UNIVERSIDAD TÉCNICA DEL NORTE
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS
CARRERA DE SOFTWARE

INFORME FINAL DEL TRABAJO DE INTEGRACIÓN CURRICULAR, MODALIDAD TRABAJO DE GRADO

TEMA:
"Integración de sistemas que comparten un alto volumen de mensajes mediante el uso de un middleware: Un estudio de caso"

Trabajo de titulación previo a la obtención del título de *INGENIERO EN SOFTWARE*

Línea de investigación: Desarrollo, aplicación de software y cyber security (seguridad cibernética).

AUTOR(A):
Kevin Zahid Aldas Proaño

DIRECTOR(A):
MSc. Xavier Mauricio Rea Peñafiel

Ibarra, febrero 2025

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1.1 IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1005004732		
APELLIDOS Y NOMBRES:	Aldas Proaño Kevin Zahid		
DIRECCIÓN:	Pana americana y Rio Amazonas		
EMAIL:	kevinaldas290621@gmail.com		
TELÉFONO FIJO:	062906431	**TELÉFONO MÓVIL:**	0988441198

DATOS DE LA OBRA	
TÍTULO:	Integración de sistemas que comparten un alto volumen de mensajes mediante el uso de un middleware: Un estudio de caso.
AUTOR (ES):	Aldas Proaño Kevin Zahid
FECHA: DD/MM/AAAA	12/02/2025
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Software
DIRECTOR:	MSc. Xavier Mauricio Rea Peñafiel
ASESOR:	PhD. Jose Antonio Quiña Mera

2.1 CONSTANCIAS

2.1 CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 12 días del mes de febrero de 2025

EL AUTOR:

.....

Nombre: Kevin Zahid Aldas Proaño

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 12 de febrero de 2025

Msc. Xavier Mauricio Rea Peñafiel
DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Haber revisado el presente informe final del trabajo de Integración Curricular, mismo que se ajusta a las normas vigentes de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) .....

Msc. Xavier Mauricio Rea Peñafiel

C.C: 1004056378

Dedicatoria

Dedico este trabajo de grado a mis padres, pilares fundamentales en mi vida, cuyo amor incondicional y apoyo incansable han iluminado cada paso de mi camino. Este logro no es solo mío, sino un triunfo compartido que honra su entrega, su fe en mis sueños y las manos que siempre me sostuvieron, incluso en los momentos más desafiantes. Porque, gracias a ellos, nunca caminé solo.

Agradecimientos

Agradezco profundamente a mis padres, cuyo amor, dedicación y esfuerzo han sido el fundamento de mi camino. Su confianza en mis capacidades y su inagotable paciencia han moldeado a la persona que soy hoy. Este trabajo es un reflejo de todo lo que me han brindado y del impacto de sus enseñanzas en mi vida, un logro que no es solo mío, sino de los tres.

A los amigos que formé durante mi etapa universitaria, quienes no solo me brindaron su apoyo y conocimientos a lo largo de este proceso, sino que también fueron una fuente constante de motivación para seguir creciendo.

A los amigos que me acompañan desde otros caminos de la vida, gracias por enseñarme valiosas lecciones y por estar siempre ahí para sostenerme en los momentos difíciles. Su apoyo constante y los gratos momentos que he vivido a su lado los convierten en una segunda familia para mí.

Quiero expresar mi profundo agradecimiento a mi tutor, Mauricio Rea, por su infinita paciencia, por compartir su conocimiento y por brindarme una guía constante. Sin su perseverancia y dedicación, este logro no hubiera sido posible. A todos mis demás profesores, gracias por su manera única de transmitir su saber, por hacer de esta etapa de mi vida una experiencia enriquecedora y llena de aprendizaje.

Tabla de contenido

Índice de figuras

Resumen

La integración de sistemas que manejan grandes volúmenes de mensajes es crucial en la tecnología moderna, sin embargo, el plan de estudios de software de la Universidad Técnica del Norte carece de estudios de casos prácticos en esta área. Esta tesis aborda esta brecha mediante la implementación de un sistema de middleware que utiliza Apache Kafka para facilitar el intercambio de datos eficiente y confiable entre sistemas.

El estudio caracteriza la integración de sistemas en entornos con gran cantidad de datos, desarrolla una arquitectura de integración y evalúa su desempeño con base en los estándares ISO/IEC 25010. Los hallazgos demuestran la eficacia de la solución propuesta para mejorar la interoperabilidad del sistema y la confiabilidad del intercambio de datos, ofreciendo importantes beneficios educativos y prácticos.

Abstract

The integration of systems handling high volumes of messages is crucial in modern technology, yet the software curriculum at Universidad Técnica del Norte lacks practical case studies in this area. This thesis addresses this gap by implementing a middleware system using Apache Kafka to facilitate efficient and reliable data exchange between systems.

The study characterizes system integration in high-data environments, develops an integration architecture, and evaluates its performance based on ISO/IEC 25010 standards. The findings demonstrate the effectiveness of the proposed solution in enhancing system interoperability and data exchange reliability, offering significant educational and practical benefits.

Introducción

La integración de los sistemas de manejo de grandes volúmenes de datos es un problema central en la era de la información. Las organizaciones modernas requieren medios eficientes para llevar a cabo un intercambio rápido, seguro y confiable de datos. Con la aparición de un número creciente de aplicaciones y sistemas que generan datos, también crece la demanda de arquitecturas de integración que puedan manejar la comunicación en tiempo real y el procesamiento de la información.

El presente trabajo tiene como objetivo principal implantar un sistema middleware que facilite la integración de sistemas en entornos con grandes volúmenes de datos. Para ello, se empleará Apache Kafka como herramienta principal, proporcionando una solución práctica y demostrando los beneficios y desafíos asociados con su uso. Este caso de estudio no solo beneficiará a los estudiantes al proporcionarles una experiencia práctica y directa, sino que también contribuirá al campo de la integración de sistemas con soluciones innovadoras y eficientes.

Antecedentes

Situación actual

En la actualidad, el pensum de la carrera de software carece de un caso de estudio específico sobre la integración de sistemas que comparten un alto volumen de mensajes. Esta carencia dificulta la enseñanza de aspectos clave relacionados con la gestión de grandes volúmenes de datos. Además, resulta complicado encontrar casos de estudio que aborden el uso de middleware como herramienta de integración, lo que limita el aprendizaje y la comprensión de estas tecnologías cruciales.

Planteamiento del problema

La integración de sistemas en entornos con grandes volúmenes de datos generados plantea desafíos significativos. De acuerdo con (Algorri Álvarez, 2018) el problema central radica en la cantidad masiva de datos que se generan y cómo manejarlos de manera eficiente y confiable entre los sistemas involucrados.

Como menciona (Martínez, 2020) en la actualidad las aplicaciones generan sus propios datos, pero otras dependen de ellas para realizar ciertas tareas, pero al compartirlos no están disponibles en tiempo real para el sistema que solicita lo que ocasiona que el sistema que requiere los datos podría quedar en estado de espera (bloqueo) e incluso que estos datos no estén actualizados. Actualmente, en el pensum académico de las materias de desarrollo de software para MIPYMES y desarrollo de software empresarial de la carrera de software, se ha identificado una falta de herramientas didácticas que permitan comprender los conceptos relacionados con la integración de sistemas que manejan un alto volumen de datos. Esta carencia dificulta diversos aspectos, tales como:

- Dificultad de encontrar material que aborde la integración de este tipo de sistemas de una manera organizada y completa.
- Dificultad en entender la arquitectura necesaria para poder realizar esta integración, complicando entender los conceptos fundamentales que se abarca.
- Planteamiento de ejercicios prácticos y guías para este tipo de integración.

Como menciona (Moreno Pina, 2020) se debe tener clara la diferencia entre lo que es middleware y un software productor de datos, siendo esto un desafío entre los estudiantes para poder usar las herramientas de integración.

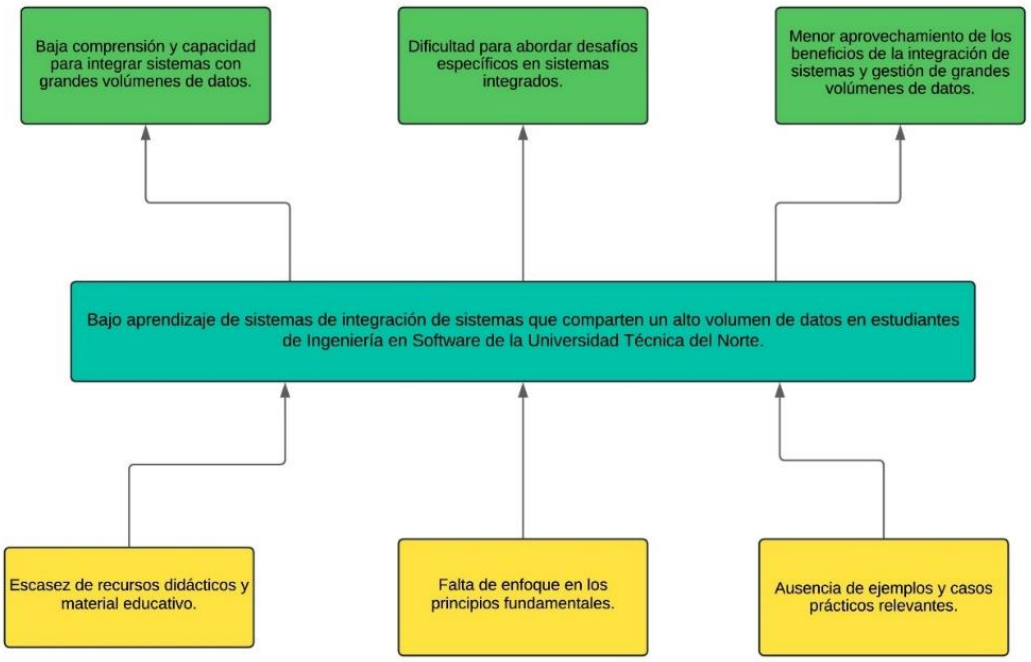

Figura 1 Árbol de problemas.
Elaborado por el autor

Objetivos

Objetivo general

Implantar un sistema middleware que permita la integración de sistemas en entornos de alto volumen de mensajes.

Objetivo especifico

- Caracterización de la integración de sistemas en entornos con grandes volúmenes de datos generados.
- Implantar una arquitectura de integración utilizando Apache Kafka como herramienta principal para que exista un intercambio de mensajes entre dos sistemas y que sirva como un caso de estudio técnico.
- Evaluar la eficiencia de desempeño de la arquitectura propuesta siguiendo los criterios establecidos en la norma en la característica de eficiencia de desempeño ISO/IEC 25010.

Alcance y metodología

Alcance

En la carrera de software de la Universidad Técnica del Norte, se emplean diversos productos de software como casos de estudio en las materias de Desarrollo de Software para MIPYMES, Desarrollo de Software Empresarial y Verificación y Validación de Software. Son los siguientes:

ERP denominado "Minimarket Demo".

Sistema de facturación.

En este contexto, surge la necesidad de desarrollar una arquitectura de integración que permita analizar un nuevo escenario empresarial donde estos dos sistemas compartan un alto volumen de datos.

Primero, se hará una investigación para comprender en detalle los requisitos, características y limitaciones de los sistemas seleccionados. Se analizarán los flujos de datos generados por cada sistema, la estructura de los datos, las interfaces de comunicación disponibles (Algorri Álvarez, 2018).

Luego, se implementará la solución propuesta, configurando y poniendo en marcha el clúster de Apache Kafka. Se desarrollarán los conectores necesarios para interactuar con los sistemas existentes y se establecerán las políticas de entrega, asegurando una transmisión confiable y ordenada de los mensajes entre los sistemas como menciona (Red Hat, 2022).

El alcance de este trabajo se centra en el desarrollo de un caso de estudio específico de la integración de los dos sistemas seleccionados como se muestra en la Figura 2, proporcionando una solución práctica y demostrando los beneficios y desafíos asociados con el uso de Apache Kafka como herramienta de integración (Algorri Álvarez, 2018).

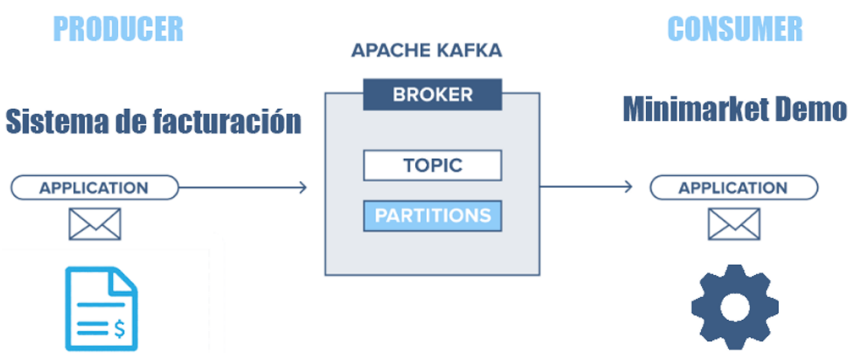

Figura 2 Estudio de caso.
Elaborado por el autor

Metodología

Revisión bibliográfica y estudio de casos: Iniciar el proceso con una revisión de la literatura existente sobre integración de sistemas, gestión de grandes volúmenes de datos y el uso de Apache Kafka. Además, estudiar casos de éxito y buenas prácticas en la implementación de soluciones similares. Esta etapa proporcionará los fundamentos teóricos necesarios.

Análisis y comprensión de los sistemas involucrados: Realizar un análisis detallado de los sistemas seleccionados como caso de estudio. Es fundamental comprender a fondo cada sistema para identificar las necesidades y los puntos críticos de integración.

Para poder cumplir el segundo objetivo específico se hará el uso de Enterprise Integration Patterns (EIP), como se menciona (Nebel, 2018) el objetivo de elaborar patrones de diseño en la integración de sistemas es proporcionar soluciones probadas y reutilizables para problemas recurrentes. Estos patrones permiten abordar los desafíos de manera sistemática y ofrecen una base sólida para el diseño de arquitecturas de integración.

En este caso, se empleará Apache Kafka como herramienta principal en la implementación de la arquitectura de integración como se menciona en (Red Hat, 2022) Apache Kafka es una plataforma distribuida que permite la transmisión de datos en tiempo real, con capacidades para publicar, suscribirse, almacenar y procesar flujos de registros. Apache Kafka está especialmente diseñado para manejar flujos de datos provenientes de múltiples fuentes y entregarlos a múltiples consumidores.

Se basa en un diseño de sistema de cola de mensajes como se muestra en la Figura 3, como menciona (Person Montero, 2017) Estos sistemas usan un paradigma de publicador-suscriptor, actuado como un buffer de mensajes producidos, estos se producen cuando los sistemas se comunican entre sí.

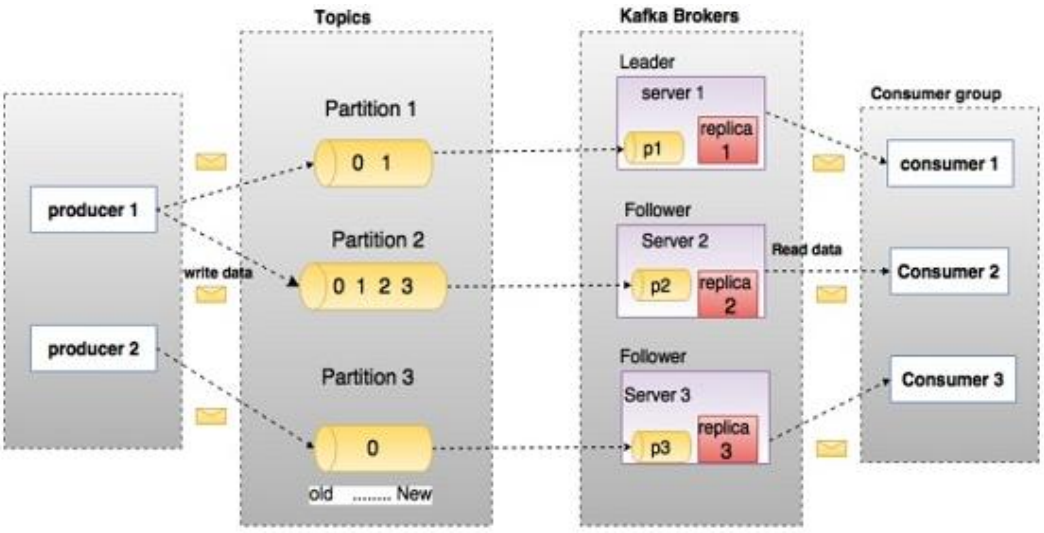

Figura 3 Árbol Sistema de cola de mensajes.
(Ukade, 2019)

La combinación de los patrones de EIP y Apache Kafka proporcionará una solución integral y robusta para optimizar la gestión de flujos de datos en el contexto de la arquitectura de integración.

Justificación

La realización de este trabajo de titulación se justifica por la relevancia y la necesidad de abordar los desafíos y obstáculos que surgen al integrar sistemas en entornos con grandes volúmenes de datos generados. Actualmente, la generación masiva de datos se ha hecho realidad en varios campos, como el comercio electrónico, las redes sociales, la industria, la salud y muchos otros. Como menciona (Algorri Álvarez, 2018) la disponibilidad de esta masiva cantidad de datos plantea oportunidades para los negocios y para crear valor de los mismo.

Justificación Tecnológica

Aumentar la investigación científica y mejorar la capacidad tecnológica de los sectores industriales de todos los países, en particular los países en desarrollo, entre otras cosas fomentando la innovación y aumentando considerablemente, de aquí a 2030, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y los gastos de los sectores público y privado en investigación y desarrollo. (Moran, 2020)

La justificación tecnológica se basa en el uso de herramientas y tecnologías avanzadas para abordar los desafíos de integración de sistemas en entornos con grandes volúmenes de datos. La implementación de un sistema middleware basado en Apache Kafka ofrece ventajas tecnológicas significativas.

Como menciona (Nebel, 2018) El diseño de una solución de integración es un desafío complejo en el que entran en juego diversos factores y existen múltiples opciones que pueden considerarse "correctas". A menudo, no se puede determinar si la arquitectura elegida fue acertada hasta varios meses o incluso años después, cuando los cambios y adiciones inevitables ponen a prueba la solidez de la arquitectura original. La justificación tecnológica de este trabajo de titulación radica en la necesidad de aplicar soluciones tecnológicas adecuadas y eficientes para superar los desafíos de la integración de sistemas en entornos con grandes volúmenes de datos generados.

El uso de Apache Kafka como tecnología principal permitirá diseñar e implementar una arquitectura de integración que optimice la gestión de flujos de datos y garantice una transmisión eficiente y confiable entre los sistemas involucrados. Además, la validación de esta arquitectura a través de un estudio de caso brindará evidencia empírica de su aplicabilidad y beneficios en este contexto específico.

