la Arquitectura

1.Base de Datos (PostgreSQL)

La base de datos relacional PostgreSQL constituye el núcleo de almacenamiento persistente del sistema. Su función es centralizar y organizar toda la información relevante, desde los datos crudos recolectados por los sensores hasta los modelos entrenados y sus asociaciones con los usuarios. El servicio se ejecuta como un contenedor Docker bajo el nombre db, accesible desde otros servicios a través de la red interna de Docker.

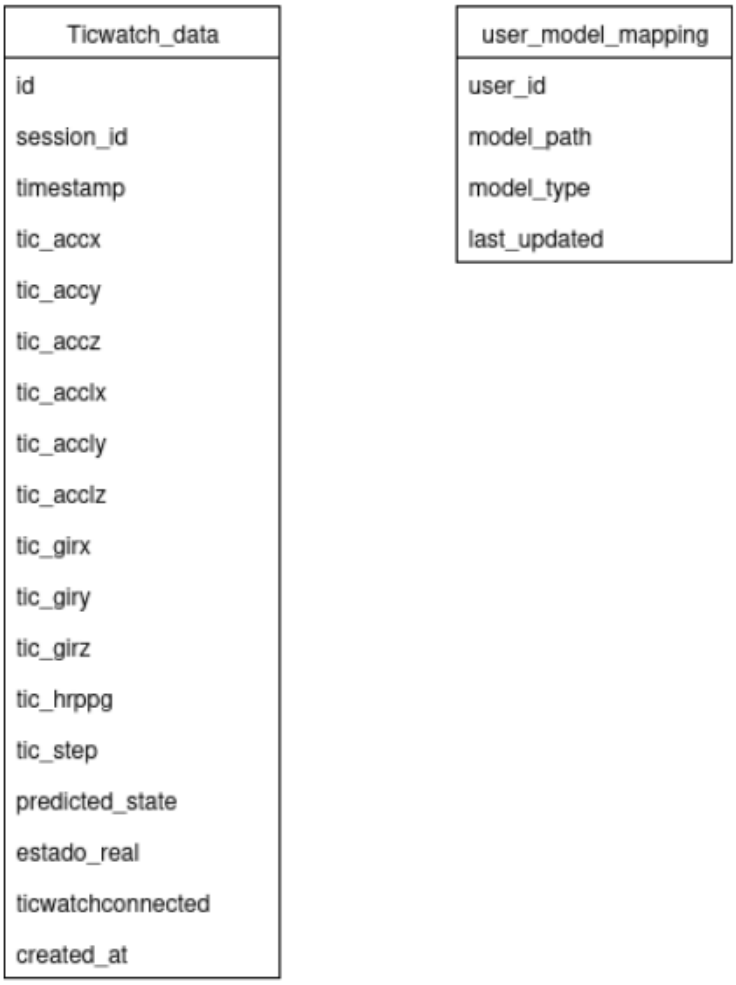


Figura 4.18: Esquema de las tablas de la base de datos

Ticwatch_data: Contiene los datos crudos y etiquetados recolectados por el dispositivo del usuario.

Campo	**Descripción del campo**
id	Identificador único del registro
session_id	Identificador de la sesión de recolección
user_id	Usuario al que pertenece el registro
timestamp	Marca temporal del dato
tic_accx, tic_accy, tic_accz	Aceleración bruta en los tres ejes
tic_acclx, tic_accly, tic_acclz	Aceleración lineal
tic_girx, tic_giry, tic_girz	Datos del giroscopio
tic_hrppg	Ritmo cardíaco
tic_step	Número de pasos
predicted_state	Estado predicho por el modelo
estado_real	Etiqueta real del estado
ticwatchconnected	Indicador de conectividad del dispositivo
created_at	Fecha de inserción del registro

Cuadro 4.7: Estructura de campos en el conjunto de datos `Ticwatchdata`

user_model_mapping: Mantiene la relación entre cada usuario y su modelo personalizado.

Campo	**Descripción del campo**
user_id	Identificador del usuario
model_path	Ruta del archivo del modelo almacenado
model_type	Tipo de modelo (genérico o personalizado)
last_updated	Fecha de la última actualización del modelo

Cuadro 4.8: Estructura de campos en la tabla `user_model_mapping`

2.Servicio de Cola de Mensajes (RabbitMQ)

RabbitMQ actúa como el sistema de mensajería asíncrona del proyecto, permitiendo una comunicación desacoplada y confiable entre los distintos nodos del sistema, especialmente entre el borde y la niebla. El servicio se ejecuta como un contenedor Docker bajo el nombre rabbitmq, accesible desde otros servicios mediante la red interna. La cola principal es edge_data_queue. Es utilizada para publicar los datos etiquetados generados por el nodo de borde. El Fog Trainer la monitorea constantemente para desencadenar procesos de fine-tuning.

3.Nodo en la Nube

El **nodo en la nube** representa el centro de control del sistema. Su función es gestionar los modelos genéricos, proporcionar una API para la interacción con los datos y modelos, y actuar como la capa de persistencia lógica y operativa.

3.1.Cloud API

Cloud API es una interfaz RESTful desarrollada con FastAPI para que otros servicios (borde, niebla) interactúen con los datos y modelos. El servicio se ejecuta como un contenedor Docker bajo el nombre cloud_api.

Endpoint	**Descripción**
`GET /data/user/{user_id}/labeled`	Obtiene datos etiquetados de un usuario
`GET /models/generic`	Descarga el modelo genérico
`GET /models/user/{user_id}`	Descarga el modelo personalizado de un usuario
`POST /models/user/{user_id}`	Sube un modelo personalizado de usuario
`GET /users/{user_id}/model_mapping`	Obtiene el mapeo del modelo de un usuario
`PUT /users/{user_id}/model_mapping`	Actualiza el mapeo del modelo de un usuario

Cuadro 4.9: Listado de endpoints clave de Cloud API

Tecnologías Utilizadas:

- FastAPI: Framework principal para la construcción de la API.
- Pandas: Para el procesamiento de datos tabulares.
- Psycopg2: Para la conexión con la base de datos PostgreSQL.

3.2.Cloud Trainer

El servicio **Cloud Trainer** desempeña un papel fundamental dentro del sistema de aprendizaje automático, ya que se encarga de entrenar el modelo genérico inicial y, en caso de ser necesario, reentrenarlo periódicamente utilizando todos los datos disponibles en la base de datos. Este proceso garantiza que el modelo se mantenga actualizado.

Se ejecuta como un servicio independiente, con el nombre dentro de Docker **cloud_trainer**.

El proceso consiste en:

- **Inicialización del modelo**: Al iniciarse, el servicio verifica si existe un modelo genérico previamente entrenado. En caso de que no se encuentre, se procede a generar uno nuevo utilizando un conjunto de datos dummy predefinido que ha sido insertado en la base de datos. Este modelo inicial actúa como una solución de respaldo para garantizar que el sistema pueda operar incluso en ausencia de datos reales.
- **Bucle de Reentrenamiento**: Este proceso puede estar basado en una programación temporal (por ejemplo, diaria o semanal) o en la acumulación de una cantidad significativa de nuevos datos. Durante este ciclo:
  - Se extraen todos los registros relevantes desde la base de datos
  - Se entrena un nuevo modelo utilizando RandomForestClassifier de Scikit-learn
  - El modelo actualizado se guarda y reemplaza al anterior, quedando disponible para su uso inmediato por otros servicios del sistema.

En cuanto a las tecnologías usadas:

- Scikit-learn: Biblioteca principal para el entrenamiento del modelo, específicamente mediante el clasificador RandomForestClassifier
- Pandas: Utilizada para la manipulación y transformación de los datos extraídos de la base de datos.
- Psycopg2: Librería para la conexión y consulta de datos en la base de datos PostgreSQL

4.Nodo en la Niebla

El **nodo en la niebla** actúa como un intermediario inteligente entre la nube y el borde, con la responsabilidad específica de adaptar los modelos de predicción a las características individuales de cada usuario. Su función principal es realizar el fine-tuning de los modelos genéricos utilizando datos etiquetados que provienen directamente del dispositivo del usuario. Este enfoque permite personalizar el comportamiento del sistema, mejorando la precisión y relevancia de las predicciones para cada caso particular.

4.1.Entrenador de la Niebla (Fog Trainer)

El **Fog Trainer** escucha mensajes entrantes desde RabbitMQ que indican la disponibilidad de nuevos datos etiquetados por parte de los usuarios. A partir de esta señal, recupera los datos correspondientes desde la Cloud

API, realiza el fine-tuning del modelo (ya sea el modelo genérico o uno previamente personalizado), y finalmente sube el nuevo modelo ajustado a la nube, actualizando el mapeo de modelos por usuario.

El servicio mantiene un bucle de escucha constante sobre una cola específica de RabbitMQ. Utilizando la librería Pika, el Fog Trainer consume mensajes que notifican la llegada de nuevos datos etiquetados. Cada mensaje contiene información suficiente para identificar al usuario y desencadenar el proceso de personalización.

Una vez recibido un mensaje, el servicio consulta la Cloud API mediante Requests para obtener los datos etiquetados del usuario correspondiente. Estos datos son procesados y validados utilizando Pandas, asegurando que cumplan con los requisitos mínimos para realizar un ajuste efectivo del modelo.

Para evitar sobre ajustes o personalizaciones poco representativas, el sistema define un umbral mínimo de muestras necesarias para proceder con el ajuste fino: MIN_SAMPLES_FOR_FINE_TUNING. Si el número de muestras etiquetadas no alcanza este umbral, el proceso se descarta temporalmente y se reintentará en futuras ejecuciones.

El flujo completo de personalización sigue estos pasos:

1. **Descarga del Modelo Genérico o Personalizado Existente**: Se obtiene el modelo base desde la Cloud API. Si el usuario ya cuenta con un modelo personalizado, se utiliza como punto de partida.
2. **Fine-tuning**: Se entrena el modelo utilizando RandomForestClassifier de Scikit-learn, incorporando los nuevos datos etiquetados del usuario.
3. **Subida del Modelo Personalizado**: El modelo ajustado se serializa y se envía de vuelta a la Cloud API, quedando disponible para su uso inmediato.
4. **Actualización del Mapeo de Modelos**: Se actualiza el registro que asocia a cada usuario con su modelo personalizado más reciente.

En cuanto a las tecnologías usadas:

- Scikit-learn: Para el entrenamiento y ajuste fino del modelo, utilizando RandomForestClassifier.
- Pandas: Para la manipulación y validación de los datos etiquetados.
- Pika: Cliente de Python para la comunicación con RabbitMQ.

- Requests: Para interactuar con la Cloud API y gestionar la descarga/-subida de modelos.

5.Nodo de Borde (Simulado)

En una implementación física, el nodo de borde tendría las siguientes responsabilidades:

- **Recolección de Datos**: Captura continua de señales fisiológicas y de movimiento generados por los sensores del dispositivo (acelerómetro, giroscopio, ritmo cardíaco, pasos, etc.).
- **Predicción Local**: Utiliza el modelo personalizado más reciente para realizar inferencias directamente en el dispositivo, permitiendo respuestas rápidas sin necesidad de conexión constante a la nube.
- **Etiquetado y Envío de Datos**

Dado que el sistema aún no está desplegado en dispositivos reales, el comportamiento del nodo de borde se encuentra actualmente simulado mediante el servicio Docker llamado fog_tester. Este servicio emula el flujo de trabajo del nodo de borde de la siguiente manera:

- Genera datos sintéticos o reproduce datos históricos previamente almacenados.
- Ejecuta predicciones utilizando el modelo personalizado del usuario (si está disponible).
- Etiqueta los datos de forma automática o semiautomática.
- Publica los datos etiquetados en la cola de RabbitMQ, desencadenando así el proceso de fine-tuning en el nodo en la niebla.

Esta simulación permite validar el comportamiento del sistema completo en un entorno controlado, facilitando el desarrollo y la depuración.

6.Nodo de Borde

Al igual que se simuló su funcionamiento para agilizar la evaluación de los servicios del sistema, se probó el funcionamiento del nodo de borde, tal y como se ve en la siguiente imagen 4.19 donde, mediante la herramienta Postman, se le notifica al nodo una serie de datos de ejemplo y obtenemos como resultado el estado que predice usando los modelos del usuario.

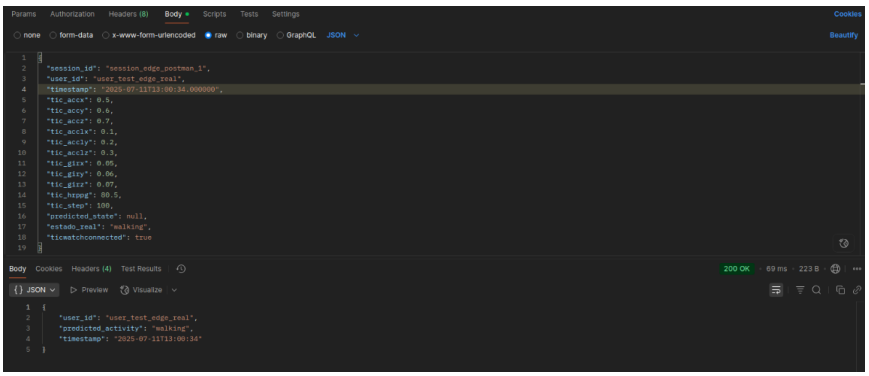


Figura 4.19: Captura de Postman enviando datos a un nodo de borde

Flujo de Datos El flujo de datos describe el recorrido que siguen los datos desde su generación en el dispositivo hasta su almacenamiento y uso en el entrenamiento de modelos. Este proceso es fundamental para garantizar la trazabilidad, integridad y disponibilidad de la información en todo el sistema.

1. **Recolección en el nodo de borde**: Podemos usar los datos recolectados directamente desde los sensores del dispositivo TicWatch o la simulación con el servicio fog_tester, que genera datos sintéticos.

2. **Envío a RabbitMQ**: Una vez generados, los datos se publican en la cola de mensajes edge_data_queue de RabbitMQ. Este mecanismo permite una comunicación asíncrona y desacoplada entre el borde y la niebla.

