Diseño de una Metodología de Diagnóstico Proactivo para Incidentes que Afectan la Disponibilidad de Aplicaciones Empresariales

Autor
Andres Felipe Acevedo Monroy

Director
Jorge Ivan Romero Gelvez

Co-Director
Mauricio Garcés Restrepo

Universidad de Bogotá Jorge Tadeo Lozano
Facultad de Ciencias Naturales e Ingeniería
Especialización en Desarrollo de Bases de Datos

Bogotá - Colombia, noviembre de 2025

Índice

Página

Resumen v

Abstract vi

Glosario vii

1. Introducción 1

2. Descripción del Problema 2

3. Objetivos 3
3.1. Objetivo General . . . 3
3.2. Objetivos Específicos . . . 3

4. Requerimientos 4
4.1. Requerimientos de Negocio . . . 4
4.2. Requerimientos Funcionales . . . 4
4.3. Requerimientos de los Usuarios . . . 4
4.4. Requerimientos de Implementación . . . 4
4.5. Requerimientos de Calidad . . . 5

5. Estado del Arte 6

6. Marco Teórico 8
6.1. Disponibilidad de Aplicaciones Empresariales . . . 8
6.2. Alta Disponibilidad en Sistemas Empresariales . . . 8
6.3. Degradación del Desempeño en Servidores . . . 8
6.3.1. Degradación por memoria . . . 8
6.3.2. Degradación por CPU . . . 9
6.3.3. Degradación por procesos o hilos . . . 9
6.3.4. Degradación por configuración . . . 9
6.3.5. Degradación en bases de datos . . . 9
6.4. Diagnóstico de Incidentes y Detección de Anomalías . . . 9
6.5. Ingeniería de Confiabilidad de Sitios (SRE) . . . 9
6.6. Entornos Tecnológicos Relevantes . . . 10
6.6.1. Linux . . . 10
6.6.2. Windows Server / .NET . . . 10
6.6.3. Servidores de Aplicaciones Java . . . 10
6.6.4. Bases de Datos Oracle y SQL Server . . . 10
6.7. Herramientas Nativas para el Diagnóstico Proactivo . . . 10

7. Solución Propuesta **11**
7.1. Descripción general de la solución 11
7.2. Modelo conceptual 11
7.3. Estándares de la solución 13
7.4. Condiciones de diseño, propuesta de implementación y evaluación 13
7.4.1. Taxonomía de incidentes de degradación 14
7.4.2. Matriz incidente–métrica–herramienta 14
7.4.3. Checklist de diagnóstico proactivo 14
7.4.4. Flujograma general del diagnóstico 16
7.4.5. Criterios para decidir si reiniciar un servicio 17
7.4.6. Propuesta de evaluación 17
7.4.7. Consideraciones finales 17
7.4.8. Integración emergente de Inteligencia Artificial en el diagnóstico proactivo 17

8. Planeación del Trabajo **20**
8.1. Descomposición de actividades WBS 20
8.2. Tabla de actividades 20
8.3. Diagrama de Gantt 21

9. Presupuesto **23**

10. Conclusiones **24**

Referencias **26**

Índice de figuras

1. Diagrama de bloques propuesto para la metodología de diagnóstico proactivo. . 11
2. Recolección de métricas de memoria con el comando free -h 12
3. Recolección de métricas de Disco / I/O con el comando df -h. 12
4. Se identifican umbrales superados, en este caso puntos de montaje con poco espacio disponible. 12
5. Como ejemplo, se aprovisiona nuevo espacio para que no supere el umbral recomendado ante imposibilidad de borrar archivos o logs. 13
6. Flujograma general del diagnóstico proactivo. 16
7. Arquitectura moderna de observabilidad basada en la propuesta de Shah [17] eBPF con integración opcional de modelos de IA para análisis interpretativo. . 18
8. Work Breakdown Structure para la implementación de la metodología de diagnóstico proactivo 20
9. Diagrama de Gantt. 22

Índice de tablas

1. Taxonomía de incidentes de degradación en aplicaciones empresariales 14
2. Matriz incidente–métrica–herramienta de diagnóstico 14
3. Tabla de actividades del proyecto 21
4. Presupuesto estimado para la implementación institucional de la metodología de diagnóstico proactivo. 23
5. Tabla de gastos resumidos del proyecto (valores coherentes con el presupuesto detallado). 23

Resumen

Este trabajo de grado se centra en el diseño de una metodología de diagnóstico proactivo para la gestión de incidentes que afectan la disponibilidad de aplicaciones empresariales desplegadas en entornos Oracle/Linux y SQL Server/Windows, tanto *on-premise* como en la nube. El objetivo principal es anticipar y detectar condiciones de degradación que comúnmente se resuelven mediante reinicios de servidor o servicios de aplicación, una acción reactiva que no aborda la causa raíz del problema.

El análisis técnico abarca la identificación de causas recurrentes de degradación, tales como: memoria (fugas, bloqueos de *heap*, saturación de *swap*); procesos (hilos colgados o consumo excesivo de recursos); CPU (sobreutilización y cargas sostenidas); configuración (parámetros inadecuados en aplicaciones o *middleware*); y base de datos (agotamiento de *pools* de conexiones e impacto en Oracle/SQL Server).

Se busca establecer una metodología que permita a los administradores anticipar incidentes mediante la evaluación de herramientas nativas disponibles en los sistemas (por ejemplo, `sar`, `journalctl`, *logs* de GC, AWR/ADDM de Oracle, PerfMon en .NET) y su integración en una guía práctica.

La propuesta metodológica se evaluará considerando su utilidad en tres componentes principales: 1) una taxonomía de incidentes de degradación; 2) la comparación funcional de herramientas nativas de diagnóstico; y 3) una guía en forma de *checklist* y flujogramas de decisión para actuar antes de recurrir a reinicios.

Abstract

This thesis proposes the design of a proactive diagnostic methodology aimed at managing incidents that affect the availability of enterprise applications deployed in Oracle/Linux and SQL Server/Windows environments, both on-premise and in the cloud. The main goal is to anticipate and detect degradation conditions that, in operational practice, are frequently mitigated through server or application service restarts—reactive actions that temporarily restore functionality but do not address the underlying root cause.

The analysis focuses on the most recurrent sources of degradation in Java EE and .NET applications, including memory leaks, heap blocking, swap saturation, hung processes, sustained CPU overutilization, misconfigurations at the application server or middleware level, and exhaustion of connection pools in Oracle and SQL Server databases.

Additionally, the study evaluates the diagnostic potential of native tools available in these environments—such as `sar`, `journalctl`, garbage collection logs, Oracle AWR/ADDM reports, and Windows PerfMon—highlighting their usefulness in detecting early deterioration before it escalates into critical failures.

The proposed methodology is structured into three components: (1) a taxonomy of degradation incidents derived from operational evidence and prior research; (2) a functional assessment of native diagnostic tools present in the evaluated environments; and (3) a practical guide composed of checklists and decision flowcharts designed to facilitate early detection and reduce the reliance on restarts as a recovery strategy.

The results aim to provide system and database administrators with a practical and academically grounded framework to transition from reactive operations toward proactive reliability practices aligned with modern availability and service continuity standards.

Glosario

Alta Disponibilidad (HA) Conjunto de prácticas, arquitecturas y mecanismos orientados a minimizar el tiempo de inactividad y asegurar la continuidad de un servicio.

Checklist Lista estructurada de verificación que permite validar de forma sistemática el estado de componentes críticos.

Deadlock Situación donde dos procesos o hilos se bloquean mutuamente al esperar recursos que el otro posee.

Enfoque Proactivo Estrategia operativa centrada en identificar y resolver problemas antes de que afecten la disponibilidad del sistema.

Flujograma Representación gráfica del proceso de diagnóstico mediante pasos, decisiones y acciones asociadas.

Fuga de Memoria Condición donde un programa retiene memoria innecesariamente, aumentando el consumo hasta causar degradación o fallo.

Latencia Tiempo de respuesta de un sistema ante una operación. Su incremento suele ser un indicador temprano de degradación.

Metodología Conjunto estructurado de pasos, técnicas y herramientas que permiten resolver un problema de manera ordenada y verificable.

Observabilidad Capacidad de un sistema para exponer información interna mediante logs, métricas y trazas, permitiendo comprender su estado sin necesidad de instrumentación adicional.

Pool de Conexiones Conjunto gestionado de conexiones a base de datos en aplicaciones para mejorar rendimiento y reutilización. Su agotamiento produce lentitud o fallos.

SRE Site Reliability Engineering. Disciplina que combina principios de software e ingeniería de sistemas para garantizar disponibilidad, rendimiento y confiabilidad en plataformas tecnológicas..

Swap Memoria virtual del sistema utilizada cuando la RAM se agota. Su uso excesivo es un indicador de degradación o fuga de memoria.

Taxonomía Clasificación sistemática de incidentes según su naturaleza, síntomas y métricas asociadas, utilizada para el diagnóstico proactivo.

Thread (Hilo) Unidad de ejecución dentro de un proceso. Su mal manejo puede causar bloqueos, saturación del CPU o deadlocks.

Throughput Cantidad de operaciones procesadas por un sistema en un intervalo de tiempo. Su caída suele indicar degradación.

Wait Events Eventos que indican qué está esperando un proceso de base de datos y dónde se encuentran los cuellos de botella.

Zero Downtime Enfoque de despliegue que garantiza actualizaciones sin interrupción del servicio, mediante técnicas como blue-green o canary releases.

1. Introducción

La disponibilidad de aplicaciones empresariales soportadas sobre bases de datos Oracle y sistemas operativos Linux/Windows constituye un factor crítico para la continuidad de los procesos organizacionales [1]. Sin embargo, estos entornos suelen enfrentar incidentes de degradación derivados de fenómenos como fugas de memoria, bloqueos de procesos o saturación de recursos. En la práctica, tales incidentes se resuelven habitualmente mediante reinicios de servidores o servicios de aplicación, una acción reactiva que permite recuperar la operación, pero que no soluciona la causa raíz ni previene su recurrencia.