Como menciona (Mollá Sirvent, 2016) las características destacables por las que se escoge Apache Kafka como middleware son:

- Rápido: Un solo Bróker puede trabajar con cientos de megabytes de lecturas y escrituras por segundo desde miles de clientes.
- Escalable: Un único clúster funciona como eje central de datos, ampliado sin tiempo muerto.
- Alto rendimiento: Tiene la capacidad de tolerar cientos de miles de mensajes por segundo, incluso mediante un hardware sencillo.
- Soporte para partición de mensajes: Se produce mediante los servidores de Kafka y el consumo distribuido en un clúster de máquinas consumidoras, manteniendo la ordenación por partición.

Riesgos

Riegos 1: Falta de experiencia en el uso de Apache Kafka.

Capacitación y formación adecuada sobre el uso de Apache Kafka, garantizando la adquisición de conocimientos necesarios para usar la plataforma eficazmente.

Riegos 2: Inadecuada configuración del clúster de Kafka.

Realizar una configuración exhaustiva del clúster de Kafka siguiendo las mejores prácticas recomendadas por la documentación oficial de Kafka y contar con expertos en la configuración de clústeres para garantizar un entorno estable y óptimo.

Riegos 3: Problemas de compatibilidad entre sistemas.

Realizar un análisis exhaustivo de los sistemas a integrar para identificar y abordar posibles problemas de compatibilidad antes de la implementación.

Riegos 4: Problemas de rendimiento y escalabilidad.

Realizar pruebas de rendimiento y escalabilidad durante la fase de desarrollo e implementar estrategias de optimización, como la partición adecuada de los datos y el uso de clústeres de Kafka con dimensionamiento apropiado para satisfacer las demandas del sistema.

Riegos 5: Inadecuada gestión de flujos de datos en tiempo real.

Diseñar una arquitectura de gestión de flujos de datos en tiempo real que tenga en cuenta la latencia y el rendimiento, utilizando técnicas como la segmentación y la optimización de la entrega de mensajes para asegurar una gestión eficiente de los flujos de datos en tiempo real.

Riegos 6: Fallas en la entrega confiable de mensajes.

Implementar mecanismos de confirmación y control de errores en la entrega de mensajes en Kafka, como el uso de confirmaciones síncronas y asíncronas, para garantizar la fiabilidad de la entrega de mensajes y la sincronización adecuada entre los sistemas integrados.

Riegos 7: Pérdida de datos o corrupción de la información.

Establecer políticas y procedimientos de respaldo y recuperación de datos, como la implementación de réplicas y copias de seguridad regulares, para minimizar el riesgo de pérdida o corrupción de datos. Además, realizar pruebas periódicas de los procedimientos de respaldo y recuperación para asegurar su eficacia.

Riegos 8: Pérdida de datos o corrupción de la información.

Al igual que en el punto anterior, implementar políticas y procedimientos de respaldo y recuperación de datos, asegurando la implementación de mecanismos de validación y

verificación de la integridad de los datos almacenados para detectar y corregir posibles corrupciones.

Tabla 1 Matriz de riesgos.
Elaborado por el autor

PROBABILIDAD	3	Alto		**R1:** Falta de experiencia en el uso de Apache Kafka	
	2	Media		**R2:** Inadecuada configuración del clúster de Kafka	**R3:** Problemas de compatibilidad entre sistemas **R4:** Problemas de rendimiento y escalabilidad **R5:** Inadecuada gestión de flujos de datos en tiempo real
	1	Baja			**R6:** Fallas en la entrega confiable de mensajes **R7:** Pérdida de datos o corrupción de la información **R8:** Pérdida de datos o corrupción de la información
			Bajo	Medio	Alto
			5	10	20
			IMPACTO		

CAPÍTULO 1
Marco teórico

1.1. Integración de sistemas

Con el crecimiento exponencial de la generación de datos y la creciente necesidad de procesar información de manera eficiente y en tiempo real, las tecnologías emergentes se han convertido en herramientas clave en el diseño e implementación de estrategias corporativas y organizacionales (Cahyaningrum, 2024). Estas tecnologías optimizan los procesos internos, permitiendo a las organizaciones adaptarse a entornos dinámicos, mantener su competitividad y responder de manera ágil a las demandas del mercado y los avances tecnológicos. Como menciona (Hyrynsalmi, 2022), el término "integración" puede resultar difícil de entender, ya que su significado puede variar según la perspectiva de cada persona. Esto se debe a que, a lo largo del tiempo, la forma de gestionar y desarrollar integraciones ha cambiado y se ha adaptado de distintas maneras.

Como señalan (Lagos et al., 2023), la integración de sistemas de gestión en las organizaciones contribuye a la optimización de recursos y a la mejora de la eficiencia operativa. Esto se traduce en una reducción de redundancias, mayor cohesión entre procesos y una respuesta más rápida y efectiva a las necesidades del entorno, lo cual facilita la alineación de los sistemas con los objetivos estratégicos de la organización, fortaleciendo su competitividad y asegurando un desempeño más eficiente y sostenible a largo plazo.

En este sentido, la interoperabilidad juega un papel crucial, ya que es esencial para la integración de sistemas de software heterogéneos. Facilita la comunicación y el intercambio de información entre sistemas basados en diferentes tecnologías. La interoperabilidad se desarrolla a través de niveles técnico, sintáctico, semántico y organizativo, asegurando la eficiencia y operatividad conjunta de los sistemas, lo que mejora la toma de decisiones y optimiza la eficiencia operativa en entornos complejos (Torres Ricaurte, 2019).

La integración de sistemas se puede abordar desde diferentes enfoques. Estos enfoques proporcionan diversas formas de interconectar sistemas y facilitar la comunicación entre ellos, permitiendo que las organizaciones elijan la estrategia más adecuada según sus necesidades y objetivos.

Como se observa en la Figura 4, los sistemas heterogéneos, que poseen características operativas distintas, pueden alcanzar una interoperabilidad efectiva mediante el uso de

estándares abiertos y middleware. Este enfoque permite interpretar y traducir las diferencias entre los sistemas, asegurando una interacción eficiente. La figura también ilustra cómo estos sistemas pueden adaptarse a cambios en el entorno, aumentando su resiliencia y capacidad de respuesta ante eventos catastróficos, lo que mejora su funcionalidad y diversidad operativa.

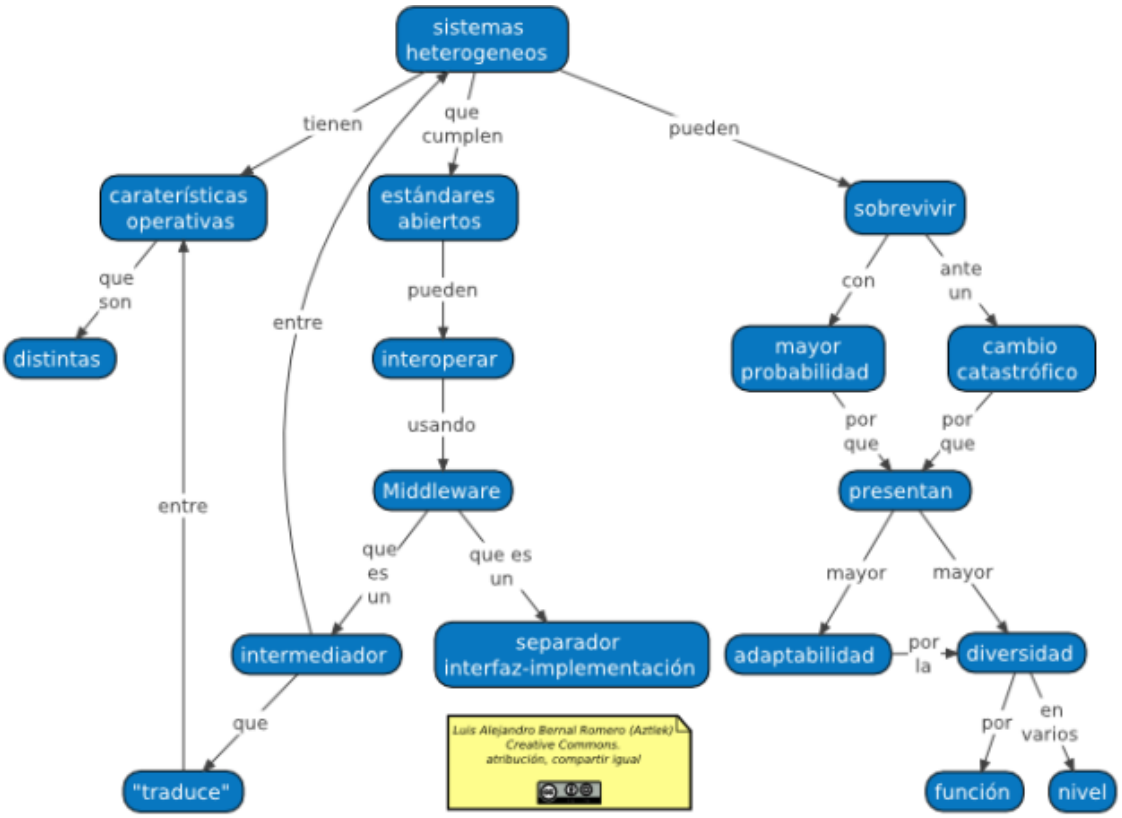

Figura 4 Mapa conceptual de la heterogeneidad de los sistemas.
(Aztlek, 2014)

1.1.1. Enfoque Point-to-Point (P2P)

El enfoque Point-to-Point es ampliamente reconocido como la forma más básica de construir una integración, ya que se basa en un esquema de datos que conecta directamente los sistemas de origen y destino a través de código. Este método facilita la transferencia de información de manera directa mediante una conexión individualizada para cada par de aplicaciones, lo que lo hace ideal para conectar únicamente dos sistemas de forma simple y efectiva (Hyrynsalmi, 2022).

Sin embargo, esta metodología presenta importantes limitaciones cuando se requiere expandir la integración para incluir más sistemas. A medida que aumenta el número de conexiones necesarias, la complejidad y el esfuerzo de mantenimiento se

incrementan de manera exponencial, lo que dificulta la escalabilidad del enfoque (Hyrynsalmi, 2022). Como se muestra en la Figura 5, cada sistema debe establecer múltiples conexiones individuales para comunicarse con los demás, lo que genera una red de dependencias difícil de gestionar y mantener.

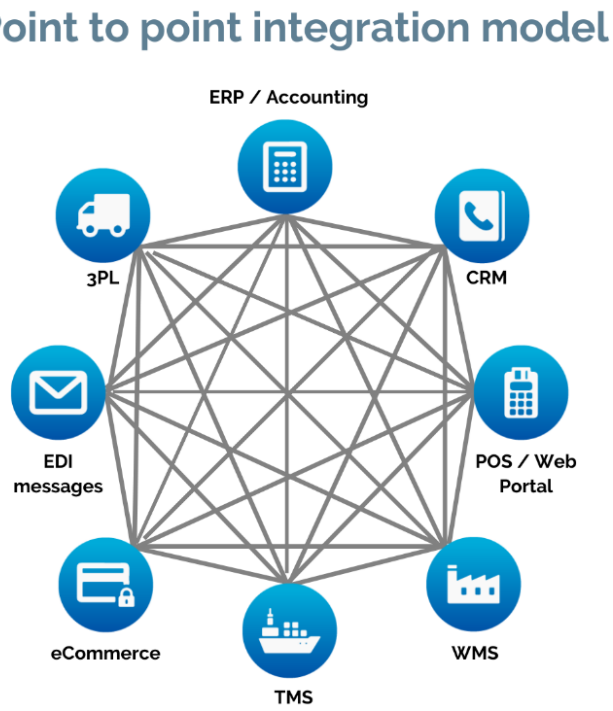

Figura 5 Point to point integration
(flowsoftware, 2022)

Con el paso del tiempo, estas limitaciones impulsaron la necesidad de implementar soluciones más eficientes en escenarios donde múltiples sistemas demandaban conectividad simultánea. Según (Laakkonen & Smolander, 2023), este desafío motivó el desarrollo de metodologías avanzadas como la Integración de Aplicaciones Empresariales (EAI). Este enfoque, mediante una arquitectura centralizada, permite la conexión de múltiples sistemas, abordando los problemas de escalabilidad y complejidad que son inherentes al modelo Point-to-Point.

1.1.2. Bus de servicios empresariales (ESB)

Un Enterprise Service Bus (ESB) es un entorno diseñado para facilitar la interconectividad entre servicios dentro de una organización. Como menciona (Pastrana Pardo et al., 2020) ESB establece una capa intermedia de procesamiento que resuelve problemas comunes relacionados con la confiabilidad, la escalabilidad y

las disparidades en las comunicaciones. Uno de los beneficios clave de la implementación de un ESB es la orquestación, entendida como la capacidad de coordinar y ejecutar de manera secuencial una serie de servicios independientes para llevar a cabo un proceso dentro de la organización (Aziz et al., 2020).

ESB se asocia con otros patrones arquitectónicos, como las comunicaciones asíncronas a través de colas, la transformación de formatos de datos en los mensajes y las fachadas de servicio, que proporcionan un punto único de inicio transaccional en el flujo de procesos (Pastrana Pardo et al., 2020). Como muestra la figura 6, el Enterprise Service Bus (ESB) actúa como una capa de integración centralizada que permite la interconexión de diferentes sistemas y aplicaciones en una organización.

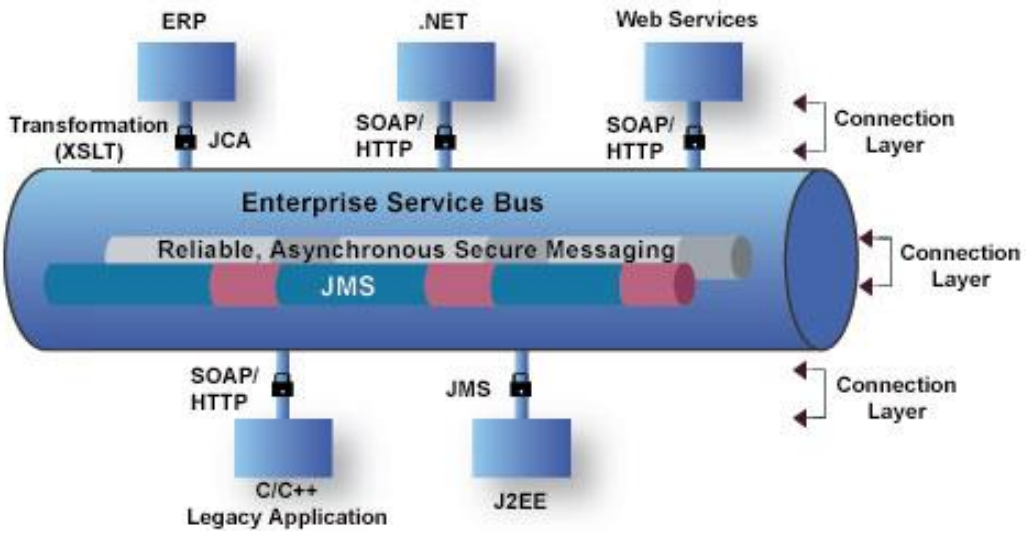

Figura 6 Enterprise Service Bus
(Tirabassi, 2021)

Las arquitecturas empresariales modernas, el Enterprise Service Bus (ESB) se configura como un componente fundamental para facilitar la integración y la interoperabilidad entre sistemas heterogéneos. Como menciona (Aziz et al., 2020) ESB proporciona una capa de abstracción que simplifica de manera significativa la integración de aplicaciones, permitiendo la comunicación eficiente entre sistemas diversos a través de un modelo de mensajería centralizado. Esta infraestructura de middleware posibilita el desacoplamiento efectivo de los servicios, lo que da lugar a una arquitectura más flexible y fácil de mantener, capaz de adaptarse ágilmente a los cambios en los requisitos empresariales.

1.1.3. Middleware

Como menciona (Krakowiak, 2009) un middleware es un software que se encuentra entre el sistema operativo y las aplicaciones, y su función principal es ayudar a que las aplicaciones puedan trabajar juntas, sin importar las diferencias en los sistemas o equipos donde se ejecutan. Facilita la comunicación y oculta la complejidad técnica, haciendo que los desarrolladores puedan centrarse en crear las funciones de sus aplicaciones sin preocuparse por cómo se conectan los sistemas entre sí.

El middleware cumple una función esencial al simplificar la comunicación y garantizar la interoperabilidad entre múltiples sistemas y aplicaciones en distintos contextos. En el ámbito de la Industria 4.0, actúa como un soporte clave para las redes inteligentes de máquinas y procesos, permitiendo diferentes grados de integración que influyen directamente en el diseño de las arquitecturas de software (Jepsen et al., 2021).

1.1.3.1. Middleware de Streaming

El middleware de streaming se define como un marco tecnológico que facilita la captura, procesamiento y análisis en tiempo real de datos provenientes de fuentes heterogéneas. Según (Akanbi & Masinde, 2020), el middleware de streaming se estructura como un marco distribuido que utiliza técnicas de big data para el análisis de datos en tiempo real. Estas plataformas emplean mecanismos de procesamiento de eventos que permiten realizar operaciones como filtrado, correlación y abstracción de datos, asegurando un alto rendimiento y una integración eficiente de información.

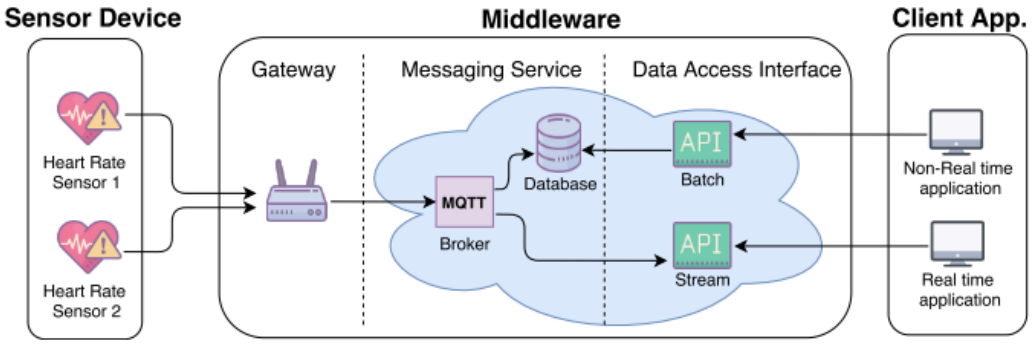

Figura 7 General system archutecture
(Bhawiyuga et al., 2020)

Este tipo de middleware permite manejar flujos continuos de datos, integrándolos de manera eficiente y garantizando la disponibilidad de los resultados de forma inmediata para diversas aplicaciones (Akanbi & Masinde, 2020). Destacando la capacidad de procesamiento distribuido, la escalabilidad y la compatibilidad con arquitecturas basadas en eventos.

