RIDUNAJ
Repositorio Institucional
Digital UNAJ

Universidad Nacional
ARTURO JAURETCHE

Práctica Profesional Supervisada

Lucero, Lucas Gabriel

APIODE (Api Alta Ordenes de Envíos)

Instituto de Ingeniería y Agronomía

Carrera: Ingeniería en Informática

2025

Esta obra está bajo una Licencia Creative Commons.
Atribución – No comercial – Sin obra derivada 4.0
https://creativecommons.org/licenses/by-nc-nd/4.0/

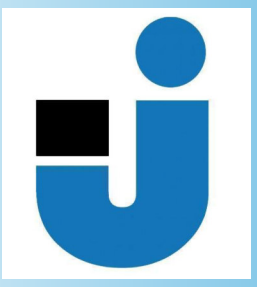
Documento descargado de RID - UNAJ Repositorio Institucional Digital de la Universidad Nacional Arturo Jauretche

Cita recomendada:
Lucero L. G. (2025). *APIODE (Api Alta Ordenes de Envíos)* [Práctica Profesional Supervisada, Universidad Nacional Arturo Jauretche]. https://rid.unaj.edu.ar/handle/123456789/3612

Universidad Nacional Arturo Jauretche

Instituto de Ingeniería y Agronomía

Carrera de Ingeniería en Informática



PRÁCTICA PROFESIONAL SUPERVISADA
Informe final

APIODE (Api Alta Ordenes de envios)

Lucas Gabriel Lucero

Florencio Varela, Agosto 2025

DATOS DEL ESTUDIANTE

Nombre y Apellido: Lucas Gabriel, Lucero.
DNI: 36.648.143
Correo Electrónico: luks671@gmail.com
Carrera: Ingeniería Informática

DATOS DE LA ORGANIZACIÓN DONDE SE REALIZA EL PIP

Nombre o Razón Social: Correo Andreani SA
Dirección: AV. LEANDRO N. ALEM 639, Piso 7, Departamento I, (1001) C.A.B.A.
Teléfono: 0800-122-1112
Sector: Arquitectura TI

TUTOR ORGANIZACIONAL

Nombre y Apellido: Lucas Maximiliano, Olivera
Correo Electrónico: lolivera@andreani.com

DOCENTE SUPERVISOR

Nombres y Apellidos: Martin Morales
Correo Electronico: mmorales@unaj.edu.ar

DOCENTE TUTOR DEL TALLER DE APOYO PARA LA PRODUCCIÓN DE TEXTOS ACADÉMICOS

Nombre y Apellido: N/A
Correo Electrónico: N/A

COORDINADOR DE LA CARRERA DE INGENIERIA EN INFORMATICA

Nombre y Apellido: Martin Morales
Correo Electronico: mmorales@unaj.edu.ar

Resumen

Durante mi práctica profesional participé en un proyecto clave para la empresa: la **migración de un sistema crítico** encargado de recibir y procesar las órdenes de envío, que constituyen la **principal fuente de ingresos de la organización** dedicada a la logística.

El sistema anterior presentaba limitaciones tecnológicas y dificultades para su mantenimiento, lo que representaba un riesgo para la operación. Frente a esto, el trabajo realizado consistió en el **diseño y desarrollo de una nueva plataforma moderna**, basada en las últimas tecnologías y buenas prácticas de la industria, priorizando la **escalabilidad, la trazabilidad y la disponibilidad continua**.

Uno de los aspectos más relevantes de la experiencia fue la **estrategia de implementación**, que permitió llevar adelante la modernización sin interrumpir la operación diaria del negocio. De esta manera, se garantiza la continuidad del servicio mientras se introducía la nueva solución.

Con esta migración, la empresa no sólo reemplazó un sistema obsoleto por otro actual, sino que también incorporó capacidades que fortalecen su **eficiencia operativa**, reducen riesgos y mejoran la experiencia de clientes y áreas internas.

Palabras clave: Andreani, Microservicios, Kafka, orden de envío.

Abstract

During my professional internship, I took part in a key company project: migrating a critical system responsible for receiving and processing shipping orders, which are the main source of revenue for this logistics-focused organization.
The legacy system had technological limitations and maintenance challenges, posing an operational risk. In response, the work involved designing and developing a modern platform based on the latest technologies and industry best practices, prioritizing scalability, traceability, and continuous availability.
A major highlight of the experience was the implementation strategy, which enabled the modernization to proceed without disrupting the business's day-to-day operations. This ensured service continuity while the new solution was introduced.
With this migration, the company not only replaced an obsolete system with a modern one, but also added capabilities that strengthen operational efficiency, reduce risk, and improve the experience for both customers and internal teams.

Títulos y agradecimientos

Al cerrar una etapa tan importante de mi formación profesional, quiero expresar mi profundo agradecimiento a todas las personas que hicieron posible este logro.

En primer lugar, a mi pareja y compañera de vida, quien me acompañó incondicionalmente desde el inicio de la carrera. Su apoyo constante, comprensión y aliento fueron fundamentales para sostenerme en cada desafío y momento de dificultad. Siento que este título nos pertenece a ambos, porque sin su respaldo este camino no habría sido más difícil de recorrer.

A mi familia, por brindarme las bases y el apoyo emocional necesario para perseverar en mis objetivos académicos y profesionales.

A mis amigos, por su compañía, comprensión y por los momentos de distensión que equilibraron las exigencias del estudio y el trabajo.

A Lucas Olivera, mi tutor organizacional, por su invaluable guía, consejos y apoyo durante el desarrollo de esta tesis. Su experiencia y conocimiento fueron esenciales para orientar este proyecto hacia el éxito, y su disposición para compartir su expertise enriqueció significativamente mi aprendizaje.

A mis colegas de Andreani, por su colaboración, paciencia y por compartir conmigo sus conocimientos del negocio, los cuales fueron fundamentales para comprender las necesidades reales que debía abordar este proyecto.

Finalmente, a la Universidad Nacional Arturo Jauretche y a todos los docentes que contribuyeron a mi formación, por proporcionarme las herramientas conceptuales y metodológicas que hicieron posible este trabajo y mi desarrollo profesional.

A todos, mi más sincero agradecimiento.

Notaciones y abreviaturas

Los términos técnicos específicos utilizados en este trabajo se encuentran definidos en el Glosario luego de la introducción del documento para facilitar la comprensión de lectores no especializados en el área.

Abreviaturas y siglas

APIODE - API Orden De Envío (Sistema desarrollado)
APM - Application Performance Monitoring (Monitoreo de Rendimiento de Aplicaciones)
API - Application Programming Interface (Interfaz de Programación de Aplicaciones)
EDA - Event-Driven Architecture (Arquitectura Orientada a Eventos)
IBM MQ - IBM Message Queue (Sistema de mensajería de IBM)
MVP - Minimum Viable Product (Producto Mínimo Viable)
SLA - Service Level Agreement (Acuerdo de Nivel de Servicio)
SOLID - Principios de diseño de software (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion)

Índice

Introducción

La empresa y el contexto

Andreani es una de las principales empresas de logística de la Argentina, dedicada al transporte, distribución y gestión integral de envíos. Su negocio se sostiene sobre la capacidad de **recibir y procesar órdenes de envío**, que representan el canal de ingreso más importante de la compañía. Sin un flujo confiable para dar de alta envíos, la operación logística se vería directamente afectada, con impacto inmediato tanto en los clientes como en la rentabilidad de la organización.

En este marco, el sistema encargado de gestionar las solicitudes de creación de envíos constituye un **activo crítico**. Sin embargo, el software que cumplía este rol había quedado desactualizado, con limitaciones técnicas, dificultades de mantenimiento y carencias de monitoreo, lo que aumentaba los riesgos operativos y reducía la capacidad de respuesta frente a incidentes o picos de demanda.

El sistema anterior

El servicio vigente concentraba en un único componente todo el proceso de alta de envíos, desde la validación inicial hasta la persistencia de datos y publicación en sistemas internos. Su diseño monolítico, sumado a una infraestructura tecnológica ya deprecada, dificultaba la escalabilidad, el mantenimiento y la visibilidad de la operación. Estas restricciones derivaban en tiempos mayores de resolución ante incidentes, menor trazabilidad y costos de operación elevados.

La nueva solución

Frente a este escenario, se diseñó e implementó una **plataforma moderna basada en microservicios**, con tecnologías alineadas al estándar corporativo y una arquitectura orientada a eventos. La estrategia no solo contempló la construcción del nuevo sistema, sino también un **plan de migración gradual** que permitió garantizar la continuidad operativa durante la transición.

El resultado es un ecosistema más robusto, escalable y observable, preparado para sostener la operación diaria y acompañar la evolución del negocio en contextos de alta demanda (como campañas comerciales estacionales). Además, la solución incorpora prácticas modernas de desarrollo y monitoreo que facilitan el mantenimiento y reducen riesgos, mejorando la experiencia tanto de clientes como de áreas internas.

Sobre este trabajo

En las secciones siguientes el lector encontrará:

- Una descripción del sistema previo y las limitaciones que motivaron su reemplazo.
- El diseño y la implementación de la nueva solución, junto con la estrategia de migración aplicada.
- Los resultados esperados y el impacto de la modernización en la operación de la empresa.

El sistema anterior

El sistema de alta de órdenes de envío en uso hasta el inicio del proyecto respondía a un diseño **monolítico**, concentrando en un único componente la validación, normalización y persistencia de los datos. Si bien durante un tiempo cumplió su propósito, con el crecimiento del negocio fue quedando desalineado de las necesidades actuales de la compañía.

Las principales limitaciones podían resumirse en los siguientes puntos:

- **Obsolescencia tecnológica**: el sistema estaba construido sobre tecnologías que ya no formaban parte del estándar corporativo, lo que dificultaba su evolución y encarecía el soporte.
- **Mantenimiento complejo**: cualquier cambio implicaba intervenir sobre el núcleo completo, aumentando riesgos y tiempos de implementación.
- **Escalabilidad limitada**: no era posible dimensionar selectivamente los recursos de acuerdo con la demanda; ante picos de carga, toda la aplicación sufría degradación.
- **Baja observabilidad**: las métricas disponibles eran fragmentadas y poco claras, lo que dificultaba el monitoreo en tiempo real y extendía los tiempos de resolución de incidentes.
- **Costos elevados**: tanto por el uso de plataformas ya discontinuadas en la organización como por la dificultad de sostener el producto a largo plazo.

Estas debilidades se traducían en un riesgo operativo significativo: al tratarse del sistema que gestiona el **principal canal de ingresos de la empresa**, cualquier indisponibilidad o degradación afectaba tanto la experiencia de los clientes como la eficiencia de las áreas internas.

La evidencia de estas limitaciones se refleja claramente en la **precaria infraestructura de observabilidad** del sistema actual. Tal como se muestra en las Figuras 1.1 a 1.4, los dashboards disponibles presentan **información fragmentada y poco intuitiva**:

- Gráficos sin correlación temporal ni contexto de negocio.
- Métricas dispersas, sin un hilo conductor que permita detectar patrones.
- Distribución de errores sin análisis de tendencias.
- Información operativa que exige análisis manual para encontrar cuellos de botella.

Esta carencia de visibilidad no solo dificulta la detección proactiva de problemas, sino que además **incrementa el tiempo medio de resolución** de incidentes,

obligando a los equipos a realizar investigaciones extensas sobre datos poco estructurados antes de identificar la causa raíz.