Diversas investigaciones han evidenciado la importancia de anticipar este tipo de degradaciones mediante enfoques de detección de anomalías. Trabajos recientes han demostrado la utilidad de métricas de CPU y memoria para detectar condiciones anómalas, proponiendo estrategias estadísticas y de *machine learning* con resultados alentadores para la identificación temprana de fugas de memoria y comportamientos anormales en servidores [2]. En paralelo, estudios sobre servidores de aplicaciones Java han señalado que la recolección de basura (*garbage collection*) representa un "talón de Aquiles" para la sostenibilidad del rendimiento, al generar caídas abruptas de *throughput* cuando no se gestiona adecuadamente [3].

El problema no se limita a un lenguaje o plataforma: comparaciones de arquitecturas .NET y Java EE han mostrado diferencias significativas en la forma en que las aplicaciones responden a cargas y degradaciones [4]. Además, en infraestructuras de gran escala, la necesidad de mantener Alta Disponibilidad (HA) ha derivado en prácticas como despliegues sin interrupción (*zero downtime release*), que buscan minimizar el impacto de reinicios y actualizaciones frecuentes en clústeres de producción [5].

La literatura también ha explorado fenómenos como el *software aging*, evidenciando cómo servidores web o de aplicaciones pierden confiabilidad con el tiempo, aumentando la probabilidad de fallas si no se aplican prácticas de mantenimiento preventivo [6]. En el ámbito organizacional, investigaciones sobre *Site Reliability Engineering* (SRE) resaltan la necesidad de integrar prácticas proactivas de monitoreo, respuesta a incidentes y planeación de capacidad para garantizar continuidad del servicio [7].

En este contexto, se justifica la necesidad de diseñar una Metodología de diagnóstico proactivo que permita anticipar incidentes de degradación en entornos empresariales con aplicaciones Java/.NET sobre Oracle/Linux. Esta propuesta se estructura en tres componentes: (i) una taxonomía de incidentes de degradación observados en servidores de aplicaciones y bases de datos; (ii) la evaluación de herramientas nativas de diagnóstico disponibles en sistemas operativos y motores de base de datos (`sar`, `journalctl`, `GC logs`, `AWR/ADDM`, `PerfMon`); y (iii) la elaboración de una guía práctica de diagnóstico en forma de *checklist* y flujogramas de decisión.

El resultado esperado es ofrecer a administradores de sistemas y bases de datos un marco práctico y académico que contribuya a la transición de un enfoque reactivo —basado en reinicios— hacia un enfoque proactivo, alineado con las tendencias modernas de confiabilidad y continuidad de servicio.

2. Descripción del Problema

La disponibilidad de las aplicaciones empresariales que operan sobre Oracle/Linux (y, en muchos casos, SQL Server/Windows) es crítica para la continuidad del negocio. Sin embargo, en la operación diaria aparecen incidentes de degradación (p. ej., fugas de memoria, bloqueos de hilos/procesos, saturación de CPU o memoria, agotamiento de *pools* de conexiones, configuraciones defectuosas) que afectan el desempeño y la estabilidad del servicio. Ante estos eventos, la práctica común es ejecutar acciones reactivas —como reinicios de servicios o de servidores— que restablecen temporalmente la operación pero no abordan la causa raíz, propiciando la recurrencia de fallas y aumentando la exposición a indisponibilidad.

El desafío se agrava por la complejidad y heterogeneidad de los entornos: capas de aplicación Java/.NET y *middleware*, sistemas operativos, redes, balanceadores y bases de datos que interactúan en infraestructuras híbridas (*on-premise* y nube). En estos escenarios, distinguir con rapidez si el origen del problema está en aplicación, sistema operativo, red o base de datos requiere evidencias diagnósticas oportunas y un proceso sistemático de decisión. Aunque existen herramientas de monitoreo y utilitarios nativos (p. ej., `sar`, `journalctl`/`dmesg`, `GC logs`/JVM, `AWR`/`ADDM`, `PerfMon`), su uso suele ser ad hoc, sin un protocolo unificado que indique qué medir, cómo interpretar y cuándo intervenir antes de recurrir a reinicios. Desde el punto de vista operativo y de negocio, esta situación conlleva consecuencias relevantes: 1) aumento del MTTR (tiempo medio de recuperación) y reincidencia de incidentes por ausencia de análisis causal; 2) riesgo de incumplimiento de SLO/SLA, afectación a la experiencia del usuario y costos operativos por paradas no planificadas; 3) pérdida de trazabilidad y oportunidades de mejora continua al carecer de criterios y *checklists* que estandaricen el diagnóstico previo a la acción; y 4) dificultad para priorizar acciones preventivas (capacidad, ajustes de configuración, afinamiento de GC/*pools*) al no contar con una taxonomía clara de degradaciones y sus indicadores.

Adicionalmente, la recomendación alinear prácticas con plataformas en soporte vigente (p. ej., Oracle Database 19c/21c, RHEL 8/9, Java 11/17), a fin de asegurar aplicabilidad y vigencia. No obstante, incluso en estos *stacks* soportados, los patrones de degradación persisten si no existe una metodología de diagnóstico proactivo que integre señales de cada capa y guíe decisiones (continuar observando, mitigar, ajustar configuración, escalar o reiniciar de forma controlada) con criterios repetibles.

En consecuencia, el problema central que aborda este trabajo es la ausencia de un marco práctico y estandarizado que: 1) clasifique los incidentes de degradación más frecuentes en aplicaciones empresariales sobre Oracle/Linux (y SQL Server/Windows); 2) operativice el uso de herramientas nativas ya disponibles para obtener evidencias mínimas y anticiparse a la indisponibilidad; y 3) estructure un protocolo de diagnóstico proactivo (*checklist* y flujos de decisión) que reduzca la dependencia de reinicios como medida reactiva, mejore la continuidad del servicio y facilite la mejora continua (*postmortems* y lecciones aprendidas).

3. Objetivos

3.1. Objetivo General

Diseñar una metodología de diagnóstico proactivo para la gestión de incidentes que afectan la disponibilidad de aplicaciones empresariales en entornos Oracle/Linux y SQL Server/Windows, con el fin de reducir la dependencia de acciones reactivas como reinicios de servidores o servicios y mejorar la continuidad operativa.

3.2. Objetivos Específicos

- Clasificar los principales incidentes de degradación que comprometen la disponibilidad de aplicaciones empresariales (Java/.NET) en entornos Oracle/Linux y SQL Server/Windows, mediante la elaboración de una taxonomía basada en literatura y casos reales.
- Identificar y analizar las herramientas nativas y ya disponibles en los sistemas (p. ej., `sar`, `journalctl`, *logs* de GC/JVM, `AWR`/`ADDM` en Oracle, `PerfMon` en Windows/.NET) evaluando su utilidad para la detección temprana de condiciones de degradación.
- Diseñar un protocolo metodológico de diagnóstico proactivo en forma de *checklist* y flujogramas de decisión, que guíe a administradores en la identificación de causas raíz antes de recurrir a reinicios.
- Validar la aplicabilidad de la metodología propuesta mediante el análisis de casos de incidentes reales, destacando su contribución para mejorar la disponibilidad y continuidad del servicio en entornos empresariales *legacy* y vigentes en soporte.

4. Requerimientos

4.1. Requerimientos de Negocio

La solución debe contribuir a la mejora de la disponibilidad de aplicaciones empresariales mediante la detección temprana de incidentes de degradación, reduciendo la dependencia de reinicios de servicios o servidores. Asimismo, debe facilitar la optimización de recursos operativos y generar insumos útiles para la toma de decisiones técnicas, evitando recurrir exclusivamente a herramientas comerciales de alto costo.

4.2. Requerimientos Funcionales

La metodología debe permitir:

- Clasificar los incidentes más frecuentes de degradación (fugas de memoria, saturación de CPU, procesos colgados, agotamiento de *pools* de conexiones, configuraciones inadecuadas).
- Identificar y utilizar herramientas nativas disponibles en los sistemas (p. ej., `sar`, `journalctl`, *logs* de GC/JVM, `AWR`/`ADDM` en Oracle, `PerfMon` en Windows/.NET) para obtener evidencias mínimas de diagnóstico.
- Proporcionar guías de diagnóstico en forma de *checklist* y flujogramas de decisión que orienten a los administradores en la búsqueda de causas raíz.
- Diferenciar si el origen del incidente corresponde a la capa de aplicación, sistema operativo, red o base de datos.

4.3. Requerimientos de los Usuarios

Los administradores de sistemas y bases de datos deben contar con:

- Un protocolo estandarizado y documentado que unifique criterios de diagnóstico previo a la acción de reinicio.
- *Checklists* y guías prácticas que faciliten el análisis rápido en situaciones críticas.
- La posibilidad de adaptar los flujos de decisión a las políticas y herramientas específicas de cada organización.

4.4. Requerimientos de Implementación

La propuesta metodológica debe:

- Presentarse en formato documentado y reutilizable, con ejemplos prácticos y referencias al estado del arte.
- Basarse en versiones soportadas de sistemas operativos (RHEL 8/9, Windows Server 2019/2022) y bases de datos (Oracle 19c/21c, SQL Server 2019/2022).
- Permitir su aplicación tanto en infraestructura *on-premise* como en entornos híbridos/nube sin necesidad de instalar software adicional.

