Documento de Diseño de Solución: Modernización de Plataforma de Integración Bancaria

Fecha: 2026-01-09 **Versión:** 1.0.0 **Estado:** Aprobado para Revisión Arquitectónica **Clasificación:** Confidencial - Uso Interno R&D

Índice de Contenido

1. **Resumen Ejecutivo**
2. **Alcance y Objetivos**
3. **Matriz de Stack Tecnológico**
4. **Estrategia de Arquitectura (Modelo C4)**
5. **Patrones de Integración y Diseño**
6. **Estrategia Multicore (Legacy vs Digital)**
7. **Seguridad y Cumplimiento Normativo (Zero Trust)**
8. **Alta Disponibilidad y Resiliencia**
9. **Gobierno de APIs y Ciclo de Vida**
10. **Análisis de Problemas y Mitigaciones (Deep Dive)**
11. **Casos de Uso Detallados**
12. **Conclusiones y Próximos Pasos**

---

1. Resumen Ejecutivo

Este documento detalla la arquitectura de referencia para la modernización tecnológica de una entidad financiera tradicional hacia un modelo de **Banca Digital Componible**. La solución propone la integración de un nuevo **Core Digital Cloud-Native** con los sistemas heredados (Legacy Mainframe/AS400), habilitando

simultáneamente capacidades de **Open Finance**, pagos en tiempo real y prevención de fraude avanzada.

La arquitectura se fundamenta en principios de **Domain-Driven Design (DDD)** alineados con los dominios de servicio de **BIAN** (Banking Industry Architecture Network), garantizando interoperabilidad estándar. Se adopta un enfoque **Event-Driven (EDA)** para el desacoplamiento de sistemas y **Zero Trust** para la seguridad transversal.

---

2. Alcance y Objetivos

2.1 Objetivos de Negocio

- **Modernización Digital:** Habilitar nuevos canales (Web, Banca Móvil) con experiencia de usuario fluida y desacoplada del ciclo de vida del Legacy.
- **Open Finance:** Exposición segura de APIs a Terceros (TPPs) cumpliendo normativas tipo PSD2.
- **Time-to-Market:** Reducción de tiempos de despliegue mediante microservicios independientes.

2.2 Alcance Técnico

La solución abarca el diseño e integración de:

1. **Core Bancario Tradicional (Legacy):** Sistema de registro y contabilidad base.
2. **Core Digital:** Microservicios para gestión de clientes, cuentas y saldos modernos.
3. **Plataforma de Pagos:** Hub centralizado para transferencias locales e internacionales (SWIFT, ACH).
4. **Sistema de Prevención de Fraudes:** Motor de decisión en tiempo real (<50ms).
5. **Canales Digitales:** BFFs (Backend for Frontend) para Web y Mobile.
6. **Open Finance Gateway:** Capa de seguridad y control para ecosistemas externos.

---

3. Matriz de Stack Tecnológico

Capa	Tecnología	Justificación Arquitectónica
Canales (Frontend)	React / React Native	Experiencia de usuario rica, componente único reutilizable (Design System).
API Gateway & Ingress	Kong Enterprise	Gestión centralizada de tráfico, Rate Limiting, Plugins de seguridad (OIDC/mTLS).
Identidad (CIAM)	Keycloak	Estándar OAuth2/OIDC, soberanía del dato, gestión de consentimiento FAPI.
Microservicios (Core)	Java 21 (Spring Boot)	Ecosistema robusto, integración nativa con Spring Cloud, madurez empresarial.
Event Bus (EDA)	Apache Kafka	Backbone de eventos para consistencia eventual, alto throughput y desacoplamiento.
Base de Datos (Transaccional)	PostgreSQL	ACID compliance, soporte JSONB para flexibilidad, amplia adopción cloud.
Base de Datos (Lectura/Cache)	Redis / MongoDB	Alta velocidad para lecturas de saldo ("Cache-Aside") y perfiles de cliente (Customer 360).
Prevención Fraude	Python (ML) + Drools	Flexibilidad para modelos de ML y reglas de negocio deterministas complejas.
Infraestructura	Kubernetes (K8s)	Orquestación de contenedores, auto-escalado y gestión declarativa (GitOps).

---

4. Estrategia de Arquitectura (Modelo C4)

La solución se modela utilizando el estándar C4 para gestionar la abstracción y el detalle.

4.1 Nivel 1: Contexto del Sistema (System Context)

Referencia: Diagrama `01. Contexto - Ecosistema Bancario` en LikeC4.

El **Core Bancario Digital** es el sistema central que orquesta las interacciones.

- **Actores:** Clientes Retail/Corporativos (inician operaciones), Analistas de Fraude, Reguladores.
- **Sistemas Externos:** Redes de Pagos (Visa, Mastercard), Core Legacy (Sistema de Registro), Red SWIFT, Burós de Crédito.
- **Integración:** El sistema actúa como un agregador inteligente, ocultando la complejidad de los múltiples backends a los canales digitales.

4.2 Nivel 2: Contenedores (Containers)

Referencia: Diagramas `02`, `03` y `04` en LikeC4.