3. **Consumo por el nodo en la niebla**: El servicio fog_trainer escucha continuamente la cola edge_data_queue. Al recibir un mensaje, desencadena el proceso de almacenamiento y evaluación de si el usuario tiene suficientes datos etiquetados para realizar un fine-tuning.

4. **Reentrenamiento del modelo general en la nube**: de forma periódica se realiza un reentrenamiento con todos los datos disponibles en la base de datos.

Configuración y Despliegue

- Limpiar cualquier estado anterior: **docker compose down -v**

- Construir y levantar los servicios principales: **docker compose up –build db cloud_trainer rabbitmq cloud_api**

- Construir el servicio del nodo en la niebla: **docker compose up –build fog_trainer**

```
cloud_trainer-1  | Initializing TicWatchPredictor and training initial model...
cloud_trainer-1  | Initial model trained.
cloud_trainer-1  | Saving initial generic model to: /app/data/models/generic_activity_model.pkl
cloud_trainer-1  | Initial generic model successfully saved to /app/data/models/generic_activity_model.pkl
cloud_trainer-1  | --- generate_initial_model: Initial model generation complete ---
cloud_trainer-1  | Cloud Trainer Init: Initial generic model generated.
cloud_trainer-1  | --- Cloud Trainer Init: Initialization Process Complete ---
cloud_trainer-1  | Cloud Trainer Init: Initial setup complete. Starting main training loop.
cloud_trainer-1  | [2025-07-08 17:18:43.162168] Cloud Trainer: Iniciando bucle principal. Verificando datos cada 1 horas.
cloud_trainer-1  | [2025-07-08 17:18:43.162189] Cloud Trainer: Running periodic re-training check...
cloud_trainer-1  | [2025-07-08 17:18:43.162197] Cloud Trainer: Iniciando el re-entrenamiento del modelo genérico.
cloud_trainer-1  | Cargando todos los datos de entrenamiento con etiquetas de verdad desde la DB...
cloud_trainer-1  | PDB: Antes de pd.read_sql_query, conn.row_factory: N/A (para psycopg2)
cloud_trainer-1  | /app/app/data/database.py:144: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connect
ion. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.
cloud_trainer-1  |   df = pd.read_sql_query(query, conn, parse_dates=['timestamp'])
cloud_trainer-1  | Datos cargados para entrenamiento global: 3000 muestras.
cloud_trainer-1  | Distribución global de clases:
cloud_trainer-1  | estado_real
cloud_trainer-1  | sleeping    1000
cloud_trainer-1  | sedentary   1000
cloud_trainer-1  | training    1000
cloud_trainer-1  | Name: count, dtype: int64
cloud_trainer-1  | Modelo genérico entrenado. Guardando...
cloud_trainer-1  | [2025-07-08 17:18:44.110549] Cloud Trainer: Modelo genérico re-entrenado y guardado en: /app/data/models/generic_activity_model.pkl
cloud_trainer-1  | [2025-07-08 17:18:44.110678] Cloud Trainer: Periodic re-training check complete. Sleeping for 1 hours.
```

Figura 4.20: Captura de la ejecución del servicio de cloud_trainer

En la captura anterior 4.20 podemos ver como el servicio de cloud trainer comienza el bucle de entrenamiento, crea el modelo genérico y pasa a un modo de "dormir"hasta que se pase el tiempo de espera y pase a reentrenar el modelo.

Pruebas del Sistema

Para probar que el sistema está funcionando correctamente podemos usar el servicio fog_tester que simula el comportamiento del nodo de borde insertando datos directamente en la base de datos y publicando mensajes en RabbitMQ. Para su uso lanzar **docker compose up –build fog_tester**

```
Attaching to fog_tester-1
fog_tester-1  | Intentando conectar a la DB (1/10)...
fog_tester-1  | Conexión a la DB exitosa.
fog_tester-1  | Intentando conectar a RabbitMQ (1/15)...
fog_tester-1  | Conexión a RabbitMQ exitosa.
fog_tester-1  | Intentando conectar a RabbitMQ (1/15)...
fog_tester-1  | Conexión a RabbitMQ exitosa.
fog_tester-1  | Inyectando datos de prueba en la DB y publicando mensajes en RabbitMQ...
fog_tester-1  | Insertados 30 puntos de datos de ejemplo para el usuario user_test_fog en la DB (con etiquetas=True).
fog_tester-1  | Mensaje publicado en RabbitMQ: user_test_fog en 2025-07-08T17:20:49.444334
fog_tester-1  | Mensaje publicado en RabbitMQ: user_test_fog en 2025-07-08T17:20:39.452177
fog_tester-1  |
fog_tester-1  | Revisa los logs del Fog Trainer. Debería procesar los mensajes e intentar el fine-tuning.
fog_tester-1  | Si todo va bien, deberías ver que se han insertado los datos en la DB y se han publicado los mensajes en RabbitMQ.
fog_tester-1 exited with code 0
jcasmar@jcasmar-ASUS:~/Documentos/4_IIFO/TFG/computing_continuum$
```

Figura 4.21: Captura del servicio fog_tester insertando datos de prueba

```
rabbitmq-1       | 2025-07-08 17:19:47.936231+00:00 [info] <0.885.0> accepting AMQP connection <0.885.0> (172.24.0.1:36732 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:19:47.937862+00:00 [info] <0.885.0> connection <0.885.0> (172.24.0.1:36732 -> 172.24.0.3:5672): user 'guest' authenticated and granted access
to vhost '/'
rabbitmq-1       | 2025-07-08 17:19:47.940834+00:00 [info] <0.885.0> closing AMQP connection <0.885.0> (172.24.0.1:36732 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
rabbitmq-1       | 2025-07-08 17:20:17.975711+00:00 [info] <0.935.0> accepting AMQP connection <0.935.0> (172.24.0.1:53782 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:20:17.979625+00:00 [info] <0.935.0> connection <0.935.0> (172.24.0.1:53782 -> 172.24.0.3:5672): user 'guest' authenticated and granted access
to vhost '/'
rabbitmq-1       | 2025-07-08 17:20:17.982524+00:00 [info] <0.935.0> closing AMQP connection <0.935.0> (172.24.0.1:53782 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
rabbitmq-1       | 2025-07-08 17:20:47.994593+00:00 [info] <0.964.0> accepting AMQP connection <0.964.0> (172.24.0.1:53158 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:20:47.997627+00:00 [info] <0.964.0> connection <0.964.0> (172.24.0.1:53158 -> 172.24.0.3:5672): user 'guest' authenticated and granted access
to vhost '/'
rabbitmq-1       | 2025-07-08 17:20:48.000267+00:00 [info] <0.964.0> closing AMQP connection <0.964.0> (172.24.0.1:53158 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
rabbitmq-1       | 2025-07-08 17:20:49.447349+00:00 [info] <0.981.0> accepting AMQP connection <0.981.0> (172.24.0.1:53160 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:20:49.448649+00:00 [info] <0.981.0> connection <0.981.0> (172.24.0.1:53160 -> 172.24.0.3:5672): user 'guest' authenticated and granted access
to vhost '/'
rabbitmq-1       | 2025-07-08 17:20:49.452130+00:00 [info] <0.981.0> closing AMQP connection <0.981.0> (172.24.0.1:53160 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
rabbitmq-1       | 2025-07-08 17:20:49.453772+00:00 [info] <0.998.0> accepting AMQP connection <0.998.0> (172.24.0.1:53174 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:20:49.454917+00:00 [info] <0.998.0> connection <0.998.0> (172.24.0.1:53174 -> 172.24.0.3:5672): user 'guest' authenticated and granted access
to vhost '/'
rabbitmq-1       | 2025-07-08 17:20:49.456491+00:00 [info] <0.998.0> closing AMQP connection <0.998.0> (172.24.0.1:53174 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
rabbitmq-1       | 2025-07-08 17:21:18.034634+00:00 [info] <0.1022.0> accepting AMQP connection <0.1022.0> (172.24.0.1:36558 -> 172.24.0.3:5672)
rabbitmq-1       | 2025-07-08 17:21:18.039062+00:00 [info] <0.1022.0> connection <0.1022.0> (172.24.0.1:36558 -> 172.24.0.3:5672): user 'guest' authenticated and granted acces
s to vhost '/'
rabbitmq-1       | 2025-07-08 17:21:18.044122+00:00 [info] <0.1022.0> closing AMQP connection <0.1022.0> (172.24.0.1:36558 -> 172.24.0.3:5672, vhost: '/', user: 'guest')
```

Figura 4.22: Captura del servicio de RabbitMQ

En estas capturas se puede ver cómo se produce la conexión por parte del servicio fog_tester 4.21 a la cola 4.22 para insertar los mensajes y

posteriormente se consumen los mensajes.

```
cloud_api-1       | Datos etiquetados recuperados para el usuario user_test_fog: 30 filas.
cloud_api-1       | ModelRepository initialized. MODELS_DIR: /app/data/models
cloud_api-1       | Starting Cloud Model API on http://0.0.0.0:5000
cloud_api-1       | INFO:     172.24.0.1:56794 - "GET /data/user/user_test_fog/labeled HTTP/1.1" 200 OK
cloud_api-1       | INFO:     172.24.0.1:56806 - "GET /models/user/user_test_fog HTTP/1.1" 404 Not Found
cloud_api-1       | INFO:     172.24.0.1:56812 - "GET /models/generic HTTP/1.1" 200 OK
cloud_api-1       | INFO:     172.24.0.1:56818 - "POST /models/user/user_test_fog HTTP/1.1" 200 OK
cloud_api-1       | Mapeo de modelo para usuario user_test_fog actualizado a http://192.168.1.141:5000/models/user/user_test_fog (personalized).
cloud_api-1       | INFO:     172.24.0.1:56834 - "PUT /users/user_test_fog/model_mapping HTTP/1.1" 200 OK
```

Figura 4.23: Captura del servicio de cloud_api para la generación de un modelo personalizado

```
fog_trainer-1  | No new messages in the queue. Waiting...
fog_trainer-1  | [2025-07-08 17:21:18.030464] Fog Trainer: Starting to consume messages...
fog_trainer-1  | CloudAPIClient initialized. Models URL: http://192.168.1.141:5000/models, Data URL: http://192.168.1.141:5000/data, Users URL: http://192.168.1.141:5000/users
fog_trainer-1  | [2025-07-08 17:21:18.044142] Fog Trainer: Consumed 2 messages.
fog_trainer-1  | Users with new labeled data: ['user_test_fog']
fog_trainer-1  | User user_test_fog: Fetching all labeled data from Cloud API...
fog_trainer-1  | Attempting to fetch labeled data for user user_test_fog from http://192.168.1.141:5000/data/user/user_test_fog/labeled...
fog_trainer-1  | Successfully fetched 30 labeled data points for user user_test_fog.
fog_trainer-1  | User user_test_fog: Fine-tuning model with 30 samples.
fog_trainer-1  | User user_test_fog: Checking for existing custom model in Cloud API...
fog_trainer-1  | Attempting to download user user_test_fog model from http://192.168.1.141:5000/models/user/user_test_fog...
fog_trainer-1  | Error downloading user user_test_fog model from http://192.168.1.141:5000/models/user/user_test_fog: 404 Client Error: Not Found for url: http://192.168.1.141
:5000/models/user/user_test_fog
fog_trainer-1  | User user_test_fog: No custom model found. Downloading generic model from Cloud API...
fog_trainer-1  | Attempting to download generic model from http://192.168.1.141:5000/models/generic...
fog_trainer-1  | Successfully downloaded generic model.
fog_trainer-1  | User user_test_fog: Loaded generic model from Cloud API.
fog_trainer-1  | TicWatchPredictor: Modelo cargado desde bytes.
fog_trainer-1  | TicWatchPredictor: Re-ajustando el RandomForestClassifier existente con nuevos datos.
fog_trainer-1  | TicWatchPredictor: Entrenamiento/fine-tuning del modelo completado.
fog_trainer-1  | User user_test_fog: Uploading fine-tuned model to Cloud API...
fog_trainer-1  | Attempting to upload user user_test_fog model to http://192.168.1.141:5000/models/user/user_test_fog...
fog_trainer-1  | Successfully uploaded user user_test_fog model: {'message': 'User model for user_test_fog uploaded successfully', 'path': '/app/data/models/users/user_test_fo
g_activity_model.pkl'}
fog_trainer-1  | Attempting to update model mapping for user user_test_fog in http://192.168.1.141:5000/users/user_test_fog/model_mapping...
fog_trainer-1  | Successfully updated model mapping for user user_test_fog: {'message': 'Model mapping for user user_test_fog updated successfully'}
fog_trainer-1  | User user_test_fog: Model mapping updated successfully in Cloud API.
fog_trainer-1  | User user_test_fog: Fine-tuning and upload complete.
fog_trainer-1  | [2025-07-08 17:21:48.287238] Fog Trainer: Starting to consume messages...
fog_trainer-1  | CloudAPIClient initialized. Models URL: http://192.168.1.141:5000/models, Data URL: http://192.168.1.141:5000/data, Users URL: http://192.168.1.141:5000/users
```

Figura 4.24: Captura del servicio de fog_trainer

En estas últimas capturas 4.23 4.24 podemos ver como el nodo en la niebla solicita al nodo de la nube los datos de un usuario específico, en este caso “user_test_fog”. Como detecta que la cantidad es suficiente, trata de conseguir el último modelo personalizado del usuario. En este caso, como no tiene, da fallo y genera el error 404 indicando que no ha encontrado el modelo personalizado de ese usuario. Tras ver esto, el nodo en la niebla procede a descargar el modelo genérico y a adaptarlo al usuario específico. Tras realizar el proceso de fine-tunning, sube el nuevo modelo a la nube y publica este cambio mapeando el modelo con el usuario.