En el ámbito de la computación moderna, el middleware de streaming se posiciona como una solución crítica para la gestión de sistemas distribuidos y basados en eventos. (Bhawiyuga et al., 2020) destaca que el middleware de streaming habilita el análisis continuo y simultáneo de grandes volúmenes de datos, contribuyendo significativamente a la eficiencia operativa de sistemas complejos. Por otra parte (García Sánchez et al., 2004), enfatiza que el middleware de streaming juega un rol esencial en el desarrollo de arquitecturas distribuidas resilientes y escalables. Estas arquitecturas permiten la integración de datos de diversas fuentes, tales como sensores IoT, aplicaciones web y sistemas de información, favoreciendo el análisis y la toma de decisiones en tiempo real.

Han surgido varias plataformas tecnológicas como ejemplos clave de middleware de streaming, entre las cuales se destacan Apache Kafka, Apache Flink y Apache Storm. Estas son conocidas por su habilidad para manejar flujos de datos distribuidos y enfrentar situaciones con grandes volúmenes de información y baja latencia. Según (Akanbi & Masinde, 2020), estas herramientas utilizan arquitecturas basadas en eventos, combinando motores de procesamiento de eventos y sistemas de mensajería para asegurar un rendimiento eficiente en entornos distribuidos.

1.2. Apache Kafka

Como menciona (Akanbi & Masinde, 2020) Apache Kafka es una plataforma de streaming de código abierto que se ofrece como parte de un modelo IaaS (Infraestructura como Servicio). A través de Kafka, es posible publicar y suscribirse a flujos de datos en tiempo real. Por esta razón, es fundamental comprender el concepto de transmisión de datos

1.2.1. Transmisión de datos.

La transmisión de datos hace referencia al flujo continuo y en tiempo real de información entre múltiples fuentes y destinos. Como menciona (Shukla, 2023) esta tecnología permite que las organizaciones procesen y analicen grandes volúmenes de datos de manera instantánea, lo que facilita la toma de decisiones basadas en la información más actual. En lugar de almacenar los datos para un procesamiento posterior, se procesan de inmediato en pequeños fragmentos. Con el aumento de la generación de datos, la capacidad de procesarlos en tiempo real se ha vuelto crucial. La transmisión de datos es esencial en arquitecturas de microservicios, ya que permite la comunicación en tiempo real entre servicios independientes, lo que optimiza el procesamiento de eventos conforme ocurren.

También se puede definir como una plataforma de mensajería, como menciona (Palino Todd et al., 2021) Apache Kafka es una plataforma de mensajería basada en el modelo de publicación/suscripción, diseñada para abordar problemas relacionados con el manejo y transmisión de datos en sistemas distribuidos diseñado para ofrecer un registro duradero de todas las transacciones, lo que permite la reconstrucción consistente del estado de un sistema. En Kafka, los datos se almacenan de manera persistente y en un orden específico, lo que permite su lectura determinista, es decir, los consumidores pueden acceder a ellos de manera predecible (Shukla, 2023).

1.2.2. Mensajes

Un mensaje en Kafka es un conjunto de bytes, sin un formato predefinido o un significado específico para Kafka (Palino Todd et al., 2021). Es decir, Kafka no interpreta ni transforma los datos contenidos en el mensaje, lo que le otorga una gran flexibilidad al sistema, permitiendo que se utilicen diversos formatos de datos, como JSON, Avro o incluso texto plano.

Cada mensaje dentro de Kafka puede tener una clave opcional, que también es un arreglo de bytes. Como menciona (Palino Todd et al., 2021) La clave no tiene un significado intrínseco dentro de Kafka, pero se usa principalmente para influir en cómo los mensajes se distribuyen a través de las particiones de un tema (topic). Utilizando una clave consistente, Kafka garantiza que los mensajes con la misma clave se escriban en la misma partición, lo que facilita el procesamiento de mensajes relacionados de manera ordenada y asegura la consistencia dentro de una partición.

1.2.3. Topics y Partitions

Los mensajes se organizan en "temas" (topics), que agrupan datos según atributos similares. Como menciona (Akanbi & Masinde, 2020) estos temas se asemejan a las tablas de una base de datos, pero sin las restricciones que suelen existir en estos sistemas. A diferencia de las bases de datos tradicionales, los temas en Kafka permiten una gran flexibilidad, ya que no están sujetos a las limitaciones de relaciones o restricciones de integridad. Cada tema es identificado de manera única por su nombre, lo que facilita la organización y acceso de los datos dentro del sistema. Además, los temas se dividen en particiones, lo que permite distribuir y ordenar los mensajes de manera eficiente.

Como se observar en la figura 8, un topic de Kafka se divide en múltiples particiones (en este caso, Partition 0, Partition 1 y Partition 2), lo que permite distribuir y ordenar los mensajes de manera eficiente. Cada partición mantiene una secuencia ordenada de mensajes, representada por los números secuenciales (0, 1, 2, 3...) que actúan como índices.

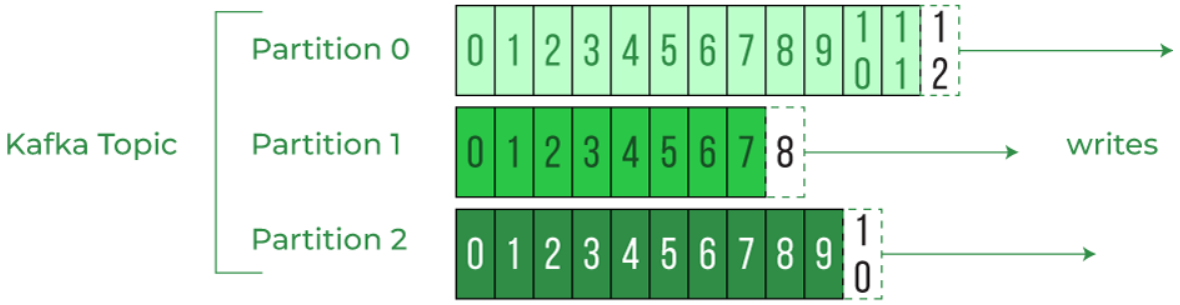

Figura 8 Topics, Partitions, and Offsets
(geeksforgeeks, 2022)

Una partición es una subdivisión de un topic y se puede comparar con un registro único en un log. Los mensajes se escriben en una partición de manera secuencial, en un formato "append-only", y se leen en el mismo orden en que fueron escritos. Como se ilustra en la figura 8, cada partición mantiene su propia secuencia numerada de mensajes, donde Partition 0 tiene mensajes del 0 al 12, Partition 1 del 0 al 8, y Partition 2 del 0 al 10. Dado que un topic puede estar compuesto por múltiples particiones, el orden de los mensajes no está garantizado a nivel de todo el topic, sino solo dentro de una partición específica. Las particiones permiten la escalabilidad y la redundancia

en Kafka, ya que pueden ser hospedadas en servidores diferentes, lo que permite distribuir la carga de trabajo y mejorar el rendimiento general del sistema (Akanbi & Masinde, 2020).

1.2.4. Productores y Consumidores

Los productores representan las fuentes primarias de datos o mensajes, encargadas de generar y enviar información a los temas correspondientes. Estos mensajes son posteriormente distribuidos en particiones y almacenados en los intermediarios del clúster. Como generadores de eventos, los productores pueden estar constituidos por sensores, dispositivos, o sistemas operados de manera manual o automática, evidenciando su versatilidad en distintos entornos tecnológicos (Akanbi & Masinde, 2020). Por defecto, los productores distribuyen los mensajes de manera uniforme entre las particiones de un tema, garantizando un equilibrio en la carga del sistema. Sin embargo, también es posible dirigir mensajes a particiones específicas utilizando una clave asociada al mensaje y un particionador que genera un hash de dicha clave. Esta estrategia permite que todos los mensajes relacionados con una misma clave se almacenen en la misma partición, preservando la coherencia en la distribución de los datos (Palino Todd et al., 2021).

Los consumidores son responsables de procesar los mensajes generados por los productores al suscribirse a uno o más temas específicos. Como menciona (Akanbi & Masinde, 2020) estos consumidores, también denominados suscriptores consumen los mensajes en el orden en que fueron producidos y utilizan un mecanismo denominado desplazamiento para llevar un registro preciso de los mensajes procesados. El desplazamiento, que es un valor entero único asignado a cada mensaje dentro de una partición, permite a los consumidores almacenar su progreso.

Como ejemplo se muestra en la Figura 9, la arquitectura de Apache Kafka implementa un modelo de mensajería distribuida donde los Productores (Productor1 y Productor2) actúan como fuentes primarias de datos, encargándose de generar y publicar mensajes hacia Topics o temas específicos, los cuales funcionan como categorías o canales de comunicación donde se organizan los mensajes por tipo o propósito. Estos mensajes son posteriormente consumidos por los Consumidores (Consumidor1, Consumidor2 y Consumidor3), que se suscriben a los temas de su interés y procesan la información en el orden en que fue producida, estableciendo así

un flujo de datos unidireccional y escalable desde los productores hasta los consumidores a través de los temas como punto central de coordinación.

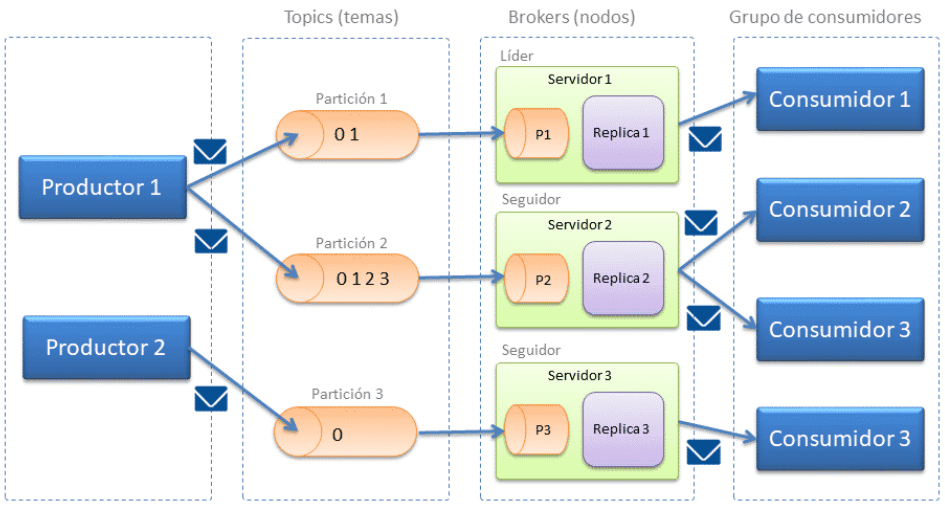

Figura 9 Arquitectura Kafka
(Diego, 2018)

1.2.5. Brokers y Clusters

Como se muestra en la Figura 10, un clúster de Apache Kafka constituye una infraestructura robusta compuesta por múltiples brokers interconectados, donde cada broker actúa como un servidor independiente dentro del ecosistema distribuido, cada broker posee un identificador numérico único que facilita su identificación y gestión dentro del clúster, permitiendo una organización eficiente de los recursos del sistema (Palino Todd et al., 2021).

Los brokers son responsables de la recepción y gestión de mensajes provenientes de los productores, a los cuales asignan identificadores secuenciales denominados offsets. Además, los brokers se encargan del almacenamiento persistente de estos mensajes en disco, garantizando la durabilidad de la información, y cada uno contiene particiones de temas (topics) que almacenan los datos de manera distribuida, lo que permite una gestión eficiente de grandes volúmenes de mensajes (Akanbi & Masinde, 2020).

Un aspecto destacable de la arquitectura del clúster es la presencia de un broker controlador, seleccionado automáticamente entre los miembros activos del clúster, que asume responsabilidades administrativas cruciales como la asignación dinámica

de particiones entre los brokers y la supervisión continua del estado de salud de los demás brokers.

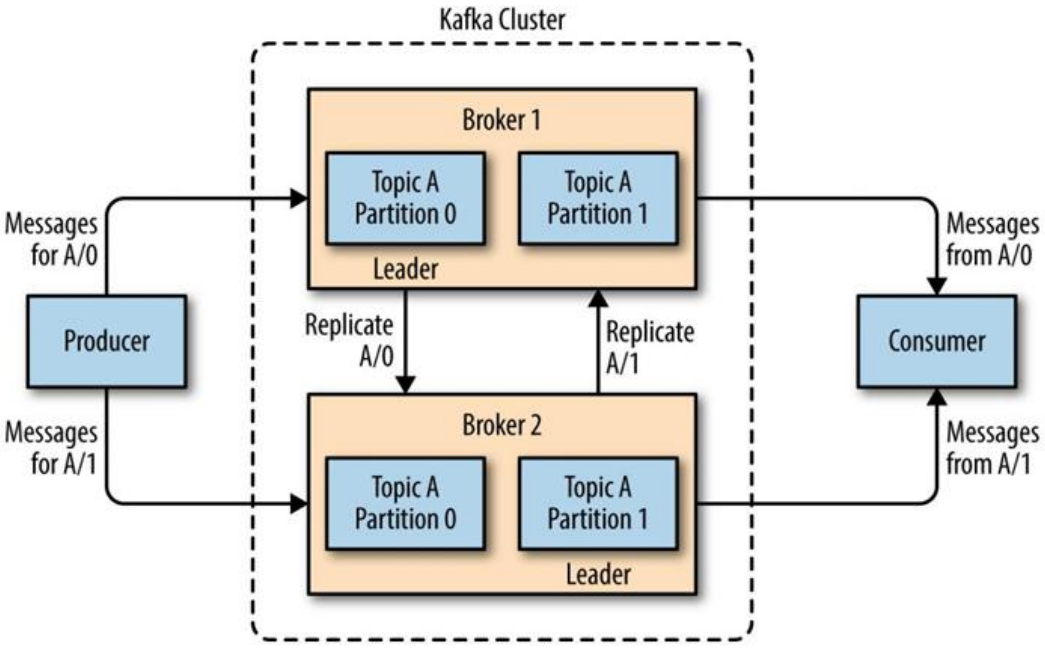

Figura 10 Replication of partitions in a cluster
(Palino Todd et al., 2021)

1.2.6. Replication

En Kafka, la replicación se implementa a nivel de partición de los topics. Cada partición se replica a través de múltiples brokers del clúster, donde uno de ellos actúa como líder (leader) y los demás como réplicas (followers). El líder es responsable de gestionar todas las operaciones de lectura y escritura para esa partición, mientras que las réplicas mantienen copias redundantes de los datos, sincronizándose continuamente con el líder, esto proporciona tolerancia a fallos, alta disponibilidad y redundancia de datos en el clúster. Este mecanismo garantiza que los eventos publicados en los topics sean persistidos de manera confiable y accesibles incluso en caso de fallos de nodos o particiones (Raptis & Passarella, 2022).

La replicación en Kafka no solo proporciona tolerancia a fallos y alta disponibilidad, sino que también mejora el rendimiento al distribuir las cargas de lectura y escritura entre múltiples brokers. Además, al mantener réplicas redundantes de los datos, Kafka facilita la recuperación ante desastres y garantiza la durabilidad de los eventos publicados en los topics (Palino Todd et al., 2021).

Como se muestra en la figura 11: el Tema 1 se divide en tres particiones (0, 1 y

2), cada una replicada en los tres brokers. Por ejemplo, el Broker 1 contiene el líder para la partición 0, el Broker 2 el líder para la partición 1 y el Broker 3 el líder para la partición 2. Cada broker también actúa como seguidor para particiones ubicadas en otros brokers, proporcionando redundancia.

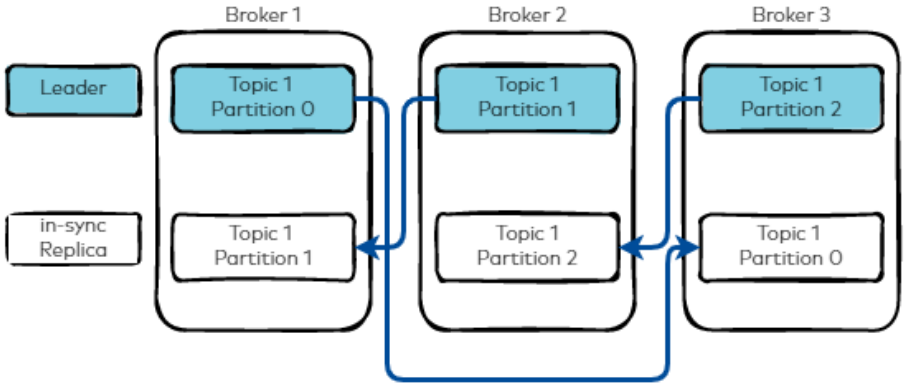

Figura 11. Kafka Replication and Committed Messages.
(confluent, s.f.)

1.3. Enterprise Integration Patterns (EIP)

Los Patrones de Integración Empresarial (EIP) representan un conjunto de principios y soluciones estandarizadas esenciales para diseñar arquitecturas de integración en entornos organizacionales, ofreciendo respuestas validadas a los desafíos comunes en la conexión de sistemas distribuidos (Hohpe & Woolf, 2003) . Estos modelos promueven la interacción no bloqueante entre aplicaciones con diferencias tecnológicas o funcionales, lo que posibilita un flujo efectivo de información y la gestión coordinada de operaciones organizacionales de alta complejidad.

En el contexto de la integración de sistemas, Apache Kafka emerge como una plataforma de mensajería distribuida que adopta el modelo de publicación-suscripción (pub/sub), diferenciándose de las soluciones convencionales mediante atributos innovadores en su diseño. Como señalan (Yadav, 2021), Kafka se fundamenta en un esquema de persistencia basado en un registro secuencial de eventos, donde los mensajes se almacenan de manera cronológica y permanente, garantizando su inmutabilidad tras ser registrados. Esta arquitectura, orientada a la escalabilidad horizontal y la tolerancia a fallos, permite no solo la transmisión asincrónica de datos en tiempo real, sino también la reconstrucción de estados históricos de los sistemas interconectados.