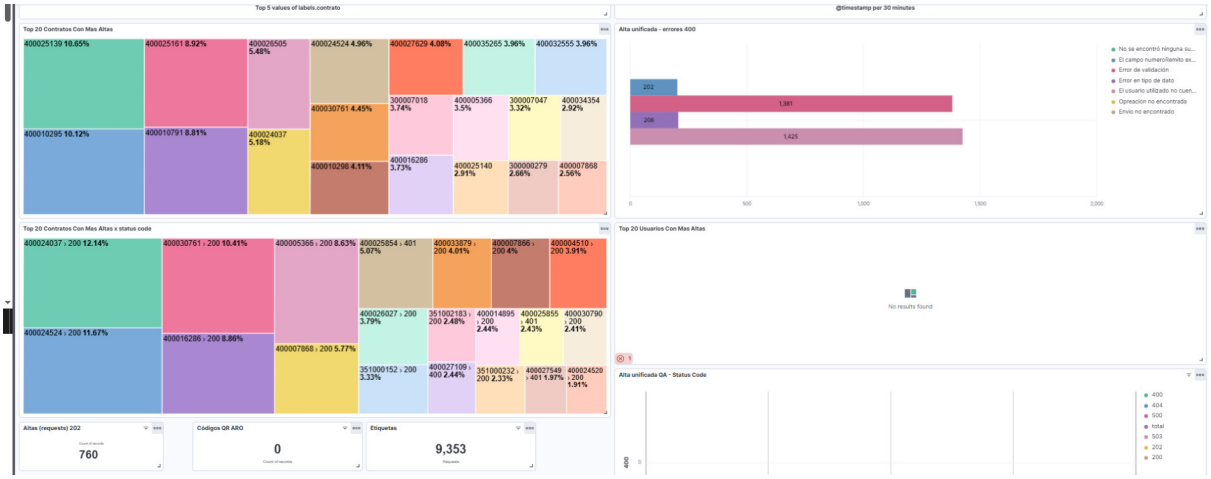


***Figura 1.1**: Dashboard actual - API de pre envíos: fragmentación de métricas por contratos y distribución básica de códigos de respuesta*

***Figura 1.2**: Análisis de estados y errores - API de pre envíos: información dispersa*

Figura 1.3: *Monitoreo de errores y métricas básicas - API de pre envíos: visualización limitada del estado*

Figura 1.4: *Visibilidad de eventos y publicaciones - API de pre envíos: métricas operativas sin correlación de performance*

El diseño y la implementación de la nueva solución

La modernización del sistema de alta de órdenes de envío se abordó mediante la construcción de una **nueva plataforma basada en microservicios**, alineada con los estándares corporativos de Andreani. El diseño se centró en desacoplar funciones críticas —validación, normalización, orquestación y persistencia— que antes se encontraban integradas en un único componente. Esta separación permitió crear servicios más simples, especializados y capaces de escalar de manera independiente, optimizando el uso de recursos y mejorando la resiliencia del sistema.

Un principio rector de la solución fue incorporar **observabilidad end-to-end**, con tableros de monitoreo que ofrecen métricas claras y centralizadas sobre el estado del flujo de creación de envíos. De esta forma, el nuevo sistema no solo resuelve las limitaciones operativas del anterior, sino que además brinda mayor trazabilidad y capacidad de respuesta ante incidentes.

La estrategia de migración aplicada

Dado que el sistema de altas constituye el **principal canal de ingresos de la empresa**, la transición hacia la nueva plataforma debía garantizar la **continuidad operativa sin interrupciones**. Para lograrlo se diseñó una **estrategia de migración gradual**, que permitió coexistir durante un tiempo el sistema anterior y la nueva solución.

La estrategia contempló tres ejes principales:

- **Implementación escalonada**: se introdujeron progresivamente los nuevos microservicios, comenzando con componentes menos críticos para validar su funcionamiento antes de avanzar sobre el flujo completo.

- **Compatibilidad dual**: se mantuvo la publicación hacia la infraestructura anterior mientras los sistemas modernizados comenzaron a operar sobre la nueva plataforma. Esto aseguró que ninguna área dependiente quedara desconectada durante la transición.

- **Acompañamiento de clientes internos y externos**: cada fase de la migración fue comunicada y coordinada previamente, habilitando canales de soporte dedicados para atender posibles incidencias.

Gracias a esta estrategia, el pasaje al nuevo ecosistema se realizó de manera **transparente para los usuarios**, sin generar cortes de servicio ni afectar la operación diaria.

En conjunto, el **nuevo diseño e implementación** permitieron pasar de un sistema rígido y difícil de mantener a una plataforma moderna, escalable y preparada para acompañar el crecimiento de la compañía, tanto en términos de volumen como de exigencias tecnológicas y de negocio.

Glosario

Apache Kafka: Plataforma de streaming de datos distribuida que permite publicar y suscribirse a flujos de eventos en tiempo real de manera escalable y confiable.

Broker: Servidor individual que forma parte del cluster de Kafka, especializado en almacenar y servir streams de datos con alta disponibilidad.

Clean Architecture: Patrón arquitectónico que organiza el código en capas concéntricas con dependencias unidireccionales hacia el núcleo, priorizando la independencia de frameworks externos.

Código: Conjunto de instrucciones escritas en un lenguaje de programación que definen el comportamiento y funcionalidad de una aplicación o sistema de software.

Consumer Group: Grupo de consumidores de Kafka que trabajan en paralelo para procesar mensajes de un tópico, distribuyendo automáticamente la carga entre miembros.

Event-Driven Architecture (EDA): Arquitectura donde los servicios se comunican mediante la publicación y consumo de eventos, eliminando dependencias síncronas directas.

Legacy: Sistemas o tecnologías heredadas de versiones anteriores que siguen en funcionamiento pero están basadas en tecnologías desactualizadas o que la organización planea reemplazar gradualmente.

Microservicio: Servicio pequeño e independiente que implementa una capacidad específica de negocio y puede ser desarrollado, desplegado y escalado de forma autónoma.

Monolito: Aplicación donde toda la funcionalidad está implementada en una única unidad de despliegue con componentes fuertemente acoplados.

Normalización: Proceso de transformar y estandarizar datos provenientes de diferentes fuentes para asegurar consistencia de formato y completitud de información.

Offset: Identificador único secuencial de cada mensaje dentro de una partición de Kafka que permite a los consumidores mantener su posición de lectura.

Orquestación: Coordinación centralizada del flujo de trabajo entre múltiples servicios, definiendo la secuencia y condiciones de ejecución de cada paso del proceso.

Partición: División lógica de un tópico de Kafka que permite procesamiento paralelo y distribución de carga entre múltiples consumidores.

Tópico: Canal nombrado en Kafka donde se organizan los mensajes por tema específico, actuando como un log distribuido de eventos.

Background: Proceso que se ejecuta en segundo plano sin intervención directa del usuario, como workers que sincronizan datos automáticamente según intervalos programados.

Worker: Proceso en background que ejecuta tareas programadas de forma automática, como la sincronización de datos entre sistemas.

Pipeline: Flujo automatizado de procesos secuenciales, típicamente utilizado en CI/CD para compilar, probar y desplegar código de manera continua.

Stack tecnológico: Conjunto específico de tecnologías, frameworks, librerías y herramientas utilizadas para desarrollar y operar una aplicación o sistema.

Template: Plantilla de código pre-configurada que proporciona estructura base y mejores prácticas para acelerar el desarrollo de nuevos proyectos manteniendo consistencia arquitectónica.

Throughput: Cantidad de datos o transacciones procesadas por unidad de tiempo, medida típicamente en mensajes por segundo o peticiones por minuto.

Sincronizar: Proceso de mantener consistencia de datos entre diferentes sistemas o bases de datos, asegurando que la información esté actualizada y alineada entre todas las fuentes.

Síncrono: Comunicación donde el emisor debe esperar la respuesta del receptor antes de continuar su procesamiento, creando dependencias directas entre servicios.

Validación de datos: Proceso de verificar que la información recibida cumple con reglas de negocio específicas, formatos requeridos y restricciones de integridad antes de procesarla o almacenarla.

Asíncrono: Comunicación donde el emisor no espera respuesta inmediata del receptor para continuar su procesamiento, permitiendo operaciones paralelas y mayor eficiencia en sistemas distribuidos.

Caché: Almacenamiento de datos de acceso frecuente en memoria de alta velocidad para reducir tiempos de respuesta y disminuir la carga sobre sistemas principales. Puede ser temporal o persistente según la configuración y estrategia implementada.

Redis: Sistema de almacenamiento de datos en memoria de código abierto que soporta persistencia, utilizado principalmente como base de datos de caché, broker de mensajes y almacén de estructuras de datos de alta performance.

Persistente: Característica de los datos que se almacenan de manera duradera en dispositivos de almacenamiento permanente (como discos), manteniendo la información disponible incluso después de reinicios del sistema o fallas de energía.

B2B (Business to Business): Modelo de negocio que describe las transacciones comerciales realizadas entre empresas, donde una organización vende productos o servicios a otra organización. Se caracteriza por volúmenes de transacciones mayores, contratos específicos, procesos de negociación más complejos y ciclos de venta más largos.

B2C (Business to Consumer): Modelo de negocio en el que las empresas venden productos o servicios directamente a consumidores finales. Se caracteriza por transacciones unitarias, procesos de compra más simples, marketing dirigido al consumidor individual y ciclos de venta más cortos.

CI/CD (Continuous Integration/Continuous Deployment): Metodología de desarrollo de software que automatiza la integración de código y el despliegue de aplicaciones. CI (Integración Continua) implica la fusión frecuente de cambios de código en un repositorio compartido, con pruebas automatizadas para detectar errores tempranamente. CD (Despliegue Continuo) automatiza la entrega y despliegue de aplicaciones a entornos de producción, reduciendo el tiempo entre el desarrollo y la puesta en producción.

2. Bases Tecnológicas

2.1 Arquitectura de Microservicios

Los microservicios forman un patrón de arquitectura que divide una aplicación en servicios pequeños, independientes y débilmente acoplados, donde cada servicio implementa una capacidad de negocio específica y puede ser desarrollado, desplegado y escalado de forma autónoma. Esta aproximación contrasta fundamentalmente con las arquitecturas monolíticas tradicionales, donde toda la funcionalidad vive en una única aplicación ejecutable con componentes fuertemente acoplados.

Aspecto	**Monolito**	**Microservicios**
Despliegue	*Una sola unidad*	*Servicios independientes*
Escalabilidad	*Todo el sistema*	*Solo servicios necesarios*
Tecnología	*Stack unico*	*Específica por servicio*
Fallas	*Compromete todo el sistema*	*Aislado por servicio*
Mantenimiento	*Cambios afectan todo el sistema*	*Modificaciones aisladas*
Equipos	*Coordinación centralizada*	*Equipos autónomos*

Ventajas

Escalabilidad Selectiva: permite dimensionar únicamente los servicios que experimentan mayor demanda, optimizando recursos y costos operativos. Por ejemplo, en sistemas como **APIODE (Api alta órdenes de envío)** los servicios que experimenten mayor demanda pueden requerir escalamiento independiente.

Resiliencia operativa: se incrementa mediante el aislamiento de fallas, donde problemas en un servicio no comprometen la operación global. Esta característica resulta crítica para mantener la continuidad del principal flujo de ingreso de órdenes, donde la disponibilidad parcial es preferible a la caída total del sistema.

Flexibilidad tecnológica: facilita la adopción de herramientas específicas para cada servicio sin impactar al resto, permitiendo una evolución gradual de los sistemas legacy hacia tecnologías modernas. La **independencia de desarrollo** acelera la entrega de funcionalidades o productos mínimos viables (MVP) al eliminar dependencias entre equipos y permitir ciclos de desarrollo paralelos.

Aplicación en el proyecto

La decisión de adoptar microservicios para APIODE se fundamenta en las limitaciones específicas del sistema monolítico existente. El sistema actual concentra validación, normalización, orquestación y persistencia en un único componente, generando cuellos de botella y dificultades de mantenimiento.

Problemática del sistema monolítico

En un sistema monolítico, estos procesos están fuertemente acoplados dentro de la misma aplicación, generando múltiples inconvenientes operacionales:

Cuellos de botella de performance: cuando el volumen de normalización de direcciones es alto, toda la aplicación se ve afectada, incluso procesos no relacionados como consultas de estado de ordenes de envio

Escalamiento ineficiente: es imposible escalar selectivamente solo la orquestación de flujos sin escalar todo el sistema, desperdiciando recursos computacionales

Rigidez en modificaciones: cambiar reglas de normalización requiere desplegar toda la aplicación, aumentando riesgos operacionales y afectando flujos no relacionados

Dependencias tecnológicas: todos los componentes deben usar el mismo stack tecnológico, limitando optimizaciones específicas

La descomposición en microservicios especializados permite:

Validación como servicio independiente: manejo optimizado de reglas de negocio complejas con capacidad de escalamiento según volumen de órdenes entrantes. Por ejemplo, el servicio de alta de ordenes de envio puede escalar a 10 instancias durante picos de carga mientras otros servicios mantienen 2 instancias.

Servicios de normalización: tratamiento especializado de datos con validaciones y transformaciones específicas sin impactar otros componentes. Por ejemplo, El servicio de normalización de localidades consulta datos cacheados persistentes en Redis para obtener información completa y estandarizada. Por ejemplo, cuando un cliente envía "CABA", el servicio consulta el caché y devuelve datos normalizados: "Ciudad Autónoma de Buenos Aires - Código Postal: C1000 - Provincia: CABA - Código INDEC: 02001". Al operar con datos cacheados, si otros servicios fallan (como APIs externas o bases de datos principales), la normalización de localidades continúa funcionando independientemente, sin afectar la validación de contratos ni la persistencia de la orden.