Resultados del Sprint

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Documentación	1/7/25	20	3	3	Replanteamiento sobre el funcionamiento de aprendizaje automático y donde se ejecuta cada cosa
Implementar entrenamiento genérico	2/7/25	8	4	4	Creación de la carpeta con las funcionalidades de cada nodo para tener todo el codigo organizado. Implemento la parte de creación del modelo de entrenamiento genérico
Implementar entrenamiento genérico	3/7/25	8	7,5	3,5	Reunión con los tutores, creación de la base de datos posgresql para el cloud y la agrego a docker compose, corrijo las consultas para la generación del modelo genérico
Crear api en el cloud para gestionar la demanda del resto de servicios que tratan de acceder a la base de datos	4/7/25	4	3,5	3,5	Añado una api al cloud que me permite gestionar los modelos para el resto de nodos
Sistema de comunicación entre el Cloud y Fog para consultar modelos en versiones anteriores	5/7/25	4	2	2	Añado un contenedor de rabbit para gestionar el paso de mensajes al fog
Sistema de comunicación entre el Cloud y Fog para consultar modelos en versiones anteriores	6/7/25	4	5,5	3,5	Correcciones sobre la api del nodo cloud para que se puedan actualizar los modelos, se notifique donde pone la ultima versión del modelo y estos endpoints son consumidos por el nodo fog. Corrección del nodo fog para que use los endpoitns de la api del cloud
Correcciones y test	7/7/25	5	4	4	Correcciones sobre el nodo cloud, su api, el nodo fog, implementación de un test para el nodo fog.
Documentación	8/7/25	20	6,5	3,5	Redacto la documentación sobre el diseño del sistema con machine learning para el fog y el cloud y ejecución de fog_tester.
Documentación	9/7/25	20	8,5	2	Reunión con los tutores, redacción de actas, planteamiento del buffer en la aplicación y corrección de documentación.
Documentación	10/7/25	20	10,5	2	Cambios en la documentación añadiendo correcciones de los tutores.
Crear servicio de ingesta de datos	11/7/25	3	2,5	2,5	Cambio para tener un servicio de ingesta, dos colas de paso de mensaje para el servicio de ingesta y para el nodo fog y puesta en marcha el funcionamiento del servicio del edge
Documentación	12/7/25	20	17,5	7	Correcciones en la documentación, añado los avances del sprin, con capturas. Continuo con la documentación sobre servicio de monitoreo con Prometheus y Grafana y comienzo a hacer planteamientos para su puesta en marcha
Documentación	13/7/25	20	25,5	8	Creación de gráficos de los esprines terminados, mejoras y correcciones en la documentación
Documentación	14/7/25	20	27,5	2	Correcciones y mejoras en la memoria
Totales		44		50,5	

Figura 4.25: Organización de tareas reales del Sprint 4

Este sprint 4.25 no ha recibido los mismos resultados de planificación que el anterior, teniendo que dedicar 6,5 horas extra debido a la cantidad de documentación necesaria para este punto, dado que esta implementación es uno de los pilares de este trabajo.

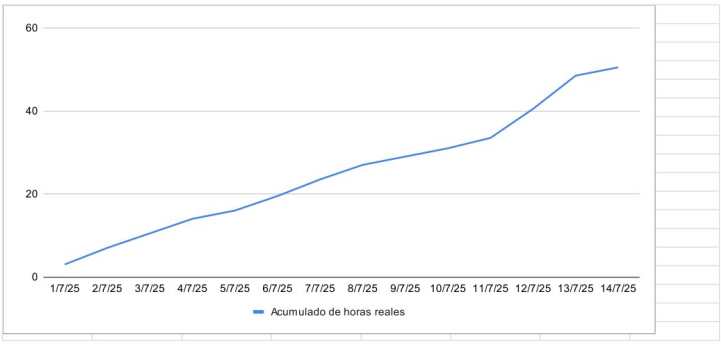


Figura 4.26: Acumulado de horas reales del Sprint 4

Como se puede observar, la evolución del trabajo no ha sido uniforme en el tramo final dado el alto volumen de documentación acumulado del sprint.

Durante este sprint se analizó el comportamiento del nodo en la niebla en la arquitectura del sistema. Inicialmente, se planteó un modelo de comunicación síncrona mediante la API Gateway, en el cual el nodo de borde notifica al nodo en la niebla cuando dispone de datos listos para procesar. Sin embargo, este enfoque presenta limitaciones al escalar: con múltiples nodos de borde enviando notificaciones simultáneas, el nodo en la niebla no puede responder de forma inmediata a todos ellos, generando esperas y bloqueos innecesarios.

Para mitigar este problema, se propone la incorporación de una cola de mensajes asíncronos utilizando **RabbitMQ**. Con esta solución, el nodo en la niebla puede consumir los mensajes a su propio ritmo, evitando cuellos de botella y permitiendo una mayor eficiencia en entornos distribuidos.

Adicionalmente, surgieron reflexiones importantes sobre la **tolerancia a fallos y la migración de servicios**. Dado el dominio de la aplicación y la frecuencia de envío de datos, se considera que no es necesaria una alta tolerancia a fallos. No obstante, se plantearon dos enfoques que podrían aplicarse en caso de escalar la robustez del sistema:

- **Proxy intermediario**: Implementar desde el inicio un proxy que reciba y almacene los datos enviados por los usuarios. Los nodos encargados del procesamiento consumirían los datos desde este proxy, garantizando independencia en caso de fallos.
- **Buffer en la aplicación cliente**: Como alternativa más liviana, se plantea que la propia aplicación mantenga un buffer temporal con los

datos enviados. En caso de fallo de un nodo en la niebla , el sistema puede redirigir los datos desde el buffer hacia un nuevo nodo asignado. En el peor de los casos, se produciría una sobreescritura de datos, aunque dada la frecuencia de envío, los valores no variarían significativamente.

4.3.6. Sprint 5: Ciclo MAPE-K Básico

- **Objetivo**: Implementar la parte de monitoreo sobre cada nodo del sistema.
- **Fecha de Inicio**: 14/7/2025
- **Fecha de Finalización**: 28/7/2025

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
T9	Implementar monitorización del estado de cada nodo	6h	6
–	Documentación	12h	–

Cuadro 4.10: Sprint backlog del Sprint 5

Implementación

Este sprint se ha centrado en incorporar capacidades de monitoreo en los nodos distribuidos del sistema, siguiendo el ciclo **MAPE-K**. El objetivo ha sido dotar a cada nodo de autonomía para exportar datos en tiempo real sobre su estado. Igualmente, se ha desarrollado el Spike 2 y el Spike 3 se ha comenzado.

Monitorización Local de Recursos

Como fase inicial, se desplegó un servicio de monitorización basado en **Prometheus y Grafana**. Este sistema permite recolectar métricas locales del nodo, incluyendo uso de CPU y RAM, y visualizar tendencias en tiempo real.

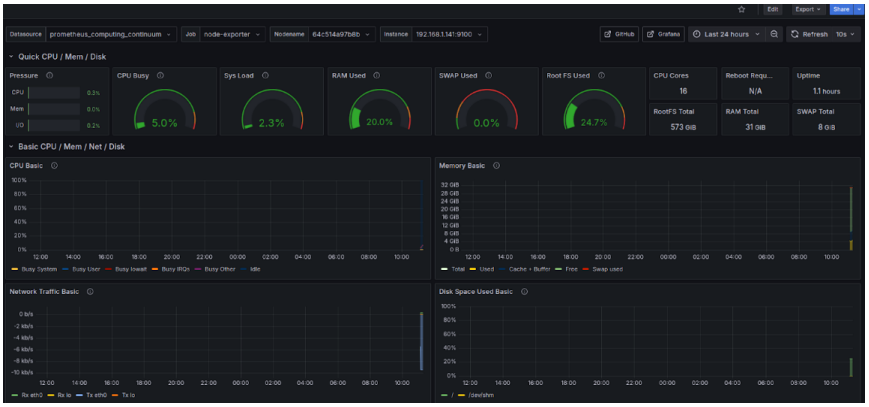


Figura 4.27: Captura del Dashboard de Grafana

En esta última imagen 4.27 se puede ver como, dado que es el comienzo del despliegue del servicio, se están comenzando a rellenar las gráficas con los datos del estado del nodo.

Para obtener datos precisos del host (y no del contenedor), se configuró un volumen que expone la información directamente del equipo anfitrión. Se realizaron pruebas comparativas entre los datos recogidos por Prometheus y los registrados por la terminal del sistema operativo, confirmando la fiabilidad de la solución propuesta. Esta capa de monitorización representa el componente "**Monitor**" del ciclo MAPE-K.

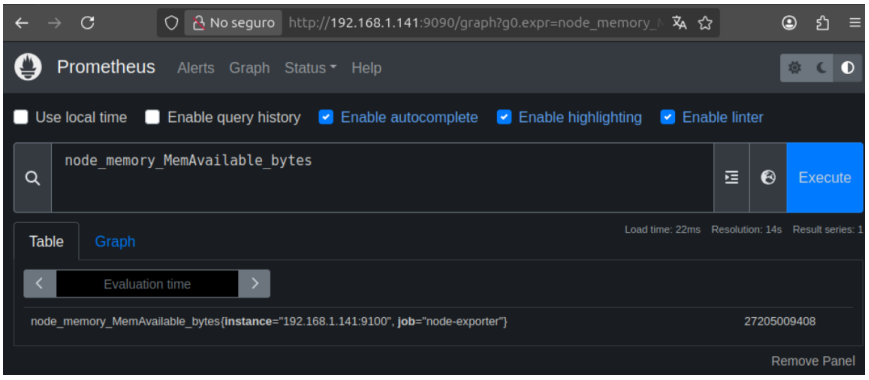
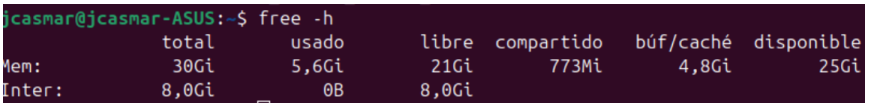


Figura 4.28: Ejecución sobre el Prometheus para consultar la memoria disponible



```
jcasmar@jcasmar-ASUS:~$ free -h
              total        usado        libre  compartido   búf/caché  disponible
Mem:           30Gi       5,6Gi        21Gi       773Mi       4,8Gi        25Gi
Inter:        8,0Gi          0B       8,0Gi
```

Figura 4.29: Ejecución del comando free -h sobre el terminal del equipo anfitrión

Comparando las dos últimas imágenes 4.28 4.29 se puede ver que node_memory_MemAvailable_bytes = 27205009408 bytes, lo cual es equivalente a 25.34 GiB, lo cual corresponde con el resultado del terminal.

Análisis y Detección de Anomalías

Se implementó una lógica básica de análisis sobre los datos recolectados, orientada a identificar posibles anomalías en el consumo de recursos. Esto incluye umbrales configurables que permiten detectar situaciones como sobrecarga del nodo, comportamiento inusual o caídas de rendimiento. Un ejemplo de estos pueden ser la detección de un nodo con un uso de CPU a más de un 90 % o el uso de memoria a más de un 80 %.

Resultados del Sprint

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Documentación	14/7/25	12	2	2	Redacción de memoria apartados del sprint anterior y planificación de este sprint
Documentación	15/7/25	12	4	2	Redacción de memoria apartados del sprint anterior
Documentación	16/7/25	12	5	1	Reunión con los tutores
Documentación	21/7/25	12	8	3	Documentación de tecnologías
Documentación	22/7/25	12	11,5	3,5	Documentación de tecnologías y conclusiones
Documentación	23/7/25	12	12,5	1	Reunión con los tutores
Implementar monitorización del estado de cada nodo	24/7/25	6	4	4	Puesta en marcha de Node Exporter, Prometheus y Grafana
Totales		18		16,5	

Figura 4.30: Organización de tareas reales del Sprint 5

Observando la tabla de tareas reales 4.30, podemos ver que el Sprint 5 se centró en la documentación técnica y en la implementación del sistema de monitorización distribuida, dando los primeros pasos hacia la integración del ciclo MAPE-K. Aunque se planificaron 18 horas de trabajo, se ejecutaron 16,5 horas reales, lo que refleja una buena aproximación al objetivo previsto.

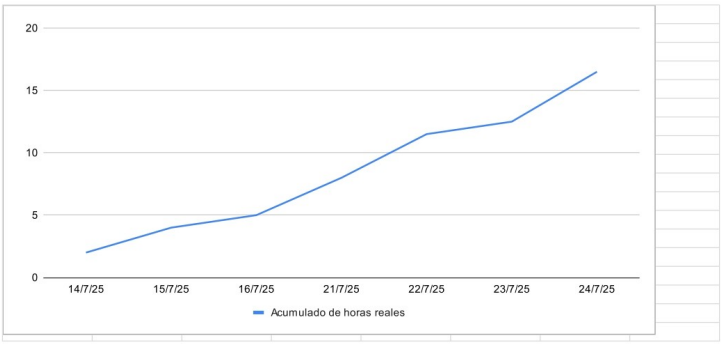


Figura 4.31: Acumulado de horas reales del Sprint 5

La gráfica del acumulado de horas mostrada en la Figura 4.31 refleja una progresión no lineal en la carga de trabajo diaria. Esta variabilidad no responde a la complejidad técnica de las tareas, sino a una distribución desigual de la dedicación, ya que la principal actividad durante este sprint ha sido la documentación técnica. Al tratarse de una tarea que permite mayor flexibilidad en su ejecución, se han concentrado bloques de trabajo en determinados días, generando picos visibles en la curva de acumulado.

El quinto sprint marcó el inicio de la implementación del ciclo **MAPE-K** en los nodos del sistema, con el objetivo de dotarlos de capacidades adaptativas basadas en su estado operativo. Se avanzó en la monitorización real de recursos, la detección de anomalías, la planificación de ajustes y la ejecución de cambios en tiempo real.

De cara a los próximos sprines se debe corregir la parte de la búsqueda de ips de los nodos, ya que esto se encuentra por el momento estático en un archivo definido y, para contemplar un despliegue distribuido, sería mejor si se añadiesen automáticamente. Por otro lado, la lógica de ejecución aún no está implementada, por lo que los nodos no reaccionan automáticamente ante anomalías.

4.3.7. Sprint 6: Orquestación Local Básica, Monitoreo Central y MAPE-K

- **Objetivo**: Establecer la orquestación básica de servicios con despliegue distribuido, comunicación entre servicios, mecanismo de autoadaptación y puesta en marcha del sistema completo.
- **Fecha de inicio**: 28/7/205
- **Fecha de finalización**: 11/8/2025

Sprint Backlog

ID Tarea	Descripción de Tarea	Estimación	HU
–	Documentación	20h	–
–	Correcciones y test	10h	–
T12	Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema	20h	11

Cuadro 4.11: Sprint backlog del Sprint 6

Implementación

El trabajo realizado durante este sprint se ha centrado en la migración y contenerización de una arquitectura de microservicios distribuida a un entorno de orquestación, junto con el desarrollo del Spike 3 y 4. Para ello, se ha seleccionado **Kubernetes** en su distribución ligera **K3s**. K3s resulta ideal para el contexto de **Computación de Borde**, ya que su bajo consumo de recursos y su naturaleza certificada lo convierten en una plataforma robusta

para gestionar cargas de trabajo en nodos con hardware limitado y baja latencia.