La implementación de estos patrones en Kafka ha mostrado alta eficacia en entornos de *Big Data* y procesamiento en tiempo real. Investigaciones como la (Hassan & Hassan,

2022) indican que organizaciones que integran Kafka con los Patrones de Integración Empresarial (EIP) logran disminuir considerablemente la latencia y aumentar la confiabilidad de sus sistemas. Esto se debe a que Kafka, al basarse en un registro persistente de mensajes inmutables y escalabilidad horizontal, materializa principios EIP como la comunicación asíncrona y la tolerancia a fallos. Su diseño permite manejar flujos masivos de datos con consistencia, garantizando la entrega ordenada y replicada de eventos, lo que fortalece la resiliencia en operaciones críticas. Así, Kafka no solo optimiza la integración en escenarios complejos, sino que se consolida como una solución estratégica alineada con estándares de arquitectura empresarial que priorizan la adaptabilidad y la eficiencia operativa.

1.4. Herramientas de monitoreo y pruebas de rendimiento

Las herramientas de monitoreo y pruebas de rendimiento son fundamentales en el desarrollo y mantenimiento de sistemas informáticos, ya que permiten evaluar, optimizar y garantizar la eficiencia de las aplicaciones bajo diversas condiciones de carga. Como menciona (Zapata & Cardona, 2011) estas herramientas permiten realizar pruebas de carga, identificar cuellos de botella y medir el comportamiento de las aplicaciones frente a distintos volúmenes de tráfico, lo cual es crucial para sistemas que manejan grandes cantidades de datos o requieren alta disponibilidad. Además, el monitoreo en tiempo real de las métricas operacionales facilita la detección proactiva de fallos y permite realizar ajustes inmediatos para mejorar la estabilidad y el rendimiento.

En este contexto, herramientas como Apache JMeter, Selenium, Prometheus y Grafana se destacan por su capacidad para evaluar tanto el rendimiento como la funcionalidad de los sistemas, integrando capacidades de automatización, monitoreo de métricas y visualización de resultados.

1.4.1. JMeter y Selenium

Apache JMeter es una herramienta de código abierto diseñada principalmente para la ejecución de pruebas de rendimiento en aplicaciones web. Como menciona (Ushakova et al., 2022) una de sus características más destacadas es su versatilidad, ya que soporta múltiples plataformas y protocolos, entre los cuales se incluyen objetos Java, Servlets, servidores FTP, HTTP, SOAP y scripts Perl, entre otros.

Como destacan (Indrianto, 2023), esta flexibilidad permite que JMeter sea una opción muy competitiva frente a soluciones comerciales, ofreciendo una alternativa accesible sin comprometer las capacidades de pruebas avanzadas. Sin embargo, a pesar de sus características destacadas, JMeter presenta ciertas limitaciones en cuanto a su capacidad de análisis en tiempo real, lo que hace recomendable su integración con herramientas adicionales de monitoreo, como Prometheus o Grafana, para obtener una visión más completa del rendimiento del sistema bajo prueba.

Selenium es un marco de automatización de pruebas de código abierto ampliamente reconocido por su capacidad para simular interacciones de usuarios en navegadores web como Chrome, Firefox, Edge y Opera. Como menciona (Do Nascimento et al., 2023) esta herramienta es esencial en el ámbito del aseguramiento de calidad (QA) debido a su enfoque en la automatización de pruebas funcionales y de regresión, garantizando que las aplicaciones web mantengan un funcionamiento estable y sin errores, incluso ante cambios frecuentes en el código.

Su flexibilidad y adaptabilidad son atributos clave, ya que permite escribir scripts de prueba en múltiples lenguajes de programación como Java, Python, C# y Ruby, lo que facilita su integración en diversos entornos de desarrollo. Asimismo, su capacidad para soportar múltiples navegadores y sistemas operativos lo convierte en una solución universal para proyectos que requieren pruebas en diferentes plataformas (García et al., 2020).

Selenium WebDriver permite una interacción directa y eficiente con los navegadores al utilizar sus capacidades de automatización nativa, lo que mejora el rendimiento y la precisión de las pruebas.

La integración de Selenium y JMeter combina las pruebas de carga con la automatización del navegador, lo que permite realizar pruebas exhaustivas de aplicaciones web. JMeter es una herramienta popular para pruebas de rendimiento, evaluando la capacidad y la estabilidad del sistema bajo varios niveles de carga (Kołtun & Pańczyk, 2020).Selenium, por su parte, destaca en la automatización de pruebas funcionales en (Do Nascimento et al., 2023). Al combinar estas herramientas, los evaluadores pueden evaluar tanto el rendimiento del sistema como la experiencia del usuario bajo cargas elevadas. Esta integración permite realizar pruebas, evaluando no solo la respuesta del sistema bajo cargas pesadas, sino también verificando la funcionalidad y la experiencia del usuario en el navegador, mejorando

así la calidad y confiabilidad general del software.

En la figura 12 se muestra como JMeter interactúa con varios Virtual Hosts para ejecutar pruebas de rendimiento en una Aplicación bajo prueba (AUT). JMeter, actuando como el Task Server, controla los Selenium Servers distribuidos en diferentes hosts virtuales. Cada Selenium Server lanza un navegador (como un cliente) para interactuar con la aplicación, simulando múltiples usuarios accediendo a ella. JMeter coordina y ejecuta las pruebas de carga, mientras que Selenium se encarga de automatizar la interacción con la aplicación, probando su funcionalidad y su rendimiento bajo diferentes condiciones.

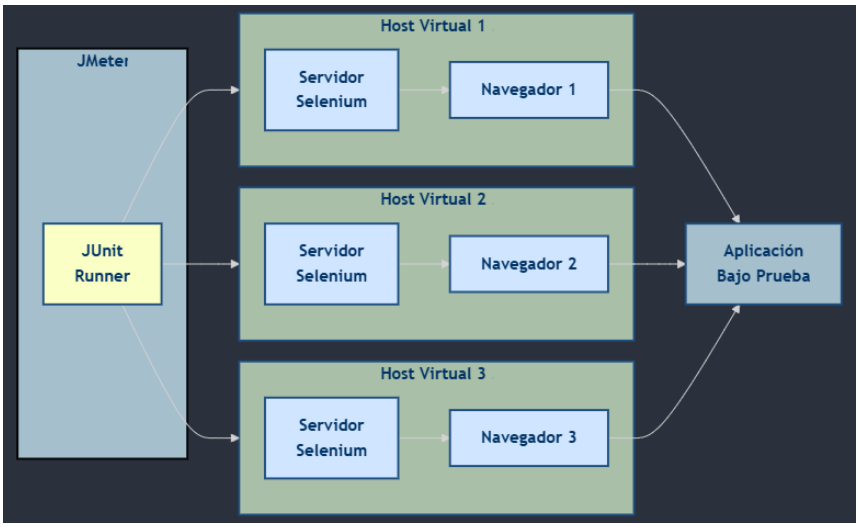

Figura 12. Interacción de Selenium en JMeter.
Elaborado por el autor.

1.4.2. Prometheus y Grafana

Prometheus es una plataforma de monitoreo de código abierto diseñada para recopilar y almacenar métricas de series temporales de manera eficiente, organizadas por etiquetas que permiten estructurar y consultar datos de forma óptima (Pai & Srinivas, 2024).

Este sistema recopila métricas directamente de trabajos instrumentados o a través de una puerta de enlace central, almacenando localmente las muestras extraídas para su posterior análisis. Además, aplica reglas a los datos para generar alertas o crear nuevas series temporales. Entre sus principales características destacan un modelo de datos multidimensional basado en pares clave/valor, el lenguaje de consultas PromQL, la independencia de nodos sin depender de un

almacenamiento distribuido, y un modelo de extracción basado en HTTP para la recolección de métricas (Dhane et al., 2024).

Estas capacidades hacen de Prometheus una herramienta fundamental para supervisar en tiempo real el rendimiento, la utilización de recursos y el estado de las aplicaciones.

Grafana es una plataforma de código abierto orientada al monitoreo y la visualización en tiempo real, diseñada para centralizar y analizar datos provenientes de diversas fuentes, como servidores, bases de datos, dispositivos IoT y aplicaciones empresariales. Como menciona (Mehdi et al., 2023) su funcionalidad principal radica en la capacidad de integrar diferentes sistemas en una única plataforma, proporcionando una visión integral de las infraestructuras y métricas clave. Con una interfaz intuitiva y adaptable, Grafana permite diseñar paneles interactivos personalizados que responden a necesidades específicas de monitoreo, posicionándola como una herramienta versátil para áreas como operaciones de TI, DevOps e inteligencia empresarial. Además, destaca por su diseño enfocado en la eficiencia, la escalabilidad y el rendimiento.

Grafana es un software de visualización y análisis de código abierto. Le permite consultar, visualizar, alertar y explorar métricas sin importar dónde estén almacenadas. En general, es una herramienta que convierte los datos de la base de datos de series temporales en hermosos gráficos y visualizaciones que serán más fáciles de analizar y entender (Kumar, 2021).

La figura 13 muestra cómo trabajan juntos Prometheus y Grafana para monitorear sistemas. Prometheus funciona como un recolector de información, visitando regularmente diferentes partes del sistema (Process A y Process B) para recoger datos sobre su funcionamiento, como si fuera un inspector que hace rondas periódicas. Por su parte, Grafana actúa como una pantalla informativa que obtiene estos datos de Prometheus y los presenta de forma visual y fácil de entender a través de sus dashboards o paneles de control.

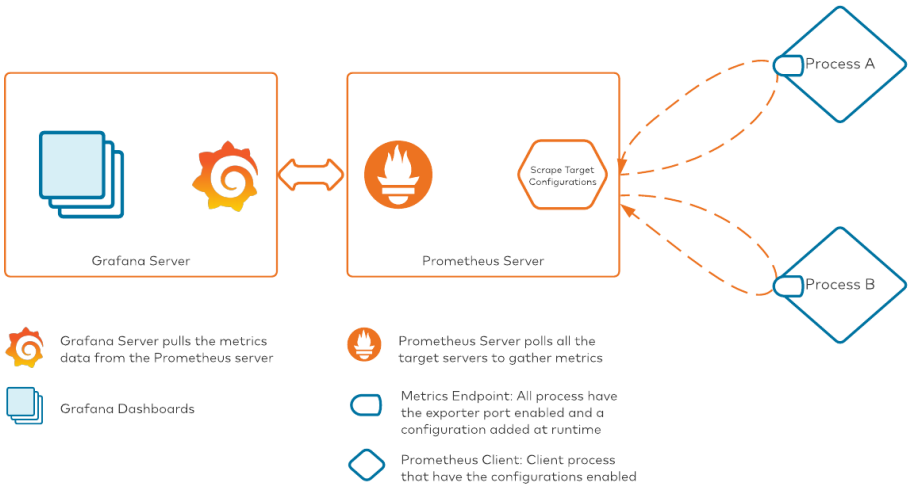

Figura 13 Connecting Grafana to Prometheus
(confluent, 2021)

1.5. ISO/IEC 25010

La norma ISO/IEC 25010, que forma parte del conjunto de normas ISO 25000, tiene como propósito establecer criterios de calidad claros y objetivos para la evaluación y el análisis de productos de software. Como menciona (Andi et al., 2024) este estándar internacional se utiliza para valorar la calidad de los sistemas de información, enfocándose en las características que definen el rendimiento y las capacidades de un producto. La aplicación de la norma ISO/IEC 25010 en la evaluación de sistemas de información se basa en la calidad del producto, entendiendo el proceso como las características que definen el sistema de información. Así, la norma ofrece un marco que permite analizar el software a partir de diversas dimensiones, tales como funcionalidad, fiabilidad, usabilidad, eficiencia de desempeño, mantenibilidad, portabilidad, seguridad y compatibilidad (Suparto & Dai, 2021).

1.5.1. Eficiencia de Desempeño

La norma ISO/IEC 25010 define la Eficiencia de Desempeño como un atributo clave en la calidad de los sistemas de software, que evalúa su capacidad para operar de manera adecuada en relación con el uso de los recursos disponibles bajo condiciones específicas. Este atributo considera aspectos como el tiempo de respuesta y el consumo de recursos, asegurando que el software funcione de manera óptima y eficiente. Su objetivo es garantizar que el sistema no solo cumpla con sus funciones, sino que lo haga utilizando la cantidad mínima de recursos necesarios, manteniendo un equilibrio entre rendimiento y estabilidad (Andi et al., 2024).

Como se menciona en (iso25000, 2018) la eficiencia de desempeño de un sistema de software se define como su capacidad para ejecutar sus funciones dentro de los parámetros de tiempo y rendimiento establecidos, utilizando de manera óptima los recursos disponibles, como CPU, memoria, almacenamiento y energía, bajo condiciones específicas. Esta característica se organiza en tres subcaracterísticas principales:

Comportamiento temporal, que evalúa el grado en que el sistema cumple con los requisitos de tiempo de respuesta y tasa de rendimiento al ejecutar sus funciones.

Utilización de recursos, que mide la cantidad y el tipo de recursos utilizados durante la ejecución, asegurando que estos no excedan los valores especificados bajo las condiciones definidas.

Capacidad, que analiza si el sistema satisface los requisitos relacionados con los límites máximos de ciertos parámetros, como el número de elementos almacenados, usuarios concurrentes o el ancho de banda de comunicaciones.

CAPÍTULO 2
Desarrollo

2.1. Rocky Linux

Para la implementación de Apache Kafka, se optó por utilizar Rocky Linux como sistema operativo en el servidor, debido a su bajo consumo de recursos, lo que permite optimizar el desempeño del middleware al destinar más capacidad al manejo de mensajes sin comprometer el rendimiento del hardware. Esta elección se basa en que Rocky Linux es una distribución optimizada para servidores, derivada de RHEL (Red Hat Enterprise Linux), que proporciona un entorno estable con un uso eficiente de los recursos. Rocky Linux es conocido por su alta compatibilidad con tecnologías empresariales, lo que lo hace adecuado para entornos de producción que requieren estabilidad y rendimiento.

Rocky Linux se instaló en una máquina virtual, garantizando así un entorno aislado y flexible para la configuración y ejecución de Apache Kafka.

Con este enfoque, se eligió la versión 8.10 Boot ISO como se muestra en la figura 14, que ofrece una instalación minimalista. Esto permitió incluir únicamente los paquetes necesarios para el entorno de desarrollo, reduciendo la sobrecarga del sistema y minimizando posibles vulnerabilidades. Esta configuración asegura un servidor ligero y eficiente, ideal para soportar las operaciones críticas que requiere Apache Kafka.

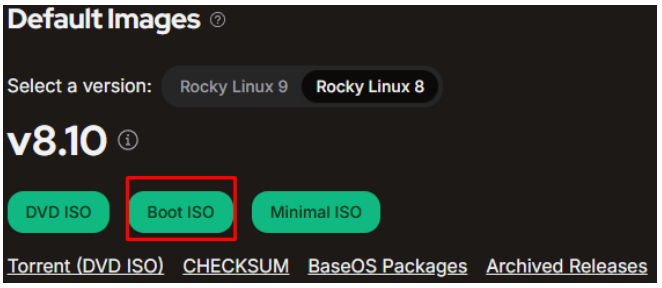

Figura 14. Rocky Linux Boot ISO.
Elaborado por el autor.

2.1.1. Configuración de Paquetes en Rocky Linux

Para complementar la instalación base de Rocky Linux y asegurar que el sistema estuviera listo para ejecutar Apache Kafka de manera eficiente, se instalaron algunos paquetes y repositorios adicionales:

a) dnf-utils:

Se instaló el paquete dnf-utils, que proporciona herramientas útiles para gestionar los repositorios y realizar tareas de administración de paquetes. Esto facilita la administración de dependencias y la configuración de los repositorios necesarios para instalar otros paquetes. Para instalarlo, se ejecutó el siguiente comando:

```
sudo dnf install dnf-utils
```

b) EPEL:

Para obtener paquetes adicionales que no están disponibles en los repositorios oficiales de Rocky Linux, se habilitó el repositorio EPEL (Extra Packages for Enterprise Linux). Esto permitió acceder a una mayor variedad de herramientas y software. La instalación del repositorio se realizó con el siguiente comando:

```
sudo dnf install epel-release
```

c) openssh

Se instaló el paquete openssh-server para habilitar el acceso remoto al servidor Rocky Linux mediante el protocolo SSH. Esta herramienta resulta esencial para la administración del entorno de forma segura, permitiendo realizar configuraciones y monitoreo del sistema sin necesidad de acceso físico a la máquina. La instalación se realizó utilizando el siguiente comando:

```
sudo dnf install -y openssh-server
```

2.2. Apache Kafka

Para la implementación de Apache Kafka, se selecciona la versión 3.4.1 debido a su estabilidad y compatibilidad con los requisitos del sistema. Para obtener esta versión, se accede al sitio web oficial de Apache Kafka, en la sección de Downloads, donde se busca la versión 3.4.1. En la subsección de Binary Downloads, se elige el paquete correspondiente a Scala 2.12, ya que es la versión recomendada para garantizar el correcto funcionamiento y la optimización del sistema.

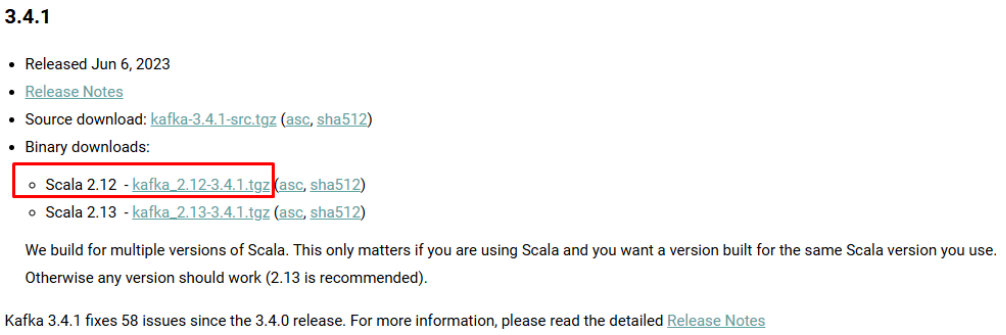

Figura 15. Apache Kafka 3.4.1.
Elaborado por el autor.

2.2.1. Instalación y configuración de Apache Kafka

Para facilitar la administración del servidor y mejorar la eficiencia en tareas como copiar y pegar comandos o configuraciones, se recomienda realizar las acciones desde otra máquina a través de SSH. Este enfoque simplifica el trabajo, ya que en algunos casos puede resultar complicado utilizar estas funciones directamente en la terminal de Rocky Linux. Al conectarse de forma remota, se mejora la experiencia al gestionar el servidor, especialmente cuando se deben realizar configuraciones o ejecutar comandos repetitivos.

a) Instalación Java.

Apache Kafka es una aplicación basada en Java, por lo que Java debe estar instalado en su servidor. Ahora ejecute el siguiente comando a continuación para instalar Java en su sistema:

Ejecutar

```
sudo dnf install java-11-openjdk
```

Verificar que se instaló correctamente:

```
java –version
```

b) Usuario para Apache Kafka.