Descomposición en unidades desplegables independientes:

1. **Core Digital Ecosystem:**
   - `API Gateway`: Punto de entrada único.
   - `Auth Service`: Identity Provider (IdP).
   - `Transaction Service`: Orquestador de movimientos monetarios.
   - `Account Service`: Gestión del ciclo de vida de productos.
2. **Payments Platform (Hub de Pagos):**
   - `Processing Engine`: Motor de estados de pagos.
   - `Connectors`: Adaptadores específicos (ISO 8583 para tarjetas, MT103 para SWIFT).
3. **Open Finance Gateway:**
   - `Public Gateway`: Expone APIs estandarizadas con mTLS.
   - `Consent Service`: Gestiona permisos de acceso a datos (Alcance/Tiempo).

4.3 Nivel 3: Componentes Críticos

Referencia: Diagrama `05. Componentes - Servicio de Transacciones`.

Detalle interno del microservicio más crítico (`Transaction Service`):

- **`Transaction Orchestrator`**: Gestiona el flujo de la operación.
- **`Saga Coordinator`**: Mantiene la consistencia distribuida (ver patrones).

- `Idempotency Guard`: Evita duplicidad de transacciones ante reintentos de red.
- `Ledger Writer`: Escritura inmutable de movimientos.

---

5. Patrones de Integración y Diseño

5.1 Arquitectura Orientada a Eventos (EDA)

Se utiliza **Apache Kafka** como columna vertebral para la comunicación asíncrona.

- **Uso:** Notificaciones, auditoría, sincronización de datos a sistemas de analítica y fraude.
- **Beneficio:** Desacopla el productor (Core Digital) del consumidor (Fraude), permitiendo análisis "Fire-and-Forget" sin impactar la latencia de la transacción del usuario.

5.2 Patrón Saga (Transacciones Distribuidas)

Dado que la transacción abarca múltiples microservicios y sistemas externos (Legacy, Redes de Pago), no existe una transacción ACID global.

- **Implementación:** Orquestación centralizada.
- **Flujo:**
    1. Reserva de fondos (Core Digital) -> Éxito.
    2. Ejecución de pago (Gateway Externo) -> Fallo.
    3. **Compensación:** El orquestador emite comando para liberar fondos en Core Digital (Rollback lógico).

5.3 BFF (Backend for Frontend)

Patrón `bffWeb` y `bffMobile` para optimizar las respuestas específicas por canal, reduciendo el "Over-fetching" de datos y simplificando la lógica en los clientes ligeros.

---

6. Estrategia Multicore (Legacy vs Digital)

Para integrar el **Core Tradicional** sin detener la operación, se aplica el patrón **Strangler Fig** combinado con **Change Data Capture (CDC)**.

6.1 Sincronización de Datos (CDC)

- **Lectura (Mainframe Offloading):** Los saldos y movimientos del Legacy se replican en tiempo real hacia una base de datos "Read-Model" en el Core Digital mediante conectores CDC (ej. Debezium/Kafka Connect).
- **Resultado:** Los canales digitales consultan esta réplica de alta velocidad (Redis/Postgres) sin saturar al Legacy (MIPS).

6.2 Escritura (Transactional)

- Las transacciones nuevas nacen en el **Core Digital**.
- Se utiliza un `Legacy Adapter` asíncrono que escribe en el Mainframe (vía colas MQ) para mantener la contabilidad oficial sincronizada a final del día (Consistencia Eventual).

---

7. Seguridad y Cumplimiento Normativo (Zero Trust)

La arquitectura sigue el enfoque "Zero Trust": *Nunca confiar, siempre verificar*.

7.1 Gestión de Identidad y Acceso (IAM)

- **Protocolos:** OAuth 2.1 y OpenID Connect (OIDC).
- **Open Finance:** Implementación del perfil **FAPI (Financial-grade API)**, requiriendo mTLS para clientes confidenciales (TPPs) y tokens de acceso vinculados al certificado (Sender Constrained Access Tokens).

7.2 Protección de Datos (LOPD / GDPR)

- **Cifrado en Reposo:** Bases de datos cifradas con claves gestionadas (KMS/Vault).

- **Cifrado en Tránsito:** TLS 1.3 mandatorio para toda comunicación externa e interna (Service Mesh).
- **Tokenización:** Los datos sensibles (PAN de tarjetas, PII de clientes) se tokenizan antes de ser almacenados o enviados a sistemas de analítica.

---

8. Alta Disponibilidad y Resiliencia

8.1 Estrategia HA/DR

- **Despliegue Multi-Zona (AZ):** El cluster de Kubernetes se extiende a través de 3 zonas de disponibilidad para tolerar la falla de un centro de datos completo.
- **Base de Datos:** PostgreSQL en configuración Cluster con Patroni para failover automático (<30s).
- **Circuit Breakers:** Implementados en todos los clientes HTTP (Feign/Retrofit) para evitar fallas en cascada cuando un sistema dependiente (ej. Legacy) responde lento.