4.5. Requerimientos de Calidad

La metodología debe:

- Ser clara, replicable y escalable a distintos entornos empresariales (con pocos o múltiples servidores).
- Mantener un enfoque proactivo, priorizando la prevención y anticipación de incidentes.
- Contribuir a la reducción del MTTR (tiempo medio de recuperación) y la disminución de reincidencias en incidentes de degradación.
- Garantizar la vigencia académica y práctica, apoyándose en el estado del arte y en casos reales de operación.

5. Estado del Arte

La disponibilidad de aplicaciones empresariales críticas depende tanto de la robustez de los sistemas de bases de datos como de la capacidad de anticipar fallos en los servidores de aplicación y la infraestructura asociada. Diversas investigaciones han abordado este problema desde enfoques complementarios, que van desde la alta disponibilidad en la nube hasta la detección temprana de anomalías y la adopción de prácticas modernas como la Ingeniería de Confiabilidad de Sitios (SRE).

En el ámbito de la alta disponibilidad en entornos *cloud*, Endo et al. [1] presentan una revisión sistemática de soluciones como redundancia, replicación, monitoreo y recuperación ante fallos, resaltando que alcanzar cinco nueves de disponibilidad (99.999 %) exige no solo infraestructura robusta, sino también procesos de detección y reacción oportunos.

En paralelo, Jäntti [2] demuestra que es posible detectar anomalías de rendimiento en servidores mediante técnicas estadísticas ligeras como *z-score* y regresión lineal, aplicadas a métricas de CPU y memoria.

Un aspecto crítico en los servidores Java EE es el impacto de la recolección de basura en el rendimiento. Xian et al. [3] identifican al *garbage collection* como un "talón de Aquiles" de los servidores Java, capaz de provocar degradaciones abruptas del *throughput*.

Desde la perspectiva de comparación tecnológica, Hamed y Kafri [4] muestran que la arquitectura Java EE puede superar en rendimiento a .NET bajo pruebas de carga controladas, lo cual subraya la relevancia de contar con métricas objetivas para anticipar cuellos de botella en ambos entornos.

El problema de los reinicios y actualizaciones sin interrupción también ha sido estudiado en infraestructuras masivas. Naseer et al. [5] proponen *Zero Downtime Release* en Facebook, mientras que Allam [8] analiza estrategias SRE como *blue-green deployments*, *canary releases* y *feature toggles* para garantizar despliegues sin *downtime*.

El modelado de disponibilidad en clústeres de servidores también ha sido explorado. Hunter y Smith [9] utilizan modelos de Markov en clústeres de dos nodos, y Handoko e Isa [10] prueban arquitecturas con MariaDB Galera Cluster, HAProxy y VRRP.

En cuanto a la persistencia de sesiones de base de datos, Barga y Lomet [11] introducen *PhoenidODBC*, un sistema que mantiene sesiones activas tras fallos del servidor.

El fenómeno del *software aging* ha sido abordado por Matias Jr. et al. [6], quienes comprobaron fugas de memoria en servidores Apache mediante modelos estadísticos, mientras que Anerousis et al. [12] proponen un sistema de monitoreo de salud para servidores de aplicaciones que detecta condiciones de degradación.

La relación entre disponibilidad de bases de datos y continuidad de negocio ha sido discutida por Tomić et al. [13], quienes comparan soluciones de Oracle, IBM y Microsoft, y por Shrestha [14], que analiza la replicación *Master-Slave* frente a *Multi-Master*.

Por último, investigaciones recientes han puesto el foco en la confiabilidad desde SRE y *cloud-native*. Allam [15] discute cómo aplicar SRE en arquitecturas *serverless*, Allam [8] en entornos IoT y *edge computing*.

De manera complementaria, el auge reciente de soluciones basadas en inteligencia artificial ha introducido nuevas alternativas para la detección automática de anomalías. Plataformas como *Hugging Face* y múltiples repositorios especializados en *GitHub* reúnen modelos orientados al análisis de logs, predicción de fallos, detección de *memory leaks* y clasificación automática de errores mediante redes neuronales recurrentes, *autoencoders* y modelos basados en *transfor-*

mers. Aunque estas herramientas presentan un alto potencial para automatizar el diagnóstico y reducir la intervención manual, su adopción en entornos empresariales *legacy* sigue siendo limitada debido a retos como la integrabilidad, la necesidad de entrenamiento con datos internos y los requisitos de seguridad. No obstante, representan una tendencia creciente que complementa las prácticas proactivas promovidas por SRE y abre la puerta a futuras metodologías híbridas de diagnóstico asistido por IA.

En síntesis, el estado del arte muestra que la problemática de la disponibilidad y el rendimiento de aplicaciones empresariales se ha abordado desde múltiples perspectivas: arquitecturas de alta disponibilidad, técnicas de diagnóstico de anomalías, estudios de degradación por *software aging* y GC, y enfoques organizacionales como SRE. Sin embargo, persiste una brecha en la documentación y sistematización de metodologías proactivas aplicables a entornos híbridos y *legacy* (Java EE, .NET, Oracle/Linux, SQL Server/Windows), donde los reinicios aún se emplean como respuesta reactiva frente a incidentes.

6. Marco Teórico

6.1. Disponibilidad de Aplicaciones Empresariales

La disponibilidad es una propiedad fundamental de los sistemas de información y se define como la probabilidad de que un servicio se encuentre operativo cuando es requerido. En entornos corporativos, una interrupción o degradación puede impactar significativamente la continuidad del servicio. Como señalan Endo et al. [1], mantener altos niveles de disponibilidad (por ejemplo, 99.999 %) requiere no solo mecanismos tradicionales de redundancia, sino también la capacidad de detectar fallos o degradaciones tempranas que preceden incidentes mayores.

La disponibilidad depende de la interacción entre múltiples capas tecnológicas: infraestructura, sistema operativo, servidores de aplicación y bases de datos. Por ello, el análisis de degradación no puede limitarse a uno solo de estos dominios.

6.2. Alta Disponibilidad en Sistemas Empresariales

La Alta Disponibilidad (HA) se basa en técnicas como redundancia, replicación, balanceo de carga y failover automático, cuyo objetivo es minimizar el tiempo de inactividad. No obstante, estudios recientes revelan que los mecanismos tradicionales de HA no siempre mitigan adecuadamente fallas de degradación progresiva. Endo et al. [1] resaltan que soluciones como checkpointing, replicación y load balancing deben complementarse con mecanismos de monitoreo proactivo.

Naseer et al. [5] muestran que incluso infraestructuras masivas, como las de Facebook, requieren estrategias avanzadas para evitar interrupciones durante reinicios programados o no programados, lo cual enfatiza la necesidad de procesos de diagnóstico anticipatorio.

6.3. Degradación del Desempeño en Servidores

La degradación del rendimiento ocurre cuando un sistema reduce gradualmente su capacidad de respuesta debido a fenómenos internos como fugas de memoria, saturación de CPU, bloqueos de hilos, configuraciones erróneas o problemas en la capa de base de datos. A continuación se describen sus principales causas.

6.3.1. Degradación por memoria

Las fugas de memoria y el uso anómalo del heap son responsables de un número considerable de incidentes. Jäntti [2] demuestra que métricas de CPU y memoria permiten detectar anomalías mediante técnicas estadísticas ligeras como *z-score*. En entornos Java EE, Xian et al. [3] identifican la recolección de basura (*garbage collection*) como un "talón de Aquiles" del rendimiento, pues puede provocar pausas prolongadas y caídas abruptas del *throughput*.

El fenómeno de *software aging*, analizado por Matias Jr. et al. [6], refuerza la importancia de monitorear fugas y recursos no liberados, pues su acumulación conduce progresivamente a fallos.

6.3.2. Degradación por CPU

La saturación de CPU puede suceder por hilos en espera activa, bucles infinitos o cargas no previstas. Este fenómeno se relaciona con lentitud en aplicaciones Java y .NET. Su detección temprana requiere la correlación de métricas del sistema operativo con procesos internos del servidor.

6.3.3. Degradación por procesos o hilos

Bloqueos de hilos, procesos zombie, deadlocks o saturación de colas internas pueden generar degradaciones severas. Anerousis et al. [12] proponen un sistema de monitoreo de salud capaz de identificar estas condiciones antes de que se conviertan en fallas críticas.

6.3.4. Degradación por configuración

Hamed y Kafri [4] demuestran que configuraciones inadecuadas del servidor de aplicaciones o del middleware pueden provocar degradaciones bajo carga, comparando arquitecturas Java EE y .NET. Parámetros como heap, tamaño de pool, concurrencia o timeouts influyen directamente en el rendimiento.

6.3.5. Degradación en bases de datos

La degradación también puede ocurrir en la capa de base de datos por agotamiento de *connection pools*, bloqueos, esperas excesivas o saturación en el almacenamiento. Tomić et al. [13] destacan que la disponibilidad del motor es esencial para la continuidad del negocio, mientras que Shrestha [14] analiza la replicación y sus implicaciones en HA.

6.4. Diagnóstico de Incidentes y Detección de Anomalías

El diagnóstico proactivo busca identificar condiciones anómalas antes de que éstas afecten la disponibilidad del servicio. La detección de anomalías consiste en identificar desviaciones respecto a patrones históricos o umbrales definidos.

Según Jäntti [2], incluso técnicas simples como *z-score* y regresión lineal pueden detectar fugas de memoria y consumo atípico de CPU. Otros estudios, como los de separación entre fallas derivadas de carga y fallas derivadas de degradación interna, demuestran que técnicas de clasificación logran altos niveles de precisión en la predicción temprana de fallas.

6.5. Ingeniería de Confiabilidad de Sitios (SRE)

SRE propone un marco moderno para garantizar disponibilidad mediante observabilidad, automatización, gestión de incidentes y presupuestos de error. Allam [15] plantea su aplicación en entornos *serverless*, mientras que Allam [8] explora su adaptación a arquitecturas distribuidas e IoT. Estas prácticas destacan la importancia de métricas como Latencia, errores y saturación para detectar problemas antes de que afecten al usuario.