Antes de crear un nuevo usuario tenemos que crear sus directorios, en este caso vamos a ocupar opt y kafka.

Creación de directorios

```
sudo mkdir /opt
```

```
sudo mkdir /opt/kafka
```

Crear nuevo usuario

```
sudo useradd -r -d /opt/kafka -s /usr/sbin/nologin kafka
```

Este comando crea un nuevo usuario del sistema llamado kafka. La opción -r indica que el usuario es del tipo de sistema (sin capacidad de inicio de sesión), -d /opt/kafka especifica el directorio principal del usuario como /opt/kafka, y -s /usr/sbin/nologin establece la shell de inicio de sesión del usuario como nologin, lo que impide que el usuario inicie sesión en el sistema.

c) Instalación de Apache Kafka.

Descarga del archivo.

```
sudo curl -fsSLo kafka.tgz https://archive.apache.org/dist/kafka/3.4.1/kafka_2.12-3.4.1.tgz
```

Utilizamos curl como herramienta para la transferencia de datos desde o hacia un servidor. La opción -f activa el modo de fallo silencioso para que no se muestren errores de descarga, -s silencia el progreso de la descarga, -S muestra mensajes de error si los hay, -L sigue redirecciones, y -o guarda el archivo descargado con el nombre que seleccionemos, en este caso, kafka.tgz.

Extraemos y movemos los archivos a nuestra carpeta del usuario.

```
tar -xzf kafka.tgz
```

```
sudo mv kafka_2.12-3.4.1/* /opt/kafka
```

Cambia el propietario y grupo de /opt/kafka y todos sus contenidos a kafka, de forma recursiva.

```
sudo chown -R kafka:kafka /opt/kafka
```

Creación de un directorio para guardar los logs.

```
sudo -u kafka mkdir -p /opt/kafka/logs
```

Editar el archivo server.properties.

```
sudo -u kafka nano /opt/kafka/config/server.properties
```

Buscamos la linea log.dirs y ponemos la ruta del directorio que creamos.

```
log.dirs=/opt/kafka/logs
```

d) Creación de Systemd para Apache Kafka y Zookeper

Creamos el servicio de Zookeper en systemd.

```
sudo nano /etc/systemd/system/zookeeper.service
```

Añadimos las siguientes instrucciones.

```
[Unit]

Requires=network.target remote-fs.target

After=network.target remote-fs.target


[Service]

Type=simple

User=kafka

ExecStart=/opt/kafka/bin/zookeeper-server-start.sh
/opt/kafka/config/zookeeper.properties

ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh

Restart=on-abnormal


[Install]

WantedBy=multi-user.target
```

Creamos el servicio de Apache Kafka en systemd.

```
sudo nano /etc/systemd/system/kafka.service
```

Añadimos las siguientes instrucciones.

```
[Unit]
```

```
Requires=zookeeper.service

After=zookeeper.service

[Service]

Type=simple

User=kafka

ExecStart=/bin/sh -c '/opt/kafka/bin/kafka-server-start.sh
/opt/kafka/config/server.properties > /opt/kafka/logs/start-
kafka.log 2>&1'

ExecStop=/opt/kafka/bin/kafka-server-stop.sh

Restart=on-abnormal


[Install]

WantedBy=multi-user.target
```

Guardamos e iniciamos nuestros nuevos servicios.

```
sudo systemctl daemon-reload

sudo systemctl start zookeeper

sudo systemctl start kafka
```

Recargamos la configuración de systemd para aplicar cualquier cambio reciente en los archivos de servicio.

Iniciamos el servicio de Zookeeper utilizando systemd.

Iniciamos el servicio de Kafka utilizando systemd.

Deshabilitar SELinux Temporalmente

En algunos casos, las políticas de seguridad de SELinux pueden interferir con el funcionamiento de Apache Kafka. Para solucionar este problema temporalmente, puedes poner SELinux en modo permisivo

```
sudo setenforce 0
```

e) Configuración de red en Apache Kafka

Para permitir que Apache Kafka sea accesible desde otras máquinas, es necesario configurar los parámetros de red adecuados en su archivo de configuración.

Antes debemos saber la IP de nuestro Rocky Linux, esto lo hacemos mediante:

```
ip a
```

Revisar y modificar el archivo server.properties:

```
sudo nano /opt/kafka/config/server.properties
```

Cambia la dirección localhost por 0.0.0.0 para que Kafka escuche en todas las interfaces de red disponibles. Así podrá aceptar conexiones desde cualquier máquina.

Especifica la IP de la máquina de Kafka (en este caso 192.168.0.104) para que los clientes sepan a qué dirección conectarse.

```
listeners=PLAINTEXT://0.0.0.0:9092

advertised.listeners=PLAINTEXT://192.168.0.104:9092
```

Configurar el firewall para permitir conexiones

```
sudo firewall-cmd --zone=public --add-port=9092/tcp --permanent

sudo firewall-cmd --reload
```

f) Creación del tema

Para comenzar a trabajar con Apache Kafka, es necesario crear un tema

donde se enviarán y recibirán los mensajes. En este caso, vamos a crear un tema llamado factura.

```
/opt/kafka/bin/kafka-topics.sh --create --topic facturas --
bootstrap-server localhost:9092 --partitions 1 --replication-
factor 1
```

El parámetro --create indica que estamos creando un nuevo tema en Apache Kafka. Al usar --topic factura, estamos especificando el nombre del tema que queremos crear, en este caso, factura. Con --bootstrap-server localhost:9092, definimos el servidor y puerto donde está corriendo Apache Kafka. Es importante asegurarse de usar la dirección correcta de tu servidor Kafka. El parámetro --partitions 1 establece el número de particiones del tema, en este caso, se utiliza una sola partición. Finalmente, --replication-factor 1 define la cantidad de réplicas para el tema, y en este caso, para pruebas locales, se utiliza una sola réplica.

2.3. Sistemas que comparten un alto volumen de mensajes

Para el desarrollo se tiene que tomar en cuenta que deben ser sistemas que compartan un alto volumen de mensajes.

Para la selección de los sistemas que compartirán el alto volumen de mensajes, se han tomado en consideración dos casos de estudio relevantes utilizados en la materia de software empresarial. Estos casos han sido elegidos debido a su representatividad y complejidad, lo que permite una evaluación del middleware en un entorno de integración real. Ambos casos ilustran escenarios comunes en el ámbito empresarial donde la eficiencia y la capacidad de manejar grandes volúmenes de datos son cruciales. Estos sistemas no solo destacan por su demanda de alto rendimiento, sino también por la necesidad de una comunicación fluida y consistente entre diversos componentes del ecosistema empresarial.

2.3.1. Dependencias

Para la implementación del sistema de integración basado en Apache Kafka, se

requieren dependencias que permiten su correcto funcionamiento.

Todas estas dependencias las podemos encontrar cuando descargamos Apache Kafka como en la figura 15, dentro de la carpeta libs.

a) Apache Kafka Client

jackson-core, jackson-annotations y jackson-databind son necesarias para la serialización y deserialización de objetos Java a formato JSON y viceversa. Este formato fue seleccionado para los mensajes enviados a través de Kafka debido a su legibilidad, flexibilidad y compatibilidad con sistemas distribuidos.

b) SLF4J

slf4j-api y slf4j-reload4j proveen una capa de abstracción para el registro de eventos dentro del sistema. Estas herramientas son utilizadas para la monitorización y depuración del código, facilitando la identificación de problemas durante la ejecución.

c) Zstandard JNI

zstd-jni proporciona soporte para la compresión de mensajes utilizando el algoritmo Zstandard, un método eficiente que permite reducir el tamaño de los datos transmitidos. Kafka utiliza esta biblioteca de manera nativa para optimizar el uso de ancho de banda y mejorar el rendimiento general del sistema.

Es importante incluir las dependencias en Assembly Directive como se muestra en la figura 16 para garantizar que todas las dependencias estén correctamente disponibles al momento de empaquetar y desplegar el proyecto. Asegurando que el proyecto sea funcional, portátil y mantenible, y evita problemas relacionados con dependencias en diferentes entornos.

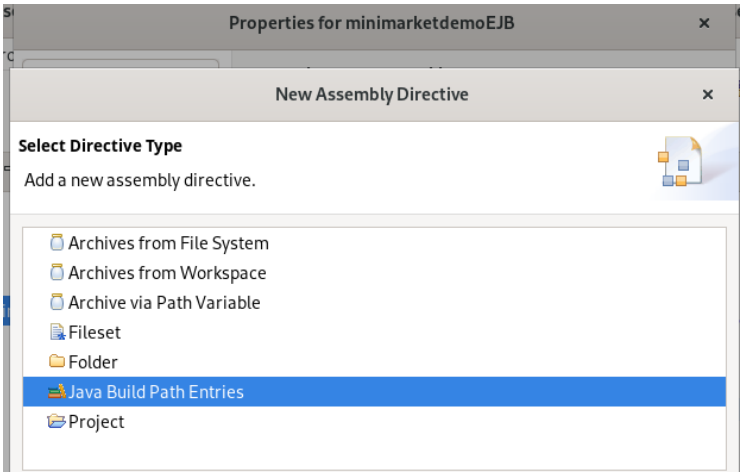

Figura 16 Assembly Directive
Elaborado por el autor.

Para ambos sistemas se creó una configuración tanto para el productor como para el consumidor de Kafka de manera modular ocupando Java, esto con el fin de crear una estructura que interactúe con kafka de a través de configuraciones de propiedades, y permiten tanto enviar mensajes como consumirlos.

2.3.2. Clases para configuración de Kafka

a) KafkaClientBase

Esta clase tiene como objetivo proporcionar una base común para la configuración de Kafka. Tiene un campo properties que es de tipo Properties, que contiene las configuraciones necesarias para la conexión con el servidor de Kafka.

b) KafkaClientConfig

La clase KafkaClientConfig tiene como función principal crear y configurar las propiedades necesarias para establecer las conexiones con Kafka, tanto para el productor como para el consumidor. Específicamente, la clase proporciona dos métodos:

- createProducerProperties(): Para configurar el productor Kafka (con el manejo de la política de "acks" y los serializadores necesarios).
- createConsumerProperties(): Para configurar el consumidor Kafka, gestionando configuraciones como el grupo de consumidores, el auto-commit de offsets y el comportamiento de lectura (leer desde el

principio en caso de no encontrar offsets).

Estas configuraciones están vinculadas al comportamiento del sistema, ya que controlan cómo se interactúa con Kafka en términos de enviar y recibir mensajes.

c) KafkaConsumerService

La clase KafkaConsumerService es el servicio responsable de consumir mensajes de Kafka. Al heredar de KafkaClientBase, esta clase reutiliza la configuración básica proporcionada por la clase base y se especializa en la lógica de consumo de los mensajes.

Esta clase implementa la interfaz Runnable, lo que permite que el servicio se ejecute de manera asíncrona o en un hilo separado, ideal para escenarios de producción, donde se requiere que el consumidor esté siempre activo y recibiendo mensajes. La lógica de consumo se implementa en el método consume(), que recoge los mensajes del topic y los procesa según se necesite. Además, el método consumeAllMessages() asegura que el consumidor siga recibiendo mensajes hasta que ya no haya más disponibles.

d) KafkaProducerService

La clase KafkaProducerService es la contraparte del consumidor, encargada de enviar mensajes a Kafka. Al igual que el consumidor, esta clase hereda de KafkaClientBase para obtener las configuraciones comunes necesarias. La lógica de producción de mensajes se implementa en el método send(), que envía un mensaje a un topic dado, gestionando el uso de la clave del mensaje y su valor. El uso de un Callback permite manejar la respuesta del servidor Kafka (ya sea éxito o error).

2.3.3. Sistema de facturación

El sistema de facturación, desarrollado como un caso de estudio utilizando Java EE, se ejecuta en un servidor WildFly. Este sistema, que será utilizado como el productor de mensajes en nuestro estudio, adaptando su arquitectura para manejar grandes volúmenes de transacciones diarias.

La elección de este sistema se debe a su arquitectura flexible, que se adapta a nuestras necesidades específicas, y a su historial de éxito en estudios previos. Además, se ha decidido que el volumen de mensajes generado corresponda a las facturas, lo que asegura una evaluación precisa y relevante del middleware en términos de producción de mensajes.

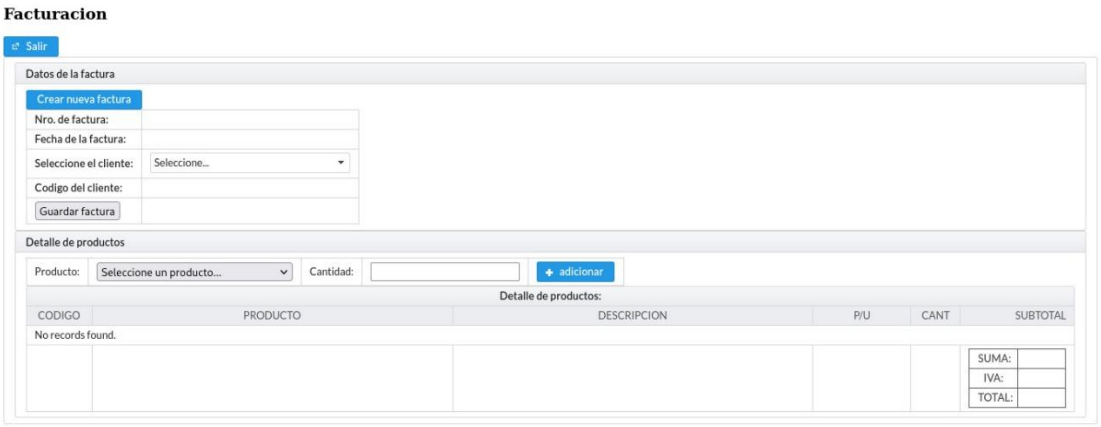

Figura 17. Vista – Facturación
Elaborado por el autor.

Una vez que la factura ha sido creada, el sistema procede a enviarla como un mensaje al tema denominado “facturas” en Apache Kafka. Este proceso implica consideraciones desde el punto de vista de la arquitectura del sistema.

a) Uso de un Data Transfer Object (DTO).

Transformamos la entidad FacturaCab en un DTO (Data Transfer Object) antes de enviarla para garantizar una adecuada separación de responsabilidades dentro del sistema, evitando exponer directamente las entidades JPA.

b) Reutilización de Clases Existentes para la Conexión con Kafka.

La implementación de clases previamente diseñadas para manejar la conexión con Apache Kafka facilita el proceso de envío del mensaje. Estas clases encapsulan la configuración y lógica necesarias para interactuar con el broker de Kafka.

c) Transformación del DTO a un Mensaje JSON.

Transformar el DTO en un mensaje JSON antes de enviarlo a Kafka. Esta transformación se realiza utilizando la librería ObjectMapper de Jackson, que permite convertir el objeto DTO en una cadena JSON.

El proceso de envío de una factura al tema “facturas” consta de:

1. Se crea una instancia de la clase FacturaCabDTO y se mapean los valores desde la entidad FacturaCab.
2. Se utiliza la clase KafkaClientConfig para generar las propiedades necesarias para el productor
3. La instancia del DTO se convierte en una cadena JSON mediante la librería ObjectMapper.
4. Con la clase KafkaProducerService, el mensaje JSON se envía al tema “facturas”. Este proceso incluye la configuración del productor, el envío del mensaje y el cierre del productor una vez completada la operación.

```
FacturaCabDTO factura =  new FacturaCabDTO();
FacturaCabDTO.Serializar(factura, facturaCabTmp);
KafkaClientConfig config = new KafkaClientConfig();
Properties producerProperties = config.createProducerProperties("all");
KafkaProducerService kafkaProducerService = new KafkaProducerService(producerProperties);

try {

    ObjectMapper objectMapper = new ObjectMapper();
    String jsonString = objectMapper.writeValueAsString(factura);
    kafkaProducerService.send("facturas", jsonString);

} catch (Exception e) {
    e.printStackTrace();

} finally {
    kafkaProducerService.close();

}
actualizarContFacturas(contFacturas);
actualizarContFacturasDet(contFacturasDet);
facturaCabTmp=null;
```

Figura 18. Fragmento de código, productor
Elaborado por el autor.

2.3.4. Minimarketdemo

El sistema de minimarketdemo, también desarrollado como un caso de estudio con Java EE, será utilizado como el consumidor de estos mensajes. La elección de este sistema se basa en su uso en las materias de desarrollo de software para MiPymes y desarrollo de software empresarial, lo que proporciona la ventaja de una familiaridad previa. Además, este sistema es escalable, lo que lo hace adecuado para manejar la carga de mensajes generados por el sistema de facturación.

Figura 19. Vista – minimarketdemo
Elaborado por el autor.

Como se muestra en la Figura 20 se presenta una implementación de un consumidor de Kafka diseñado para procesar mensajes de manera continua y asincrónica, destacando aspectos clave en su arquitectura y funcionalidad. En primer lugar, el método run() sobrescribe el comportamiento estándar de un hilo, estableciendo un bucle condicionado por la variable running, lo que garantiza una ejecución persistente mientras la aplicación esté activa.

Dentro de este bucle, se realiza una consulta periódica (poll) al servidor de Kafka con intervalos de 10 milisegundos, optimizando la eficiencia en la recuperación de registros sin bloquear recursos críticos. Al recibir mensajes, estos son procesados iterativamente: cada registro se imprime, almacena en una lista de mensajes (messages), y se deserializa mediante un objectMapper en un objeto FacturaCabDTO, facilitando su manipulación en formato de datos estructurados. La invocación del método tiempo() con el número de factura sugiere una integración con lógica de negocio para gestionar tiempos o validaciones

específicas. Adicionalmente, el manejo de excepciones dentro de un bloque try-catch asegura la resiliencia del servicio, capturando errores sin interrumpir el flujo principal.

```java
@Override
public void run() {
    while (running) {
        try {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(10));

            if (!records.isEmpty()) {
                for (ConsumerRecord<String, String> record : records) {
                    messages.add(record.value()); // Agregar el valor del mensaje a la lista
                    factura = objectMapper.readValue(record.value(), FacturaCabDTO.class);

                    tiempo(factura.getNumeroFactura());
                }


            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    close();
}
```

Figura 20. Fragmento de código – Consumidor mensajes
Elaborado por el autor.