Orquestación distribuida: coordinación del flujo de trabajo mediante eventos asíncronos eliminando dependencias que antes eran sincrónicas. Ejemplo practico en APIODE: el flujo orquestado incluye: validar contrato → asignar numeración → normalizar localidad → normalizar sucursal → persistir en MongoDB → publicar en Kafka. Si falla la normalización de localidad, se puede reintentar ese paso específico sin repetir la validación de contrato ya completada.

Persistencia especializada: Gestión optimizada de bases de datos con patrones específicos para cada tipo de información. Por ejemplo, datos de caché persistentes en Redis para consultas frecuentes y MongoDB para persistencia duradera de órdenes de envio.

Esta arquitectura facilita además la migración gradual desde IBM MQ hacia Kafka, permitiendo que los servicios nuevos utilicen Apache Kafka y mientras se mantienen compatibilidad con sistemas que aún requieren el protocolo legacy. La implementación de patrones de resiliencia (como el Retry ya implementando en la librería de Arquitectura) asegura estabilidad operacional en un entorno distribuido que maneja el flujo crítico de ingreso de órdenes de Andreani.

2.2 Clean Architecture y Principios SOLID

Clean Architecture conceptualiza un patrón de organización de código que estructura las aplicaciones en capas concéntricas con dependencias unidireccionales hacia el centro, priorizando la independencia de frameworks, bases de datos y elementos externos. Esta aproximación arquitectónica separa claramente la lógica de negocio de los detalles de implementación, facilitando la mantenibilidad, testabilidad y evolución de sistemas complejos.

Organización por capas

La estructura se compone de capas concéntricas donde cada una tiene responsabilidades específicas.

En el **núcleo** residen las **Entidades**, que contienen las reglas de negocio fundamentales del dominio, como las características esenciales de una orden de envío o las validaciones críticas que raramente cambian.

Los **Casos de Uso** rodean este núcleo, implementando la lógica de aplicación específica que orquesta las reglas de negocio para cumplir requerimientos particulares del sistema.

Los **Adaptadores de Interfaz** actúan como traductores entre la lógica interna y el mundo exterior, incluyendo controladores web, repositorios de datos y servicios externos. Finalmente, la capa externa de **Infraestructura** contiene frameworks, bases de datos, APIs externas y otros detalles técnicos que pueden modificarse sin impactar las capas internas.

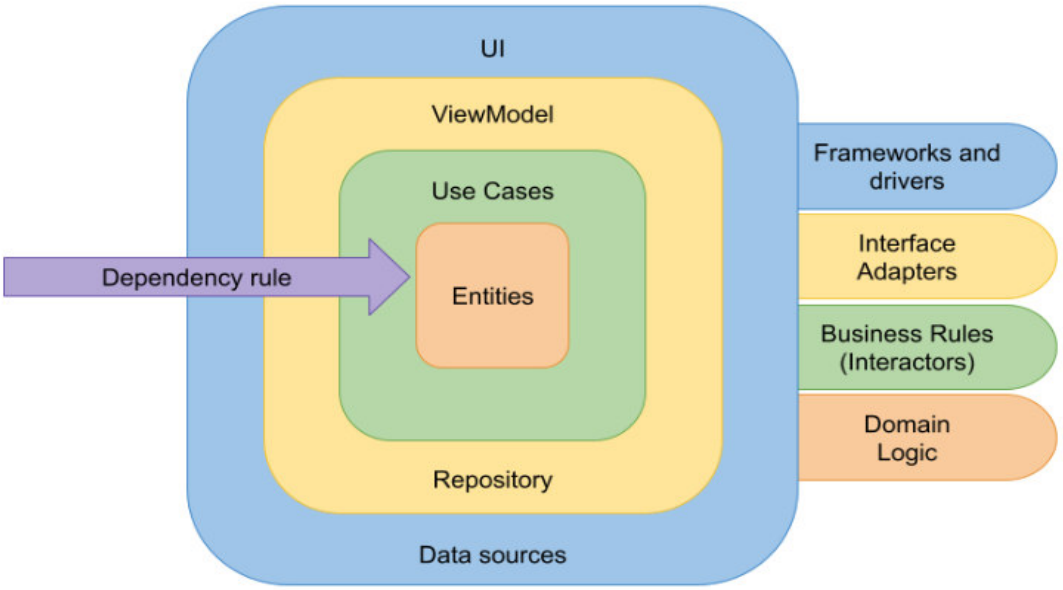


Figura 2.1: *Arquitectura Clean - Organización en capas concéntricas con dependencias unidireccionales hacia el núcleo*

Ventajas para arquitecturas de microservicios

Independencia tecnológica: permite cambiar frameworks, bases de datos o librerías sin afectar la lógica de negocio, facilitando actualizaciones tecnológicas y migraciones graduales.

Testabilidad mejorada: se logra al poder probar las reglas de negocio de forma aislada, sin dependencias externas como bases de datos o servicios web.

Mantenibilidad: se incrementa significativamente al tener código organizado predeciblemente, donde cada desarrollador puede localizar funcionalidades específicas siguiendo la estructura establecida.

Reutilización de componentes: se maximiza al centralizar la lógica de negocio en el núcleo del sistema, permitiendo que múltiples interfaces (web, móvil, APIs, servicios batch) compartan las mismas reglas de dominio sin duplicación de código ni inconsistencias entre canal

Aplicación en APIODE

La adopción de Clean Architecture en APIODE responde a la necesidad de organizar microservicios complejos con múltiples responsabilidades. Cada servicio implementa la misma estructura: las **Entidades** encapsulan reglas como validaciones de contratos o normalización de direcciones, los **Casos de Uso** orquestan flujos como “Validar que el contrato del usuario esté habilitado”, “Asignar Numeración” o “Validar tipo orden de envío”, y los **Adaptadores** manejan la comunicación con MongoDB, Redis y Kafka.

El template de Andreani materializa estos principios mediante una estructura de directorios estandarizada que organiza cada microservicio en capas claramente definidas: **Domain** para entidades y reglas de negocio, **Application** para casos de uso y lógica de aplicación, **Infrastructure** para persistencia y servicios externos, y **WebAPI** para controladores y presentación.

Figura 2.2: *Estructura de directorios del template corporativo de Andreani basado en Clean Architecture. Fuente: Documentación oficial de Arquitectura de Andreani*

Esta implementación práctica asegura que todos los microservicios de APIODE mantengan la misma organización, facilitando el mantenimiento y la colaboración entre equipos.

Esta organización facilita además la integración de nuevas funcionalidades y la modificación de componentes existentes sin riesgo de efectos colaterales no deseados. La separación clara de responsabilidades simplifica la implementación de patrones como inyección de dependencias y permite que cada capa evolucione independientemente según los requerimientos específicos del negocio.

Principios SOLID

Los principios SOLID complementan Clean Architecture proporcionando un conjunto de buenas prácticas que pueden ayudar a escribir código legible, mantenible y escalable.

1. Single Responsibility Principle (SRP): asegura que cada clase tenga una única razón para cambiar, simplificando mantenimiento y debugging.

2. Open/Closed Principle (OCP): permite extender funcionalidad sin modificar código existente, facilitando la adición de nuevas validaciones o transformaciones.

3. Liskov Substitution Principle (LSP): garantiza que las implementaciones derivadas puedan reemplazar sus abstracciones sin alterar el comportamiento del sistema.

4. Interface Segregation Principle (ISP): evita dependencias innecesarias al crear interfaces específicas para cada cliente. El

5. Dependency Inversion Principle (DIP): asegura que módulos de alto nivel dependen de abstracciones en lugar de implementaciones concretas.

Aplicación APIODE

El template corporativo de Andreani facilita la implementación de estos principios mediante la separación clara de capas de Clean Architecture.

Por ejemplo, el **SRP** se materializa al separar la lógica de negocio de las órdenes de envío (Application) de su persistencia (Infrastructure), cada una con responsabilidad específica. El **OCP** se evidencia en la capacidad de poder extender nuevos tipos de conexiones sin modificar las existentes, mientras que el **DIP** se implementa mediante interfaces en la capa Application que son implementadas por servicios concretos en Infrastructure, permitiendo que la lógica de negocio dependa de abstracciones y no de implementaciones específicas como MongoDB o Kafka.

Beneficios en el desarrollo de APIODE

La aplicación conjunta de Clean Architecture y principios SOLID adoptada en el template corporativo de Andreani facilita la **escalabilidad del equipo** al proporcionar una estructura predecible que acelera la incorporación de nuevos desarrolladores. La **calidad del código** se mantiene mediante principios claros que previenen acoplamiento excesivo y responsabilidades difusas.

La **facilidad de testing** permite implementar pruebas unitarias comprensivas al aislar lógica de negocio de dependencias externas. La **evolución gradual** del sistema se simplifica al poder modificar implementaciones específicas sin impactar la arquitectura general, es una característica fundamental para sistemas críticos que requieren actualizaciones continuas sin interrupciones operacionales.

2.3 Arquitectura Orientada a Eventos

Arquitectura Orientada a Eventos (EDA)

Es un patrón de comunicación donde los servicios interactúan mediante la producción y consumo de eventos, eliminando dependencias síncronas directas entre estos. En este paradigma, los eventos representan hechos significativos que han ocurrido en el sistema, como la creación o modificación de un recurso de sistema, como puede ser el “envío”, y fluyen a través de una plataforma de eventos permitiendo que múltiples servicios reaccionen de manera independiente.

Esta aproximación contrasta fundamentalmente con la comunicación síncrona tradicional (request-response), donde un servicio debe esperar la respuesta de otro antes de continuar su procesamiento. Por ejemplo, en este tipo de comunicación, si el servicio B falla, el servicio A queda bloqueado esperando respuesta.

En EDA, los servicios publican eventos cuando completan sus operaciones y consumen los eventos de interés para ejecutar sus propias lógicas de negocio, creando un sistema más resiliente y débilmente acoplado.

Los eventos actúan como contratos entre servicios, definiendo la estructura de datos y el significado del hecho ocurrido, sin exponer detalles de implementación interna. Un servicio productor puede publicar "CrearOrden" sin conocer los servicios interesados que consumen este evento, permitiendo agregar nuevos consumidores sin modificar el productor original.

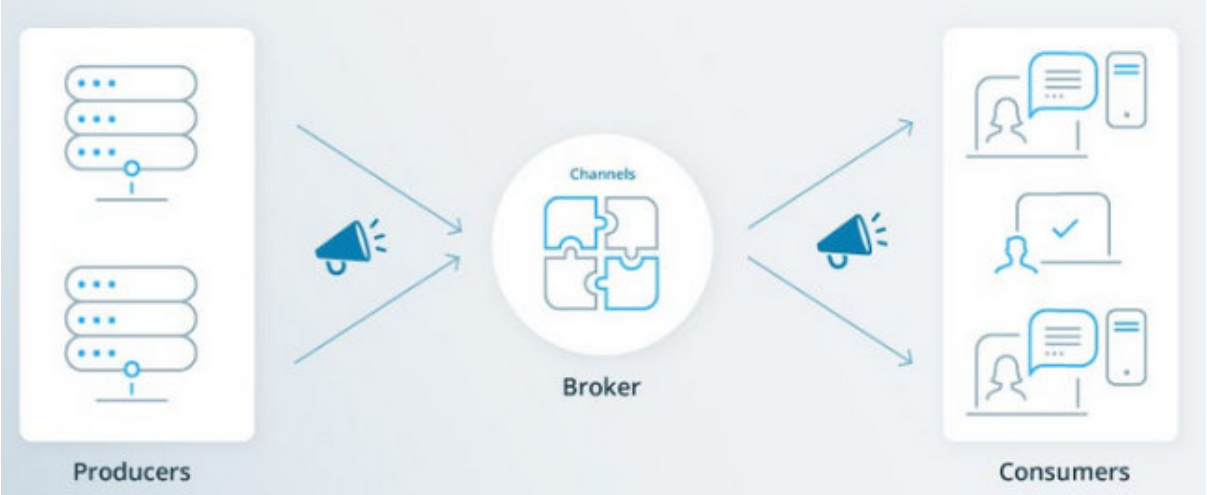


Figura 2.3: *Arquitectura Orientada a Eventos (EDA) - Comunicación asíncrona mediante broker de eventos donde productores y consumidores están desacoplados*

Apache Kafka: Plataforma de Eventos

Kafka implementa el paradigma EDA funcionando como una plataforma de streaming distribuida diseñada para manejar flujos de datos en tiempo real.

Opera como un sistema de mensajería pub/sub (publicación/suscripción) de alto rendimiento, donde los mensajes se almacenan de manera duradera y pueden ser procesados múltiples veces por diferentes aplicaciones.

A diferencia de los sistemas de mensajería tradicionales como IBM MQ, que operan como colas donde los mensajes se consumen una vez y desaparecen, Kafka

mantiene un historial persistente de eventos que permite replay y procesamiento la cantidad de veces que sean necesarias por múltiples consumidores independientes.