6.6. Entornos Tecnológicos Relevantes

6.6.1. Linux

Herramientas como `sar`, `vmstat`, `journalctl` y los registros del sistema permiten obtener métricas históricas y en tiempo real sobre CPU, memoria, procesos y disco. Su interpretación es fundamental para anticipar degradaciones.

6.6.2. Windows Server / .NET

Windows ofrece *Performance Monitor* (PerfMon), que permite analizar contadores como CPU, memoria CLR, hilos, handles, colas de disco y eventos de GC. Estos datos son esenciales para analizar degradación en aplicaciones .NET.

6.6.3. Servidores de Aplicaciones Java

En servidores como JBoss, WebLogic u Oracle Application Server, el análisis de *thread dumps*, *GC logs*, pools de conexiones y parámetros del heap es fundamental. Xian et al. [3] destacan el impacto del GC mal configurado en la degradación.

6.6.4. Bases de Datos Oracle y SQL Server

Oracle incluye herramientas como AWR, ASH y ADDM que ofrecen métricas detalladas del motor. SQL Server provee indicadores a través de DMVs y Extended Events. Shrestha [14] resalta la relevancia de estas herramientas en la disponibilidad continua.

6.7. Herramientas Nativas para el Diagnóstico Proactivo

Las herramientas nativas del sistema operativo y del servidor de aplicaciones permiten:

- detectar patrones tempranos de degradación,
- correlacionar métricas entre distintas capas,
- reducir el uso de soluciones externas,
- intervenir con acciones preventivas antes de recurrir a reinicios.

Su correcta interpretación constituye un pilar teórico esencial para sustentar la metodología de diagnóstico proactivo propuesta en este trabajo.

7. Solución Propuesta

7.1. Descripción general de la solución

La solución propuesta consiste en el diseño de una **metodología de diagnóstico proactivo** orientada a anticipar incidentes que afectan la disponibilidad de aplicaciones empresariales desplegadas en entornos Oracle/Linux y SQL Server/Windows. Esta metodología integra herramientas nativas del sistema operativo, del servidor de aplicaciones y del motor de bases de datos, con el fin de identificar patrones de degradación antes de que sea necesario recurrir a reinicios de servicios o servidores.

La propuesta se estructura en tres bloques principales:

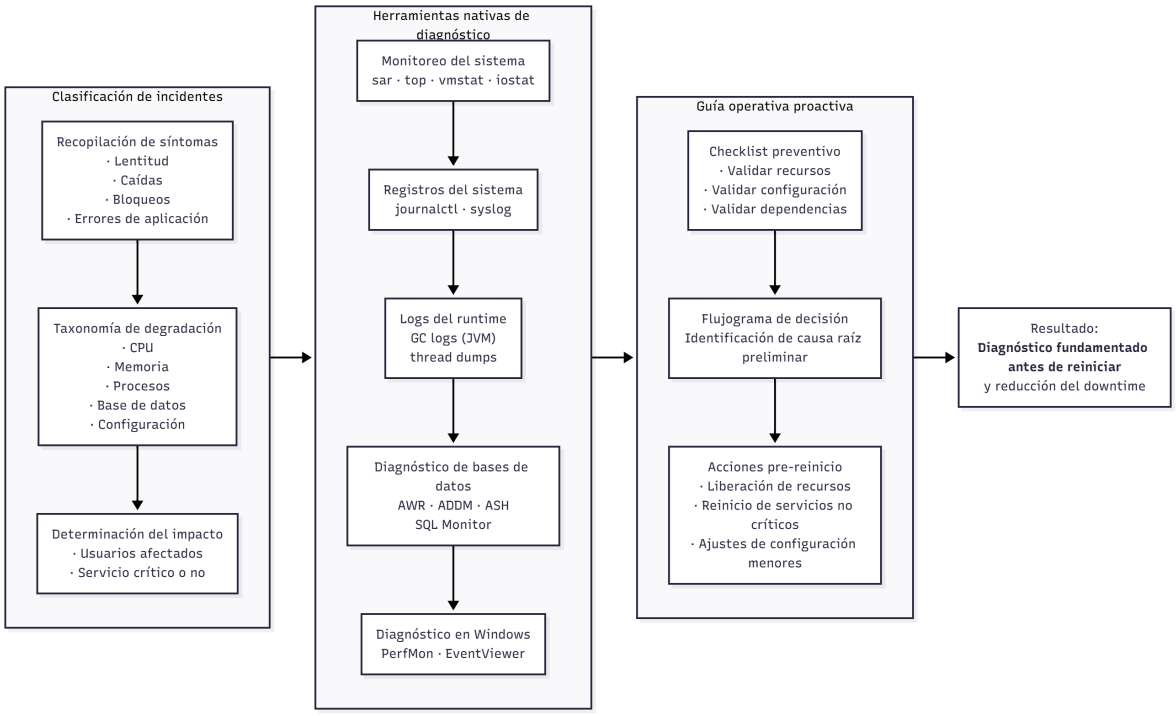


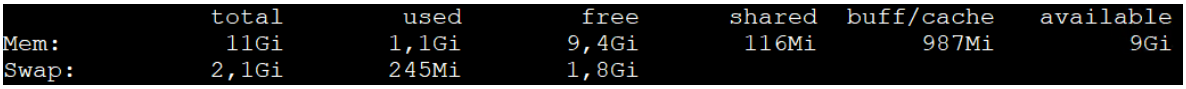
Figura 1: Diagrama de bloques propuesto para la metodología de diagnóstico proactivo.

7.2. Modelo conceptual

El modelo conceptual se fundamenta en los principios de diagnóstico proactivo, observabilidad y prácticas de Ingeniería de Confiabilidad de Sitios (SRE). Este modelo incorpora elementos identificados en el estado del arte, tales como detección temprana de anomalías [2], efectos de *software aging* [6] y degradación asociada a procesos de recolección de basura en servidores Java [3].

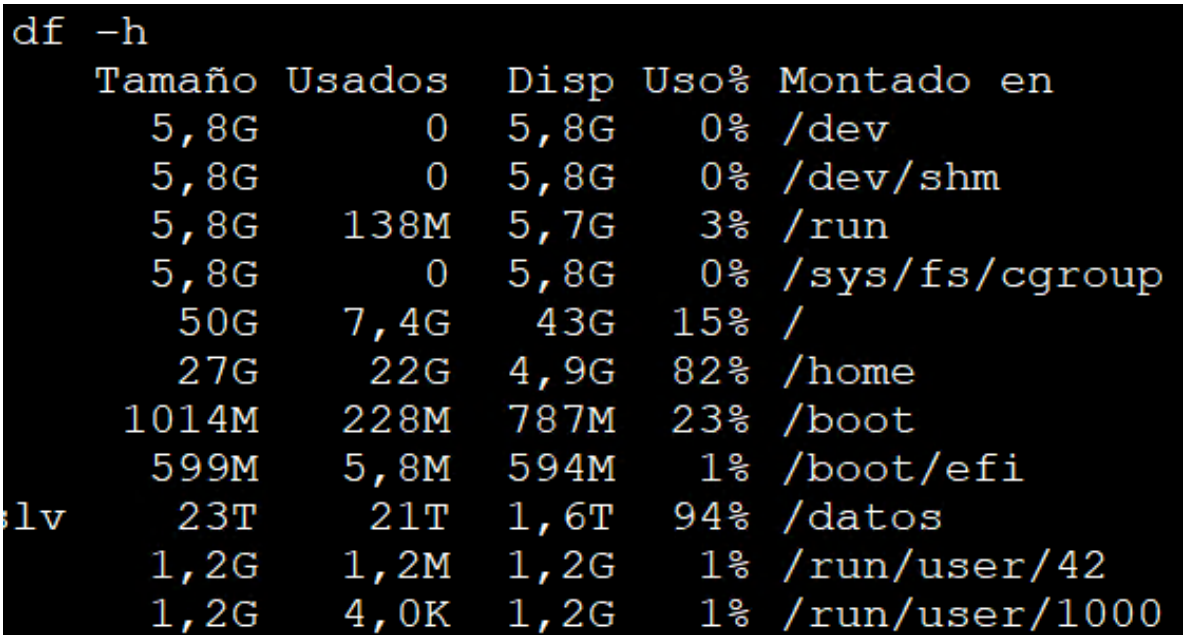
El modelo se organiza en tres capas:

1. **Capa de Observación:** Recolección de métricas del sistema operativo (CPU, memoria, I/O), del servidor de aplicaciones (GC, hilos, pools) y del motor de base de datos (AWR, ADDM, DMVs).



	total	used	free	shared	buff/cache	available
Mem:	11Gi	1,1Gi	9,4Gi	116Mi	987Mi	9Gi
Swap:	2,1Gi	245Mi	1,8Gi			

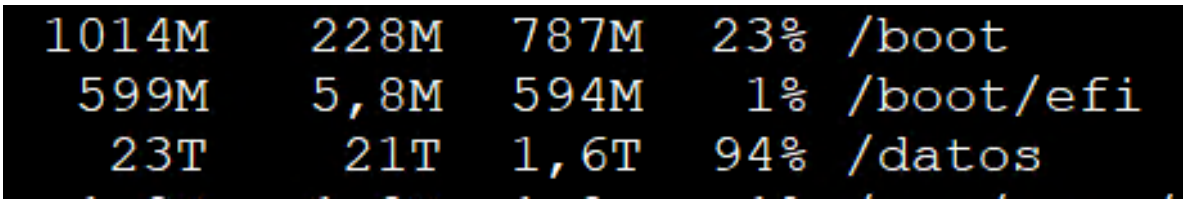
Figura 2: Recolección de métricas de memoria con el comando free -h



```
df -h
    Tamaño Usados  Disp Uso% Montado en
      5,8G       0  5,8G   0% /dev
      5,8G       0  5,8G   0% /dev/shm
      5,8G   138M  5,7G   3% /run
      5,8G       0  5,8G   0% /sys/fs/cgroup
       50G   7,4G   43G  15% /
       27G    22G  4,9G  82% /home
     1014M   228M  787M  23% /boot
      599M   5,8M  594M   1% /boot/efi
lv     23T    21T  1,6T  94% /datos
      1,2G   1,2M  1,2G   1% /run/user/42
      1,2G   4,0K  1,2G   1% /run/user/1000
```

Figura 3: Recolección de métricas de Disco / I/O con el comando df -h.