Por otro lado en la figura 21, el método inicializar(), anotado con @PostConstruct, coordina la configuración inicial del consumidor, definiendo propiedades específicas de Kafka (como el grupo de consumidores y el tema "facturas") y lanzando el servicio en un hilo independiente, lo que permite una operación concurrente y no bloqueante. La inicialización explícita de la lista facturas como ArrayList garantiza la disponibilidad de una estructura mutable para almacenar los datos consumidos.

```java
@PostConstruct
public void inicializar() {
    System.out.println("inicializa");
    KafkaClientConfig config = new KafkaClientConfig();
    Properties consumerProperties = config.createConsumerProperties("factura-co2nsumer-group");

    consumerService = new KafkaConsumerService(consumerProperties, "facturas");


    facturas = new ArrayList<>();
    new Thread(consumerService).start();
}
```

Figura 21. Fragmento de código – Consumidor
Elaborado por el autor

CAPÍTULO 3
Validación de resultados

3.1. Descripción de los componentes

3.1.1. Productor

Para llevar a cabo la producción de mensajes, se utilizó el sistema operativo Fedora, se utilizó una laptop con las siguientes especificaciones técnicas.

a) Procesador (CPU)

El procesador utilizado en el sistema de prueba es un AMD Ryzen 5 3450U con Radeon Vega Mobile Gfx. Las características detalladas del procesador son las siguientes:

- Nombre: AMD Ryzen 5 3450U with Radeon Vega Mobile Gfx
- Número de núcleos: 4
- Número de procesadores lógicos: 8
- Velocidad de reloj máxima: 2100 MHz

b) Memoria RAM

El sistema cuenta con dos módulos de memoria RAM de diferentes fabricantes. Las especificaciones de los módulos de memoria son las siguientes:

Fabricante: Hynix

- Velocidad: 3200 MHz
- Capacidad: 8 GB (8589934592 bytes)

Fabricante: Kingston

- Velocidad: 3200 MHz
- Capacidad: 8 GB (8589934592 bytes)

c) Unidad de Almacenamiento

El dispositivo de almacenamiento principal del sistema es un SSD NVMe con las siguientes características:

- Modelo: PC SN530 NVMe WDC 256GB
- Capacidad: 256 GB (256052966400 bytes)
- Tipo de medio: Fixed hard disk media

3.1.2. Consumidor

Para el consumidor se lo hizo en un sistema operativo Fedora, pero en este caso en una máquina virtual.

a) Procesador (CPU)

El sistema utiliza un procesador con las siguientes características:

- Número de núcleos: 4

b) Memoria RAM

La memoria base del sistema cuenta con:

- Capacidad total: 8895 MB (~8.7 GB)

c) Unidad de Almacenamiento

El dispositivo de almacenamiento principal del sistema es un SSD NVMe con las siguientes características:

- Capacidad: 60.46 GB

3.1.3. Apache Kafka

Para Apache Kafka se utilizó en un sistema operativo Rocky Linux, en una máquina virtual.

a) Procesador (CPU)

El sistema utiliza un procesador con las siguientes características:

- Número de núcleos: 2

b) Memoria RAM

La memoria base del sistema cuenta con:

- Capacidad total: 2048 MB (~2 GB)

c) Unidad de Almacenamiento

El dispositivo de almacenamiento principal del sistema es un SSD NVMe con las siguientes características:

- Capacidad: 20.96 GB

3.2. Jmeter con Selenium

Se ha configurado JMeter para simular las interacciones de un usuario real en la aplicación web, generando facturas. Se seleccionó el navegador Firefox para la automatización, y se utilizan las herramientas de localización de elementos proporcionadas por Selenium, adaptándolas a las necesidades del proyecto.

Esta integración permite una prueba eficiente del rendimiento del sistema, garantizando que el proceso de generación de facturas se ejecute de manera continua y fluida.

Partiendo desde el login para los vendedores se debe identificar los componentes a seleccionar como se muestra en la figura X

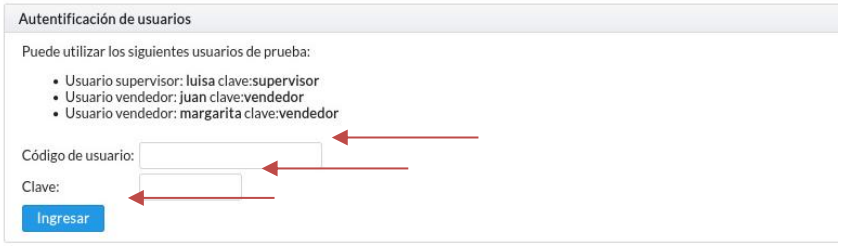

Figura 22. Vista – Login componentes
Elaborado por el autor

Se identifica tres componentes principales: dos de ellos están destinados a la entrada de información, específicamente el campo para el código de usuario y el campo para la clave, mientras que el tercer componente es el botón que permite la acción de ingreso.

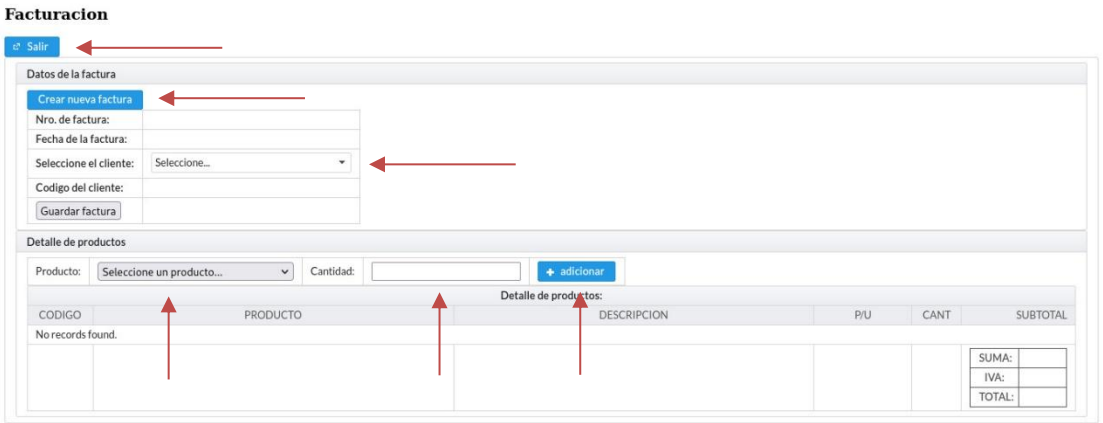

Figura 23. Vista – Facturación componentes
Elaborado por el autor

Se identifican un total de siete componentes clave dentro de la interfaz de la aplicación. De estos,

- Dos son listas desplegables: una para seleccionar clientes y otra para elegir productos.
- Cuatro botones con funciones específicas: uno para salir y regresar a la vista de inicio de sesión, otro para añadir productos a la factura, un tercero para generar los datos de la factura, y el cuarto para guardar la factura.
- Un campo destinado a ingresar la cantidad de cada producto.

Hay que tener en cuenta el flujo que tiene que seguir el sistema para crear una factura adecuadamente:

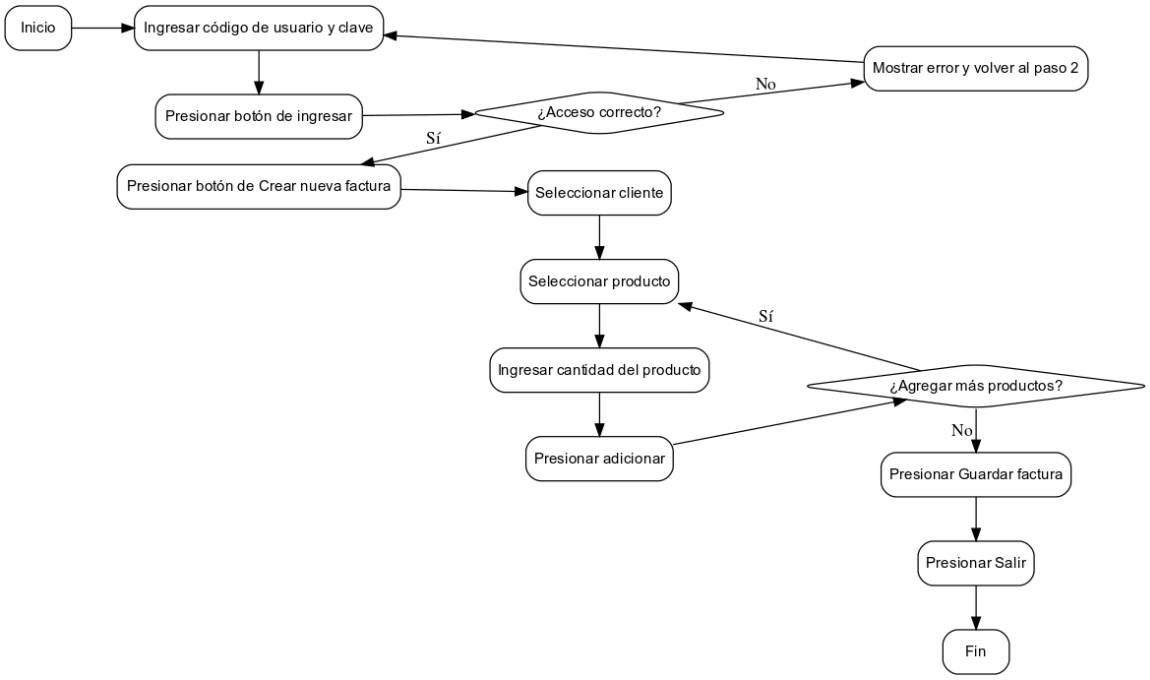

Figura 24. Diagrama de flujo de facturación
Elaborado por el autor

En el análisis del flujo de trabajo, se debe considerar que cada componente inicia su operación a partir del término de la acción de otro componente. Además, es importante tener en cuenta que los selectores tienen dos funcionalidades distintas: una para desplegar el menú del selector y otra para seleccionar un elemento dentro de dicho menú.

Cuando se acaba de crear una factura se tiene que volver a la vista de login para así seleccionar otro vendedor para así asegurar que diferentes vendedores interactúen.

Para determinar aspectos como el vendedor que ingresa, el cliente a seleccionar, el producto a seleccionar y la cantidad de producto se toma un enfoque que lo haga de manera aleatoria. Siguiendo este enfoque también para el numero de facturas que generara el script.

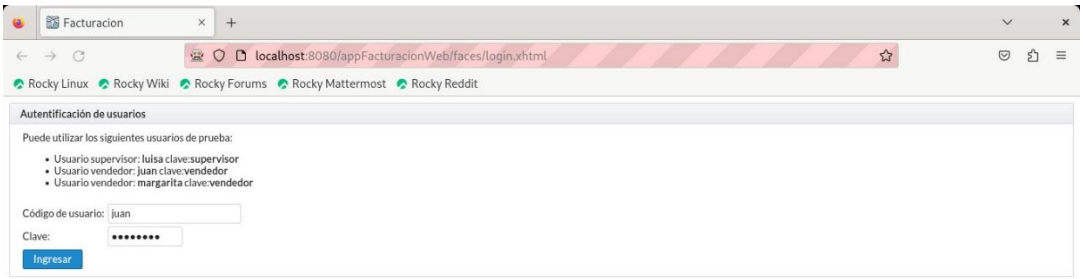

Figura 25. Automatización de login facturación
Elaborado por el autor

La automatización del proceso de inicio de sesión implica la introducción de credenciales de acceso, donde el código de usuario debe variar aleatoriamente entre los diferentes vendedores disponibles a través del script de Selenium.

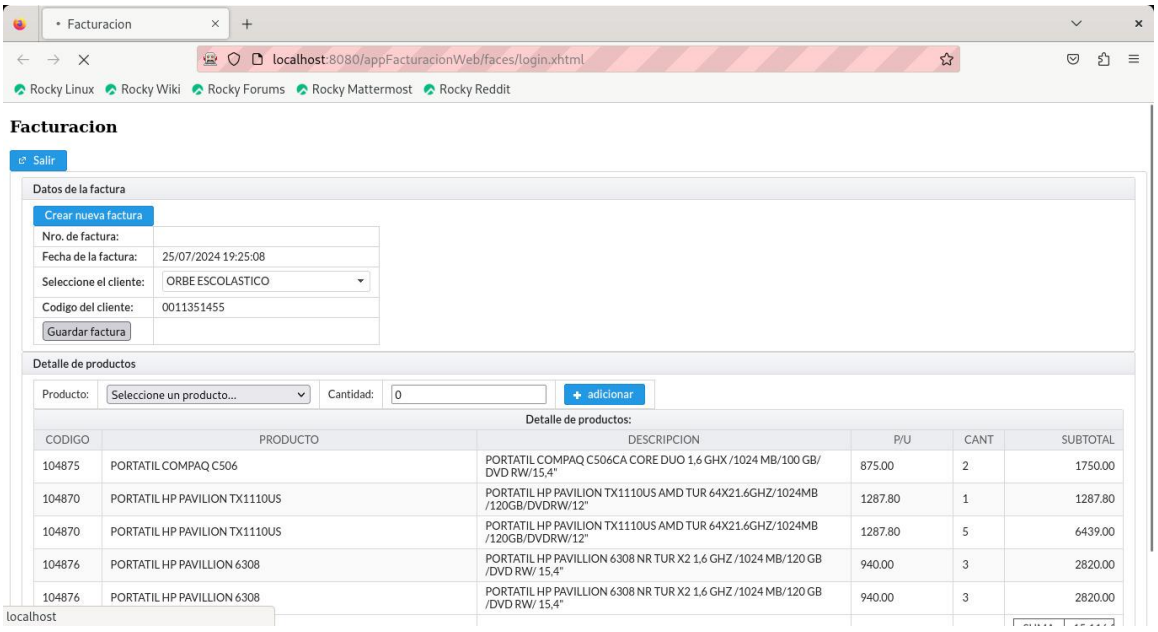

Figura 26. Automatización de facturación
Elaborado por el autor

3.3. ISO

Para el cumplimiento el apartado de eficiencia de desempeño se toma en cuenta las métricas mas importantes para nuestro caso.

3.3.1. Tiempo de espera

Según lo establecido por la norma, para realizar el cálculo es necesario determinar la diferencia entre el momento en que se completa el trabajo, que corresponde al consumo del mensaje y el momento en que se inicia, en este caso la producción del mensaje. Con base en esto, se utiliza la siguiente fórmula:

$$X = B - A$$

A: *Tiempo de productor*

B: *Tiempo de consumidor*

Para obtener la medición del tiempo de espera, se implementó un mecanismo que registró las marcas de tiempo en dos puntos: cuando el productor envía el mensaje y cuando el consumidor lo recibe. Esta información se almacenó en un archivo de texto para su posterior análisis.

El proceso consistió en capturar la hora exacta del sistema en dos momentos clave para cada mensaje:

- Hora de envío en el productor: Se registró la marca de tiempo cuando el productor envió el mensaje a Kafka.
- Hora de consumo en el consumidor: Se registró la marca de tiempo cuando el consumidor recibió y procesó el mensaje.

La captura de las marcas de tiempo se realizó utilizando la clase LocalDateTime de Java, que permite obtener la fecha y la hora exacta con precisión hasta los milisegundos. La fecha y hora fueron formateadas en el patrón yyyy-MM-dd HH:mm:ss.SSS, que incluye año, mes, día, hora, minuto, segundo y milisegundo.

Una vez que las marcas de tiempo fueron capturadas, se calculó el tiempo de espera para cada mensaje restando el tiempo de envío del productor del tiempo de consumo del consumidor. Este cálculo se realizó de manera individual para cada mensaje como se muestra en la tabla X.

Hora Productor	Hora Consumidor	Diferencia(ms)
2025-01-22 02:00:02.028	2025-01-22 02:00:02.050	22.0
2025-01-22 02:00:02.500	2025-01-22 02:00:02.518	18.0
2025-01-22 02:00:02.599	2025-01-22 02:00:02.613	14.0
2025-01-22 02:00:03.097	2025-01-22 02:00:03.118	21.0
2025-01-22 02:00:04.952	2025-01-22 02:00:04.991	39.0
2025-01-22 02:00:05.383	2025-01-22 02:00:05.399	16.0
2025-01-22 02:00:06.042	2025-01-22 02:00:06.061	19.0
2025-01-22 02:00:06.112	2025-01-22 02:00:06.133	21.0
2025-01-22 02:00:06.413	2025-01-22 02:00:06.427	14.0
2025-01-22 02:00:07.982	2025-01-22 02:00:08.000	18.0

a) Promedio

El promedio general del tiempo de espera fue de 26.39 ms, lo que indica que, en promedio, el sistema pudo procesar los mensajes en un tiempo relativamente bajo. Este resultado sugiere que Apache Kafka es eficiente en términos de latencia, permitiendo que la mayoría de los mensajes se transmitan rápidamente a través del sistema.

$$X = Tiempo\ de\ consumidor - Tiempo\ de\ productor = 0.02639\ segundos$$

b) Mediana

La mediana del tiempo de espera fue de 21.0 ms. Este valor es especialmente útil porque refleja el comportamiento típico del sistema sin verse afectado por valores extremos. Un valor cercano al promedio muestra que la mayoría de los mensajes fueron procesados rápidamente.

c) Tiempo Mínimo y Máximo de Espera

El tiempo mínimo de espera registrado fue de 8.0 ms, lo que representa el

mejor caso de procesamiento de mensajes, cuando las condiciones del sistema son óptimas.

El tiempo máximo de espera alcanzó 643.0 ms, lo que indica que, en algunos casos, se produjeron demoras significativas en el procesamiento de los mensajes. Este valor puede señalar situaciones de sobrecarga del sistema o picos de latencia.

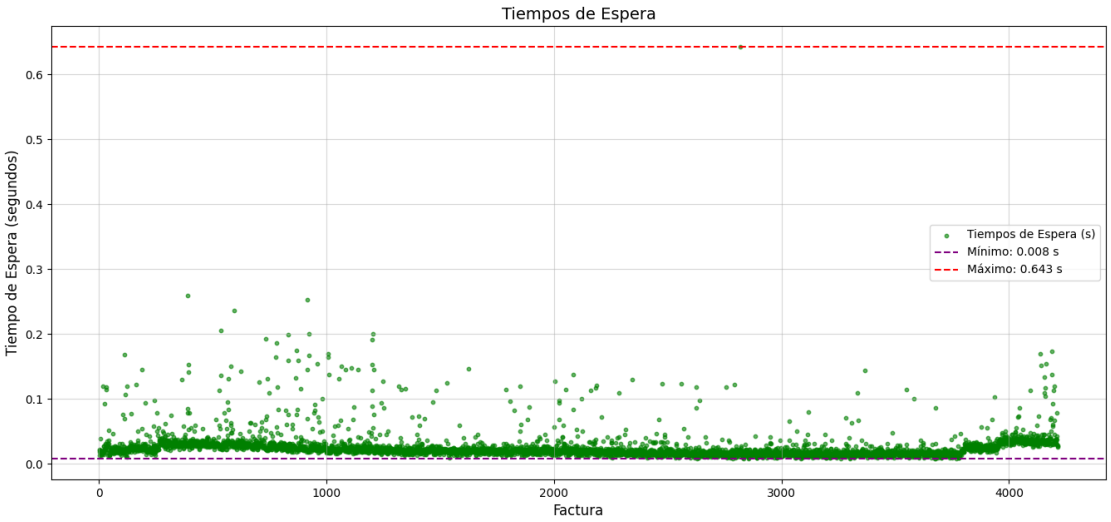

Figura 27. Tiempo de espera
Elaborado por el autor

d) Desviación Estándar.