Conceptos fundamentales de Kafka

Brokers: constituyen los servidores individuales que forman el cluster de Kafka, siendo análogos a servidores tradicionales pero especializados en almacenar y servir streams de datos. Cada broker maneja múltiples tópicos y mantiene réplicas para garantizar disponibilidad. Un cluster típico de producción incluye múltiples brokers distribuidos geográficamente para máxima resiliencia.

Tópicos: son como canales nombrados donde se organizan los mensajes por tema, como "ApiAltaOrdenes-CrearOrdenDeEnvio" o "Pagos-TransaccionCompletada". Cada tópico se divide en **particiones** que permiten procesamiento paralelo y distribución de carga entre múltiples consumidores. Las particiones también mantienen el orden de los mensajes dentro de cada división.

Productores: son aplicaciones que publican mensajes en tópicos específicos, determinando en qué partición se almacenará cada mensaje mediante claves (key) de particionamiento.

Consumidores: leen mensajes de los tópicos, manteniendo su propia posición de lectura mediante **offsets**, que representan identificadores únicos secuenciales para cada mensaje dentro de una partición.

Consumer Groups: permiten que múltiples instancias de una aplicación (miembros) trabajen en paralelo, distribuyendo automáticamente las particiones entre los miembros del grupo para maximizar el rendimiento. Si una instancia falla, las particiones se redistribuyen automáticamente entre los miembros restantes del grupo.

Características técnicas clave

La **persistencia** en Kafka es configurable por tópico, permitiendo almacenamiento temporal para casos de uso como notificaciones en tiempo real, o persistencia permanente para sistemas que requieren auditoría completa y capacidad de replay.

Los datos se almacenan en disco con configuraciones de retención basadas en tiempo o tamaño.

Los **offsets** no solo identifican la posición del mensaje dentro de la partición de un tópico sino que también permiten que los consumidores procesen datos desde cualquier punto en el tiempo, facilitando el reprocesamiento y recuperación ante fallas. Un consumidor puede reiniciar desde el offset más antiguo para reprocesar todos los eventos históricos, o desde algún punto en el tiempo más reciente para procesar solo eventos nuevos.

Las **réplicas** garantizan alta disponibilidad mediante la duplicación automática de datos entre múltiples brokers. Cada partición mantiene una réplica líder que maneja las escrituras y lecturas, mientras las réplicas seguidoras sincronizan los datos. El factor de replicación es configurable según los requerimientos de disponibilidad versus performance.

La **garantía exactly-once delivery** asegura que cada mensaje se entregue una única vez, eliminando problemas de publicación duplicada que podrían generar inconsistencias en sistemas críticos como el procesamiento de pagos o creación de órdenes.

Ventajas para arquitecturas de microservicios

Escalabilidad horizontal: permite agregar brokers dinámicamente para manejar incrementos de carga, mientras que el **particionamiento de los tópicos** permite la distribución de la carga entre múltiples consumidores.

Alto rendimiento: kafka puede manejar millones de mensajes por segundo con latencias de milisegundos, superando significativamente las capacidades de sistemas de mensajería tradicionales.

Desacoplamiento: permite que productores y consumidores operen a velocidades diferentes sin bloqueos mutuos y sin necesidad de conocerse directamente facilitando la integración de nuevos sistemas.

Durabilidad y disponibilidad: La replicación configurable asegura que los eventos persistan incluso ante fallas de brokers individuales, manteniendo la continuidad operativa de flujos críticos.

Implementación en APIODE

En el sistema APIODE, Kafka reemplaza gradualmente a IBM MQ como plataforma de mensajería, implementando dos tópicos principales con configuración optimizada para producción:

ApiAltaOrdenes-CrearOrdenEnvio: Tópico donde se publican las órdenes de envío recién persistidas en MongoDB. Configurado con 10 particiones para permitir hasta 10 consumidores paralelos y 3 réplicas para alta disponibilidad.

ApiAltaOrdenes-OrdenEnvioSolicitada: Tópico donde se publican órdenes ya procesadas y normalizadas, disponibles para múltiples sistemas consumidores. Configurado con la misma especificación de particiones y réplicas para consistencia operacional.

El flujo completo de eventos en APIODE ilustra la potencia de la arquitectura orientada a eventos:

1. **API de Preenvíos Alta** recibe y valida órdenes de clientes, persistiéndolas en MongoDB con estado "Pendiente".
2. **Sync-Kafka** (worker) se ejecuta cada 30 segundos, consulta MongoDB por órdenes pendientes de publicación y las publica en lotes en ApiAltaOrdenes-CrearOrdenEnvio.
3. **PublisherTMS** consume eventos de ApiAltaOrdenes-CrearOrdenEnvio, realiza normalización y formateo de datos, y publica en ApiAltaOrdenes-OrdenEnvioSolicitada.
4. **Sync-Kafka-MQ** consume de ApiAltaOrdenes-OrdenEnvioSolicitada y publica en IBM MQ para mantener compatibilidad con sistemas legacy durante la transición.

5. **Múltiples consumidores externos** se suscriben a ApiAltaOrdenes-OrdenEnvioSolicitada para realizar operaciones específicas: seguimiento, notificaciones, integraciones con sistemas.

La implementación incluye tópicos de retry automáticos que manejan fallos temporales. Cuando un consumidor no puede procesar un evento, el evento se mueve automáticamente a un tópico de retry donde se reintenta el procesamiento después de un delay configurable, asegurando que fallas temporales no resulten en pérdida de datos.

Los grupos de consumidores permiten escalabilidad horizontal donde múltiples instancias de los servicios consumidores y publicadores procesan mensajes en paralelo sin conflictos.
La estrategia de compatibilidad dual publica eventos tanto en Kafka como en IBM MQ mediante el componente Synk-kafka-mq, facilitando migración gradual sin interrupciones operacionales. Los servicios legacy continúan consumiendo de MQ mientras los modernizados aprovechan las capacidades de Kafka para implementar patrones más sofisticados de procesamiento de eventos.
La configuración de retención permite realizar auditorias para órdenes críticas mientras optimiza se puede optimizar el procesamiento para eventos de menor importancia. Los patrones de resiliencia como retry automático y configuraciones de timeout manejan fallas temporales, mientras que la capacidad de reprocesamiento facilita la recuperación ante errores de lógica de negocio o problemas de infraestructura.

2.4 .NET 8 y Template de Desarrollo

.NET 8 representa la evolución más reciente de la plataforma de desarrollo de Microsoft, es multiplataforma, alto rendimiento, permite desarrollar aplicaciones

rápidamente. Esta versión LTS (Long Term Support) ofrece estabilidad y soporte extendido hasta 2026, características fundamentales para sistemas empresariales críticos donde la continuidad y predictibilidad del soporte son requisitos no negociables.

La adopción de .NET 8 como estándar corporativo en Andreani responde no solo a consideraciones técnicas sino también a aspectos estratégicos relacionados con un ecosistema de herramientas maduro y alineación con la arquitectura empresarial existente.

Características técnicas destacadas

Performance optimizado: Las mejoras introducidas en .NET 8 resultan especialmente relevantes para aplicaciones de alta concurrencia como el sistema de alta de envíos. Las optimizaciones en el garbage collector, compilación ahead-of-time (AOT) y manejo de memoria reducen significativamente la latencia y mejoran el rendimiento del sistema, características críticas para mantener tiempos de respuesta consistentes bajo carga variable.

Containerización optimizada: La compatibilidad mejorada con Docker y contenedores permite despliegues consistentes a través de diferentes ambientes, desde desarrollo local hasta producción en Kubernetes. Las optimizaciones específicas para contenedores reducen el tiempo de startup y el footprint de memoria, aspectos fundamentales para arquitecturas de microservicios que requieren escalamiento rápido.

Ecosistema de librerías maduro: ASP.NET Core proporciona un framework web de alto rendimiento, mientras que Entity Framework Core facilita el acceso a datos con soporte para múltiples proveedores de bases de datos incluyendo MongoDB a través de drivers oficiales. La integración con sistemas de configuración permite la gestión centralizada de settings específicos por ambiente, facilitando la administración de secretos y configuraciones en entornos empresariales complejos.

Características del lenguaje avanzadas: La compatibilidad con C# 12 introduce características como records, pattern matching mejorado y nullable reference types

que contribuyen a la escritura de código más seguro y expresivo. Estas características lingüísticas, combinadas con las robustas herramientas de Visual Studio y el ecosistema de análisis estático, establecen una base sólida para el desarrollo de software de calidad empresarial.

Ventajas para microservicios empresariales

Desarrollo acelerado: El ecosistema maduro de .NET acelera significativamente el tiempo de desarrollo mediante librerías estables, documentación comprensiva y herramientas integradas que automatizan tareas repetitivas.

Integración empresarial: La compatibilidad nativa con estándares empresariales como OAuth 2.0, OpenAPI/Swagger, y protocolos de monitoreo facilita la integración con sistemas existentes y herramientas corporativas sin requerir adaptadores o middleware adicionales.

Escalabilidad y mantenibilidad: El modelo de hosting flexible permite desde aplicaciones monolíticas hasta microservicios distribuidos, mientras que la inyección de dependencias nativa y el soporte para patrones como mediator facilitan arquitecturas limpias y testeables.

Seguridad integrada: Las características de seguridad built-in incluyen protección automática contra vulnerabilidades comunes como XSS, CSRF y SQL injection, reduciendo la superficie de ataque y simplificando el cumplimiento de estándares de seguridad corporativos.

Template Corporativo de Andreani

El template desarrollado por el área de Arquitectura de Software materializa las mejores prácticas de .NET 8 en una estructura estandarizada que implementa Clean Architecture de manera consistente. Esta plantilla no solo acelera el desarrollo al proporcionar diversas librerías preconfiguradas y normalizadas, sino que también asegura uniformidad arquitectónica entre todos los microservicios de la organización.

La estructura del template organiza cada proyecto en capas claramente definidas: **Domain** para entidades y reglas de negocio, **Application** para casos de uso y lógica de aplicación, **Infrastructure** para persistencia y servicios externos, y **WebAPI** para controladores y presentación. Esta organización facilita la navegación del código y establece contratos claros entre capas que previenen violaciones arquitectónicas.

El template incluye configuraciones pre-establecidas para logging estructurado con Serilog, métricas con APM, manejo de errores centralizado, documentacion automatica de endpoints con Swagger, y patrones de resiliencia como circuit breakers y retry policies, estandarización del uso de Kafka. Estas configuraciones eliminan código repetitivo y aseguran que todos los servicios implementen las mismas prácticas de observabilidad y manejo de errores.

Aplicación en APIODE

La implementación de APIODE aprovecha integralmente el template corporativo y las capacidades de .NET 8. Cada microservicio de APIODE utiliza la misma estructura base, facilitando la colaboración entre desarrolladores y reduciendo la curva de aprendizaje para nuevos integrantes del equipo.

Las características específicas implementadas incluyen controllers con documentación automática, servicios de dominio que encapsulan lógica de negocio compleja como validación de contratos, localidades, sucursales y asignación de numeración, y repositorios que abstraen el acceso a MongoDB y Redis manteniendo la independencia de la lógica de negocio respecto a detalles de persistencia.

La integración con Kafka se implementa mediante la librería corporativa AMQStrams que se ejecutan en los servicios de APIODE , aprovechando las capacidades de la plataforma de eventos **“detalladas en la seccion 2.2 Apache kafka y Arquitectura Orientada a Eventos”**. Los patrones de retry configurables manejan fallos temporales de conectividad, mientras que el logging estructurado proporciona trazabilidad completa de cada operación para facilitar debugging y la utilización del APM para el monitoreo operacional.

2.5 Monitoreo/Observabilidad: Kibana y APM

La observabilidad en arquitecturas de microservicios trasciende el monitoreo tradicional, constituyendo una capacidad fundamental para comprender el comportamiento emergente de sistemas distribuidos complejos. En el contexto de Andreani, la observabilidad se implementa a través del Elastic Stack, donde Kibana actúa como la interfaz principal para visualización y análisis, mientras que APM (Application Performance Monitoring) proporciona insights granulares sobre la performance y comportamiento de las aplicaciones.

Kibana permite la creación de dashboards interactivos que correlacionan métricas operacionales, logs de aplicación y eventos de negocio en una vista unificada en tiempo real.

Esta capacidad resulta crítica para sistemas como APIODE, donde la comprensión del flujo end-to-end de una orden de envío requiere visibilidad en cada uno de los microservicios, colas de mensajes y sistemas de persistencia.

Los dashboards desarrollados incluyen métricas de latencia por servicio, rendimiento de procesamiento, distribución de códigos de error y correlaciones temporales que facilitan la identificación proactiva de degradaciones de performance.