El proceso de implementación ha consistido en traducir las definiciones de servicios de Docker Compose a manifiestos de Kubernetes (archivos YAML), adaptando cada componente para su correcto funcionamiento en el nuevo entorno orquestado.

1. Requisitos previos

Antes de proceder con el despliegue en el clúster K3s, es fundamental asegurar una serie de requisitos y configuraciones iniciales que garantizan un proceso de implementación fluido y libre de errores.

- **Imágenes de Docker en un Registro Público**: Se requiere que las imágenes de Docker de los servicios estén construidas y disponibles en un registro de contenedores público, como Docker Hub. Esto permite que el clúster K3s pueda descargar las imágenes necesarias de manera fiable. Ejemplo de construcción y subida.
  - Construcción de la imagen: docker build -t jmcasanovam/cloud_node:v1.0 -f ./cloud_node/Dockerfile .
  - Para subirlo a Docker Hub: docker push jmcasanovam/cloud_node:v1.0
- **Clúster K3s Operativo**: Un clúster K3s debe estar configurado y en estado funcional, ya sea en un entorno de múltiples nodos o en un nodo único para pruebas.
- **Acceso y Herramientas**: Es necesario tener la herramienta de línea de comandos de Kubernetes, kubectl, instalada en la máquina desde la que se gestionará el clúster.
- **Manifiestos de Despliegue (YAML)**: Se deben tener preparados todos los archivos de manifiesto en formato YAML. Estos archivos contienen las especificaciones detalladas para cada recurso de Kubernetes que se va a desplegar.

2. Estructura y Creación de Namespace

Para una organización y gestión eficiente del clúster, se han creado dos namespaces (espacios de nombres) principales, lo que permite aislar lógicamente los diferentes componentes del sistema:

- monitoring: Dedicado a los servicios de monitorización del clúster.
- core: Namespace principal donde residen los microservicios de la arquitectura.

Manifiesto de ejemplo (core):

```
# core/namespace.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: core
```

Para ejecutar su despliegue, se haría con el siguiente comando:

```
kubectl apply -f core/namespace.yaml
```

3. Despliegue de los Servicios de Monitorización (Namespace Monitoring)

La migración de los servicios de monitorización de Prometheus, Node-Exporter y Grafana se realizó mediante manifiestos de Kubernetes que, en la mayoría de los casos, utilizan imágenes oficiales.

- **Grafana**: Se configura un Deployment para gestionar las réplicas, un PersistentVolumeClaim (PVC) para la persistencia de los dashboards y los datos, y un Service que expone la interfaz de usuario.
- **Node-Exporter**: Se despliega como un DaemonSet. Esta elección técnica es crucial, ya que un DaemonSet garantiza que un pod de Node-Exporter se ejecute automáticamente en cada nodo del clúster, permitiendo la recolección de métricas a nivel de sistema. También se configura un Service para que Prometheus pueda recolectar sus métricas.
- **Prometheus**: Se utiliza un ConfigMap para definir la configuración, incluyendo las reglas de recolección de métricas (scrape_configs), el intervalo de tiempo y la lógica para que descubra automáticamente los pods de node-exporter en cada nodo.

4.Despliegue de los Servicios del Sistema (Namespace core)

Esta sección detalla el despliegue de los microservicios principales. Cada servicio se ha adaptado a la lógica de Kubernetes, utilizando Deployments para gestionar sus réplicas, Services para la comunicación interna y PersistentVolumeClaims cuando se requiere persistencia de datos.

4.1.Servicio de Base de Datos (db)

Se utiliza una imagen de PostgreSQL oficial. Para el despliegue, se configura un PVC para asegurar la persistencia de los datos, un Secret para las credenciales de acceso seguras y un Deployment que se encarga de gestionar la réplica.

La implementación incluye la asignación de roles de nodo para una optimización de recursos. Se utiliza la directiva nodeSelector para asegurar que

el pod de la base de datos se despliegue únicamente en nodos etiquetados con el rol db-node.

Comando para Asignar el Rol:

```
kubectl label node jcasmar-asus role=db-node
```

```
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f deployment.yaml
deployment.apps/db configured
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get pods -n core
NAME                    READY   STATUS    RESTARTS   AGE
db-7f7465bcf4-qnmdl     1/1     Running   0          4m2s
db-85c6976858-wp7m4     0/1     Running   0          9s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get pods -n core
NAME                    READY   STATUS    RESTARTS   AGE
db-7f7465bcf4-qnmdl     1/1     Running   0          4m9s
db-85c6976858-wp7m4     0/1     Running   0          16s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get pods -n core
NAME                    READY   STATUS    RESTARTS   AGE
db-85c6976858-wp7m4     1/1     Running   0          34s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$
```

Figura 4.32: Migración de versiones del servicio de base de datos

En esta imagen 4.32 se puede ver cómo se produce una migración progresiva, donde en un primer momento conviven las dos versiones y tras un periodo de migración (crear la segunda versión) se ve como desaparece la primera versión y solo queda la segunda.

4.2.Servicios de Mensajería y Almacenamiento

- RabbitMQ: Se configura con un Deployment y un Service para la comunicación entre microservicios, además de un Secret para gestionar las credenciales de acceso de forma segura.

```
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/rabbitmq$ kubectl get pods -n core
NAME                        READY   STATUS    RESTARTS      AGE
db-85c6976858-wp7m4         1/1     Running   2 (42m ago)   46h
rabbitmq-5bdb7df864-js8dv   1/1     Running   0             39s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/rabbitmq$
```

Figura 4.33: Comprobación del despliegue de RabbitMQ

  En esta imagen 4.33 se puede ver el resultado del despliegue del pod con rabbitmq junto a la base de datos.

- Cloud_trainer y Cloud_api: Para estos servicios se utiliza la misma imagen de Docker. Se ha configurado un volumen persistente (PVC) compartido para que ambos servicios puedan acceder a los modelos de machine learning que se generan. El cloud_trainer genera los modelos y el cloud_api los lee para servirlos.

```
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$ kubectl get pvc -n core
NAME          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
db-pvc        Bound    pvc-28793acc-16b6-40be-9c30-afc82d9c7c5f   5Gi        RWO            local-path     <unset>                 47h
models-pvc    Bound    pvc-9b7d58d5-bab4-403c-906f-efa2461136d1   1Gi        RWO            local-path     <unset>                 3m5s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$
```

Figura 4.34: Comprobación de la creación del volumen

En esta captura 4.34 se puede ver como el volumen para el almacenamiento de modelos se ha creado correctamente y convive junto con el de la base de datos.

Una vez que el pod ya está funcionando, podemos pasar a revisar los logs. Con el nombre podemos comprobarlo:

```
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$ kubectl apply -f ./cloud_trainer/deployment.yaml
deployment.apps/cloud-trainer created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$ kubectl get deployment,pods -n core -l app=cloud-trainer
NAME                                   READY   STATUS    RESTARTS   AGE
pod/cloud-trainer-798bb6fddc-wpgcn     1/1     Running   0          12s
```

Figura 4.35: Consulta del nombre del pod

```
jcasmar@jcasmar-ASUS:~/k8s-deploy/core$ kubectl logs pod/cloud-trainer-798bb6fddc-wpgcn -n cor
e
--- cloud_trainer_init_and_loop.py START ---
--- Cloud Trainer Init: Starting Initialization Process ---
Checking for DB directory: postgresql://user:password@db.core.svc.cluster.local:5432
Cloud Trainer Init: Ensuring database tables are created...
Tablas de la base de datos creadas/verificadas (PostgreSQL).
Cloud Trainer Init: Database tables checked.
Checking for generic model at: /app/data/models/generic_activity_model.pkl
Cloud Trainer Init: Generic model not found. Generating initial model...
--- generate_initial_model: Starting initial model generation ---
```

Figura 4.36: Revisión de los logs del pod. Parte 1.

```
Datos cargados para entrenamiento global: 3000 muestras.
Distribución global de clases:
estado_real
sleeping     1000
sedentary    1000
training     1000
Name: count, dtype: int64
Modelo genérico entrenado. Guardando...
[2025-08-07 19:06:20.958177] Cloud Trainer: Modelo genérico re-entrenado y guardado en: /app/data/models/generic_activity_model.pkl
[2025-08-07 19:06:20.958368] Cloud Trainer: Periodic re-training check complete. Sleeping for 1 hours.
```

Figura 4.37: Revisión de los logs del pod. Parte 2.

Viendo estas últimas imágenes (4.35 4.36 4.37) podemos afirmar que se ha desplegado correctamente el servicio de cloud_trainer, ya que obtenemos resultados coherentes y similares a los obtenidos con un despliegue con Docker Compose.

4.3.Servicios Fog_trainer y Data_ingestor

Estos servicios se despliegan de forma similar, con un Deployment que define sus características y un Service para la comunicación. Sus archivos de despliegue contienen la configuración necesaria para que puedan acceder a

los servicios de soporte, como RabbitMQ, la base de datos (db) y Cloud_api.

4.4.Edge_service

Este servicio se despliega con un Deployment y un Service que lo expone en el puerto 3000 para que sea accesible para los usuarios finales. Al igual que los otros servicios, sus manifiestos incluyen la configuración para que pueda comunicarse con RabbitMQ y Cloud_api para obtener los modelos.

4.5.Manager-mapek

Este es el componente central de la autoadaptación. Su despliegue requiere una configuración detallada para su correcto funcionamiento.

- **Deployment**: Se define el Deployment del manager-mapeka, especificando la imagen del contenedor, el servicio y las variables de entorno necesarias para su correcto funcionamiento.
- **Service**: Un Service expone la API en el puerto 8001 para que otros servicios puedan interactuar con él.
- **Configuración de Permisos**: Para que el manager-mapek pueda interactuar con el clúster (por ejemplo, para listar nodos y escalar deployments), necesita permisos explícitos. Para ello, se crea un ClusterRole y un ClusterRoleBinding que le otorgan los permisos necesarios para realizar estas acciones.

A la hora de listar los nodos, en el desarrollo se ha tenido en cuenta las etiquetas asignadas a cada nodo. Para ello, se debe añadir ésta previamente con el siguiente comando, sustituyendo "nombre" por el nombre del nodo y "tipo" por el tipo de nodo (edge, fog, cloud):

```
kubectl label node nombre node-type=tipo
```

Para la creación de este componente se ha continuado con el desarrollo del código MAPE-K planteado en el sprint anterior.

- **Fase de Monitorización:** La fase de monitoreo se mantiene gracias a la integración con Prometheus, que actúa como la fuente de verdad del sistema. A través de consultas a su API, el servicio manager-mapek obtiene datos de telemetría en tiempo real sobre el estado de cada nodo del clúster, incluyendo la carga de CPU, el porcentaje de uso de memoria y otros indicadores de rendimiento.
- **Fase de Análisis y Planificación:** El análisis se ha enriquecido con la definición de umbrales para detectar anomalías. Para fines de demostración y prueba, se establecieron los siguientes umbrales:

- Carga de CPU: Un umbral del 50 % para la carga de CPU. Superar este valor indica que el nodo está experimentando un aumento significativo en la demanda.
- Uso de Memoria: Un umbral del 85 % para el porcentaje de memoria utilizada. Exceder este valor puede sugerir un consumo de recursos excesivo en un pod.

Basándose en estos análisis, el plan de acción se ejecuta de la siguiente manera:

- Si la carga de CPU supera el 50 %, se activa una acción de scale_up.
- Si el uso de memoria supera el 85 %, se planifica un reinicio del pod afectado para liberar los recursos.

- **Fase de Ejecución:** La fase de ejecución traduce el plan de acción en comandos directos de Kubernetes. Estos comandos son orquestados por el manager-mapek para modificar el estado del clúster de forma automática.

  - Para el scale_up:

```
kubectl scale deployment/edge-service --replicas
    =+1
```

  - Para el reinicio del pod:

```
kubectl rollout restart deployment/edge-service
```

- **Fase de Conocimiento:** Cada cambio o acción ejecutada se registra y se almacena en una lista de cambios. Este registro sirve como una base de conocimiento histórica que puede ser utilizada en futuras iteraciones del proyecto para refinar la estrategia de adaptación, ajustar los umbrales de manera predictiva y optimizar el rendimiento del sistema a largo plazo.

Arquitectura final del sistema

Tras el desarrollo del último sprint, se ha completado el despliegue de los microservicios y la implementación de las comunicaciones entre componentes. A continuación, se presenta el diagrama de arquitectura final obtenido tras la ejecución de todos los sprints.

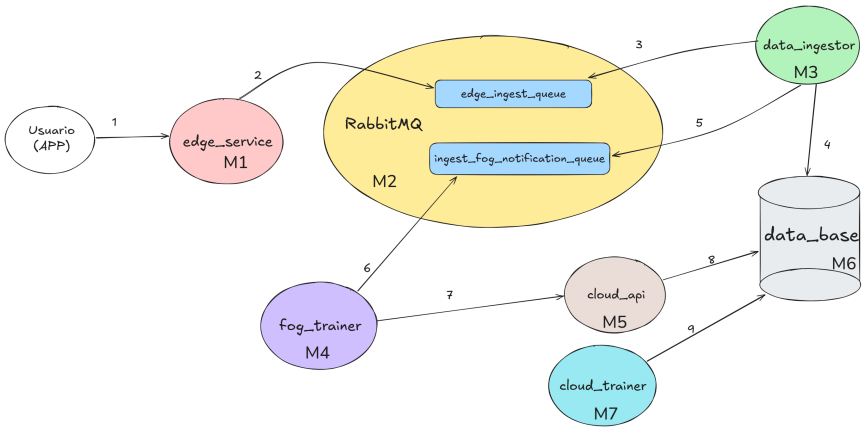


Figura 4.38: Diagrama de arquitectura. Elaboración Propia.