---

9. Gobierno de APIs y Ciclo de Vida

Se establece una estrategia de **API-First Design**:

1. **Definición:** Contratos OpenAPI 3.1 (Swagger) definidos antes de codificar.
2. **Gobierno:**
    - **System APIs:** Exponen datos crudos de backends (Internas).
    - **Process APIs:** Orquestan lógica de negocio (Internas).
    - **Experience APIs:** Exponen capacidades a canales (Públicas/Partner).
3. **Versionado:** Estrategia semántica en URL (`/v1/payments`) manteniendo compatibilidad hacia atrás de al menos 2 versiones (N-2).

---

10. Análisis de Problemas y Mitigaciones (Deep Dive)

Esta sección detalla cómo la arquitectura propuesta aborda los desafíos endémicos de la integración bancaria moderna.

10.1 Problema: La "Espaguetización" de la Integración (Point-to-Point)

- **Desafío:** En sistemas tradicionales, cada nuevo canal (Móvil, Web, ATM) se conecta directamente al Core, creando una malla inmanejable de dependencias (`N x M` conexiones).
- **Solución Arquitectónica: API Gateway & Event Bus**.
- **Mitigación:** Se introduce un bus de integración agnóstico. El Core Legacy solo "habla" una vez con el Bus (vía adaptadores). Los canales consumen APIs estandarizadas del Bus.
- **Impacto:** Reducción del acoplamiento en un 80%. Añadir un nuevo canal (ej. Chatbot) ya no requiere cambios en el Core.

10.2 Problema: Latencia en Sistemas Legacy (Cuello de Botella Mainframe)

- **Desafío:** Los Mainframes (AS400/zOS) no escalan horizontalmente para soportar la carga masiva de consultas de usuarios móviles (Checking compulsivo de saldo).
- **Solución Arquitectónica: Mainframe Offloading (CQRS)**.
- **Mitigación:** (Ver Sección 6.1) Las consultas de lectura se desvían a una base de datos Réplica de alta velocidad (Redis/Mongo) que se mantiene sincronizada con el Legacy en tiempo real.
- **Impacto:** El Legacy se libera del 90% del tráfico de lectura (`SELECTs`), dedicándose solo a procesar transacciones críticas de escritura (`UPDATEs`).

10.3 Problema: Inconsistencia de Datos en Transacciones Distribuidas

- **Desafío:** Una transferencia involucra: Debitar cuenta origen (Core), Acreditar destino (Banco Externo), Notificar Usuario (Email). Si el paso 2 falla, el dinero

"desaparece" si no se revierte el paso 1.

- **Solución Arquitectónica: Saga Pattern (Orquestación)**.
- **Mitigación:** El `Transaction Orchestrator` mantiene el estado. Si un paso falla, ejecuta automáticamente transacciones de reversión (Compensating Transactions) en orden inverso.
- **Impacto:** Garantía de "Eventual Consistency". El dinero nunca se pierde, siempre retorna al origen o llega al destino.

---

11. Casos de Uso Detallados

Ejemplos prácticos de cómo la arquitectura resuelve flujos de negocio complejos.

11.1 Caso de Uso: Onboarding Digital 100% (KYC)

- **Escenario:** Un usuario nuevo descarga la App y abre una cuenta en 3 minutos.
- **Flujo Arquitectónico:**
  1. **Mobile:** Sube foto de ID y Selfie -> `API Gateway`.
  2. **Auth Service:** Valida identidad con biométricos (Servicio Cognitivo).
  3. **Account Service:** Crea el registro de cliente en **Core Digital** (Postgres).
  4. **Legacy Adapter (Async):** Un proceso en segundo plano "cola" la creación del cliente en el **Mainframe** para cumplimiento regulatorio, sin bloquear al usuario.
  5. **Beneficio:** Experiencia de usuario instantánea ("Time-to-Glass" < 2s) mientras el proceso pesado ocurre offline.

11.2 Caso de Uso: Pago QR en Comercio (Alta Concurrencia)

- **Escenario:** "Black Friday", millones de usuarios pagando simultáneamente.
- **Flujo Arquitectónico:**
  1. **Transaction Service:** Recibe la intención de pago.
  2. **Fraud Engine:** Evalúa riesgo en <50ms (Cache de Redis rules). Si `Low Risk` -> Aprueba preliminarmente.

3. **Event Bus (Kafka):** Publica evento `Payment.Created`.
4. **Ledger:** Consume el evento y actualiza saldos.
5. **Push Notification:** El usuario recibe confirmación.

- **Beneficio:** El sistema de fraude no bloquea el hilo principal de ejecución gracias a la evaluación reactiva y reglas cacheadas, permitiendo miles de TPS (Transacciones por Segundo).

---

12. Conclusiones y Próximos Pasos

La arquitectura propuesta ofrece una ruta evolutiva clara para la modernización bancaria. Al desacoplar los canales digitales de los sistemas de registro heredados mediante una capa de integración basada en eventos y APIs, el banco gana la agilidad necesaria para innovar ("Time-to-Market") manteniendo la robustez y seguridad requerida por el regulador. La adopción de **LikeC4** como herramienta de modelado vivo asegura que la documentación evolucione a la par del código, cerrando la brecha entre arquitectura e implementación.