El componente APM del Elastic Stack proporciona trazas distribuidas automáticas que siguen flujos individuales a través de toda la arquitectura de microservicios. Esta funcionalidad permite identificar cuellos de botella específicos, analizar patrones de latencia y comprender las dependencias entre servicios. La instrumentación automática disponible en .NET 8 minimiza el overhead de implementación, generando trazas detalladas sin requerir modificaciones extensivas en el código de aplicación. Las métricas de APM incluyen tiempo de respuesta detallados, tasa de errores por endpoint, y análisis de dependencies que son fundamentales para mantener SLAs (Acuerdo de Nivel de Servicio) operacionales en un sistema crítico como el de alta de envíos.

La integración entre Kibana y APM facilita la creación de alertas proactivas basadas en umbrales dinámicos y análisis de tendencias. Estas alertas permiten la detección temprana de anomalías operacionales, reduciendo significativamente el tiempo medio de resolucion y detección de incidentes que podrían impactar la operación comercial de Andreani.

2.6 SonarQube

SonarQube constituye una plataforma integral para la gestión continua de la calidad de código, proporcionando análisis estático automatizado que identifica vulnerabilidades de seguridad, code smells, bugs potenciales y violaciones de estándares de codificación. En el contexto del desarrollo de APIODE, SonarQube se integra como una herramienta fundamental del pipeline de CI/CD, asegurando que todo código que ingresa al repositorio principal cumple con los estándares de calidad establecidos por el área de Arquitectura de Software de Andreani.

La plataforma analiza múltiples dimensiones de calidad incluyendo mantenibilidad, confiabilidad y seguridad.

Análisis de mantenibilidad: identifican código duplicado, complejidad excesiva y violaciones de principios de Clean Code, facilitando la refactorización proactiva antes de que estos problemas impacten la productividad del equipo de desarrollo.

Análisis de confiabilidad: detectan bugs potenciales, condiciones de null pointer exceptions y problemas de manejo de recursos que podrían causar fallas en tiempo de ejecución.

Análisis de seguridad: identifican vulnerabilidades conocidas, patrones de código inseguro y dependencias con problemas de seguridad documentados.

La integración de SonarQube con el template corporativo de .NET 8 permite la aplicación automática de reglas específicas alineadas con los estándares de la

empresa. Estas reglas incluyen convenciones de nomenclatura y mejores prácticas específicas para el desarrollo de microservicios.

El Quality Gate configurado establece límites específicos que deben cumplirse antes de permitir el merge de código, incluyendo cobertura mínima de pruebas unitarias del 80%, ausencia de vulnerabilidades de seguridad críticas y límites máximos de complejidad por método.

Los reportes generados por SonarQube proporcionan métricas objetivas sobre la evolución de la calidad del código a lo largo del tiempo, facilitando la identificación de tendencias y la toma de decisiones informadas sobre refactorización y mejoras técnicas. Esta visibilidad resulta particularmente valiosa en proyectos de modernización como APIODE, donde la migración desde un sistema legacy requiere asegurar que la nueva implementación mantiene o mejora los estándares de calidad establecidos.

3. Diseño e implementación del sistema APIODE

3.1 Arquitectura general

El sistema APIODE materializa los principios arquitectónicos establecidos en el capítulo anterior mediante una implementación práctica que transforma el flujo monolítico de alta de envíos en un **ecosistema de microservicios especializados**. La arquitectura resultante no solo resuelve las limitaciones técnicas del sistema anterior, sino que también establece las bases para futuras evoluciones e integraciones empresariales.

La solución se organiza en un conjunto de microservicios que colaboran mediante **eventos asíncronos**, cada uno con responsabilidades específicas y métricas de desempeño que evidencian su efectividad.

Tabla 3.1 – Microservicios principales de APIODE

Microservicio	Función principal	Métrica clave / Resultado
API de Preenvíos Alta	Punto de entrada. Recibe órdenes, válida contratos/sucursales, normaliza localidades y asigna numeración.	Latencia promedio 136 ms; throughput > 100 TPS; disponibilidad alta y estable.
Sync-Kafka	Pública en batch las órdenes pendientes desde MongoDB hacia Kafka, con control de concurrencia.	Latencia 121 ms; throughput ~114 transacciones/min; 0% errores de duplicación.
PublisherTMS	Transforma y normaliza datos de órdenes, publicando en el tópico final para consumo interno/externo.	Latencia 31 ms; throughput ~96 transacciones/min; escalabilidad horizontal con 10 particiones.
Sync-Kafka-MQ	Replica eventos de Kafka hacia IBM MQ para mantener compatibilidad con sistemas legacy.	3.817 publicaciones exitosas en MQ; 0 fallos de envío.

Worker-Contratos	Mantiene actualizado el caché de contratos en Redis para validaciones rápidas.	Latencia 127 ms; hit rate >95%.
Worker-Localidades	Mantiene actualizado el caché de localidades normalizadas.	Latencia 180 ms; throughput ~2 transacciones/min; hit rate >95%.
Worker-Numeración	Gestiona numeración incremental de envíos en Redis con consultas SQL atómicas.	Latencia 21 ms; throughput ~0.3 transacciones/min.

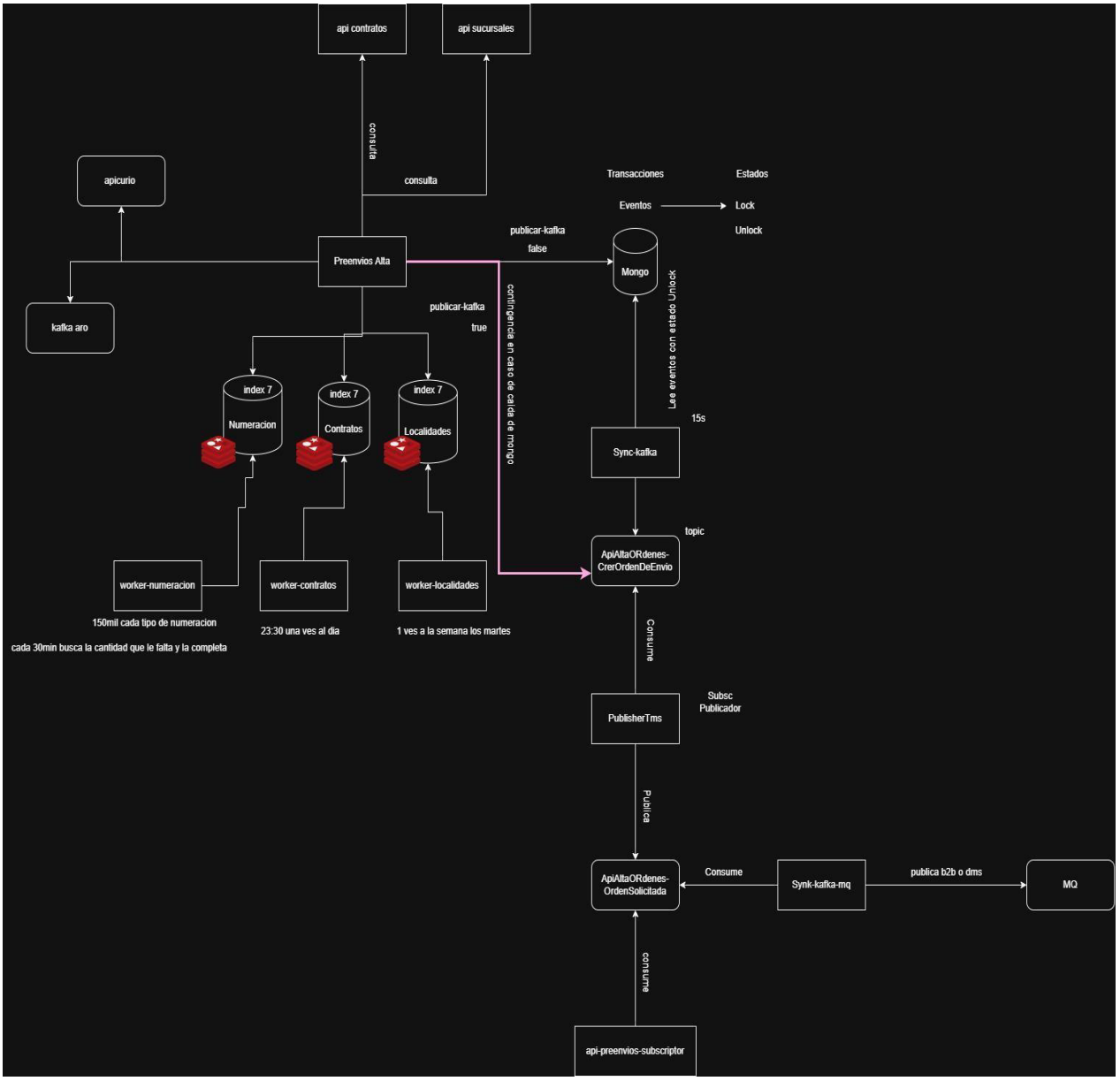


Figura 3.1 *ilustra el* ***flujo general del sistema****, donde una orden de envío atraviesa distintas etapas: recepción y validación, publicación asíncrona, procesamiento y normalización, compatibilidad con sistemas legacy y consumo distribuido por distintas áreas*

Tras esta visión general de la arquitectura, el siguiente apartado (3.2) profundiza en cada componente de la solución, mostrando cómo se materializan los principios arquitectónicos en microservicios concretos.

3.2 Microservicios Implementados

La implementación de APIODE materializa los principios arquitectónicos mediante siete microservicios especializados que colaboran para transformar el flujo monolítico en un ecosistema distribuido y resiliente. Cada componente implementa el template corporativo de .NET 8 basado en Clean Architecture, asegurando consistencia y mantenibilidad en todos los microservicios y componentes de la solución.

3.2.1 API de Preenvíos Alta - Punto de entrada principal

La API de Preenvíos constituye el servicio que expone los endpoints para recepción de órdenes de envío. Implementa validaciones complejas de negocio incluyendo verificación de contratos y sucursales mediante APIs externas, normalización de localidades consultando cachés persistentes de Redis, y asignación de numeración según tipo de envío (B2C/B2B).

Las métricas de APM demuestran performance consistente con latencia promedio estable y alta disponibilidad, evidenciando la solidez de la implementación. El servicio persiste órdenes en MongoDB con estado "Pendiente" y estado de sincronización "Unlock", habilitándolas para procesamiento posterior por el worker Sync-Kafka.

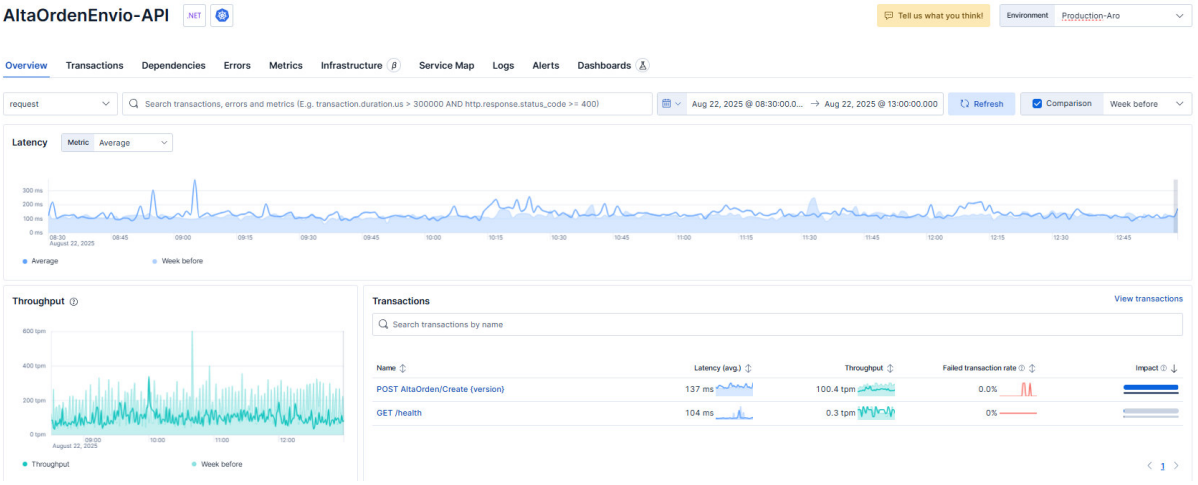


Figura 3.2 : *APM - API de Preenvíos Alta: Métricas de performance mostrando latencia promedio de 136ms, throughput de 116.2 TPS.*