2. **Capa de Análisis:** Identificación temprana de patrones de degradación mediante umbrales, tendencias y correlaciones entre métricas.



```
 1014M   228M  787M  23% /boot
  599M   5,8M  594M   1% /boot/efi
   23T    21T  1,6T  94% /datos
```

Figura 4: Se identifican umbrales superados, en este caso puntos de montaje con poco espacio disponible.

3. **Capa de Acción Proactiva:** Definición de acciones recomendadas y verificación de causa raíz sin recurrir inicialmente a reinicios.

```
pvs
fdisk -l
pvcreate /dev/sdw
vgextend datosvg  /dev/sdw
lvresize -r -l+100%FREE /dev/datosvg/datoslv
df -h
pvs
fdisk -l
```

Figura 5: Como ejemplo, se aprovisiona nuevo espacio para que no supere el umbral recomendado ante imposibilidad de borrar archivos o logs.

Este modelo permite anticipar posibles fallos y se alinea con enfoques contemporáneos de confiabilidad del servicio [8], [15].

7.3. Estándares de la solución

La metodología se construye sobre buenas prácticas y lineamientos ampliamente aceptados en sistemas empresariales:

- **Estándares de sistemas operativos Linux y Windows** para análisis de rendimiento y diagnóstico.
- **Principios SRE** relacionados con observabilidad, gestión de incidentes y transición del enfoque reactivo al proactivo [15].
- **Buenas prácticas de Oracle y Microsoft** en análisis de rendimiento y disponibilidad (AWR/ADDM, DMV, eventos extendidos).
- **Lineamientos de alta disponibilidad** derivados de la literatura [1], [9].

Estos estándares proporcionan el marco normativo y técnico que justifica la estructura de la metodología.

7.4. Condiciones de diseño, propuesta de implementación y evaluación

La metodología se diseña para ser aplicable sin requerir herramientas externas, apoyándose exclusivamente en utilidades nativas de los sistemas operativos, servidores de aplicaciones y bases de datos. Las principales condiciones de diseño son:

- **Proactividad:** anticipar fallas antes de que afecten la disponibilidad del servicio.
- **Uso exclusivo de herramientas nativas:** `sar`, `journalctl`, GC logs, AWR/ADDM, PerfMon, entre otras.
- **Enfoque multiplataforma:** Java EE, .NET, Oracle, SQL Server, Linux y Windows.

- **Independencia del entorno:** aplicable tanto en infraestructura on-premise como en la nube.
- **Accionabilidad y trazabilidad:** permitir documentar síntomas, decisiones y acciones preventivas.

7.4.1. Taxonomía de incidentes de degradación

La Tabla 1 presenta la taxonomía propuesta de incidentes de degradación, construida a partir del análisis de literatura especializada y casos reales.

Tabla 1: Taxonomía de incidentes de degradación en aplicaciones empresariales

Categoría	**Descripción y ejemplos**
Memoria	Fugas de memoria, crecimiento no controlado del heap, saturación de swap, ciclos excesivos de GC.
CPU	Cargas sostenidas, procesos en bucle, hilos colgados, saturación por compilación JIT.
Procesos/Hilos	Deadlocks, espera activa, colas saturadas, hilos bloqueados.
Configuración	Pools mal ajustados, parámetros de JVM incorrectos, configuraciones obsoletas.
Base de datos	Contención, I/O lento, pools agotados, estadísticas desactualizadas.

7.4.2. Matriz incidente–métrica–herramienta

Tabla 2: Matriz incidente–métrica–herramienta de diagnóstico

Incidente	**Métricas relevantes**	**Herramientas**
Memoria	Heap, GC, swap, RSS	GC logs, `sar -r`, `top`, AWR, PerfMon
CPU	%usr, %sys, run queue	`sar -u`, `mpstat`, AWR, PerfMon
Procesos/Hilos	Conteo de hilos, wait states, locks	`ps`, `jstack`, DMVs, AWR
Configuración	Pools, parámetros JVM, sysctl	Archivos de configuración, logs del servidor
Base de datos	Wait events, I/O, locks	AWR/ADDM, DMVs, alert log

7.4.3. Checklist de diagnóstico proactivo

1. Verificar disponibilidad y tiempos de respuesta.
2. Revisar CPU, memoria, swap y carga del sistema.
3. Analizar ciclos de GC y tamaño del heap.

4. Validar estado del pool de conexiones.
5. Examinar logs críticos (`journalctl`, GC logs, alert log).
6. Identificar procesos con consumo anómalo.
7. Detectar posibles *deadlocks* o contención.
8. Comparar el incidente con patrones definidos en la taxonomía.

7.4.4. Flujograma general del diagnóstico

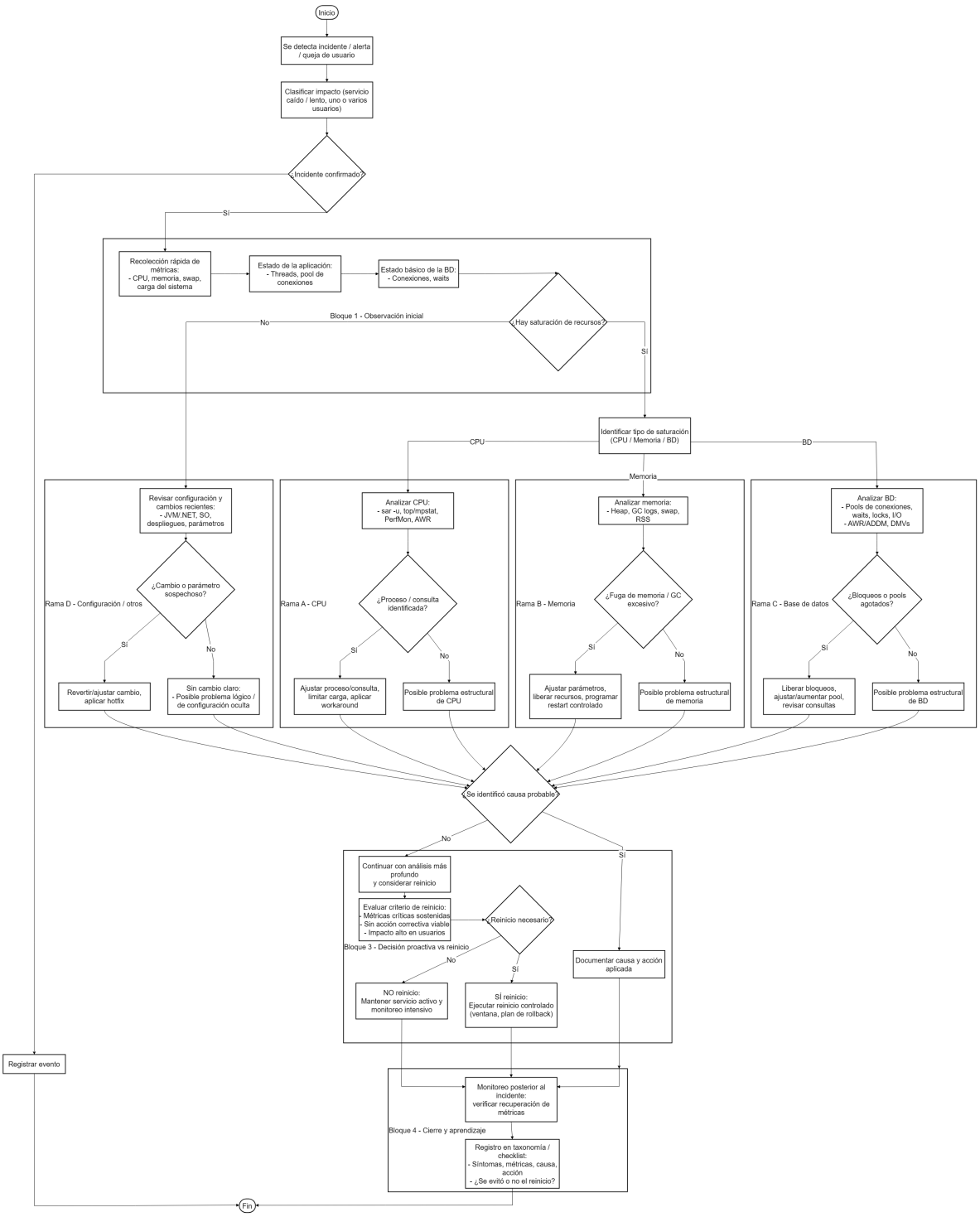


Figura 6: Flujograma general del diagnóstico proactivo.

7.4.5. Criterios para decidir si reiniciar un servicio

- Las métricas permanecen en niveles críticos tras acciones correctivas.
- Existen bloqueos o deadlocks irresolubles manualmente.
- Se confirma una fuga de memoria avanzada mediante análisis del heap.
- Bases de datos presentan contención severa documentada en AWR/ADDM o DMVs.
- El tiempo de recuperación manual es mayor al de un reinicio controlado.