La desviación estándar fue de 22.84 ms, lo que muestra que existe cierta variabilidad en los tiempos de espera. Aunque la variabilidad no es extremadamente alta, sugiere que en algunos casos el tiempo de procesamiento de los mensajes fue considerablemente más largo que el promedio, lo que podría estar relacionado con la carga del sistema o el comportamiento dinámico de los consumidores.

e) Número de Outliers.

Se identificaron 262 outliers en el conjunto de datos, lo que representa un número de casos donde los tiempos de espera fueron más altos que el promedio. Estos outliers podrían deberse a factores como la sobrecarga del sistema, cuellos de botella o eventos aislados que afectan el rendimiento.

f) Distribución del Tiempo de Espera

Para obtener una visión más clara del comportamiento del sistema, se analizó la distribución de los tiempos de espera en diferentes rangos. Los resultados fueron los siguientes:

- Rango 0-100 ms: El 97.84% de los mensajes fueron procesados en este intervalo, lo que sugiere que la gran mayoría de los mensajes fueron manejados con tiempos de espera bajos, dentro de un rango considerado eficiente.
- Rango 100-500 ms: Solo el 2.14% de los mensajes se ubicaron en este rango, lo que indica que los tiempos de espera moderados fueron poco frecuentes. A pesar de ser una pequeña fracción, este valor revela que en algunos casos el sistema experimentó latencias más altas, pero aún dentro de un rango tolerable.
- Más de 500 ms: Un 0.02% de los mensajes excedieron los 500 ms de tiempo de espera. Este valor extremadamente bajo sugiere que las demoras severas son muy poco comunes.

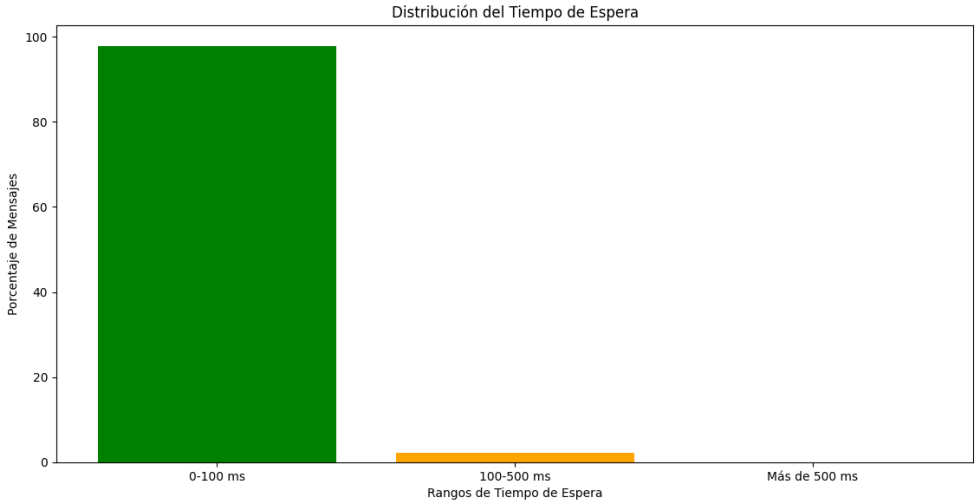

Figura 28. Distribución del tiempo de espera
Elaborado por el autor

Los resultados obtenidos durante las pruebas de tiempo de espera reflejan un sistema generalmente eficiente y con una latencia baja en la mayoría de los casos. La mayoría de los mensajes fueron procesados en menos de 100 ms (97.84%), lo que

demuestra un rendimiento adecuado para un alto volumen de datos. Sin embargo, la presencia de outliers (262 casos) y un tiempo máximo de 643 ms indican que en ciertos momentos el sistema puede experimentar picos de latencia.

La desviación estándar de 22.84 ms muestra que existe cierta variabilidad en los tiempos de espera, lo cual podría estar relacionado con fluctuaciones en la carga del sistema o en el comportamiento de los consumidores. Aunque los tiempos de espera óptimos fueron prácticamente inexistentes, el sistema mostró un rendimiento general positivo, con un porcentaje mínimo de mensajes procesados con demoras excesivas.

3.3.2. Rendimiento

Según lo que establece la norma, debemos determinar el número de tareas, que en este caso se definen como el envío y consumo de un mensaje. Además, se requiere definir un intervalo de tiempo, el cual se establece tomando la hora del primer mensaje producido como inicio y la hora del último mensaje consumido como fin.

$$X = \frac{A}{T}$$

Donde:

- A: *Numero de mensajes producidos/consumidos*
- T: *Intervalo de tiempo*

El intervalo de tiempo inicia con el primer mensaje producido, registrado a las 02:00:02.028 del 2025-01-22, y finaliza con el último mensaje consumido, registrado a las 03:03:12.651 de la misma fecha. Esto da como resultado un intervalo total de 1:03:10.623, que al ser convertido en segundos equivale a 3790.623 segundos.

El número total de mensajes producidos y consumidos durante este período es de 4216. Al aplicar los valores en la fórmula, se obtiene el rendimiento:

$$X = \frac{A}{T} = \frac{4216}{3790.623} = 1.11$$

Este cálculo refleja que el sistema tiene la capacidad de procesar aproximadamente 1.11 mensajes por segundo en las condiciones evaluadas.

Para esta métrica, es importante considerar las capacidades computacionales del productor al momento de utilizar JMeter. Además, el incremento en el número de

usuarios simultáneos impactó directamente en el desempeño del sistema, haciendo que el productor perdiera progresivamente la capacidad de generar mensajes de manera simultánea y eficiente.

3.3.3. Utilización de CPU

En esta métrica se analiza cuánto tiempo real utilizó la CPU para completar una tarea. En este caso específico, nos enfocamos en el tiempo empleado para la producción y el consumo de mensajes. El enfoque seguido fue el siguiente:

Primero, se utilizó Prometheus para obtener los datos del porcentaje de uso de CPU por parte de Apache Kafka durante el intervalo de tiempo estudiado. Este monitoreo se realizó segundo a segundo, permitiendo registrar con precisión cómo se comportó el uso de la CPU a lo largo del tiempo.

Una vez que se recopilaron los datos, se procedió a realizar las siguientes conversiones:

1. **Conversión del porcentaje de uso de CPU**

Para interpretar los datos, el porcentaje de uso de CPU (CPU%CPU\%CPU%) se convirtió a una fracción decimal.

$$x = \frac{CPU\%}{100\%}$$

De esta manera, el porcentaje se transforma en un valor más manejable, adecuado para calcular el tiempo real que la CPU estuvo activa.

2. **Cálculo del tiempo real de CPU utilizado por segundo**

El valor obtenido en decimal se multiplica por la duración de cada intervalo, que en este caso fue de 1 segundo.

$$x = \frac{CPU\%}{100\%} * 1s$$

Para ilustrarlo con un ejemplo, tomemos el primer valor registrado:

$$x = \frac{0.0413\%}{100\%} * 1s = 0{,}000413s$$

Esto indica que, durante ese segundo, Apache Kafka utilizó aproximadamente

0.000413 segundos de tiempo real de CPU para llevar a cabo sus operaciones.

3. Suma de los tiempos de CPU utilizados

El cálculo anterior se aplica a cada segundo registrado durante el intervalo de tiempo analizado. Al finalizar, se suman todos los valores obtenidos para determinar el tiempo total que la CPU estuvo activa.

$$x = s1 + s2 + \cdots sn$$

En este caso, al sumar los valores de todos los segundos monitoreados, se obtuvo un tiempo total de:

$$x = 0.1473\ segundos$$

Esto significa que, durante el período analizado, Apache Kafka utilizó un total de 0.1473 segundos de tiempo real de CPU para procesar la producción y el consumo de mensajes.

Una vez determinada la cantidad de tiempo que la CPU realmente utilizó para realizar la tarea, procedemos a calcular el tiempo de operación correspondiente. Según lo establecido en la métrica de rendimiento, el tiempo de operación registrado fue de 3790.623 segundos, mientras que el tiempo efectivo de uso de la CPU fue de 0.1473 segundos. Con base en estos datos, se aplica la fórmula de la métrica de utilización de CPU:

$$x = \frac{B}{A}$$

- *A: La cantidad de tiempo de CPU que realmente es usado.*
- *B: Tiempo de operación.*

$$x = \frac{3790.623}{0.1473} = 0.000038859$$

El valor obtenido para x es de aproximadamente 0.000038859, lo que indica que la CPU utilizó un porcentaje sumamente bajo del tiempo total de operación para ejecutar la tarea. Esto refleja un nivel de utilización de recursos de CPU extremadamente eficiente, dado que la mayoría del tiempo de operación no implicó un uso activo de la CPU.

Para facilitar una mejor comprensión, se presenta la figura X, en la cual se pueden observar tanto los picos de subida en el uso del CPU como los períodos en los que se mantuvo con valores constantes.

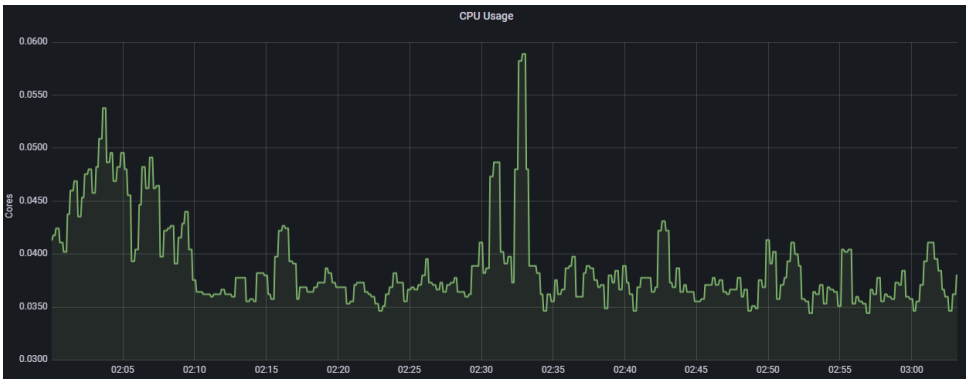

Figura 29. Uso de CPU
Elaborado por el autor

3.3.4. Utilización de la memoria

Para determinar esta métrica, es fundamental disponer tanto del total de espacio de memoria asignado como de la memoria efectivamente utilizada en la ejecución de una tarea. Para ello, se emplean las expresiones de Prometheus *jvm_memory_bytes_used* y *jvm_memory_bytes_max*, dado que Apache Kafka se ejecuta sobre la Java Virtual Machine (JVM). Esto implica que la gestión de la memoria, el procesamiento de hilos y la ejecución del código de Kafka están sujetos al funcionamiento y administración de la JVM.

$$X = B - A$$

A: Espacio de memoria que realmente es usado.

B: Cantidad total de espacios de memoria

Donde determinamos que

0<=X<=Memoria máxima

El más cercano a 0 es el mejor

Para determinar este valor, se generó un listado con los datos de jvm_memory_bytes_used y jvm_memory_bytes_max, recopilados en intervalos de un segundo dentro del rango de tiempo establecido.

Memoria máxima	Memoria usada	Diferencia
536870912	279856128	257014784
536870912	279856128	257014784
536870912	279856128	257014784
536870912	279856128	257014784
536870912	279856128	257014784
...	...	...
536870912	200774200	336096712
536870912	200774200	336096712
536870912	200774200	336096712
536870912	200774200	336096712
536870912	200774200	336096712

Con estos datos podemos determinar el promedio de la diferencia el cual nos da un valor de:

222623468.2668073 bytes

Siendo que el máximo de memoria es de *536870912 bytes*, podemos determinar que:

0<=222623468.2668073<=536870912

El más cercano a 0 es el mejor

A partir del análisis de los valores de jvm_memory_bytes_used y jvm_memory_bytes_max, se ha podido determinar la diferencia promedio entre la memoria total asignada y la memoria efectivamente utilizada por Apache Kafka en la JVM. El resultado obtenido, *222623468.27 bytes*, se encuentra dentro del rango definido entre 0 y *536870912 bytes*, lo que indica que el uso de memoria es significativamente inferior al límite máximo disponible.

Dado que un menor valor de esta diferencia implica una mayor eficiencia en la asignación de recursos, se concluye que la gestión de memoria de Kafka en este entorno presenta un comportamiento adecuado.

Podemos entender mejor el comportamiento del sistema al observar la Figura X. En ella, se nota que el valor de la memoria máxima permanece constante, mientras que el uso de memoria varía, mostrando picos y caídas que reflejan la actividad del sistema. Además, se incluyen los promedios calculados: el uso promedio de memoria es de *314247443.73 bytes,* y la memoria disponible promedio es de *222623468.27 bytes.*

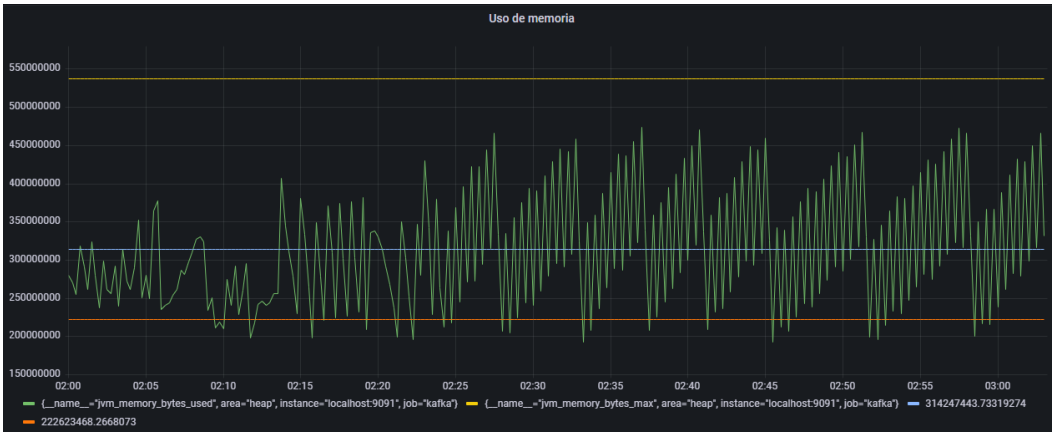

Figura 30. Uso de memoria
Elaborado por el autor

3.3.5. Sistema de transmisión de ancho de banda

Para esta métrica, es necesario determinar el ancho de banda total procesado dentro del rango de tiempo seleccionado. Para ello, se utilizarán las expresiones kafka_server_brokertopicmetrics_bytesout_total y kafka_server_brokertopicmetrics_bytesin_total, las cuales proporcionan el total de bytes transmitidos y recibidos, respectivamente.

$$X = A/T$$

A: Cantidad de transmisión de datos

B: Tiempo de operación

Para poder sacar la cantidad de cada expresión hacemos uso de Grafana y vemos cuantos bytes había al inicio del rango del tiempo y cuantos al final.

a) kafka_server_brokertopicmetrics_bytesin_total

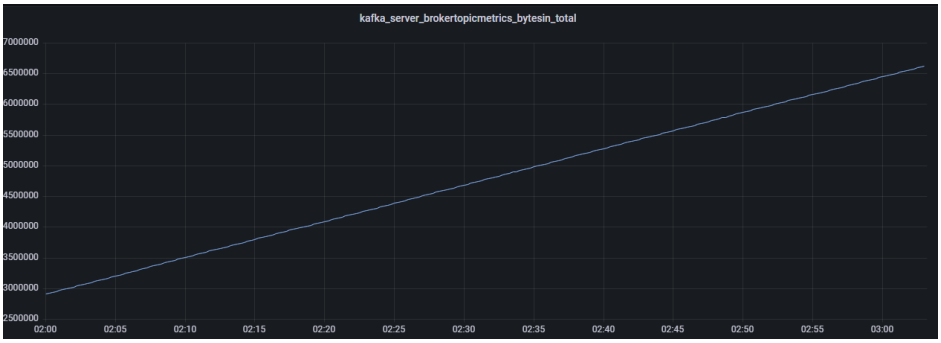

Figura 30. Bytes de entrada
Elaborado por el autor

En base a esto determinamos que el inicio es de 2911241 bytes y el final de 6618682 bytes:

$$2911241 \text{ bytes} - 6618682 \text{ bytes} = 3707441 \text{ bytes}$$

b) kafka_server_brokertopicmetrics_bytesin_total

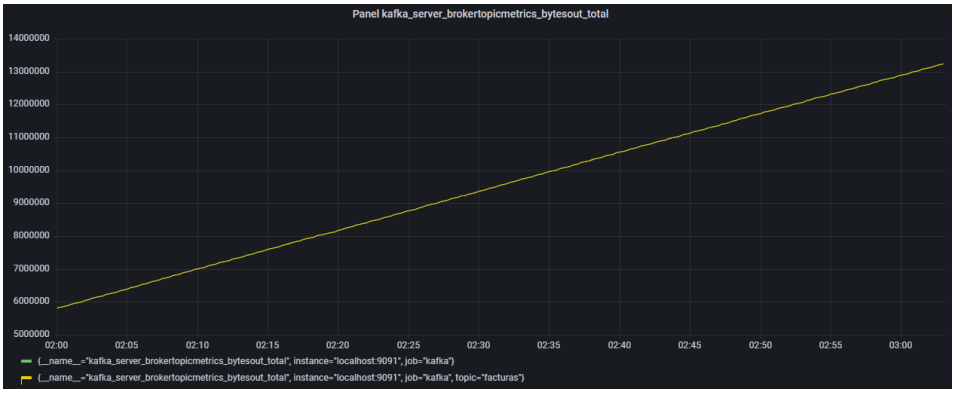

Figura 31. Bytes de salida
Elaborado por el autor

En base a esto determinamos que el inicio es de 5822482 bytes y el final de 13237364 bytes:

$$5822482 \text{ bytes} - 13237364 \text{ bytes} = 7414882 \text{ bytes}$$

Una vez tenemos esto sumamos los bytes de salida como de entrada:

$$3707441 \text{ bytes} + 7414882 \text{ bytes} = 11122323 \text{ bytes}$$

Una vez tenemos estos datos podemos determinar la métrica:

A: 11.122323 MB

B: 3790.623s

$$X = \frac{7414882\ \text{B}}{3790.623\ \text{s}} = 1.956{,}111\ B/s$$

Los resultados obtenidos permiten determinar el ancho de banda total procesado en el periodo analizado, evidenciando una tasa de transmisión promedio de 1.956 KB/s. Este valor proporciona una referencia cuantitativa sobre el volumen de datos gestionado por Kafka en el intervalo de tiempo seleccionado. También podemos tomar de referencia la figura X, en donde vemos el comportamiento de entrada y salida en el rango de tiempo.

