Diseño de Arquitectura: Sistema de Banca por Internet BP

Fecha: 3 de Octubre, 2025

Preparado para: BP

Preparado por: Sebastián Ruiz

Tabla de Contenidos

1. Resumen Ejecutivo

Este documento detalla el diseño de arquitectura para un nuevo sistema de banca en línea para BP. La solución propuesta se basa en tecnologías modernas nativas de la nube, una arquitectura de microservicios, y prácticas de seguridad estándar de la industria con el fin de entregar una plataforma escalable, segura, y que cumpla con las normativas vigentes.

Decisiones Arquitectónicas Clave:

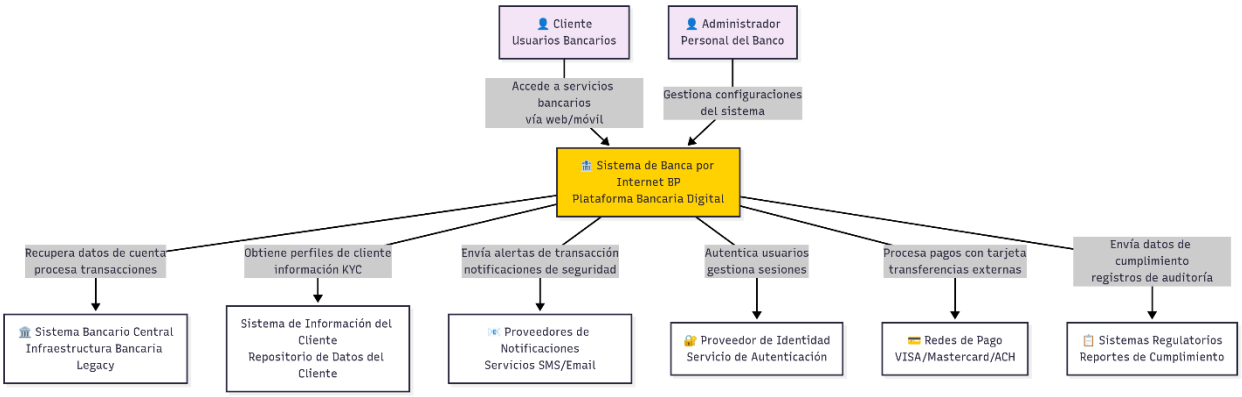
- **Proveedor de la Nube:** AWS
- **Framework Móvil:** Flutter
- **Autenticación:** Auth0 con OAuth 2.0 + PKCE
- **Patrón de Backend:** Microservicios a través de un API Gateway
- **Estrategia de Datos:** Enfoque multi-base de datos con PostgreSQL y Redis
- **Infraestructura:** Despliegue en contenedores sobre AWS ECS/EKS

2. Descripción General y Diagramas C4

Nivel 1: Diagrama de Contexto del Sistema

Audiencia: Usuarios no técnicos.
Propósito: Ofrecer una vista de alto nivel de las interacciones del sistema.

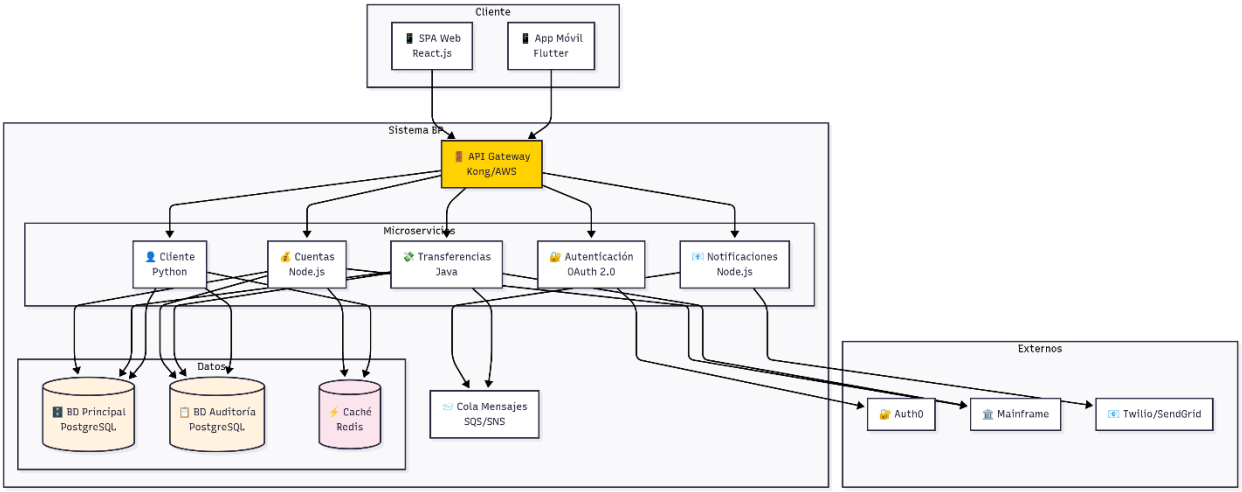
Descripción:
El diagrama de contexto muestra al Sistema de Banca en Línea de BP como un *hub* central, conectando a los clientes (a través de interfaces web y móvil) con los servicios bancarios, a la vez integrándose con sistemas externos esenciales para ejecutar las operaciones de banca necesarias.