7.4.6. Propuesta de evaluación

La metodología se evaluará mediante análisis teórico aplicado a escenarios simulados:

- Fuga de memoria en servidor Java.
- Saturación de CPU por proceso descontrolado.
- Agotamiento del pool de conexiones Oracle.
- Contención de I/O en motor de base de datos.

Para cada escenario se evaluará:

- Tiempo de detección.
- Capacidad de identificar causa raíz.
- Efectividad del checklist y flujograma.
- Posibilidad de evitar reinicios reactivos.

7.4.7. Consideraciones finales

La metodología propuesta permite identificar de manera anticipada las condiciones que originan degradaciones severas en aplicaciones empresariales. Su enfoque basado en herramientas nativas evita costos adicionales y facilita la transición hacia un modelo operativo proactivo alineado con estándares modernos de confiabilidad.

7.4.8. Integración emergente de Inteligencia Artificial en el diagnóstico proactivo

La aplicación de **modelos de Inteligencia Artificial (IA)** en operaciones de TI representa una línea de desarrollo reciente dentro del campo de *AIOps* (Artificial Intelligence for IT Operations). Estos modelos, en particular los *Large Language Models* (LLM), han comenzado a utilizarse para interpretar telemetría del sistema, correlacionar métricas y generar diagnósticos preliminares con rapidez, sirviendo como apoyo al análisis tradicional basado en herramientas nativas.

Un ejemplo representativo de esta tendencia es el modelo Linnix 3B Distilled, desarrollado por Shah [16], disponible en Hugging Face, especializado en la detección de incidentes del sistema operativo Linux tales como saturación de CPU, fugas de memoria, tormentas de procesos,

saturación de I/O y riesgos de *Out of Memory*. Dicho modelo puede transformar un breve resumen de telemetría en un **diagnóstico estructurado** en formato JSON con recomendaciones accionables, lo que acelera el análisis inicial del incidente sin reemplazar el criterio experto.

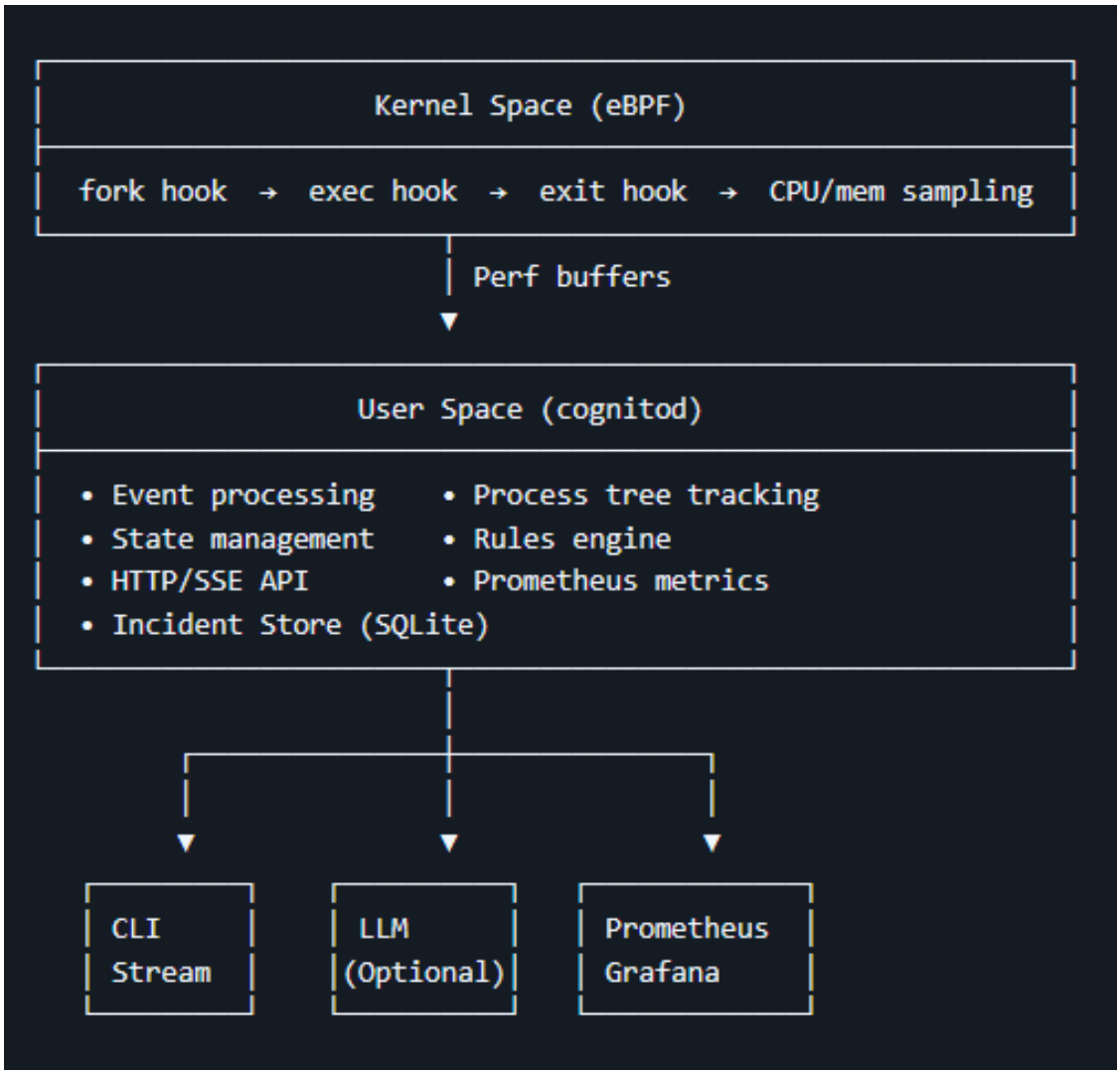


Figura 7: Arquitectura moderna de observabilidad basada en la propuesta de Shah [17] eBPF con integración opcional de modelos de IA para análisis interpretativo.

La arquitectura típica de estas soluciones integra la telemetría recolectada mediante herramientas nativas o mecanismos de bajo nivel como eBPF, que posteriormente se resumen y envían como entrada a un modelo LLM para obtener un análisis interpretativo:

$$\text{Kernel/eBPF} \rightarrow \text{Telemetría resumida} \rightarrow \text{Modelo IA} \rightarrow \text{Insight JSON}$$

Este enfoque aún se encuentra en evolución, pero ofrece oportunidades relevantes para complementar la metodología propuesta, especialmente en entornos con alta carga operativa. En particular, la IA puede:

- apoyar la **clasificación rápida del incidente**,
- sugerir **rutas de análisis** en función del patrón detectado,
- priorizar incidentes por criticidad,
- fortalecer la documentación y trazabilidad de eventos.

Aunque su integración no sustituye las herramientas nativas ni el análisis experto, sí constituye una **línea emergente de innovación** que puede potenciar la eficiencia del diagnóstico proactivo. Su adopción en fases experimentales resulta viable debido a que modelos como Linnix 3B pueden ejecutarse en CPU sin requerir infraestructura especializada.

8. Planeación del Trabajo

8.1. Descomposición de actividades WBS

A continuación, se presenta el diagrama de descomposición del trabajo (Work Breakdown Structure, WBS) correspondiente a la implementación de la metodología de diagnóstico proactivo en un entorno empresarial. Este WBS organiza las fases, actividades y subactividades necesarias para adoptar la metodología en una organización real.

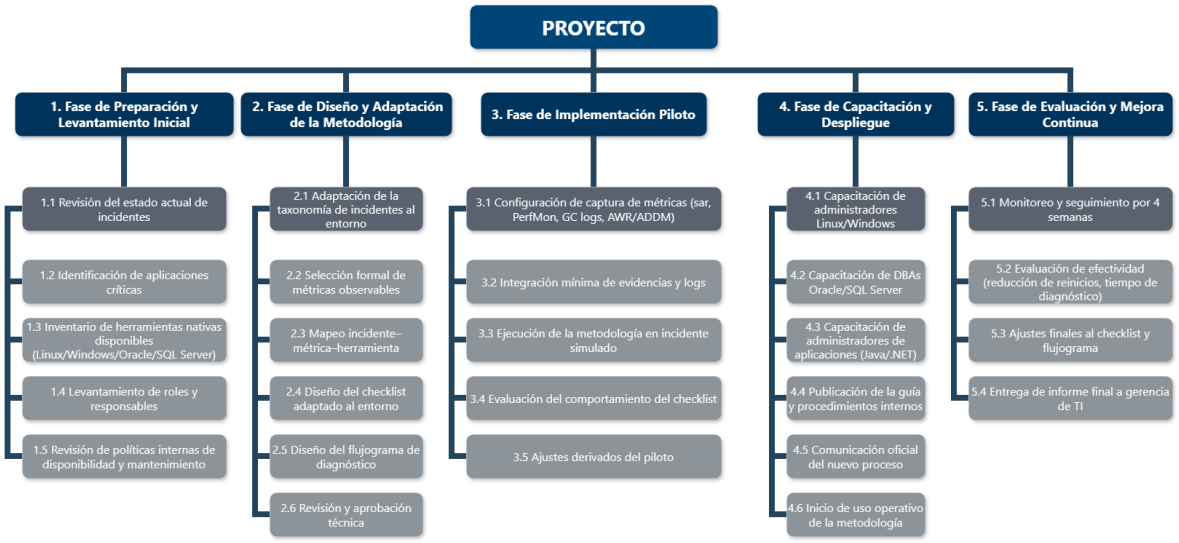


Figura 8: Work Breakdown Structure para la implementación de la metodología de diagnóstico proactivo

8.2. Tabla de actividades

A continuación, se presenta la tabla de actividades que organiza las fases y subactividades necesarias para la implementación de la metodología de diagnóstico proactivo. Esta planificación incluye duración estimada, fechas tentativas y predecesoras asociadas.