La arquitectura resultante, tal y como se puede ver en la figura 4.1, consiste en:

1. **Recepción de Datos por Edge_service**: El proceso se inicia en el borde. La aplicación del usuario (Usuario (APP)) envía los datos de actividad física directamente al Edge_service. Este servicio actúa como la puerta de entrada inicial para la información proveniente del dispositivo móvil.

2. **Envío a la Cola de Ingesta:** Una vez que el Edge_service recibe los datos, los envía a la cola edge_ingest_queue dentro del broker de mensajes RabbitMQ. Este paso es fundamental para garantizar la robustez del sistema, ya que la comunicación asíncrona desacopla el Edge_service del resto del flujo, evitando la pérdida de datos si los servicios posteriores no están disponibles de inmediato.

3. **Ingesta de Datos por Data_ingestor:** El servicio Data_ingestor escucha constantemente los mensajes en la cola edge_ingest_queue. Cuando un nuevo mensaje (con los datos del usuario) está disponible, lo consume y lo procesa para su posterior almacenamiento.

4. **Almacenamiento en la Base de Datos**: Los datos procesados por el Data_ingestor son finalmente persistidos en la base de datos (data_base). Esto asegura que la información del usuario se almacena de forma segura y se mantiene accesible para otros servicios.

5. **Notificación a la Cola de Notificaciones:** Una vez que el Data_ingestor ha almacenado los datos, envía una notificación a una cola diferente, ingest_fog_notification_queue. Esta notificación actúa como una señal para los servicios de entrenamiento que operan en la niebla.

6. **Procesamiento por Fog_trainer**: El Fog_trainer escucha la cola de notificaciones. Al recibir una señal, inicia un proceso de entrenamiento ligero para generar un modelo personalizado para el usuario.

7. **Acceso a la API en la Nube**: Durante el entrenamiento, el Fog_trainer puede interactuar con el Cloud_api para acceder a modelos base o recursos que están centralizados en la nube. Esto permite un entrenamiento más eficiente y actualizado sin la necesidad de transferir grandes volúmenes de datos.

8. **Uso de la Base de Datos por Cloud_api**: El Cloud_api también puede acceder a la base de datos para obtener información necesaria para el entrenamiento o para servir los modelos a los servicios que lo soliciten.

9. **Interacción de Cloud_trainer con la Base de Datos**: El Cloud_trainer es responsable del entrenamiento de modelos genéricos, más complejos y pesados. Los datos de la base de datos (data_base) son utilizados por este servicio para llevar a cabo estos entrenamientos intensivos en un entorno de nube con mayores recursos.

Por otro lado, el flujo para el monitoreo quedaría así:

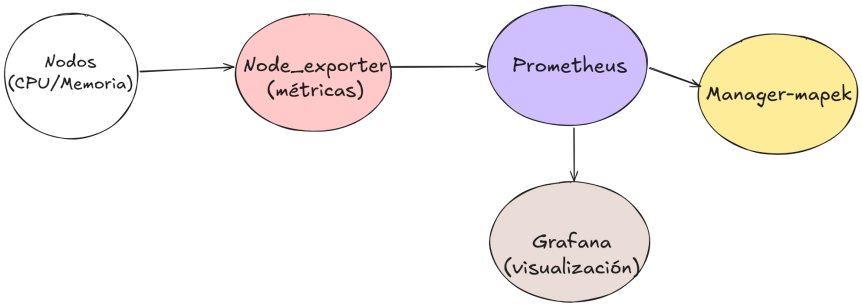


Figura 4.39: Diagrama de arquitectura de monitoreo. Elaboración Propia.

Con el diagrama 4.39 podemos ver que los Nodos exportan sus métricas mediante Node_exporter. Los datos exportados son recogidos por Prometheus, que es la fuente de datos para la herramienta de visualización Grafana y para el microservicio Manager-mapek que, a partir de la información recolectada, realiza el ciclo MAPE-K.

Resultados del Sprint

Descripción Tarea	Día	Tiempo Planificado (H)	Tiempo Acumulado(H)	Tiempo Real (H)	Comentario
Documentación	28/7/25	20	6,5	6,5	Correcciones sobre la memoria, sprines anteriores y documentación inicial del sprint 6
Documentación	29/7/25	20	12,5	6	Correcciones sobre el enfoque mapek, cambios en la memoria, documentación sobre K3s y Kubernetes, redacción de conclusiones
Documentación	30/7/25	20	16,5	4	Reunión con tutores y cambios en la memoria
Documentación	31/7/25	20	19	2,5	Documentación sobre Kubernetes, cambios en la memoria para añadir nueva información de tecnologías
Correcciones y test	2/8/25	10	5	5	Correcciones sobre manager, mapek, despliegue de servicios separados, pruebas sobre comunicaciones
Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.	3/8/25	20	3,5	3,5	Comienzo el despliegue con k3s del servicio de monitoreo con prometheus, grafana y node exporter
Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.	4/8/25	20	6,5	3	Continuo con correcciones sobre el servicio de grafana desplegado con k3s y migro el servicio de db y lo lanzo
Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.	7/8/25	20	10,5	4	Continuo con k3s, despliego el servicio de rabitmq, conexión a docker hub para subir la imagen de cloud_node, despliego servicio de cloud_trainer y de cloud_api
Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.	8/8/25	20	16	5,5	Continuo con k3s, despliego fog_trainer, data_ingestor, edge_service, correcciones en los archivos de configuración, pruebas sobre el funcionamiento
Configurar los aspectos básicos de Kubernetes para el despliegue, la gestión y la puesta en marcha del sistema.	11/8/25	20	21,5	5,5	Continuo con k3s, despliego y cambio manager-mapek, creado desde 0
Correcciones y test	12/8/25	10	8,5	3,5	Correcciones sobre manager-mapek, despliegue y prueba
Documentación	13/8/25	20	21,5	2,5	Documentación sobre el despliegue de K3s
Totales		50		51,5	

Figura 4.40: Organización de tareas reales del Sprint 6

Durante el Sprint 6 4.40 se consolidaron tres grandes bloques de trabajo: **documentación técnica, correcciones y pruebas funcionales, y despliegue del sistema distribuido con K3s.** La planificación inicial contemplaba 50 horas de trabajo, alcanzándose un total de 51,5 horas reales, lo que refleja una ejecución cercana a lo previsto.

Figura 4.41: Acumulado de horas reales del Sprint 6

Mediante esta imagen 4.41 podemos ver que este sprint se ha desarrollado con jornadas de trabajo más homogéneas que otros sprines, aproximándose a una gráfica lineal.

El sprint ha concluido satisfactoriamente, cumpliendo los objetivos principales de migrar y desplegar la arquitectura de microservicios en un entorno de orquestación en el borde con K3s. Los resultados obtenidos demuestran la viabilidad técnica y la correcta implementación del sistema de autoadaptación propuesto.

La monitorización del clúster a través de Prometheus y Grafana ha proporcionado telemetría en tiempo real sobre el estado y la carga de cada nodo y servicio. Esto no solo valida que la arquitectura está funcionando, sino que también proporciona las bases para futuros análisis del rendimiento. La implementación del DaemonSet de Node-Exporter ha demostrado ser una solución eficiente para recolectar métricas de cada nodo de forma automática.

4.4. Problemas Durante el Desarrollo

Problema	**Descripción**	**Actualización realizada**	**Duración del diagnóstico**
Incompatibilidad de versiones en la aplicación HDRescuer	Se usó el código fuente de una aplicación desarrollada previamente para la conexión con pulseras inteligentes y el almacenamiento de sus datos. La incompatibilidad entre las versiones del compilador original y la versión más reciente utilizada impidió la compilación directa del código.	Se realizó una migración progresiva a una versión intermedia del compilador, editando archivos de compilación y asegurando la compatibilidad para permitir la compilación.	5 días, debido a la necesidad de encontrar una versión compatible y la dificultad de editar archivos de compilación.
Incompatibilidad con las pulseras Empatica E4	Las pulseras inteligentes Empatica E4 ofrecen una amplia gama de datos fisiológicos, pero requieren una cuenta en la plataforma Empatica, una API Key y la vinculación de los dispositivos. El cese del soporte impidió la creación de nuevas cuentas y API Keys, imposibilitando la conexión y extracción de datos.	Se sustituyeron las pulseras Empatica E4 por un reloj TicWatch Pro, que permitía la recopilación de datos fisiológicos y la integración con la plataforma de desarrollo.	Cerca de 2 semanas, debido a la dificultad inicial para identificar la causa de la falta de conexión.

Cuadro 4.12: Problemas encontrados durante el desarrollo del sistema

4.5. Evaluación de la solución

La evaluación se ha realizado combinando la observación directa del sistema desplegado con el análisis de documentación técnica y estudios previos sobre arquitecturas distribuidas y monolíticas. Aunque no se ha implementado una arquitectura monolítica para realizar pruebas comparativas directas, se ha planteado una comparación teórica basada en el comportamiento observado del sistema distribuido y en las características reconocidas de cada enfoque.

La arquitectura distribuida autoadaptativa desarrollada presenta claras ventajas en términos de escalabilidad, resiliencia y tolerancia a fallos. Gracias a la separación de servicios y su despliegue en múltiples nodos, el sistema puede redistribuir la carga, escalar dinámicamente y continuar operando incluso ante la caída de uno o varios componentes. Esto contrasta con una arquitectura monolítica, donde la dependencia de un único nodo central puede provocar una interrupción total del servicio ante cualquier fallo crítico.

Además, el uso del ciclo MAPE-K permite una gestión proactiva de los recursos, monitorizando continuamente el estado del sistema y ejecutando acciones correctivas en tiempo real. Esta capacidad de adaptación es especialmente útil en entornos con carga variable o impredecible.

No obstante, es importante reconocer que una arquitectura monolítica puede ofrecer ventajas en ciertos contextos. Por ejemplo, al estar todo el sistema integrado en un único proceso, los tiempos de respuesta pueden ser más bajos en escenarios simples, ya que se eliminan las latencias asociadas a la comunicación entre microservicios. Asimismo, el despliegue y mantenimiento de una solución monolítica puede ser más sencillo en entornos controlados o de pequeña escala, donde la adaptabilidad no es un requisito crítico.

En conjunto, la solución distribuida propuesta está mejor preparada para entornos dinámicos y exigentes, donde la continuidad del servicio y la capacidad de adaptación son esenciales. La evaluación realizada permite concluir que, para el caso de uso abordado, la arquitectura distribuida ofrece una respuesta más robusta y flexible frente a posibles fallos y variaciones en la carga de trabajo.

Capítulo 5

Conclusiones, Trabajos Futuros y Potencial Emprendedor

5.1. Conclusiones

El presente proyecto pretende poner de manifiesto el potencial de las soluciones basadas en el paradigma de la Continuidad Computacional, que integra de forma transparente y cohesiva dispositivos IoT, nodos de borde, niebla y nube. A lo largo del desarrollo, hemos demostrado que es posible diseñar una plataforma híbrida y autoadaptativa capaz de distribuir de forma inteligente las cargas de trabajo según eficiencia energética, requisitos de latencia y disponibilidad de recursos. El resultado ha sido un prototipo en ejecución real, que valida la viabilidad técnica del paradigma y abre nuevas avenidas para la gestión dinámica de sistemas distribuidos.

En el desarrollo de este proyecto se han alcanzado todos los objetivos que inicialmente se plantearon, tanto a nivel técnico como formativo, logrando una evolución progresiva y estructurada desde el análisis teórico hasta la implementación práctica. A continuación se detalla el grado de cumplimiento de cada uno de ellos:

1. **Estudio e investigación sobre el paradigma de la Continuidad Computacional**

   Se ha realizado una revisión de literatura y marcos conceptuales, incluyendo características, retos y soluciones actuales en los entornos de borde, niebla y nube. Hemos documentado los requisitos para soportar nodos volátiles, adaptabilidad al cambio y tolerancia a fallos en condiciones reales.

2. **Identificación de estrategias de gestión de recursos de computación en sistemas distribuídos**

   Integrando conceptos del ciclo MAPE-K, se desarrollaron políticas de monitorización (Monitor), detección de anomalías (Analyze), ajuste de parámetros (Plan) y acciones de reconfiguración (Execute). Esto asegura que el sistema puede reaccionar proactivamente a cambios en el entorno o la carga de trabajo.

3. **Propuesta de infraestructura para la gestión de la Continuidad Computacional**

   Se ha creado un prototipo funcional que incluye un Nodo Manager como orquestador central, nodos de borde con capacidad de preprocesamiento e inferencia local, y nodos de niebla y nube preparados para escalabilidad y coordinación global.

4. **Uso de algoritmos para el análisis de datos (Machine Learning)**

   Se hace uso de modelos personalizados por usuario mediante scikit-learn, permitiendo una personalización de un modelo genérico para cada usuario.

5. **Implementación del Caso de Uso con Pulsera Inteligente** El prototipo incluye el procesamiento en tiempo real de datos fisiológicos, recoge y preprocesa información en el borde, ejecuta inferencias sobre estado y sincroniza con el nodo en la niebla/nube.

6. **Visualización del comportamiento del sistema**

   Se han medido métricas clave como latencia de respuesta, uso de CPU y RAM en dispositivos de borde y consumo energético estimado. Los resultados evidencian que la arquitectura distribuida incrementa la resiliencia del sistema y minimiza la dependencia de un único nodo, favoreciendo una operación más robusta y tolerante a fallos.

7. **Recomendaciones y conclusiones**

   Se han identificado estrategias clave para optimizar la eficiencia y resiliencia en sistemas distribuidos. Se recomienda separar procesamiento y almacenamiento, aplicar ciclos de auto-adaptación distribuidos (MAPE-K), integrar métricas en tiempo real y usar orquestadores.

8. **Desarrollo del proyecto de forma ágil con herramientas de gestión de proyectos**

   La aplicación del enfoque SCRUM, junto con el empleo de plataformas como GitHub y Jira, así como la elaboración sistemática de actas de reunión, ha facilitado una coordinación eficaz del proyecto. Esta combinación de herramientas ha contribuido a garantizar la transparencia en el proceso y la trazabilidad de las decisiones técnicas y metodológicas tomadas durante su desarrollo.