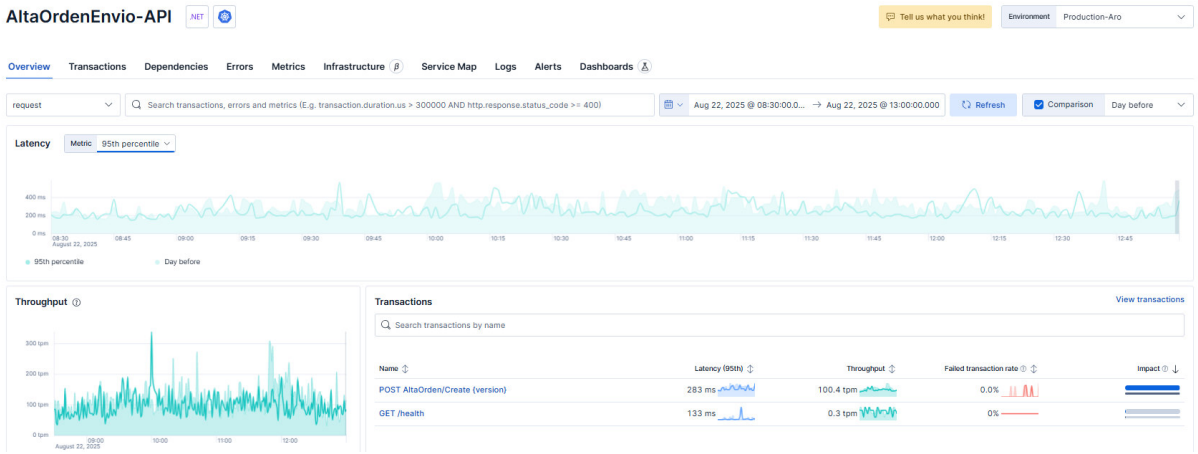


Figura 3.3: *APM - API de Preenvíos Alta: Métricas de performance mostrando latencia promedio de 136ms, throughput de 100.4 TPS y distribución de latencia con percentil 95 bajo los 400ms cumpliendo el SLA operacional.*

Visibilidad operacional en tiempo real:

La implementación incluye dashboards operacionales especializados que proporcionan visibilidad granular del comportamiento de la API en tiempo real. Como se evidencia en las Figuras 3.4 y 3.5, estos dashboards permiten:

Identificación inmediata por cliente: Distribución de altas por contrato con nombres de clientes para identificación rápida (AZTRAZENECA SOCIEDAD ANONIMA, FORD ARGENTINA, TARJETA NARANJA, etc.)

Segmentación operacional inteligente: Proporción clara entre envíos B2B (84.1%) y B2C (15.9%), facilitando análisis de patrones de uso

Métricas de volumen granulares: Cantidad de bultos por envío y distribución por contrato, permitiendo identificación proactiva de clientes con alta demanda

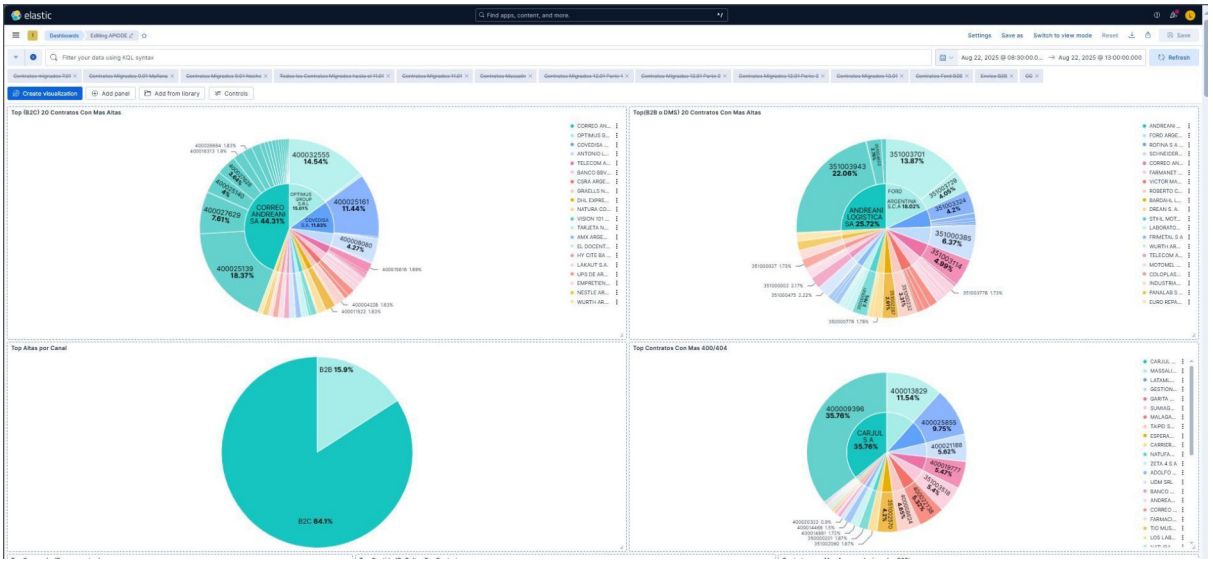


Figura 3.4: *Dashboard APIODE - Distribución operacional: Top contratos con mayor volumen de altas, segmentación por cliente y análisis de tipos de envío con identificación inmediata de patrones comerciales.*

Figura 3.5: Dashboard APIODE - Análisis granular: Distribución de bultos por contrato, segmentación B2B/B2C detallada, y métricas de contratos DMS vs legacy para gestión de migración gradual.

El servicio implementa patrones de fallback para publicación directa en Kafka cuando la base de datos presenta indisponibilidad temporal, asegurando continuidad operativa del flujo crítico.

3.2.2 Sync-Kafka - Sincronizador de eventos con control de concurrencia

El worker Sync-Kafka implementa un patrón de polling cada 15 segundos desde que finalizó la última publicación de eventos , también incluye mecanismos de control de concurrencia para evitar condiciones de carrera. Las métricas muestran latencia promedio de 121ms con throughput de 114.3 transacciones por minuto, reflejando su naturaleza de procesamiento batch optimizado.

El mecanismo de sincronización opera mediante estados atómicos en MongoDB: consulta documentos en estado "**Unlock**", los marca como "**Lock**" usando consultas transaccionales para atomicidad, y procesa los eventos resultantes. Este patrón permite escalamiento horizontal seguro cuando se requieren múltiples instancias del worker sin duplicación de procesamiento.

La implementación utiliza consultas paginadas configurables para evitar degradación de performance en MongoDB cuando existen grandes volúmenes de

órdenes pendientes. El procesamiento incluye deserialización de eventos y publicación batch en el topico **ApiAltaOrdenes-CrearOrdenEnvio**, manteniendo una tasa de errores del 0% como demuestra el APM, y manejo de reintentos configurables ante fallas temporales de Kafka.

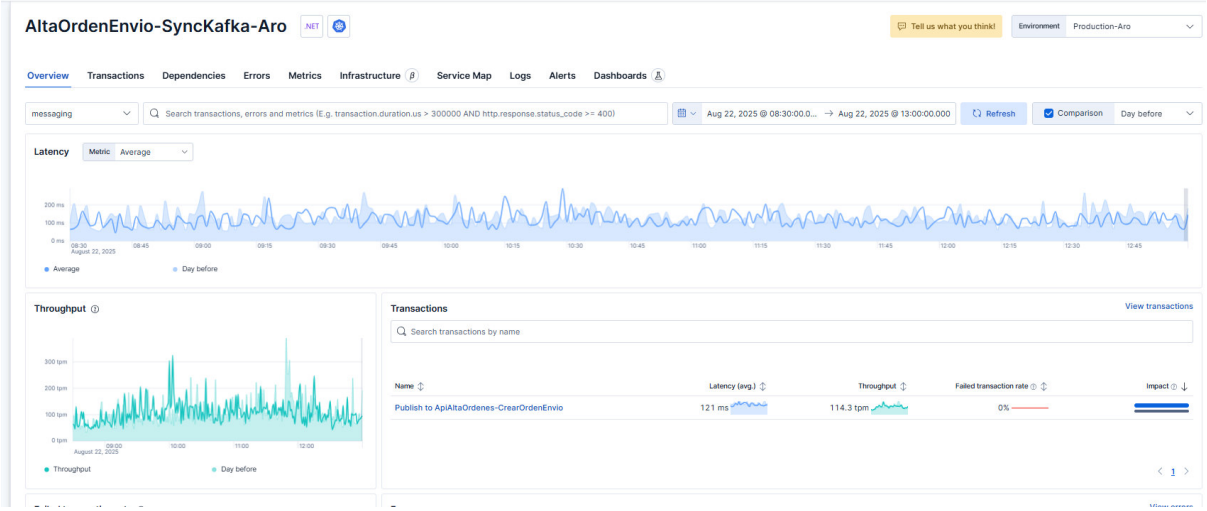


Figura 3.6: *APM - Sync-Kafka Worker: Métricas de performance mostrando latencia estable 121ms, throughput de 114.3 transacciones por minuto con picos de hasta 300+ tpm durante períodos de alta demanda, y 0% de errores en el procesamiento batch.*

3.2.3 PublisherTMS - Transformador de estructura de datos

PublisherTMS actúa como consumidor del topico ApiAltaOrdenes-CrearOrdenEnvio y publicador de ApiAltaOrdenes-OrdenEnvioSolicitada, implementando la lógica de mapeo y transformación entre estructuras de datos.

El servicio ejecuta transformaciones específicas mapeando la estructura CrearOrdenEnvio (estado inicial) hacia OrdenEnvioSolicitada (estado Solicitada), normalizando formatos y completando información requerida por sistemas. Las trazas de APM muestran el procesamiento detallado: procesamiento y consumo de tópico (1.6ms) → publicación resultado (31ms).

Figura 3.7: *APM - PublisherTMS: Métricas de performance mostrando latencia estable 31ms, throughput de 96.4 transacciones por minuto para el consumidor y latencia estable de 1.6ms y throughput de 95.7 transacciones por minuto, y 0% de errores en el procesamiento.*

La capacidad de procesar múltiples eventos en paralelo mediante Consumer Groups permite escalabilidad horizontal que se puede ajustar según la demanda, aprovechando las 10 particiones configuradas en los tópicos para distribución eficiente de carga.

3.2.4 Sync-Kafka-MQ - Puente de compatibilidad legacy

Este componente especializado consume eventos del topico **AltaOrdenEnvio-OrdenEnvioSolicitada** y los replica en IBM MQ, manteniendo compatibilidad con sistemas legacy durante la transición tecnológica. Las métricas muestran latencia promedio de 43ms con throughput de 17.5 transacciones por minuto.

La implementación incluye transformaciones de formato específicas para IBM MQ y manejo de reconexiones automáticas ante fallas temporales de conectividad. Las trazas distribuidas evidencian el flujo: consumo y procesamiento Kafka (50ms) → publicación MQ (334ms).

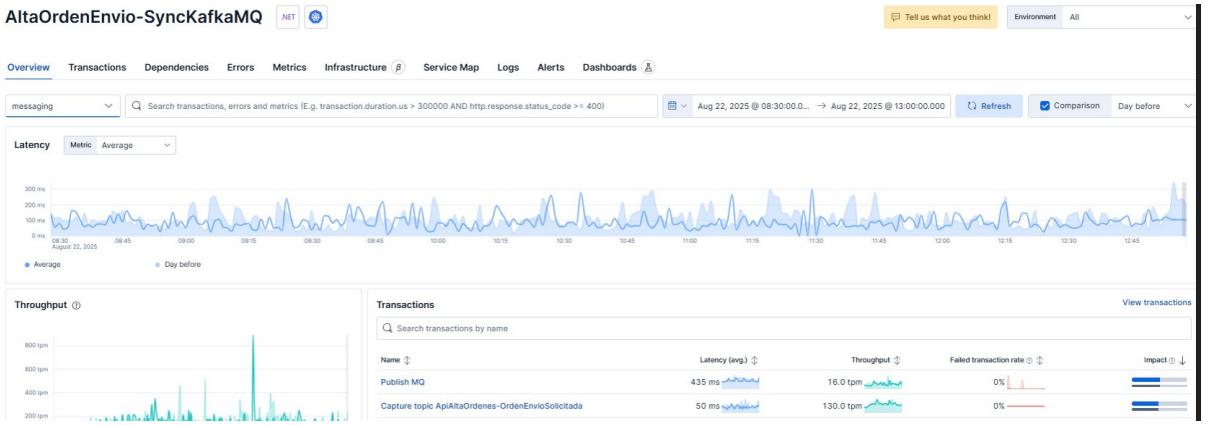


Figura 3.8: *APM - Sync-Kafka-MQ Worker: Métricas de performance mostrando latencia estable 435ms, throughput de 16 transacciones por minuto para la publicacion en MQ y latencia estable 50ms, throughput de 130 transacciones por minuto en el consumidor*

Como se evidencia en la **Figura 3.9**, el dashboard operacional muestra 3,817 publicaciones exitosas en MQ con 0 fallos, demostrando la confiabilidad de la estrategia de compatibilidad dual implementada para facilitar migración gradual sin interrupciones.