ID	Fase / Actividad	Duración (Semanas)	Inicio	Fin	Predecesora
1	**Fase de Preparación y Levantamiento Inicial**	4	2025-09-01	2025-09-26	–
1.1	Revisión del estado actual de incidentes	1	2025-09-01	2025-09-05	1
1.2	Identificación de aplicaciones críticas	1	2025-09-08	2025-09-12	1.1
1.3	Inventario de herramientas nativas disponibles	1	2025-09-15	2025-09-19	1.2
1.4	Levantamiento de roles y responsables	0.5	2025-09-22	2025-09-24	1.3
1.5	Revisión de políticas internas de disponibilidad	0.5	2025-09-25	2025-09-26	1.4
2	**Fase de Diseño y Adaptación de la Metodología**	5	2025-09-29	2025-11-03	1
2.1	Adaptación de la taxonomía de incidentes	1	2025-09-29	2025-10-03	1.5
2.2	Selección formal de métricas observables	1	2025-10-06	2025-10-10	2.1
2.3	Mapeo incidente–métrica–herramienta	1	2025-10-13	2025-10-17	2.2
2.4	Diseño del checklist adaptado al entorno	1	2025-10-20	2025-10-24	2.3
2.5	Diseño del flujograma de diagnóstico	1	2025-10-27	2025-11-03	2.4
3	**Fase de Implementación Piloto**	4	2025-11-04	2025-11-28	2
3.1	Configuración de captura de métricas	1.5	2025-11-04	2025-11-13	2.5
3.2	Integración mínima de evidencias y logs	1	2025-11-14	2025-11-20	3.1
3.3	Ejecución en incidente simulado	0.5	2025-11-21	2025-11-25	3.2
3.4	Evaluación del checklist y resultados del piloto	0.5	2025-11-26	2025-11-26	3.3
3.5	Ajustes derivados del piloto	0.5	2025-11-27	2025-11-28	3.4
4	**Fase de Capacitación y Despliegue**	3	2025-12-01	2025-12-19	3
4.1	Capacitación de administradores SO Linux/Windows	1	2025-12-01	2025-12-05	3.5
4.2	Capacitación de DBAs Oracle/SQL Server	1	2025-12-08	2025-12-12	4.1
4.3	Capacitación de admins de aplicaciones Java/.NET	0.5	2025-12-15	2025-12-16	4.2
4.4	Publicación de guía operativa interna	0.5	2025-12-17	2025-12-17	4.3
4.5	Comunicación oficial del nuevo proceso	0.5	2025-12-18	2025-12-19	4.4
5	**Fase de Evaluación y Mejora Continua**	3	2025-12-22	2026-01-10	4
5.1	Monitoreo continuo por 4 semanas	2	2025-12-23	2026-01-02	4.5
5.2	Evaluación de efectividad de la metodología	0.5	2026-01-05	2026-01-07	5.1
5.3	Ajustes finales y documentados	0.5	2026-01-08	2026-01-08	5.2
5.4	Entrega del informe final	0.2	2026-01-09	2026-01-09	5.3

Tabla 3: Tabla de actividades del proyecto

8.3. Diagrama de Gantt

La Tabla 3 presenta el detalle de las fases y actividades definidas para el desarrollo del proyecto, incluyendo su duración estimada y las dependencias entre ellas. A partir de esta pla-

nificación se construyó el diagrama de Gantt que resume visualmente la distribución temporal de las actividades.

La Figura 9 muestra el diagrama de Gantt general del proyecto, elaborado en Microsoft Project a partir de la estructura de fases presentada en la Tabla 3. En dicho diagrama se observa la secuencia de ejecución, las relaciones de precedencia y la duración estimada de cada una de las actividades.

Id	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1	**Fase 1 - Preparación y Levantamiento Inici**	**20 días**	**lun 1/09/25**	**vie 26/09/25**	
2	Revisión del estado actual de incidentes	5 días	lun 1/09/25	vie 5/09/25	
3	Identificación de aplicaciones críticas	5 días	lun 8/09/25	vie 12/09/25	2
4	Inventario de herramientas nativas disponibles (Linux/Windows/Oracle/SQL Server)	5 días	lun 15/09/25	vie 19/09/25	3
5	Levantamiento de roles y responsables	3 días	lun 22/09/25	mié 24/09/25	4
6	Revisión de políticas internas de disponibilidad y mantenimiento	2 días	jue 25/09/25	vie 26/09/25	5
7	**Fase 2 - Diseño y Adaptación de la Metodología**	**26 días**	**lun 29/09/25**	**lun 3/11/25**	
8	Adaptación de la taxonomía de incidentes al entorno	5 días	lun 29/09/25	vie 3/10/25	6
9	Selección formal de métricas observable	5 días	lun 6/10/25	vie 10/10/25	8
10	Mapeo incidente–métrica–herramienta	5 días	lun 13/10/25	vie 17/10/25	9
11	Diseño del checklist adaptado al entorn	5 días	lun 20/10/25	vie 24/10/25	10
12	Diseño del flujograma de diagnóstico	6 días	lun 27/10/25	lun 3/11/25	11
13	**Fase 3 - Implementación Piloto**	**19 días**	**mar 4/11/25**	**vie 28/11/25**	
14	Configuración de captura de métricas (sar, PerfMon, GC logs, AWR/ADDM)	8 días	mar 4/11/25	jue 13/11/25	12
15	Integración mínima de evidencias y logs	5 días	vie 14/11/25	jue 20/11/25	14
16	Ejecución de la metodología en incidente simulado	3 días	vie 21/11/25	mar 25/11/25	15
17	Evaluación del comportamiento del checklist	1 día	mié 26/11/25	mié 26/11/25	16
18	Ajustes derivados del piloto	2 días	jue 27/11/25	vie 28/11/25	17
19	**Fase 4 - Capacitación y Despliegue**	**15 días**	**lun 1/12/25**	**vie 19/12/25**	
20	Capacitación de administradores Linux/Windows	5 días	lun 1/12/25	vie 5/12/25	18
21	Capacitación de DBAs Oracle/SQL Server	5 días	lun 8/12/25	vie 12/12/25	20
22	Capacitación de administradores de aplicaciones (Java/.NET)	2 días	lun 15/12/25	mar 16/12/25	21
23	Publicación de la guía y procedimientos internos	1 día	mié 17/12/25	mié 17/12/25	22
24	Comunicación oficial del nuevo proceso	2 días	jue 18/12/25	vie 19/12/25	23
25	**Fase 5 - Fase de Evaluación y Mejora Continua**	**15 días**	**lun 22/12/25**	**vie 9/01/26**	
26	Monitoreo y seguimiento por 4 semanas	10 días	lun 22/12/25	vie 2/01/26	24
27	Evaluación de efectividad (reducción de reinicios, tiempo de diagnóstico)	3 días	lun 5/01/26	mié 7/01/26	26
28	Ajustes finales al checklist y flujograma	1 día	jue 8/01/26	jue 8/01/26	27
29	Entrega de informe final a gerencia de T	1 día	vie 9/01/26	vie 9/01/26	28

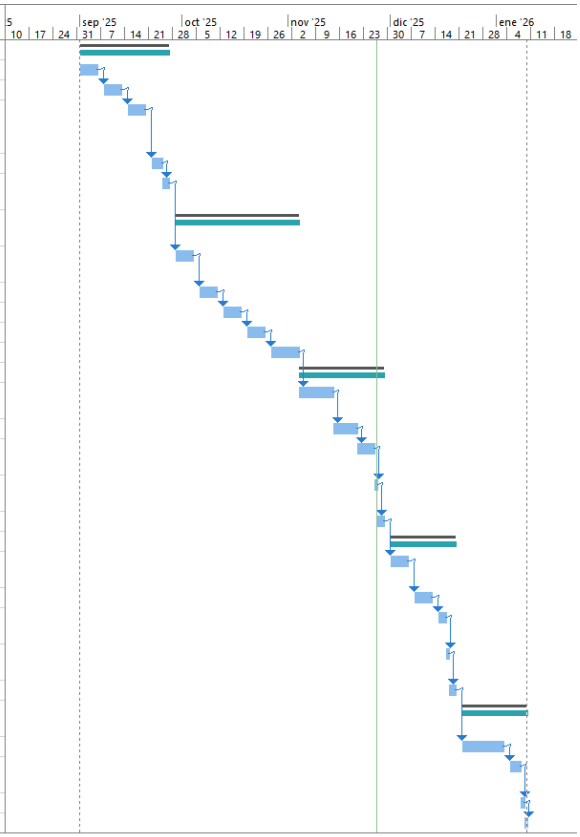


Figura 9: Diagrama de Gantt.

9. Presupuesto

El presente presupuesto estima los costos asociados a la implementación institucional de la metodología de diagnóstico proactivo propuesta. Debido a que la solución se basa en el uso de herramientas nativas de los sistemas operativos, servidores de aplicación y motores de bases de datos, no se contemplan costos de adquisición de software o infraestructura adicional. Los costos se concentran en horas de trabajo del personal, capacitación, documentación y potencial consumo incremental de almacenamiento y logs.

Presupuesto			
Ítem	**Cantidad**	**Precio Unitario (COP)**	**Total (COP)**
Horas de trabajo analista de infraestructura (OTIC)	160	$45.000	$7.200.000
Horas de trabajo administrador de bases de datos	120	$55.000	$6.600.000
Horas de trabajo del equipo de aplicaciones Java/.NET	100	$50.000	$5.000.000
Horas del líder técnico / arquitecto	40	$70.000	$2.800.000
Horas de consultoría externa (opcional)	20	$150.000	$3.000.000
Capacitación interna (sesiones de 3 grupos técnicos)	3	$1.200.000	$3.600.000
Documentación, manuales y estandarización	1	$2.500.000	$2.500.000
Publicación y adopción del proceso (comunicaciones)	1	$1.200.000	$1.200.000
Almacenamiento adicional en servidores (logs/metrics)	200 GB	$300/COP/GB	$60.000
Incremento en retención de AWR/ADDM y logs	1	$500.000	$500.000
Contingencias (10 %)	–	–	$3.646.000
TOTAL			**$39.106.000**

Tabla 4: Presupuesto estimado para la implementación institucional de la metodología de diagnóstico proactivo.