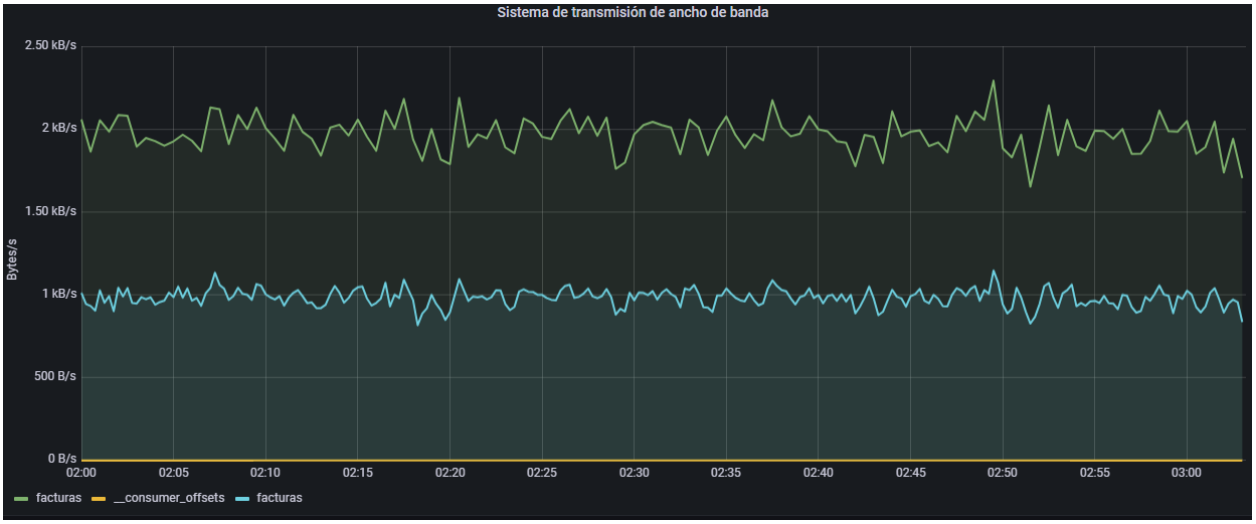

Figura 32. Ancho de banda
Elaborado por el autor

Conclusiones

- Apache Kafka demostró un tiempo promedio de procesamiento de mensajes de 26.39 ms, con un 97.84% de casos bajo 100 ms, validando su capacidad para manejar alto volumen con baja latencia. Sin embargo, la presencia de outliers (hasta 643 ms) y una desviación estándar de 22.84 ms revelan que factores externos como la configuración de consumidores en máquinas virtuales (2 GB RAM) produce variabilidad. Esto resalta la importancia de ajustar recursos hardware (ej. memoria en consumidores).

- Kafka tuvo un uso mínimo de CPU (0.000038% de utilización efectiva) y un consumo estable de memoria (41.5% del máximo asignado), evidenciando su eficiencia. No obstante, el rendimiento registrado (1.11 mensajes/segundo) no refleja su potencial real, sino limitaciones del entorno experimental: el cuello de botella en el productor (laptop con JMeter y 16 GB RAM) y la configuración de usuarios simultáneos en Selenium. Esto sugiere que la infraestructura de prueba actuó como factor restrictivo, resaltando la necesidad de distribuir cargas en nodos dedicados.

- La tasa de ancho de banda (1.956 KB/s) y la gestión de memoria en la JVM (222.62 MB disponibles en promedio) confirmaron que Kafka opera sin estrés de recursos incluso bajo demanda sostenida. Sin embargo, el bajo volumen de datos transmitidos (11.12 MB en 1 hora) y la infrautilización de CPU sugieren que la prueba no alcanzó los límites operativos del sistema. Esto refuerza que, para escenarios empresariales reales, se requieren configuraciones escalables (ej. clusters en lugar de máquinas virtuales aisladas) y perfiles de carga acordes a la capacidad de Kafka (ej. miles de mensajes/segundo).

Recomendaciones

- La implementación de un cluster distribuido de Kafka, en reemplazo de la arquitectura monolítica basada en máquinas virtuales (2 núcleos, 2 GB RAM). Este enfoque permitiría desplegar nodos dedicados con capacidades superiores (≥8 GB RAM, 4+ núcleos por nodo), aprovechando la escalabilidad horizontal de Kafka. Productores en escenarios con alto rendimiento de red (evitando limitaciones de hardware como las observadas en la laptop con JMeter) y consumidores en entornos con mayor capacidad de procesamiento (≥8 GB RAM). Esta estrategia mitigaría los picos de latencia (643 ms) y reduciría la desviación estándar registrada (22.84 ms), al distribuir equilibradamente la carga y evitar cuellos de botella asociados a recursos compartidos.

- Para superar las limitaciones identificadas (ej. 11.12 MB transmitidos en 1 hora), adoptar un enfoque de pruebas distribuido mediante herramientas como JMeter, emulando cargas realistas (miles de usuarios concurrentes). Implementar pruebas de estrés progresivo incrementando gradualmente el volumen de mensajes de 1k a 100k/segundo, permitiendo identificar puntos de ruptura antes de alcanzar latencias críticas (>500 ms).

- Para optimizar el rendimiento de Apache Kafka en entornos de alto volumen de mensajes, se recomienda automatizar la limpieza de mensajes y priorizar aquellos de mayor criticidad. Para ello, es conveniente segmentar los mensajes en tópicos diferenciados según su nivel de prioridad, permitiendo que las transacciones críticas se procesen de manera independiente.

Referencias

Aguila, E. (2016). DESARROLLO E IMPLEMENTACIÓN DE UN MÓDULO DE GESTIÓN DE EVENTOS, ALERTAS, MENSAJES Y TAREAS DE CONTROL INTEGRADO AL SISTEMA IOTMACH. UTMACH, UNIDAD ACADÉMICA DE INGENIERÍA CIVIL, Machala. Recuperado el 14 de junio de 2023, de http://repositorio.utmachala.edu.ec/bitstream/48000/7557/1/TTUAIC_2016_ISIST_CD0002.pdf

Algorri Álvarez, M. (2018). Caracterización de tecnologías de procesamiento de datos en streaming sobre una arquitectura orientada al dato. Universidad de Cantabria, Facultad de Ciencias , Cantabria. Obtenido de https://repositorio.unican.es/xmlui/bitstream/handle/10902/16283/AlgorriMiguel_TFM.pdf?sequence=1

Apache Kafka - Quick Guide. (s.f.). Recuperado el 6 de junio de 2023, de tutorialspoint: https://www.tutorialspoint.com/apache_kafka/apache_kafka_quick_guide.htm

Berzal, F. (s.f.). Middleware y sistemas distribuidos. Universidad de Granada, Departamento de Ciencias de la Computación e I.A, Granada. Obtenido de http://elvex.ugr.es/decsai/information-systems/slides/50%20Middleware.pdf

ISO 25000. (s.f.). Eficiencia de desempeño. Obtenido de ISO 25000: https://iso25000.com/index.php/normas-iso-25000/iso-25010/21-eficiencia-de desempeno

Martínez, A. (2020). Diseño e implementación de un middleware CoAP-MQTT-HTTP para la mejora de la interoperabilidad de los protocolos de aplicación en redes IoT. Universidad de Zaragoza, Escuela de Ingeniería y Arquitectura (EINA), Zaragoza. Obtenido de https://zaguan.unizar.es/record/96261/files/TAZ-TFM-2020-1104.pdf

Mollá Sirvent, R. A. (2016). Estudio y Mejora del Rendimiento del Backend. Universidad de Alicante, Escuela Politécnica Superior, Escuela Politécnica Superior. Obtenido de https://rua.ua.es/dspace/bitstream/10045/57068/1/Estudio_y_mejora_del_rendimiento_del_backend_MOLLA_SIRVENT_RAFAEL_ALEJANDRO.pdf

Moran, M. (17 de junio de 2020). Infraestructura - Desarrollo Sostenible. Obtenido de Desarrollo Sostenible: https://www.un.org/sustainabledevelopment/es/infrastructure/

Moreno Pina, P. (2020). Middleware de caché orientado a microservicios. E.T.S.I. de Sistemas Informáticos (UPM), Sistemas Informáticos, Madrid. Recuperado el 7 de junio de 2023, de https://oa.upm.es/65150/1/TESIS_MASTER_PABLO_MORENO_PINA.pdf Navas, F. (2020). DESARROLLO DE UNA APLICACIÓN MÓVIL COMO HERRAMIENTA DE

SISTEMATIZACIÓN DE PRODUCCIÓN Y COMERCIALIZACIÓN AGROECOLÓGICA A TRAVÉS DE LAS TECNOLOGÍAS IONIC, ANGULAR, MYSQL, MICROSERVICIOS MEDIANTE LARAVEL Y AWS EC2, DIRIGIDOS A PRODUCTORES RESPONSABLES CON EL. Universidad Técnica del Norte, Facultada de Ingeniería en Ciencia Aplicadas - FICA, Ibarra. Recuperado el 14 de junio de 2023, de http://repositorio.utn.edu.ec/handle/123456789/1041

Nebel, A. (2018). Enterprise integration patterns: Designing, building, and deploying messaging solutions. Universidad de la República, Facultad de Ingeniería Instituto de Computación, Montevideo. Recuperado el 17 de junio de 2023, de https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/20586/1/tm nebel.pdf

Nesmachnow, S. (2018). Computación distribuida y procesamiento de grandes volúmenes de datos. Universidad de la república de Uruguay. Obtenido de https://eva.fing.edu.uy/pluginfile.php/196530/mod_resource/content/1/1 ComputacionDistribuida-Introducci%C3%B3n.pdf

NILSSON, E., & PREGÉN, V. (2020). Performance evaluation of message-oriented middleware Utvärderdering av prestanda för meddelandeorienterade mellanprogramvaror. Obtenido de https://www.diva portal.org/smash/get/diva2:1473908/FULLTEXT01.pdf

Person Montero, T. (2017). Autoría de aplicaciones móviles para el análisis de datos. UNIVERSIDAD DE CADIZ, ESCUELA SUPERIOR DE INGENIERÍA, Cádiz. Obtenido de https://rodin.uca.es/bitstream/handle/10498/19961/TFM_Tatiana_Person.pdf?s equence=1&isAllowed=y

Ramos Huarachi, V. A. (2020). MODELO DE COMPUTACIÓN EN STREAMING PARA REDUCIR LA COMPLEJIDAD COMPUTACIONAL DEL SOFTWARE USADO EN EL SECTOR RETAIL. UNIVERSIDAD MAYOR DE SAN ANDRÉS. Obtenido de https://repositorio.umsa.bo/bitstream/handle/123456789/28780/TM 3884.pdf?sequence=1&isAllowed=y

Red Hat. (10 de octubre de 2022). Integration: What is Apache Kafka? Recuperado el 5 de junio de 2023, de What is Apache Kafka?: https://www.redhat.com/en/topics/integration/what-is-apache-kafka

Rodríguez, O. (2017). MIDDLEWARE GINGA EN EL DESARROLLO DE APLICACIONES INTERACTIVAS PARA LA TELEVISIÓN DIGITAL TERRESTRE. UNIVERSIDAD TÉCNICA DEL NORTE, FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS, Ibarra. Obtenido de http://repositorio.utn.edu.ec/bitstream/123456789/6068/1/04%20ISC%20432%

20TRABAJO%20DE%20GRADO.pdf

Akanbi, A., & Masinde, M. (2020). A distributed stream processing middleware framework for real-time analysis of heterogeneous data on big data platform: Case of environmental monitoring. *Sensors (Switzerland)*, *20*(11), 1–25. https://doi.org/10.3390/s20113166

Andi, K., Ramirez, D., & Chango Sailema, W. G. (2024). Comparativa de frameworks más usados de JavaScript mediante la creación de una aplicación web. *Mikarimin. Revista Científica Multidisciplinaria*, *10*(3), 81–100. https://doi.org/10.61154/mrcm.v10i3.3245

Aziz, O., Farooq, M. S., Abid, A., Saher, R., & Aslam, N. (2020). Research Trends in Enterprise Service Bus (ESB) Applications: A Systematic Mapping Study. *IEEE Access*, *8*, 31180–31197. https://doi.org/10.1109/ACCESS.2020.2972195

Bhawiyuga, A., Kharisma, S. A., Santoso, B. J., Kartikasari, D. P., & Kirana, A. P. (2020). Cloud-based middleware for supporting batch and stream access over smart healthcare wearable device. *Bulletin of Electrical Engineering and Informatics*, *9*(5), 1990–1997. https://doi.org/10.11591/eei.v9i5.1978

Cahyaningrum, Y. (2024). Optimization of Information Technology Through Intelligent System Integration: Comprehensive Exploration. *Journal of Intelligent Systems and Information Technology*, *1*(1), 2024. https://jurnalapik.id/index.php/jisit

Dhane, A., Joshi, A., Borgaonkar, C., & Deshpande, M. (2024). Application Monitoring using Prometheus and Grafana. *International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)*.

Do Nascimento, E. D., Da Silva, J. A. X. B., & Lima, Â. T. P. (2023). O PANORAMA DO USO DE SELENIUM E RANOREX STUDIO COMO FERRAMENTAS DE TESTES. *Revista Contemporânea*, *3*(12), 31047–31071. https://doi.org/10.56083/rcv3n12-317

García, B., Gallego, M., Gortázar, F., & Munoz-Organero, M. (2020). A survey of the selenium ecosystem. *Electronics (Switzerland)*, *9*(7), 1–29. https://doi.org/10.3390/electronics9071067

García Sánchez, A. J., García Sánchez, F., Díaz Jiménez, F. J., & Haro, J. G. (2004). *Plataformas Middleware comerciales para la integración de flujos de vídeo bruto*. http://www.cs.wustl.edu/~schmidt/TAO.html

Hassan, A. A., & Hassan, T. M. (2022). Real-Time Big Data Analytics for Data Stream Challenges: An Overview. *European Journal of Information Technologies and Computer Science*, *2*(4), 1–6. https://doi.org/10.24018/COMPUTE.2022.2.4.62

Hohpe, G., & Woolf, B. (2003). *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc.

Hyrynsalmi, S. (2022). *Pathway to the successful integration platform management*.

Indrianto. (2023). PERFORMANCE TESTING ON WEB INFORMATION SYSTEM USING APACHE JMETER AND BLAZEMETER. *Jurnal Ilmiah Ilmu Terapan Universitas Jambi*, *7*(2), 138–149. https://doi.org/10.22437/jiituj.v7i2.28440

iso25000. (2018, June 15). *Eficiencia de Desempeño*. https://iso25000.com/index.php/normas-iso-25000/iso-25010/21-eficiencia-de-desempeno

Jepsen, S. C., Worm, T., Mork, T. I., & Hviid, J. (2021). Industry 4.0 Middleware Software Architecture Interoperability Analysis. *Proceedings - 2021 IEEE/ACM 3rd International Workshop on Software Engineering Research and Practices for the IoT, SERP4IoT 2021*, 32–35. https://doi.org/10.1109/SERP4IoT52556.2021.00012

Kołtun, A., & Pańczyk, B. (2020). *Comparative analysis of web application performance testing tools Analiza porównawcza narzędzi do badania wydajności aplikacji interne-towych*.

Krakowiak, S. (2009). *Middleware Architecture with Patterns and Frameworks*.

Kumar, S. (2021). *A Comprehensive study on Data Visualization tool-Grafana*.

Laakkonen, T., & Smolander, K. (2023). *Transforming integrations of an organization from point-to-point to iPaaS*.

Lagos, N. M., Fernanda Gómez, L., Londoño, D. C., Moreno, I. C., & Camacho Camacho, H. (2023). *Metodología para la integración de sistemas de gestión: revisión de literatura*. *15*. https://doi.org/10.15332/24631140

Mehdi, A., Bali, M. K., Abbas, S. I., & Singh, M. (2023). Unleashing the Potential of Grafana: A Comprehensive Study on Real-Time Monitoring and Visualization. *2023 14th International Conference on Computing Communication and Networking Technologies, ICCCNT 2023*. https://doi.org/10.1109/ICCCNT56998.2023.10306699

Pai, K., & Srinivas, B. K. (2024). Enhanced Visibility for Real-time Monitoring and Alerting in Kubernetes by Integrating Prometheus, Grafana, Loki, and Alerta. *International Journal of Scientific Research in Engineering and Management*. https://doi.org/10.55041/IJSREM35639

Palino Todd, Narkhede Neha, & Shapira Gwen. (2021). *"Kafka The Definitive Guide - Real-Time Data and Stream Processing at Scale* (O'Reilly Media).

Pastrana Pardo, M. A., Cifuentes Calderón, F. A., & Ordoñez Eraso, H. A. (2020). Arquitectura de Integración para Servicios y Soluciones Smart Campus. *Inge CuC*, *16*(2), 267–276. https://doi.org/10.17981/ingecuc.16.2.2020.21

Raptis, T. P., & Passarella, A. (2022). *On Efficiently Partitioning a Topic in Apache Kafka*. https://doi.org/10.1109/CITS55221.2022.9832981

Shukla, S. (2023). Exploring the Power of Apache Kafka: A Comprehensive Study of Use Cases suggest topics to cover. *International Journal of Latest Engineering and Management Research (IJLEMR) Www.Ijlemr.Com* ||, *08*, 71–78. www.ijlemr.com

Suparto, H. S., & Dai, R. H. (2021). Evaluasi Kualitas Sistem Informasi Pengukuran Prestasi Kerja Berdasarkan ISO/IEC 25010. *Jambura Journal of Informatics*, *3*(2), 109–120. https://doi.org/10.37905/JJI.V3I2.11744

Torres Ricaurte, D. M. (2019). *Formulating a theory about interoperability among heterogeneous software systems, based on the Semat kernel*.

Ushakova, I., Plokha, O., & Skorin, Y. (2022). Approaches to web application performance testing and real-time visualization of results. *Bulletin of Kharkov National Automobile and Highway University*, *96*, 71. https://doi.org/10.30977/bul.2219-5548.2022.96.0.71

Yadav, P. S. (2021). *Latency Reduction Techniques in Kafka for Real- Time Data Processing Applications*. https://doi.org/10.5281/ZENODO.13762396

Zapata, C. M., & Cardona, C. (2011). *Comparación de las características de algunas herramientas de software para pruebas de carga*.