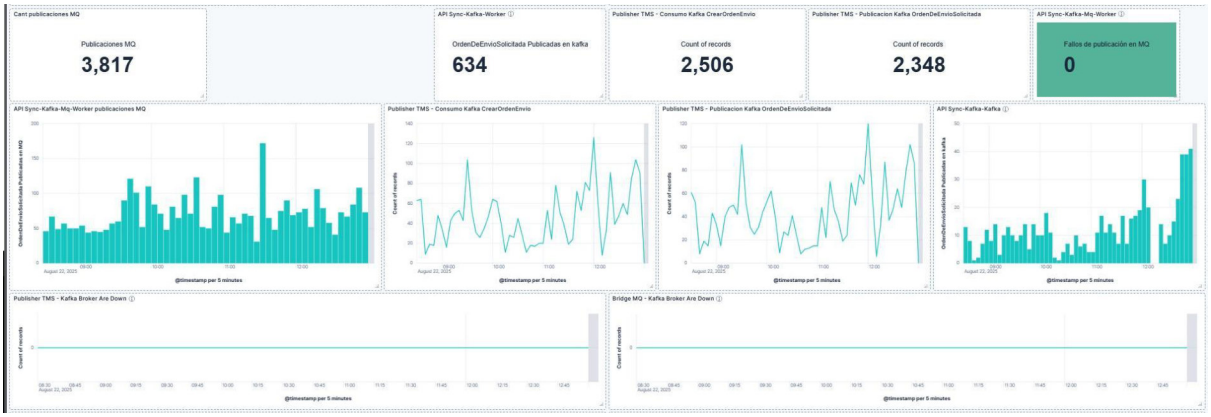


Figura 3.9: *Dashboard APIODE - Métricas consolidadas: Publicaciones MQ (3,817), órdenes solicitadas en Kafka (634), throughput por servicio y 0 fallos de publicación, evidenciando la efectividad del puente de compatibilidad legacy.*

3.2.5 Workers especializados - Gestión de cachés persistentes

Los tres workers (Contratos, Localidades, Numeración) mantienen cachés persistentes actualizados en Redis para optimizar performance de consultas frecuentes. Worker-Contratos ejecuta con latencia de 127ms y throughput de 0.3 transacciones por minuto, mientras Worker-Localidades opera con 180ms y 2.0 transacciones por minuto.

Worker-Numeración demuestra la mayor eficiencia con latencia de solo 21ms y throughput de 0.3 transacciones por minuto, reflejando la simplicidad de sus operaciones de incremento atómico. Las trazas muestran consultas SQL optimizadas (285ms) seguidas de actualización Redis (11ms), evidenciando el patrón cache-aside implementado.

Los workers implementan estrategias de retry configurables y logging detallado para facilitar debugging operacional. La sincronización periódica asegura consistencia eventual entre fuentes autoritativas y cachés persistentes*, mientras que las métricas de Redis confirman hit rates superiores al 95% para consultas de la API principal.

3.3 Estrategia de Migración y Implementación

La migración desde el sistema monolítico hacia APIODE se ejecutó siguiendo una estrategia de implementación gradual que priorizó la continuidad operativa y la mitigación de riesgos. Esta aproximación permitió validar cada componente del nuevo sistema antes de escalar a toda la base de clientes, manteniendo siempre un mecanismo de rollback disponible ante cualquier imprevisto.

3.3.1 Fase de Desarrollo y Validación Interna

El desarrollo de APIODE se completó exitosamente utilizando el template corporativo de .NET 8 y siguiendo los principios de Clean Architecture establecidos por el área de Arquitectura. Cada microservicio se implementó como un proyecto

independiente con su propio pipeline de CI/CD, permitiendo desarrollo paralelo sin interferencias entre componentes.

Durante esta fase, se establecieron métricas base y configuraciones de observabilidad que serían críticas para el monitoreo posterior. Los dashboards de Kibana se configuraron para capturar métricas específicas de cada servicio, incluyendo latencia, throughput, tasa de errores y trazas distribuidas mediante APM.

3.3.2 Implementación del Cliente Piloto

La primera fase de la migración consistió en habilitar APIODE para un cliente interno específico mediante una ruta dedicada expuesta en el AWS Gateway. Esta estrategia permitió validar el comportamiento del sistema completo bajo condiciones reales de tráfico sin exponer riesgos a la base de clientes externa.

El cliente piloto seleccionado representaba un volumen de transacciones significativo pero controlado, permitiendo identificar y resolver cualquier inconsistencia operacional antes de la migración masiva. Durante esta fase, se observaron intensivamente las métricas de performance y se ajustaron configuraciones específicas de timeouts, pools de conexiones y políticas de retry según los patrones de tráfico observados.

Resultados de la fase piloto:

Latencia promedio: Reducción aproximada del 25% en promedio y del 80% durante altas demandas comparado con el sistema anterior

Disponibilidad: 99% durante el período de prueba de 2 meses

Trazabilidad: Visibilidad end-to-end completa de cada transacción mediante APM

Errores de validación: 6 casos menores detectados y resueltos en menos de 3 horas

3.3.3 Migración General y Redireccionamiento del Gateway

Una vez validada la estabilidad con el cliente piloto, se procedió al redireccionamiento del AWS Gateway desde el sistema monolítico hacia APIODE. Esta operación se realizó fuera del horario pico para minimizar el impacto potencial, con un plan de rollback configurado para ejecutarse en caso de detectar degradación de métricas críticas.

El proceso de migración se ejecutó exitosamente, transfiriendo inmediatamente todo el tráfico de producción hacia el nuevo sistema. Las métricas post-migración demostraron una mejora sustancial en performance y observabilidad comparado con el sistema legacy.

Incidentes menores y resolución rápida: Durante las primeras 48 horas posteriores a la migración, se reportaron validaciones incorrectas por parte de algunos clientes específicos. El análisis mediante las trazas distribuidas de APM permitió identificar rápidamente la causa raíz: diferencias en la lógica de validación de códigos postales para localidades específicas. La resolución se implementó y desplegó en menos de 15 minutos, demostrando la efectividad de la arquitectura de microservicios para implementar correcciones focalizadas sin impactar otros componentes del sistema.

3.3.4 Estrategia de Compatibilidad Dual: Kafka + IBM MQ

Para asegurar una transición sin interrupciones hacia sistemas que aún dependían de IBM MQ, se implementó una estrategia de compatibilidad dual mediante el componente Sync-Kafka-MQ. Esta decisión arquitectónica permitió que APIODE opere nativamente con Kafka mientras mantenía la compatibilidad con sistemas legacy.

El componente Sync-Kafka-MQ replica cada evento publicado en el tópico **ApiAltaOrdenes-OrdenEnvioSolicitada** hacia las colas correspondientes de IBM MQ, realizando las transformaciones de formato necesarias (JSON a XML) y manteniendo la semántica de mensajería original.

Esta estrategia permitió que los sistemas continuarán operando sin modificaciones mientras se planifica su eventual migración hacia consumo directo de Kafka, eliminando la dependencia de IBM MQ de manera gradual y controlada.

4. Resultados Operacionales y Evidencia de Éxito

La implementación de APIODE ha demostrado mejoras sustanciales en todos los indicadores operacionales críticos, validando las decisiones arquitectónicas tomadas durante el diseño del sistema, tal como se evidenció en los resultados de la prueba piloto en la sección 3.3.2.

4.1 Mejoras en Observabilidad y Monitoreo

El contraste entre los dashboards del sistema legacy **(Figuras 1.1 a 1.4)** y los del nuevo sistema APIODE evidencia una transformación radical en capacidades de

observabilidad. Mientras el sistema anterior presentaba métricas fragmentadas sin correlación temporal clara, APIODE proporciona visibilidad granular de cada componente del flujo de procesamiento.

Dashboard - Capacidades operacionales avanzadas

La nueva arquitectura ha establecido capacidades de monitoreo que trascienden la simple recolección de métricas, proporcionando **inteligencia operacional** para la gestión proactiva del sistema crítico de altas de envíos.

Gestión proactiva de errores y resolución acelerada

Las Figuras 4.1 y 4.2 muestran la capacidad de APIODE para proporcionar contexto detallado ante situaciones de error, contrastando con la opacidad operacional del sistema anterior.

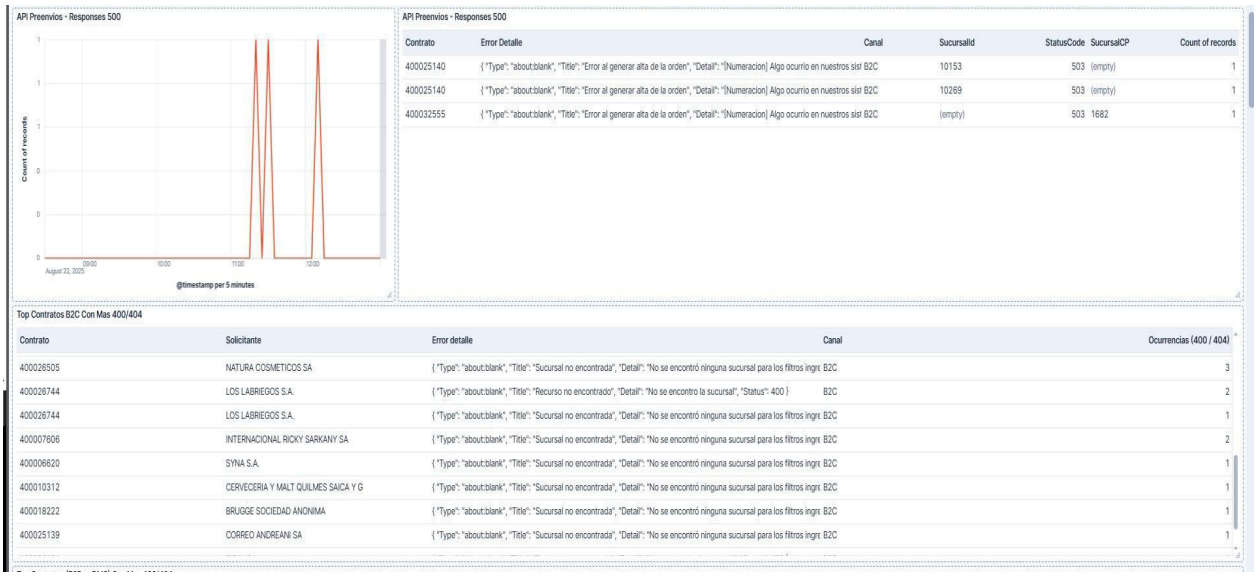
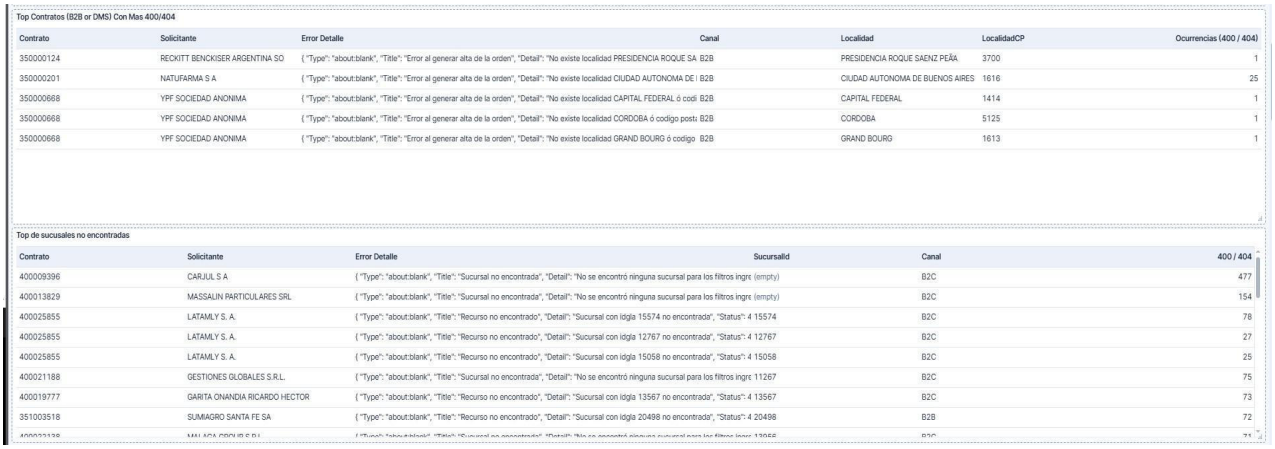