En paralelo a los objetivos técnicos, se han cumplido los objetivos formativos que permitían desarrollar competencias transversales:

- **Aprendizaje Básico sobre Técnicas de Machine Learning:**

  El desarrollo del sistema ha implicado profundizar en técnicas de aprendizaje supervisado para clasificación de estados fisiológicos, familiarizándome con el uso de bibliotecas como scikit-learn y técnicas de personalización de modelos.

- **Estudio y puesta en práctica de Metodologías Ágiles:**

  La gestión del proyecto se ha basado en la metodología Scrum, organizando el trabajo en sprints y gestionando el avance mediante product backlog, historias de usuario y reuniones periódicas. Esta metodología ha permitido una planificación flexible y una mejora continua en el desarrollo.

- **Exploración de tecnologías y herramientas que ayuden a dar soporte a la Continuidad Computacional:**

  Se han adquirido conocimientos prácticos en tecnologías clave como Docker, Kubernetes, Prometheus, Grafana, RabbitMQ y técnicas de orquestación, que permiten desplegar y gestionar sistemas distribuidos en entornos reales.

- **Aplicación de técnicas para la integración e interoperabilidad de Sistemas Heterogéneos:**

  El proyecto ha requerido integrar distintos tipos de dispositivos (pulseras inteligentes y Raspberry Pi), distintos entornos de cómputo (borde, niebla y nube) y protocolos de comunicación, consolidando habilidades en la interoperabilidad entre sistemas heterogéneos.

Con ello se garantiza que el trabajo ha cumplido no solo los objetivos definidos en su inicio, sino que además ha supuesto una experiencia formativa integral en las competencias necesarias para abordar proyectos en el ámbito de la Continuidad Computacional y los sistemas distribuidos inteligentes.

5.2. Trabajo Futuro

Consideramos que el sistema desarrollado ofrece muchas posibilidades de mejora y ampliación, no sólo en términos funcionales, sino también en cuanto a la aplicabilidad y robustez a nivel tecnológico. A continuación se propone una serie de líneas futuras:

Ampliar la Resiliencia y Seguridad del Sistema

- Definir mecanismos de alta disponibilidad para el Node Manager mediante despliegue en clúster con balanceo.
- Incorporar políticas de autenticación y autorización (OAuth 2.0, JWT) para proteger el acceso a datos sensibles.
- Implementar cifrado en tránsito (TLS) y en repositorios de checkpoints.

Automatización CI/CD y Pruebas

- Integrar pipelines automáticos para construir, testear y desplegar imágenes de Docker en entornos de borde y niebla.
- Añadir pruebas de carga para validar el sistema completo ante caídas de nodos y tráfico elevado.
- Versionar modelos ML y automatizar su despliegue y retroalimentación mediante pipelines.

Evolución del Ciclo MAPE-K

- Explorar estrategias adaptativas basadas en aprendizaje reforzado para mejorar decisiones del nodo de borde.
- Permitir que los propios usuarios definan políticas de tolerancia o precisión (por ejemplo, mayor frecuencia de muestreo).

- Implementar una capa de "Knowledge" que registre decisiones y facilite aprendizaje a largo plazo.

Visibilidad y Control del Sistema

- Desarrollar un dashboard interactivo con Grafana y alertas inteligentes para administradores y usuarios.
- Añadir visualización histórica de métricas, trazabilidad de cambios y eventos relevantes.
- Incorporar interfaces de control (p.ej. límites de batería, nivel de privacidad) que permitan ajustes manuales.

Optimización y Pruebas en Condiciones Reales

- Ampliar el testbench para incluir nodos más heterogéneos como Jetson Nano o dispositivos industriales.
- Medir impacto de desconexiones reales, calidad de conexión móvil y rendimiento con múltiples usuarios.
- Comparar rendimiento energético y tolerancia en escenarios reales vs. simulación.

Integración con Plataformas Externas

- Permitir que la app móvil reciba visualización preliminar de datos y estadísticas a través del nodo de borde.
- Crear APIs que faciliten la exportación de datos a plataformas médicas.

Expansión del Caso de Uso

- Extender el procesamiento a otras wearables (gafas inteligentes, balanzas, pulsioxímetros).
- Evolucionar hacia un sistema distribuido para cuidados en remoto, teleasistencia o salud preventiva.
- Integración con Smart Cities para responder a multitud de agentes conectados.

5.3. Potencial Emprendedor

El sistema desarrollado en el marco de este Trabajo de Fin de Grado no solo tiene un valor académico y quizás algo de investigación, sino que también puede presentar cierto potencial para su proyección en el ámbito empresarial y de emprendimiento tecnológico. La combinación de dispositivos IoT, procesamiento distribuido mediante el paradigma de Continuidad Computacional y capacidades de adaptación en tiempo real ofrece una base para la creación de un producto innovador con aplicaciones en distintos sectores.

Un posible enfoque de despliegue comercial sería el desarrollo de una plataforma destinada a la monitorización avanzada de la salud y el bienestar, especialmente orientada a usuarios con necesidades de seguimiento continuo como deportistas, personas mayores o pacientes con enfermedades crónicas. El sistema permitiría ofrecer

un servicio personalizado.

Desde una perspectiva empresarial, este producto podría ser ofrecido bajo un modelo de suscripción mensual o pago por servicio, orientado tanto al mercado B2C (business to customers) como al mercado B2B (business to business), ofreciendo soluciones a residencias de mayores, aseguradoras de salud, empresas deportivas o incluso instituciones médicas interesadas en programas de tele-medicina y prevención.

Además, gracias a la modularidad del sistema, es posible ampliar el catálogo de servicios incluyendo:

- Integración con Smart Cities, ofreciendo a las administraciones datos agregados para diseñar entornos urbanos más saludables.
- Servicios de analítica avanzada, orientados a centros de investigación o empresas de salud que deseen realizar estudios poblacionales sobre hábitos y patrones fisiológicos.

El carácter autoadaptativo, la tolerancia a fallos y la optimización de recursos convierten este prototipo en un candidato idóneo para ser evolucionado hacia un producto comercial escalable.

Asimismo, la creciente preocupación social por la salud preventiva y el bienestar personal proporciona un contexto de mercado favorable, que podría ser aprovechado con una estrategia adecuada de financiación, desarrollo tecnológico y captación de clientes.

En definitiva, el proyecto cuenta con los elementos técnicos y conceptuales necesarios para transformarse en una solución de mercado con alto valor añadido, sostenibilidad tecnológica y proyección internacional, especialmente en el contexto de la digitalización de la salud y el auge de la computación distribuida inteligente.

5.4. Reflexión Final

El desarrollo de este trabajo ha permitido constatar el enorme potencial que encierra el paradigma de la Continuidad Computacional como respuesta a los desafíos actuales de procesamiento distribuido en entornos IoT, borde, niebla y nube. La creciente necesidad de sistemas que sean capaces de adaptarse a la variabilidad de los recursos disponibles, a las condiciones de red y a las necesidades específicas de cada usuario, encuentra en este paradigma un enfoque sólido y eficiente que conjuga rendimiento, resiliencia y flexibilidad.

A lo largo del proyecto, no solo se ha demostrado la viabilidad técnica de construir un sistema distribuido, modular y autoadaptativo, sino que se ha puesto de relieve la importancia de adoptar arquitecturas que permitan la migración dinámica de tareas y la orquestación eficiente de recursos dispersos geográficamente. La combinación de dispositivos IoT, nodos de borde y niebla, junto con la nube, ha evidenciado que la distribución inteligente de la carga puede mejorar tanto la la-

tencia como la eficiencia energética, ofreciendo soluciones sostenibles y escalables.

Además, este trabajo ha servido como una experiencia enriquecedora que ha permitido combinar conocimientos de distintas disciplinas: ingeniería de software, Machine Learning, infraestructuras distribuidas y gestión de sistemas complejos. Esta integración multidisciplinar ha sido clave para afrontar los retos que surgen al diseñar sistemas adaptativos y autónomos, donde cada decisión técnica tiene un impacto directo en la eficacia global del sistema.

Cabe destacar que, si bien el sistema implementado es un prototipo, su diseño modular y la arquitectura sobre la que se sustenta lo convierten en una base sólida para futuras evoluciones. La adaptabilidad, la escalabilidad y la capacidad de recuperación ante fallos no son solo características deseables en entornos distribuidos, son auténticas necesidades que, en un futuro inmediato, marcarán la diferencia entre sistemas eficientes y obsoletos.

En conclusión, el trabajo realizado no solo aporta una solución funcional, sino que también sienta las bases para reflexionar sobre cómo deben evolucionar los sistemas computacionales en un mundo cada vez más interconectado y dinámico. Las oportunidades de mejora son amplias, pero también lo es el camino abierto hacia una computación ubicua, inteligente y autoadaptativa que se aproxime a la visión de sistemas verdaderamente autónomos y resilientes.

Bibliografía

[1] Areeg Al-Dulaimy, Marten Jansen, Björn Johansson, Animesh Trivedi, Alexandru Iosup, Mohammad Ashjaei, André B. Bondi, and Peng Li. The computing continuum: From IoT to the cloud. *Internet of Things*, 27:101272, 2024.

[2] Amazon Web Services (AWS). Diferencia entre arquitectura monolítica y de microservicios. `https://aws.amazon.com/es/compare/the-difference-between-monolithic-and-microservices-architecture/`. Consulta realizada el 21 de marzo de 2025.

[3] Arsys. Python: ¿Qué es y para qué sirve? `https://www.arsys.es/blog/python-que-es-y-para-que-sirve`, 2025. Consulta realizada el 22 de julio de 2025.

[4] Hesham F. Atlam, Robert J. Walters, and Gary B. Wills. Fog Computing and the Internet of Things: A Review. *Big Data and Cognitive Computing*, 2(2):10, 2018.

[5] Myriam Bendechache, Sergejs Svorobej, Pedro T. Endo, and Trevor Lynn. Simulating Resource Management across the Cloud-to-Thing Continuum: A Survey and Future Directions. *Future Internet*, 12(6):95, 2020.

[6] Ignacio Bernabé, Alberto Fernández, Helder Billhardt, and Stanislaw Ossowski. Towards Semantic Modelling of the Edge-Cloud Continuum. In A. González-Briones et al., editors, *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection. Communications in Computer and Information Science*, volume 1678, pages 69–82. Springer, Cham, 2022.

[7] Fabiana Borges Oliveira, Massimo Di Felice, and Carlos Kamienski. IoTDeploy: Deployment of IoT Smart Applications over the Computing Continuum. *Internet of Things*, 28:101348, 2024.

[8] Jin Chen and Xian Ran. Deep Learning With Edge Computing: A Review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[9] Cloud Native Computing Foundation (CNCF). Kubernetes on the edge: Getting started with KubeEdge and Kubernetes for edge computing. `https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing` 2022. Consulta realizada el 21 de julio de 2025.

[10] Codemotion. Mejores bibliotecas y frameworks para Machine Learning. `https://www.codemotion.com/magazine/`

es/inteligencia-artificial/aprendizaje-automatico/mejores-bibliotecas-y-frameworks-machine-learning/, 2025. Consulta realizada el 22 de julio de 2025.

[11] Marco Danelutto, Paolo Dazzi, and Marco Torquati. Structuring the Continuum. In Leonard Barolli, editor, *Advanced Information Networking and Applications. AINA 2024. Lecture Notes on Data Engineering and Communications Technologies*, volume 203, pages 401–404. Springer, Cham, 2024.

[12] DataCamp. Guía para el desarrollo backend en Python. https://www.datacamp.com/es/tutorial/python-backend-development, 2024. Consulta realizada el 22 de julio de 2025.

[13] Desconocido. Carguero de contenedores. https://i.blogs.es/1181c7/cargo-ship/1366_2000.jpeg, 2025. Consulta realizada el 26 de febrero de 2025.

[14] Desconocido. Ejemplos de iot. https://innovacion-tecnologia.com/wp-content/uploads/2020/08/dispositivos-IoT-Home.jpg, 2025. Consulta realizada el 26 de febrero de 2025.

[15] Desconocido. Esquema de cloud computing, fog computing y edge computing. https://img.datacentermarket.es/wp-content/uploads/2019/06/05122736/Edge-computing-y-fog-computing2-763x720.jpeg.webp, 2025. Consulta realizada el 26 de febrero de 2025.

[16] Desconocido. Esquema de computación en la nube. https://media.licdn.com/dms/image/v2/C4D12AQH0dqjjxgbknA/article-cover_image-shrink_720_1280/article-cover_image-shrink_720_1280/0/1592546362101?e=2147483647&v=beta&t=HC7rgsS6aAYkXGy-TVh3_1zPeXqqoFkPEG9hwwgMzhQ, 2025. Consulta realizada el 26 de febrero de 2025.

[17] Desconocido. Representación de ciudad inteligente. https://s03.s3c.es/imag/_v0/1200x655/1/b/b/ciudad-inteligente.jpg, 2025. Consulta realizada el 25 de febrero de 2025.

[18] DIGI. Fibra óptica de 1 GB. https://www.digimobil.es/fibra-optica/1gb, 2025. Consulta realizada el 22 de julio de 2025.

[19] Pradeep K. Donta, Ilir Murturi, Victor C. Pujol, Benedikt Sedlak, and Schahram Dustdar. Exploring the Potential of Distributed Computing Continuum Systems. *Computers*, 12(10):198, 2023.

[20] Door3. Diferentes modelos de desarrollo de software. https://www.door3.com/es/blog/different-software-development-models, 2023. Consulta realizada el 22 de marzo de 2025.

[21] Schahram Dustdar, Victor C. Pujol, and Pradeep K. Donta. On distributed computing continuum systems. *IEEE Transactions on Knowledge and Data Engineering*, 35(4):4092–4105, 2023.

[22] Empatica. Empatica E4 Band Manual. https://archive.arch.ethz.ch/esum/downloads/manuals/emaptics.pdf, 2025. Consulta realizada el 22 de julio de 2025.

[23] David Fernández-Cerero, Adam Jakóbik, Daniel Grzonka, Joanna Kołodziej, and Antonio Fernández-Montes. Security supportive energy-aware scheduling and energy policies for cloud environments. *Journal of Parallel and Distributed Computing*, 119:191–202, 2018.