Categoría	**Costo Unitario (COP)**	**Cantidad**	**Costo Total (COP)**
Personal técnico (OTIC, DBAs, Apps, Líder técnico)	$21.600.000	1	$21.600.000
Consultoría externa (opcional)	$3.000.000	1	$3.000.000
Capacitación interna	$1.200.000	3	$3.600.000
Documentación y estandarización	$2.500.000	1	$2.500.000
Costos de almacenamiento y logs	$560.000	1	$560.000
Contingencias (10 %)	$3.846.000	1	$3.846.000
TOTAL			**$39.106.000**

Tabla 5: Tabla de gastos resumidos del proyecto (valores coherentes con el presupuesto detallado).

10. Conclusiones

La disponibilidad de las aplicaciones empresariales y sus bases de datos constituye un pilar fundamental para la continuidad operativa de las organizaciones modernas. En este trabajo se desarrolló una metodología de diagnóstico proactivo enfocada en anticipar incidentes de degradación en entornos Oracle/Linux y SQL Server/Windows, integrando buenas prácticas de observabilidad, lineamientos SRE y el uso sistemático de herramientas nativas del sistema operativo y del motor de bases de datos. El proyecto permitió establecer una estructura conceptual y operativa capaz de guiar a los administradores en la identificación temprana de síntomas críticos, reduciendo la dependencia histórica de los reinicios como mecanismo reactivo de recuperación.

En primera instancia, se logró definir una taxonomía clara de incidentes de degradación que agrupa y caracteriza las problemáticas más frecuentes (memoria, CPU, procesos, configuración y base de datos). Esta taxonomía, junto con la matriz incidente–métrica– herramienta, constituye un aporte práctico que facilita el análisis estructurado de eventos operativos reales y se convierte en un insumo directo para ambientes de misión crítica.

Asimismo, se evidenció que herramientas nativas como `sar`, `journalctl`, Free/DF, top, PerfMon y los logs de recolección de basura de la JVM poseen un potencial subutilizado en muchas organizaciones. La metodología propuesta demuestra que, cuando se integran de forma ordenada y alineada con una guía operativa, estas herramientas permiten detectar tendencias de degradación (por ejemplo, fugas de memoria o saturación progresiva de CPU) con suficiente antelación para evitar fallas graves o reinicios no planificados.

El proyecto también permitió explorar el papel emergente de la inteligencia artificial en el diagnóstico de incidentes. Plataformas como HuggingFace, modelos de lenguaje y agentes autónomos muestran un potencial considerable para complementar el análisis tradicional mediante clasificación automática de logs, detección de anomalías basada en aprendizaje profundo y recomendaciones generadas a partir de patrones históricos. Aunque la implementación de estos modelos no formó parte del alcance del trabajo, se identifican como una vía futura altamente prometedora para automatizar la correlación y priorización de alertas, especialmente en infraestructuras con altos volúmenes de datos operativos.

En términos de logros alcanzados, el trabajo consolidó una propuesta metodológica coherente, aplicable y alineada con estándares contemporáneos de confiabilidad, demostrando que es posible mejorar la resiliencia operacional sin inversión adicional en software especializado. La metodología es factible tanto en entornos on-premise como en la nube, y se adapta especialmente bien a entornos *legacy* dominados por Java EE, .NET y Oracle.

Como líneas de mejora futura se recomienda: (i) integrar modelos de IA para detección automática de anomalías en métricas y logs; (ii) extender la metodología a arquitecturas basadas en contenedores y microservicios; (iii) automatizar la ejecución del checklist mediante agentes de observabilidad; y (iv) desarrollar un módulo de retroalimentación que permita calibrar umbrales dinámicos basados en comportamiento histórico.

Finalmente, se concluye que la transición de un enfoque reactivo a uno proactivo no solo es posible, sino necesaria para garantizar la continuidad del servicio en organizaciones con alta dependencia tecnológica. La metodología propuesta constituye un paso significativo hacia la construcción de una cultura de confiabilidad, donde las decisiones se basan en evidencia, métricas y análisis sistemático, minimizando las interrupciones y fortaleciendo la operación misional de la organización.

Referencias

[1] P. T. Endo, M. Rodrigues, G. E. Gonçalves, J. Kelner, D. H. Sadok y C. Curescu, «High availability in clouds: systematic review and research challenges,» *Journal of Cloud Computing: Advances, Systems and Applications*, vol. 5, n.º 16, págs. 1-22, 2016. DOI: 10.1186/s13677-016-0066-8.

[2] N. Jäntti, «Detecting anomalies in server performance,» [Online]. Available: https://trepo.tuni.fi/bitstream/handle/10024/121617/JanttiNiko.pdf?sequence=24, Tesis de mtría., Tampere University, Faculty of Information Technology y Communication Sciences, mayo de 2020.

[3] F. Xian, W. Srisa-an y H. Jiang, «Garbage collection: Java application servers' Achilles heel,» *Science of Computer Programming*, vol. 70, n.º 2–3, págs. 89-110, feb. de 2008. DOI: 10.1016/j.scico.2007.07.008.

[4] O. Hamed y N. Kafri, «Performance Prediction of Web Based Application Architectures: Case Study .NET vs. Java EE,» *International Journal of Web Applications*, vol. 1, n.º 3, págs. 146-155, sep. de 2009, [Online]. Available: https://www.researchgate.net/publication/220500854_Performance_Prediction_of_Web_Based_Application_Architectures_Case_Study_NET_vs_Java_EE.

[5] U. Naseer, L. Niccolini, U. Pant, A. Frindell, R. Dasineni y T. A. Benson, «Zero Downtime Release: Disruption-free Load Balancing of a Multi-Billion User Website,» en *Proc. ACM SIGCOMM*, ago. de 2020, págs. 1-13. DOI: 10.1145/3387514.3405885.

[6] R. M. Jr., P. J. F. Filho, L. Guedes y A. Dias, «Estimation of Web Servers' Reliability with Symptoms of Software Aging,» en *Proceedings of the Experimental Software Engineering Latin American Workshop (ESELAW)*, 2005, págs. –.

[7] M. Saarinen, «Evaluation of Reliability in IoT and Edge Computing Platforms through SRE Practices,» CC-BY 4.0 License, Master's Thesis, University of Oulu, Faculty of Information Technology y Electrical Engineering, Oulu, Finland, jun. de 2024. dirección: https://oulurepo.oulu.fi/handle/10024/50800.

[8] H. Allam, «Reliability at the Edge: SRE for Distributed Cloud and IoT Platforms,» *International Journal of Engineering Research & Emerging Technology (IJERET)*, vol. 6, n.º 2, págs. 39-52, mayo de 2025. DOI: 10.63282/3050-922X.IJERET-V6I2P106.

[9] S. W. Hunter y W. E. Smith, «Availability Modeling and Analysis of a Two Node Cluster,» en *Proc. 5th Int. Conf. on Information Systems, Analysis and Synthesis (ISAS)*, Orlando, FL, USA, 1999.

[10] H. Handoko, S. M. Isa y S. I. Si, «High Availability Analysis with Database Cluster, Load Balancer and Virtual Router Redundancy Protocol,» en *Proceedings of the 2018 3rd International Conference on Computer and Communication Systems (ICCCS)*, abr. de 2018, –. DOI: 10.1109/CCOMS.2018.8463263.

[11] R. S. Barga y D. B. Lomet, «Phoenix/ODBC: A Recoverable, Scalable ODBC Interface for Transaction Processing,» en *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, IEEE, 2001, págs. 339-348. DOI: 10.1109/ICDE.2001.914810.

[12] N. Anerousis, A. Black, S. Hanson, L. Mummert y G. Pacifici, «Health Monitoring and Control for Application Server Environments,» en *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Hawthorne, NY, USA: IEEE, 2005, págs. 385-398. DOI: 10.1109/NOMS.2005.102.

[13] D. Tomić, B. Markić, M. Mabić y D. Gašpar, «Continuous Database Availability as a Precondition for Business Continuity,» en *Proceedings of the Conference on Business, Economics and Information Technology*, University of Mostar, Bosnia y Herzegovina, 2007. dirección: https://www.researchgate.net/publication/270100780_Continuous_Database_Availability_as_a_Precondition_for_Business_Continuity.

[14] R. Shrestha, «High Availability and Performance of Database in the Cloud: Traditional Master-Slave Replication versus Modern Cluster-Based Solutions,» en *Proceedings of the 7th International Conference on Cloud Computing and Services Science (CLOSER)*, Porto, Portugal: SciTePress, 2017, págs. 385-392, ISBN: 978-989-758-243-1. DOI: 10.5220/0006294604130420.

[15] H. Allam, «Cloud-Native Reliability: Applying SRE to Serverless and Event-Driven Architectures,» *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, vol. 5, n.º 3, págs. 68-79, 2024, ISSN: 3050-9262. DOI: 10.63282/3050-9262.IJAIDSML-V5I3P108.

[16] P. Shah, *Linnix 3B Distilled: Incident Detection Model*, Fine-tuned Qwen2.5-3B for system observability., 2025. dirección: https://huggingface.co/parth21shah/linnix-3b-distilled.

[17] P. Shah, *Linnix Observability Pipeline Diagram*, https://github.com/linnix-os/linnix, Diagrama de arquitectura del pipeline eBPF–User Space–LLM., 2025.