Figura 4.1: *Dashboard APIODE - Gestión temporal de errores: Timeline de respuestas 500 con correlación temporal precisa, permitiendo identificación inmediata de ventanas de degradación y análisis de patrones de fallas.*



Top Contratos (B2B or DMS) Con Mas 400/404

Contrato	Solicitante	Error Detalle	Canal	Localidad	LocalidadCP	Ocurrencias (400 / 404)
350000124	RECKITT BENCKISER ARGENTINA SO	{ "Type": "about:blank", "Title": "Error al generar alta de la orden", "Detail": "No existe localidad PRESIDENCIA ROQUE SA	B2B	PRESIDENCIA ROQUE SAENZ PEÃA	3700	1
350000201	NATUFARMA S A	{ "Type": "about:blank", "Title": "Error al generar alta de la orden", "Detail": "No existe localidad CIUDAD AUTONOMA DE I	B2B	CIUDAD AUTONOMA DE BUENOS AIRES	1616	25
350000668	YPF SOCIEDAD ANONIMA	{ "Type": "about:blank", "Title": "Error al generar alta de la orden", "Detail": "No existe localidad CAPITAL FEDERAL ó codi	B2B	CAPITAL FEDERAL	1414	1
350000668	YPF SOCIEDAD ANONIMA	{ "Type": "about:blank", "Title": "Error al generar alta de la orden", "Detail": "No existe localidad CORDOBA ó código posta	B2B	CORDOBA	5125	1
350000668	YPF SOCIEDAD ANONIMA	{ "Type": "about:blank", "Title": "Error al generar alta de la orden", "Detail": "No existe localidad GRAND BOURG ó codigo	B2B	GRAND BOURG	1613	1

Top de sucusales no encontradas

Contrato	Solicitante	Error Detalle	SucursalId	Canal	400 / 404
400009396	CARJUL S A	{ "Type": "about:blank", "Title": "Sucursal no encontrada", "Detail": "No se encontró ninguna sucursal para los filtros ingre	(empty)	B2C	477
400013829	MASSALIN PARTICULARES SRL	{ "Type": "about:blank", "Title": "Sucursal no encontrada", "Detail": "No se encontró ninguna sucursal para los filtros ingre	(empty)	B2C	154
400025855	LATAMLY S. A.	{ "Type": "about:blank", "Title": "Recurso no encontrado", "Detail": "Sucursal con idgla 15574 no encontrada", "Status": 4 15574		B2C	78
400025855	LATAMLY S. A.	{ "Type": "about:blank", "Title": "Recurso no encontrado", "Detail": "Sucursal con idgla 12767 no encontrada", "Status": 4 12767		B2C	27
400025855	LATAMLY S. A.	{ "Type": "about:blank", "Title": "Recurso no encontrado", "Detail": "Sucursal con idgla 15058 no encontrada", "Status": 4 15058		B2C	25
400021188	GESTIONES GLOBALES S.R.L.	{ "Type": "about:blank", "Title": "Sucursal no encontrada", "Detail": "No se encontró ninguna sucursal para los filtros ingre 11267		B2C	75
400019777	GARITA ONANDIA RICARDO HECTOR	{ "Type": "about:blank", "Title": "Recurso no encontrado", "Detail": "Sucursal con idgla 13567 no encontrada", "Status": 4 13567		B2C	73
351003518	SUMIAGRO SANTA FE SA	{ "Type": "about:blank", "Title": "Recurso no encontrado", "Detail": "Sucursal con idgla 20498 no encontrada", "Status": 4 20498		B2B	72

Figura 4.2: Dashboard APIODE - Trazabilidad completa de errores: Detalle granular por contrato, cliente, sucursal y código postal, con mensajes específicos de error ("Sucursal no encontrada", "Error al generar alta de la orden") que permiten resolución dirigida en minutos en lugar de horas.

Ventajas operacionales evidenciadas

Resolución dirigida: Los errores incluyen contexto específico (contrato, cliente, localidad) eliminando el tiempo de investigación manual

Identificación de patrones: Visualización de errores por cliente permite detectar problemas sistemáticos vs. incidentes aislados

Gestión predictiva: Los timelines de errores facilitan correlación con eventos de infraestructura o cambios de sistema

Aspecto	Sistema Legacy	Sistema APIODE
Tiempo promedio de identificación de causa raíz	mayor a 60 minutos	2-5 minutos
Contexto de errores	Código genérico sin detalle	Contrato, cliente, localidad específica
Visibilidad por cliente	No disponible	Inmediata con identificación comercial
Correlación temporal	Manual, fragmentada	Automática con timeline preciso

Esta evolución en observabilidad no solo reduce el tiempo medio de resolución de incidentes, sino que establece las bases para mantenimiento predictivo y optimización continua del sistema de mayor criticidad comercial de Andreani

4.2 Resiliencia y Confiabilidad

La arquitectura de microservicios ha demostrado resiliencia superior al sistema anterior:

Aislamiento de fallas: Los incidentes menores reportados durante la migración afectaron únicamente las validaciones específicas sin impactar la persistencia de órdenes o la publicación de eventos. Esta capacidad de degradación gradual representa una mejora significativa sobre el comportamiento del sistema legacy.

Recuperación automática: Los mecanismos de retry configurables implementados han manejado exitosamente fallas temporales de conectividad, manteniendo la disponibilidad general del sistema por encima del 99%.

Escalabilidad comprobada: La utilización de Consumer Groups de Kafka con 10 particiones ha permitido escalar horizontalmente el procesamiento de eventos según la demanda, sin modificaciones de código ni interrupciones de servicio.

4.3 Impacto en la operación y el negocio

La implementación exitosa de APIODE ha generado beneficios tangibles que trascienden las mejoras técnicas, impactando directamente en la capacidad operativa de la empresa y la calidad de servicio brindada a los clientes.

Mejora en la atención y experiencia al cliente

La nueva arquitectura facilitó una respuesta más ágil ante requerimientos y consultas de clientes. La visibilidad compleja del flujo de órdenes permite al equipo de soporte identificar rápidamente el estado y ubicación de cualquier envío, reduciendo significativamente los tiempos de resolución de consultas. Los clientes experimentan una mayor confiabilidad en el servicio, evidenciada por la reducción de incidentes y la capacidad de brindar información precisa en tiempo real.

La observabilidad granular del sistema permite anticipar y prevenir degradaciones de servicio antes de que impacten a los usuarios finales, mejorando la percepción de calidad y profesionalismo por parte de los clientes corporativos.

Facilitación para incorporación de nuevos clientes

La arquitectura modular y la documentación automatizada de endpoints facilitan significativamente el proceso de onboarding de nuevos clientes. Los integradores pueden comprender rápidamente la estructura de las APIs e implementar sus integraciones de manera más eficiente, reduciendo el tiempo para nuevas incorporaciones.

Reducción de esfuerzo en desarrollos y modificaciones

La separación en microservicios especializados permite implementar cambios y nuevas funcionalidades sin afectar componentes no relacionados. Las modificaciones que anteriormente requerían despliegues completos del sistema ahora se limitan a servicios específicos, reduciendo el riesgo y el tiempo de implementación.

Los desarrolladores pueden trabajar de manera paralela en diferentes componentes sin generar conflictos, acelerando significativamente la velocidad de desarrollo. La implementación de nuevas validaciones de negocio o integraciones con sistemas

externos se realiza de manera focalizada, requiriendo menor esfuerzo de testing y coordinación.

Transferencia de conocimiento y autonomía técnica

A diferencia del sistema anterior, donde el conocimiento se concentraba en unos pocos especialistas, la nueva implementación está completamente documentada y utiliza tecnologías estándar de la empresa. Esto ha permitido que el equipo interno desarrolle expertise completa sobre todos los componentes del sistema.

La capacitación del personal se ve facilitada por la utilización del template corporativo y patrones arquitectónicos consistentes, permitiendo que nuevos integrantes del equipo se incorporen productivamente en menor tiempo. La documentación técnica generada automáticamente y los dashboards operacionales proporcionan una base sólida para la transferencia de conocimiento y el mantenimiento autónomo del sistema.

Capacidad de evolución y adaptabilidad

La nueva arquitectura proporciona una base sólida para futuras evoluciones del negocio. La incorporación de nuevos tipos de envío, modificaciones en procesos de validación, o integraciones con sistemas adicionales puede realizarse de manera incremental sin comprometer la estabilidad del ecosistema existente.

Esta flexibilidad arquitectónica posiciona a la empresa para responder ágilmente a cambios regulatorios, oportunidades de mercado, o requerimientos operacionales emergentes, manteniendo la competitividad y capacidad de innovación en el sector logístico.

Conclusiones

La implementación exitosa de APIODE demuestra la efectividad de una estrategia de migración gradual y planificada. Los resultados operacionales evidencian mejoras sustanciales en performance, observabilidad y resiliencia.

Las métricas presentadas validan que la modernización no solo resolvió las limitaciones técnicas del sistema anterior, sino que estableció una base sólida para futuras evoluciones e integraciones en Andreani. La estrategia de compatibilidad dual ha permitido una transición sin interrupciones, minimizando riesgos operacionales mientras maximiza los beneficios de las nuevas capacidades tecnológicas.

Más allá del éxito técnico del proyecto, el desarrollo de APIODE representó un desafío personal de crecimiento profesional que transformó mi perspectiva como ingeniero de software. Inicialmente, mi enfoque desde el Área de Arquitectura era predominantemente técnico, pero terminé descubriendo que el verdadero desafío era el de comprender las necesidades funcionales del negocio.

El proceso de construcción de dashboards operacionales ejemplifica esta evolución: no bastaba con mostrar métricas técnicas como latencia o throughput. Era fundamental entender qué información del envío resultaba crítica para el área comercial, cómo correlacionar datos técnicos con impacto de negocio, y presentar la información de manera que fuera útil tanto para equipos técnicos como comerciales. Esto requirió desarrollar conocimientos específicos del dominio logístico que no había adquirido durante la formación académica.

Un ejemplo concreto fue el trabajo con el sistema core que consumía eventos desde IBM MQ. Desde lo técnico, la integración era directa, pero funcionalmente debía comprender exactamente qué segmentos del mensaje eran procesables por el sistema legacy, qué transformaciones de formato eran necesarias, y cómo mantener la semántica del negocio durante la migración. Este conocimiento funcional resultó tan crítico como la implementación técnica para entregar una solución exitosa.

Esta experiencia me proporcionó una **visión holística** que integra perspectivas técnicas, funcionales y de impacto en el cliente final. Aprender a prestar atención a detalles que impactan la percepción del cliente sobre el sistema, comprender cómo las decisiones técnicas afectan procesos comerciales, y desarrollar la capacidad de comunicar conceptos técnicos complejos a audiencias no técnicas, representó un crecimiento profesional significativo.

El reconocimiento de este desarrollo se reflejó en mi evolución profesional: de Desarrollador Senior a Especialista, convirtiéndome en referente para otros sectores cuando necesitan comprender aspectos específicos de APIODE para sus propios diseños. Jefes y líderes de distintas áreas me consultan tanto sobre aspectos técnicos como funcionales del sistema, validando que la combinación de conocimientos técnicos profundos con comprensión del negocio genera un perfil profesional más completo y valioso.

Para concluir la implementación de APIODE fue exitosa no solo en términos de objetivos técnicos cumplidos, sino como experiencia de crecimiento profesional integral. El proyecto demostró que la modernización arquitectónica efectiva requiere tanto excelencia técnica como comprensión profunda del dominio de negocio. Esta combinación de saberes técnicos y funcionales constituye una base sólida para enfrentar futuros desafíos de modernización en entornos empresariales complejos.

Referencias Bibliográficas

- Apache Kafka
  - https://kafka.apache.org/documentation/
- Net Core 8
  - https://learn.microsoft.com/es-es/dotnet/core/compatibility/8.0
  - https://learn.microsoft.com/es-es/dotnet/core/whats-new/dotnet-8
- Clean Architecture
  - https://www.netmentor.es/entrada/clean-architecture
  - https://juanjoblog.com/que-es-clean-architecture-y-cuales-son-sus-beneficios-y-desventajas/
- SOLID
  - https://www.codigo6.com/principios-solid-en-programacion-explicacion-clara-y-completa/
- SonarQube
  - https://imaginaformacion.com/tutoriales/que-es-sonarqube
- Arquitectura basada en eventos
  - https://medium.com/@diego.coder/introducci%C3%B3n-a-la-arquitectura-orientada-a-eventos-a532c71c9945
- Arquitectura de Microservicios
  - https://www.supermicro.com/es/glossary/microservices-architecture