Nivel 2: Diagrama de Contenedores

Audiencia: Personal técnico (desarrolladores, operaciones).
Propósito: Descomponer el sistema en sus principales contenedores tecnológicos.

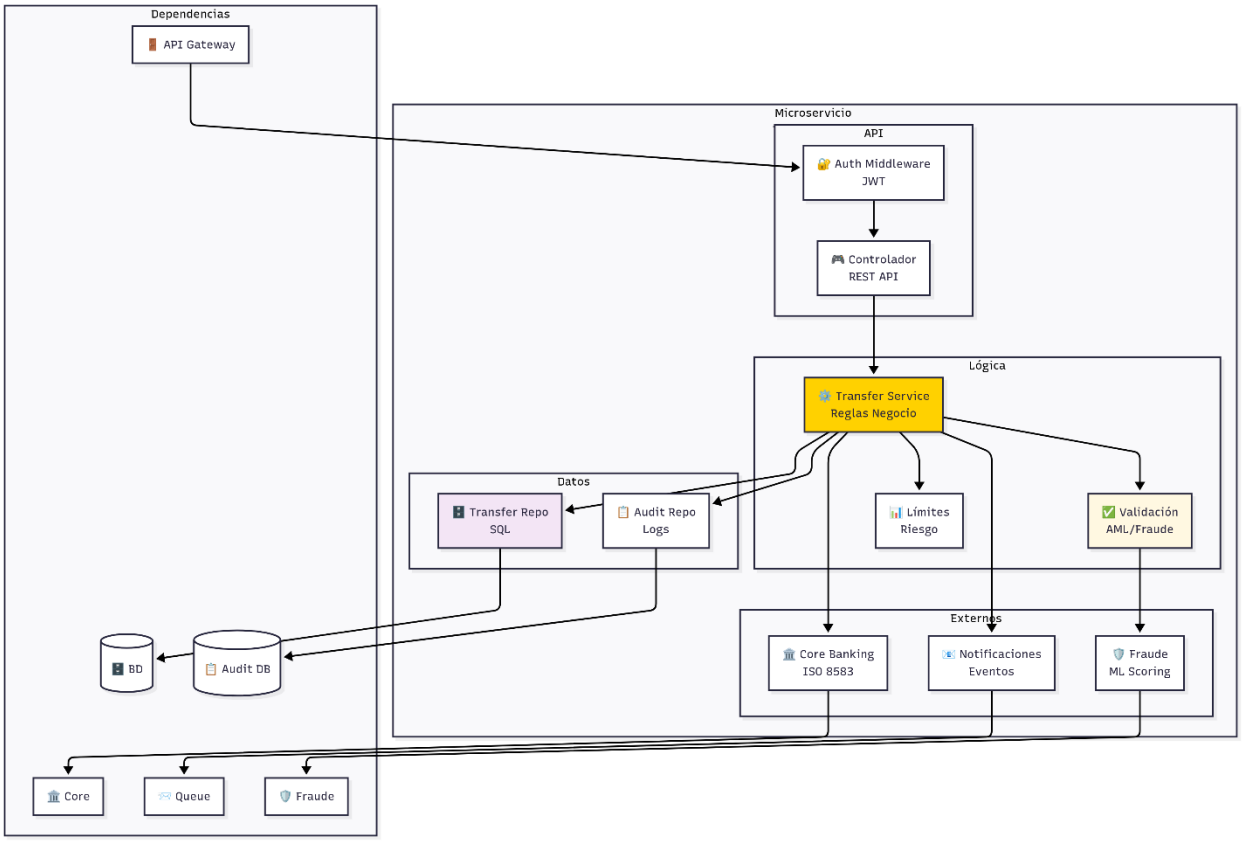
Descripción:
Este diagrama desglosa el sistema en sus contenedores clave, mostrando cómo la Aplicación Web (SPA), la App Móvil, el API Gateway y los mircroservicios colaboran para otorgar las funcionalidades bancarias.