[24] Christos K. Filelis-Papadopoulos, George A. Gravvanis, and Pantelis E. Kyziropoulos. A Framework for Simulating Large Scale Cloud Infrastructures. *Future Generation Computer Systems*, 79:703–714, 2018.

[25] Glassdoor. Sueldos de administrador de sistemas. `https://www.glassdoor.es/Sueldos/administrador-de-sistemas-sueldo-SRCH_KO0,25.htm`, 2025. Consulta realizada el 22 de julio de 2025.

[26] Google. WearOS by Google. `https://wearos.google.com/`, 2025. Consulta realizada el 22 de julio de 2025.

[27] Grafana. Documentación de Grafana. `https://grafana.com/docs/grafana/latest/`, 2025. Consulta realizada el 21 de julio de 2025.

[28] iArtificial. Scikit-learn: Aprendizaje automático en Python. `https://iartificial.blog/desarrollo/scikit-learn-aprendizaje-automatico-en-python/`, 2025. Consulta realizada el 22 de julio de 2025.

[29] Integrate.io. SQL vs. NoSQL. `https://www.integrate.io/blog/the-sql-vs-nosql-difference/`, 2025. Consulta realizada el 22 de julio de 2025.

[30] Carlos Kamienski, Jukka-Pekka Soininen, Michael Taumberger, Ricardo Dantas, Ayslan Toscano, Tiziana Salmon Cinotti, Rodrigo Filev Maia, and Alfredo Torre Neto. Smart Water Management Platform: IoT-Based Precision Irrigation for Agriculture. *Sensors*, 19(2):276, 2019.

[31] Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[32] Komodor. Kubernetes on Edge: Key Capabilities, Distros and Best Practices. `https://komodor.com/learn/kubernetes-on-edge-key-capabilities-distros-and-best-practices/`, 2025. Consulta realizada el 21 de julio de 2025.

[33] Kubernetes. Conceptos de Kubernetes. `https://kubernetes.io/es/docs/concepts/`, 2021. Consulta realizada el 21 de julio de 2025.

[34] Kubernetes. Namespaces. `https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/`, 2024. Consulta realizada el 21 de julio de 2025.

[35] Kubernetes. Pods. `https://kubernetes.io/docs/concepts/workloads/pods/`, 2025. Consulta realizada el 21 de julio de 2025.

[36] Kubernetes. Service. `https://kubernetes.io/docs/concepts/services-networking/service/`, 2025. Consulta realizada el 21 de julio de 2025.

[37] Kubernetes. Volumes. `https://kubernetes.io/docs/concepts/storage/volumes/`, 2025. Consulta realizada el 21 de julio de 2025.

[38] Jihwan Lee, Dong-min Kim, Hyung-Yoon Ryoo, and Byung-Soo Shin. Sustainable Wearables: Wearable Technology for Enhancing the Quality of Human Life. *Sustainability*, 8(5):466, 2016.

[39] Domingo López. HDRescuer: Herramienta para la recuperación de datos. `https://github.com/DomingoLopez/HDRescuer`, 2022. Accedido: 20 de febrero de 2025.

[40] Somayya Madakam, R. Ramaswamy, and S. Tripathi. Internet of Things (IoT): A Literature Review. *Journal of Computer and Communications*, 3(5):164–173, 2015.

[41] Manifest Infotech. Vue.js: A powerful Fronted Technology for modern web development. `https://manifestinfotech.com/blog/vue-js-a-powerful-frontend-technology-for-modern-web-development/`, 2025. Consulta realizada el 22 de julio de 2025.

[42] Medium Multimedia. Por qué usar Java para backend. `https://www.mediummultimedia.com/web/por-que-usar-java-para-backend/`, 2023. Consulta realizada el 22 de julio de 2025.

[43] MetricFire. Prometheus vs Zabbix. `https://www.metricfire.com/blog/prometheus-vs-zabbix/`, 2025. Consulta realizada el 21 de julio de 2025.

[44] MilesWeb. Docker vs. RKT: A Comparison. `https://www.milesweb.com/blog/technology-hub/docker-vs-rkt/`, 2024. Consulta realizada el 21 de julio de 2025.

[45] Mobvoi. TicWatch Pro 3. `https://www.mobvoi.com/sp/pages/ticwatchpro3gps`, 2025. Consulta realizada el 22 de julio de 2025.

[46] Alessio Morichetta, Victor C. Pujol, and Schahram Dustdar. A roadmap on learning and reasoning for distributed computing continuum ecosystems. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 25–31. IEEE, 2021.

[47] Postman. Página oficial de Postman. `https://www.postman.com/`, 2025. Consulta realizada el 22 de julio de 2025.

[48] Prometheus. Página oficial de Prometheus. `https://prometheus.io/`, 2025. Consulta realizada el 21 de julio de 2025.

[49] Victor C. Pujol, Pradeep K. Donta, Alessio Morichetta, Ilir Murturi, and Schahram Dustdar. Edge Intelligence–Research Opportunities for Distributed Computing Continuum Systems. *IEEE Internet Computing*, 27(4):53–74, 2023.

[50] Victor C. Pujol, Peter Raith, and Schahram Dustdar. Towards a new paradigm for managing computing continuum applications. In *2021 IEEE Third International Conference on Cognitive Machine Intelligence (CogMI)*, pages 180–188. IEEE, 2021.

[51] Tayyab Qayyum, Ahsan Waheed Malik, Muhammad Asif Khan Khattak, Osman Khalid, and Shahzad Umair Khan. FogNetSim++: A Toolkit for Modeling and Simulation of Distributed Fog Environment. *IEEE Access*, 6:63570–63583, 2018.

[52] ReactJS. Introducción a JSX. https://es.legacy.reactjs.org/docs/introducing-jsx.html, 2025. Consulta realizada el 22 de julio de 2025.

[53] Squadcast. Prometheus vs. Zabbix: A Comprehensive Comparison of Monitoring Tools. https://www.squadcast.com/compare/prometheus-vs-zabbix-a-comprehensive-comparison-of-monitoring-tools, 2025. Consulta realizada el 21 de julio de 2025.

[54] Squadcast. Zabbix vs. Grafana: A Comprehensive Comparison. https://www.squadcast.com/compare/zabbix-vs-grafana-a-comprehensive-comparison, 2025. Consulta realizada el 21 de julio de 2025.

[55] StackShare. Docker vs. RKT. https://stackshare.io/stackups/docker-vs-rkt, 2019. Consulta realizada el 21 de julio de 2025.

[56] Technoscriber. Python vs. Java: A Comprehensive Comparison. https://www.technoscriber.com/python-vs-java-a-comprehensive-comparison/, 2025. Consulta realizada el 22 de julio de 2025.

[57] TechRepublic. Prometheus vs. Zabbix. https://www.techrepublic.com/article/prometheus-vs-zabix/, 2022. Consulta realizada el 21 de julio de 2025.

[58] Tecla. Why React Continues to Lead Front-End Development in 2025. https://www.tecla.io/blog/react-leads-front-end-development-in-2025, 2025. Consulta realizada el 22 de julio de 2025.

[59] The Linux Foundation. Opportunities and Challenges in Edge Computing under Kubernetes. https://training.linuxfoundation.org/blog/opportunities-and-challenges-in-edge-computing-under-kubernetes/, 2022. Consulta realizada el 21 de julio de 2025.

[60] Emanuele Tinto and Tullio Vardanega. A runtime infrastructure for the Continuum of Computing. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24)*, pages 401–404. Association for Computing Machinery, 2024.

[61] Unite.AI. 10 mejores bibliotecas de Python para aprendizaje automático e IA. https://www.unite.ai/es/10-mejores-bibliotecas-de-python-para-aprendizaje-autom%C3%A1tico-ai/, 2025. Consulta realizada el 22 de julio de 2025.

[62] UpGuard. Docker vs. CoreOS. https://www.upguard.com/blog/docker-vs-coreos, 2024. Consulta realizada el 21 de julio de 2025.

[63] Vue.js. Página oficial de Vue.js. https://vuejs.org/, 2025. Consulta realizada el 22 de julio de 2025.

[64] Li Wang, Jianfeng Tao, Matthias Kunze, Alexandra Castellanos, David Kramer, and Wolfgang Karl. Scientific Atlanta, Cloud Computing: Early Definition and Experience. In *2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 825–830. IEEE, 2008.

[65] Danny Weyns. *An Introduction to Self-Adaptive Systems: A Contemporary Software Engineering Perspective*. Wiley, 2020. Chapters 1-4.

[66] Zabbix. Página oficial de Zabbix. https://www.zabbix.com/, 2025. Consulta realizada el 21 de julio de 2025.

Apéndice A

Exploración de Escenarios de Despliegue

A.1. Implementación Simulada VS Implementación Física

La complejidad inherente a la Continuidad Computacional plantea desafíos significativos en el desarrollo y despliegue de sistemas distribuidos. Uno de los dilemas centrales que enfrentamos en este trabajo es la decisión entre implementar físicamente el sistema o recurrir a simulaciones para su validación y prueba. Ambos enfoques presentan un conjunto único de ventajas y desventajas que deben ser cuidadosamente consideradas.

A.1.1. Implementación Simulada

Ventajas:

- Proporcionan un entorno controlado y adaptable, facilitando la experimentación y la optimización del sistema antes de su despliegue físico.
- Rapidez en la iteración: Es posible probar y ajustar algoritmos y políticas sin necesidad de configurar hardware físico.
- Reproducibilidad: Los escenarios simulados permiten repetir pruebas bajo condiciones controladas para comparar resultados de manera objetiva.

Desventajas:

- Limitación en el realismo: Los simuladores se centran en ciertos aspectos técnicos pero no pueden abarcar toda la heterogeneidad de un entorno físico que integra IoT, computación de borde, niebla y nube.
- Integración de componentes: Es difícil encontrar o integrar un simulador único que cubra la totalidad de la Continuidad Computacional. Por ello es necesario combinar varios simuladores, añadiendo complejidad al planteamiento del sistema.

- Flexibilidad: La simulación, aunque es poderosa para el análisis, puede limitar la experimentación con aspectos prácticos como la configuración de red reales y problemas de hardware.

A.1.2. Implementación Física

Ventajas:

- Realismo: Permite probar el sistema en condiciones reales, enfrentándose a problemas reales de comunicación, latencia, interferencias y gestión de recursos.
- Flexibilidad: Se puede ajustar y experimentar con distintos hardware, protocolos y configuraciones sin depender de las limitaciones de un simulador.
- Validación práctica: Es más fácil evidenciar la viabilidad y efectividad de la solución.

Desventajas:

- Costos y tiempo: Requiere inversión en hardware y tiempo en el montaje, configuración y mantenimiento de los dispositivos.
- Escalabilidad limitada: Experimentar a gran escala (múltiples nodos, escenarios de fallo) puede ser más costoso y logísticamente complejo.

A.1.3. Conclusión

- **Simulación**: Es ideal para análisis teóricos, validación de modelos y escalabilidad en escenarios controlados, pero puede no capturar toda la complejidad y variabilidad de un entorno real.
- **Implementación Física**: Aunque más costosa, ofrece resultados más realistas y flexibilidad para experimentar con configuraciones y problemas prácticos.

Dada la complejidad del sistema, lo ideal sería hacer una implementación física respaldandose, siempre que sea posible, en planteamientos testeados en entornos simulados. De esta forma obtendremos:

- Contrastes entre los resultados obtenidos en la simulación con los del entorno físico.
- Desarrollo incremental: desarrollo y corrección de algoritmos de gestión de recursos, auto-adaptación y comunicación en un entorno simulado.
- Reducción de Costos y Riesgos.

No obstante, teniendo en cuenta las limitaciones temporales del proyecto, se optará finalmente por una implementación física directa, priorizando la obtención de resultados concretos en condiciones reales.

A.2. Enfoque del Funcionamiento del Sistema

Durante la fase de diseño se identificó la necesidad de definir cómo se gestionaría el aprendizaje automático en el contexto de la Continuidad Computacional. En este sentido, se debatieron dos posibles enfoques para distribuir la carga de procesamiento en el sistema:

- **Enfoque Tradicional:** Este modelo consiste en centralizar el proceso de aprendizaje automático en la nube. La mayor parte del procesamiento se asigna a nodos de alta potencia, que reproducen el aprendizaje de máquina de manera centralizada. Aunque este enfoque ha sido el predominante durante años, presenta importantes desventajas, tales como la alta latencia en la transmisión de datos, la dependencia de conexiones robustas y la posible saturación de recursos en la nube. Además, al concentrar la carga en pocos nodos, se generan cuellos de botella que pueden afectar la escalabilidad y la eficiencia del sistema.
- **Enfoque Distribuido:** El enfoque distribuido propone asignar tareas de aprendizaje a múltiples nodos ubicados en diferentes niveles del continuum. Cada nodo no solo procesa, sino que también filtra y pre procesa la información antes de enviarla a la nube para un análisis final. Este método permite reducir la latencia y optimizar el uso del ancho de banda, ya que los datos son tratados de forma local y solo se transmiten los resultados ya refinados. Asimismo, la distribución de la carga mejora la resiliencia del sistema y facilita una adaptación dinámica a cambios en el entorno.

A.2.1. Conclusión

Tras analizar ambos enfoques, se optó por el enfoque distribuido. Esta decisión se fundamenta en que dicho modelo se alinea con las tendencias actuales en sistemas distribuidos y ofrece ventajas significativas en términos de eficiencia, escalabilidad y capacidad de respuesta en tiempo real. Al permitir que los nodos locales realicen un preprocesamiento inteligente, se optimiza el filtrado de datos y se reduce la sobrecarga en la nube, abordando de forma efectiva los desafíos de la Continuidad Computacional.

Apéndice B

Guía de Despliegue del Sistema

El despliegue de la infraestructura se divide en dos fases principales: la instalación de la aplicación en el dispositivo del usuario y el despliegue de la arquitectura de microservicios en el clúster K3s.

B.1. Despliegue de la Aplicación de Recolección de Datos

Esta sección describe el proceso para preparar y desplegar la aplicación móvil encargada de la recolección de datos de los usuarios.