Nivel 3: Diagrama de Componentes – Microservicio de Transferencias

Audiencia: Personal con alto perfil técnico.
Propósito: Detallar la estructura interna de un servicio crítico.

Descripción:
Este diagrama detalla la arquitectura interna del microservicio de transferencias, mostrando el flujo de una solicitud a través de las capas de validación, lógica de negocio e integración con sistemas externos.



---

3. Arquitectura Frontend

Selección de Framework Móvil

Recomendación: Flutter

Justificación:

1. **Eficiencia:** Flutter permite desarrollar para iOS y Android desde una única base de código en Dart, reduciendo el tiempo de desarrollo y garantizando consistencia en la experiencia de usuario.
2. **Rendimiento y Seguridad:** Flutter compila código nativo ARM, ofreciendo un rendimiento cercano al nativo, crucial para una aplicación financiera. Su arquitectura basada en widgets y su sistema de tipado fuerte minimizan errores en tiempo de ejecución.

Alternativas Consideradas:

- **React Native:** Aunque es una tecnología madura con bastante soporte, su arquitectura basada en bridge puede generar cuellos de botella en operaciones intensivas.
- **Desarrollo Nativo (Swift/Kotlin):** Ofrece el máximo rendimiento, pero duplica el esfuerzo de desarrollo e incrementa el riesgo de inconsistencias entre plataformas.

Arquitectura de la Aplicación Web

Recomendación: React.js (Single Page Application)

Justificación:

1. **Ecosistema Empresarial:** El ecosistema maduro de React, con extensas librerías de grado empresarial (como react-spring para animaciones y Formik para el manejo de formularios complejos) y un sólido soporte para TypeScript, garantiza la seguridad de tipos de crítica para los cálculos financieros y el manejo de datos.
2. **Rendimiento y SEO:** El DOM virtual de React y las capacidades de code-splitting permiten tiempo de carga rápidos, esenciales para la retención de clientes.

Alternativas Consideradas:

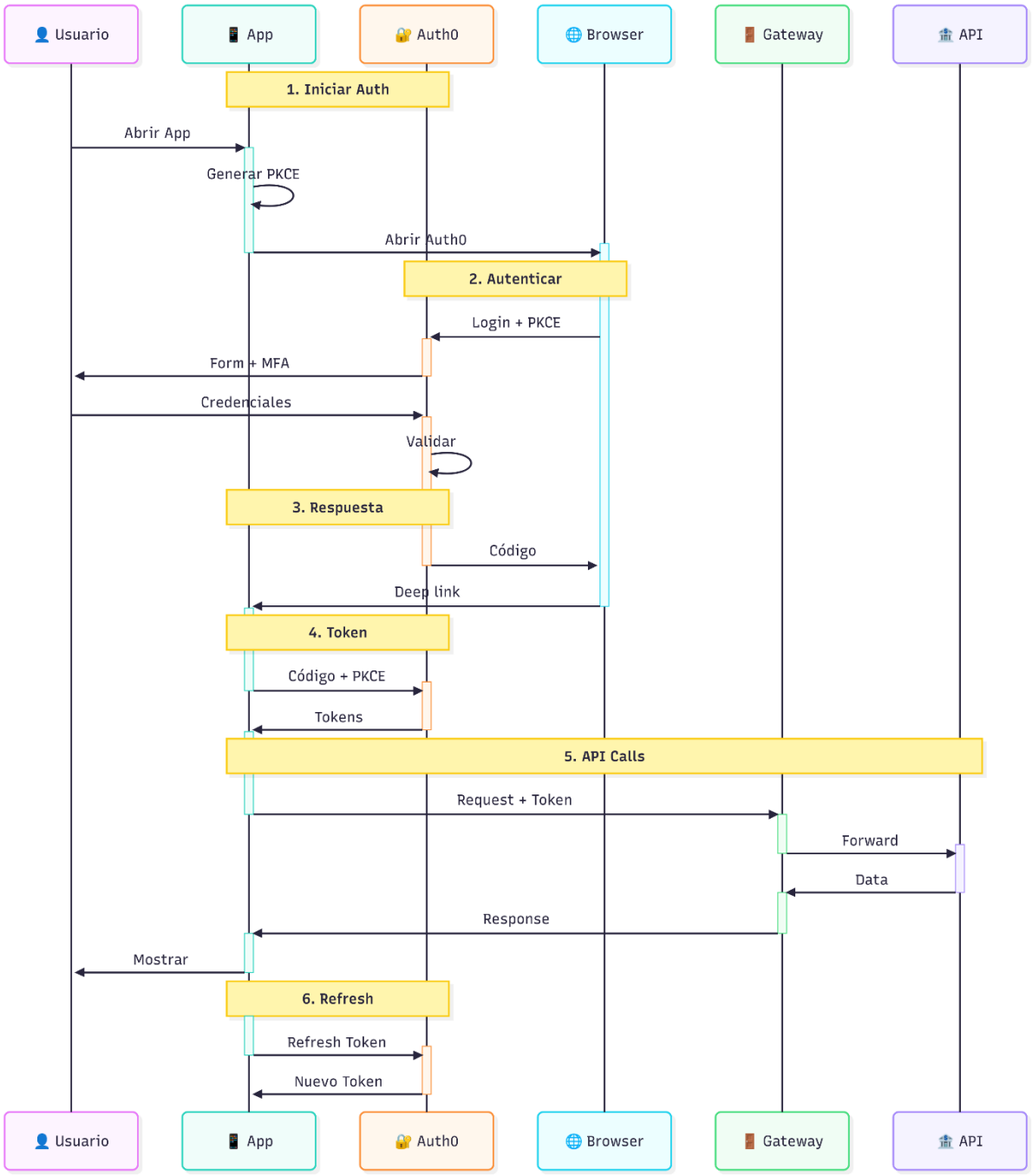
- **Angular:** Excelente para aplicaciones empresariales, pero con una curva de aprendizaje más pronunciada y un "bundle size" inicial más grande.