Para instalar la aplicación móvil en un dispositivo Android, este debe tener Android 8.0 o superior y disponer de Android Studio. Basta con descargar el proyecto, abrir Android Studio, importar la carpeta HDRescuer desde File - New - Import Project, compilarlo y conectar el dispositivo al ordenador. Una vez conectado, se puede ejecutar el proyecto para instalar la app en el teléfono.

Para iniciar el sistema, es necesario contar con la dirección del nodo principal donde se coordina el despliegue de los microservicios. Esta dirección debe estar correctamente configurada en los archivos de comunicación interna, como gateway.config.yml, common/endpoints y common/Constants, para asegurar la conectividad entre servicios. Una vez ajustada la configuración, basta con ejecutar el comando *sh start.sh* para lanzar los contenedores necesarios.

B.2. Despliegue de la Infraestructura en K3s

Esta sección guía paso a paso el despliegue de toda la arquitectura de microservicios en un clúster K3s, asumiendo que el clúster está completamente vacío.

B.2.1. Instalación de K3s

Para configurar el clúster, se debe instalar K3s en cada máquina, diferenciando entre el nodo principal (manager) y los nodos secundarios (agentes).

Instalación del Nodo Principal (Manager)

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

Este comando instala K3s y configura el nodo como el servidor principal.

Instalación de Nodos Secundarios (Agentes)
Primero, se debe obtener el token de conexión del nodo manager ejecutando en el manager:

```
sudo cat /var/lib/rancher/k3s/server/node-token
```

Luego, en cada nodo agente, se ejecuta el comando de instalación, reemplazando la IP y el token por los valores correctos:

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

Una vez finalizada la instalación en todos los nodos, es posible verificar la conexión entre equipos.

B.2.2. Verificación del Clúster

Desde el nodo manager, se puede verificar que todos los nodos están conectados y listos para trabajar.

```
sudo kubectl get nodes
```

El resultado de este comando debe mostrar todos los nodos con el estado ‘Ready‘. Un ejemplo de resultado es el que se ve en la imagen B.1. En este caso lo he creado con mi portátil ASUS y una raspberry pi 5. Pero voy a añadir otro portátil para tener un clúster con 3 equipos.

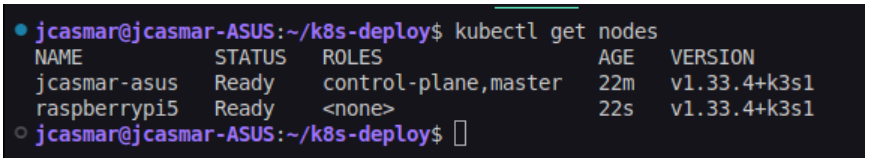


Figura B.1: Verificación de los nodos del clúster

B.2.3. Despliegue de la Arquitectura de Microservicios

Todos los archivos de manifiesto (‘.yaml‘) necesarios para el despliegue se encuentran en el repositorio de GitHub del proyecto. El primer paso es clonarlo en

una máquina con acceso al clúster.

Paso 1: Clonar el Repositorio de Código

```
git clone https://github.com/jmcasanovam/computing_continuum.git
cd computing_continuum/k8s-deploy
```

Paso 2: Creación de Namespaces

Se crean los ‘namespaces‘ ‘core‘ y ‘monitoring‘ para organizar los recursos.

```
kubectl apply -f monitoring/namespace.yaml
kubectl apply -f core/namespace.yaml
```

Verificación:

```
kubectl get namespaces
```

Debemos conseguir algo similar a lo que se ve en la imagen B.2.

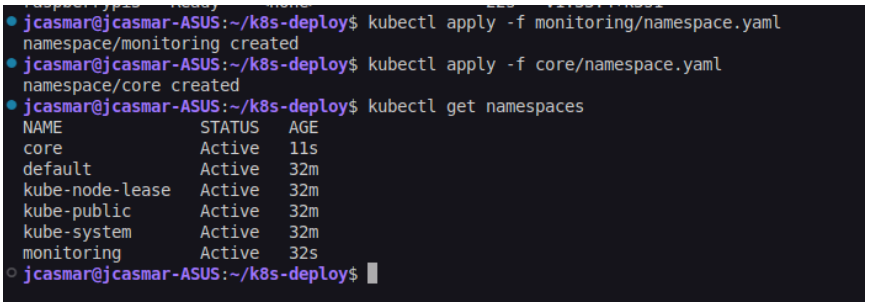


Figura B.2: Verificación de los namespaces

Paso 3: Despliegue del Stack de Monitorización

Se despliegan los servicios de Prometheus, Node-Exporter y Grafana en el namespace ‘monitoring‘.

Node Exporter

```
kubectl apply -f monitoring/node-exporter/
```

Verificación de que Node Exporter está corriendo en todos los nodos:

```
kubectl -n monitoring get daemonset node-exporter
kubectl -n monitoring get pods -l app=node-exporter
```

Debemos conseguir algo similar a lo que se ve en la imagen **??**, donde podemos ver como Node Exporter está corriendo en los 3 nodos del clúster.

Prometheus

```
kubectl apply -f monitoring/prometheus/
```

Verificación de que Prometheus está corriendo correctamente:

```
kubectl -n monitoring get pods -l app=prometheus
```

Debemos conseguir algo similar a lo que se ve en la imagen B.3, donde podemos ver que Prometheus está corriendo en un nodo, en el manager. De igual forma, podemos confirmar que está corriendo navegando a la URL http://IP_MANAGER:30090

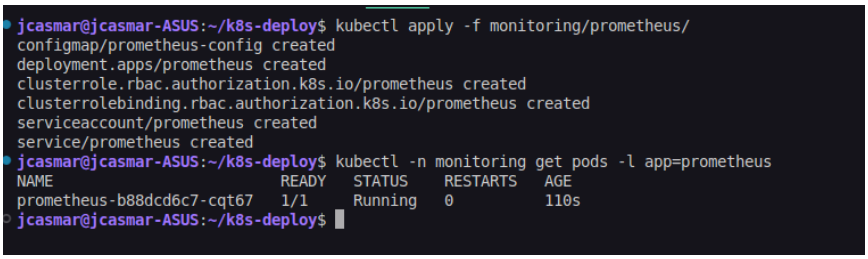


Figura B.3: Verificación de Prometheus

Grafana

```
kubectl apply -f monitoring/grafana/
```

Verificación de que Grafana está corriendo correctamente:

```
kubectl -n monitoring get deployment grafana
kubectl -n monitoring get pods -l app=grafana
kubectl -n monitoring get svc grafana
```

Debemos conseguir algo similar a lo que se ve en la imagen B.4, donde podemos ver que Grafana está corriendo en un nodo correctamente y aparece el mapeo de puertos. En este caso Grafana usa el puerto 30300. De igual forma, podemos confirmar que está corriendo navegando a la URL http://IP_MANAGER:30300.

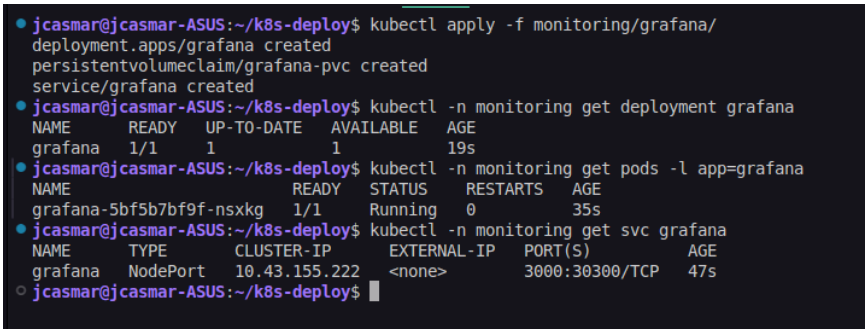


Figura B.4: Verificación de Grafana

Si queremos utilizar la interfaz de Grafana, debemos de configurarla:

1. Conectar a Prometheus como Data Source

- En Grafana, ve a Configuration / Data Sources.
- Haz clic en Add data source y selecciona Prometheus.
- En el campo URL, introduce http://prometheus.monitoring.svc.cluster.local:9090
- Haz clic en Save & Test.

2. Agregar un nuevo Dashboard:

- En la barra de búsqueda consultar Ïmport Dashboard".
- Sugiero usar el Dashboard con ID 1860.
- Usar la fuente de datos creada antes.

3. Acceder al Dashboard

Debemos conseguir algo similar a lo que se ve en la imagen B.5. Dentro de "nodename"podremos navegar entre los distintos nodos del sistema. La información se irá rellenando y actualizando automáticamente.

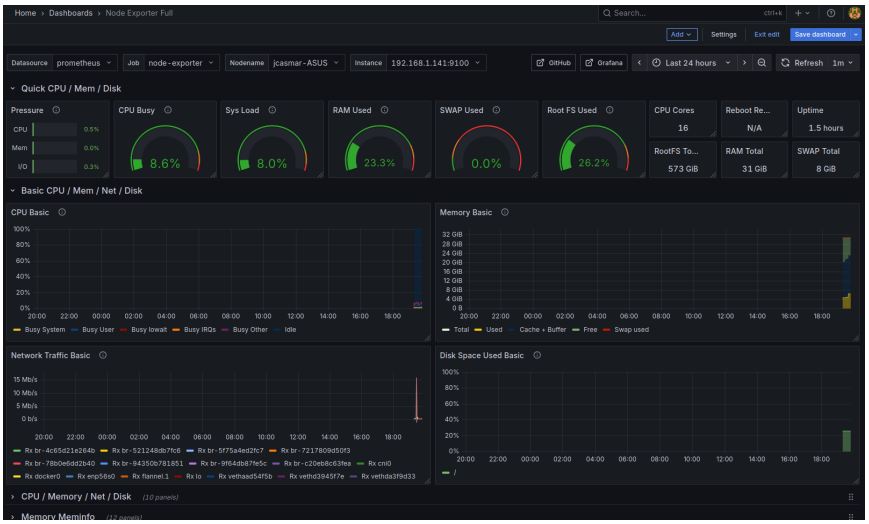


Figura B.5: Dashboard en Grafana

Paso 4: Despliegue de Servicios de Soporte (DB y RabbitMQ)

Se despliegan los servicios de base de datos y mensajería, que son dependencias críticas para el resto de la arquitectura. Como para que se asigne al nodo que yo quiero la base de datos hay que crear un rol para esto, lo primero que hay que hacer es asignar este rol al nod y luego desplegar todo.

```
kubectl label node jcasmar-asus role=db-node
```

Despliegue de db

1. Entrar en core/db y desplegar los componentes de almacenamiento y configuración

```
kubectl apply -f pvc.yaml
kubectl apply -f secret.yaml
kubectl apply -f configmap.yaml
```

2. Desplegar el servicio y el pod

```
kubectl apply -f service.yaml
kubectl apply -f deployment.yaml
```

Verificación

```
kubectl get pvc -n core
kubectl get configmap,secret -n core
kubectl get pods -n core
kubectl get service -n core
```

Debemos conseguir algo similar a lo que se ve en la imagen B.6. En la imagen podemos ver como se ha creado un volumen persistente, se le ha asignado a un nodo con el rol db-node y está corriendo correctamente.

```
jcasmar@jcasmar-ASUS:~/k8s-deploy$ kubectl label node jcasmar-asus role=db-node
node/jcasmar-asus labeled
jcasmar@jcasmar-ASUS:~/k8s-deploy$ cd core/db
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f pvc.yaml
persistentvolumeclaim/db-pvc created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f secret.yaml
secret/postgres-secret created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f configmap.yaml
configmap/postgres-config created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f deployment.yaml
deployment.apps/db created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl apply -f service.yaml
service/db created
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get pods -n core
NAME                  READY   STATUS    RESTARTS   AGE
db-85c6976858-jdrr2   1/1     Running   0          78s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get service -n core
NAME   TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
db     ClusterIP   10.43.167.58   <none>        5432/TCP   28s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$ kubectl get pvc -n core
NAME      STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
db-pvc    Bound    pvc-c8864203-2139-4682-886c-3fa6260f3e0f   5Gi        RWO            local-path     <unset>                 2m20s
jcasmar@jcasmar-ASUS:~/k8s-deploy/core/db$
```

Figura B.6: Verificación del despliegue de db

Despliegue de RabbitMQ

```
cd core
kubectl apply -f rabbitmq
```

Verificación

```
kubectl -n core get pods -l app=rabbitmq
```

Paso 5: Despliegue de los Microservicios de la Aplicación

Se despliegan los microservicios restantes que componen la lógica de la aplicación. Recomiendo ir lanzando uno a uno y entre medias ejecutar el comando de verificación y revisar los logs para saber si ya han terminado de correr los primeros pasos, ya que por ejemplo cloud-trainer intentará crear el modelo genérico inicial ni no existe todavía.

Para el edge_service es necesario decir que nodos pueden ser seleccionados. Esto se hace con el comando:

```
kubectl label nodes <nombre-del-nodo> edge-service=true
```

Adicionalmente, para la simulación, se debe asignar una etiqueta que indique la localización lógica del nodo dentro del paradigma de Continuidad Computacional (edge, fog, cloud). Esto se hace con el siguiente comando:

```
kubectl label node <nombre-del-nodo> node-type=<tipo>
```

Una vez configuradas las etiquetas, se procede al despliegue de los microservicios necesarios para el sistema:

```
kubectl apply -f ./core/cloud_trainer
kubectl apply -f ./core/cloud_api
kubectl apply -f ./core/data_ingestor
kubectl apply -f ./core/fog_trainer
kubectl apply -f ./core/edge_service
```

Para revisar los logs de un pod:

1. Consultar el nombre del pod con:

   ```
   kubectl get pods -n core
   ```

2. A partir del nombre conseguido, ejecutar:

   ```
   kubectl logs -f NOMBRE -n core
   ```

Verificación

```
kubectl get pods -n core
```

Paso 6: Despliegue del Servicio ‘manager-mapek‘

Finalmente, se despliega el ‘manager-mapek‘, ya que depende de la infraestructura de monitorización y de los servicios a gestionar.

```
kubectl apply -f core/manager-mapek
```

Verificación

```
kubectl get pods -n core
```

Si queremos ver en qué nodo está corriendo cada pod, podemos usar el comando:

```
kubectl get pods -n core -o wide
```

Una vez que todos los pods muestren el estado ‘Running‘, la arquitectura estará completamente desplegada y funcional.