---

4. Arquitectura de Autenticación y Onboarding

Diagrama de Flujo de Autenticación

Flujo: OAuth 2.0 Authorization Code con PKCE



Justificación de Flujo y Proceso de Onboarding

Por qué OAuth 2.0 + PKCE es el estándar más seguro:

1. **Protección PKCE:** PKCE previene ataques de intercepción de códigos de autorización al requerir que el cliente pruebe la posesión del *code_verifier* que corresponde al *code_challenge* enviado en la solicitud inicial. Esto es crucial para aplicaciones móviles donde los esquemas de URL personalizados pueden ser vulnerables a aplicaciones maliciosas.

2. **No Exposición de Secretos de Cliente:** A diferencia de los flujos OAuth tradicionales que requieren un secreto de cliente, PKCE elimina la necesidad de incrustar secretos en las aplicaciones móviles, eliminando una vulnerabilidad de seguridad importante, ya que las aplicaciones móviles no pueden almacenar secretos de forma segura.
3. **Seguridad del Navegador del Sistema:** Utilizar el navegador del sistema en lugar de vistas web incrustadas proporciona un mejor aislamiento de seguridad, acceso a las credenciales guardadas y la configuración de seguridad del usuario, y protección contra ataques de phishing a través de indicadores de seguridad del navegador.

Proceso de Onboarding con Reconocimiento Facial

Flujo Seguro de Establecimiento de Identidad

1. **Registro Inicial:**

- El cliente visita una sucursal de BP con su cédula de identidad, pasaporte, o alguna otra forma de identificación emitida por el gobierno.
- El personal del banco captura una imagen facial de alta resolución utilizando un scanner biométrico certificado.
- La plantilla facial es cifrada y almacenada en una base de datos biométrica segura.
- La identidad del cliente es verificada contra las bases de datos gubernamentales.
- Se generan credenciales temporales para el primer acceso a la aplicación.

2. **Onboarding en la App:**

- El cliente descarga la aplicación e ingresa las credenciales temporales.
- La aplicación solicita la inscripción biométrica utilizando el enclave seguro del dispositivo.
- Se realiza un escaneo de reconocimiento facial con la cámara del dispositivo, incluyendo detección de vida.
- La plantilla biométrica cifrada es enviada a Auth0 para la verificación de identidad.
- Auth0 compara el escaneo móvil contra la plantilla de referencia del banco utilizando la API de FaceID (o equivalente).
- Tras un match exitoso, se aprovisionan credenciales permanentes en Auth0.
- El cliente configura métodos de autenticación multifactor adicionales (SMS, aplicación de autenticación).

3. **Autenticación Continua:**

- Los inicios de sesión posteriores utilizan el reconocimiento facial como factor principal.
- Los datos biométricos nunca abandonan el enclave seguro del dispositivo.
- Auth0 solo recibe la prueba criptográfica de la coincidencia biométrica.
- La autenticación basada en riesgo activa factores adicionales para transacciones de alto riesgo.

5. Arquitectura Backend y de Datos

Visión General de la Arquitectura Backend

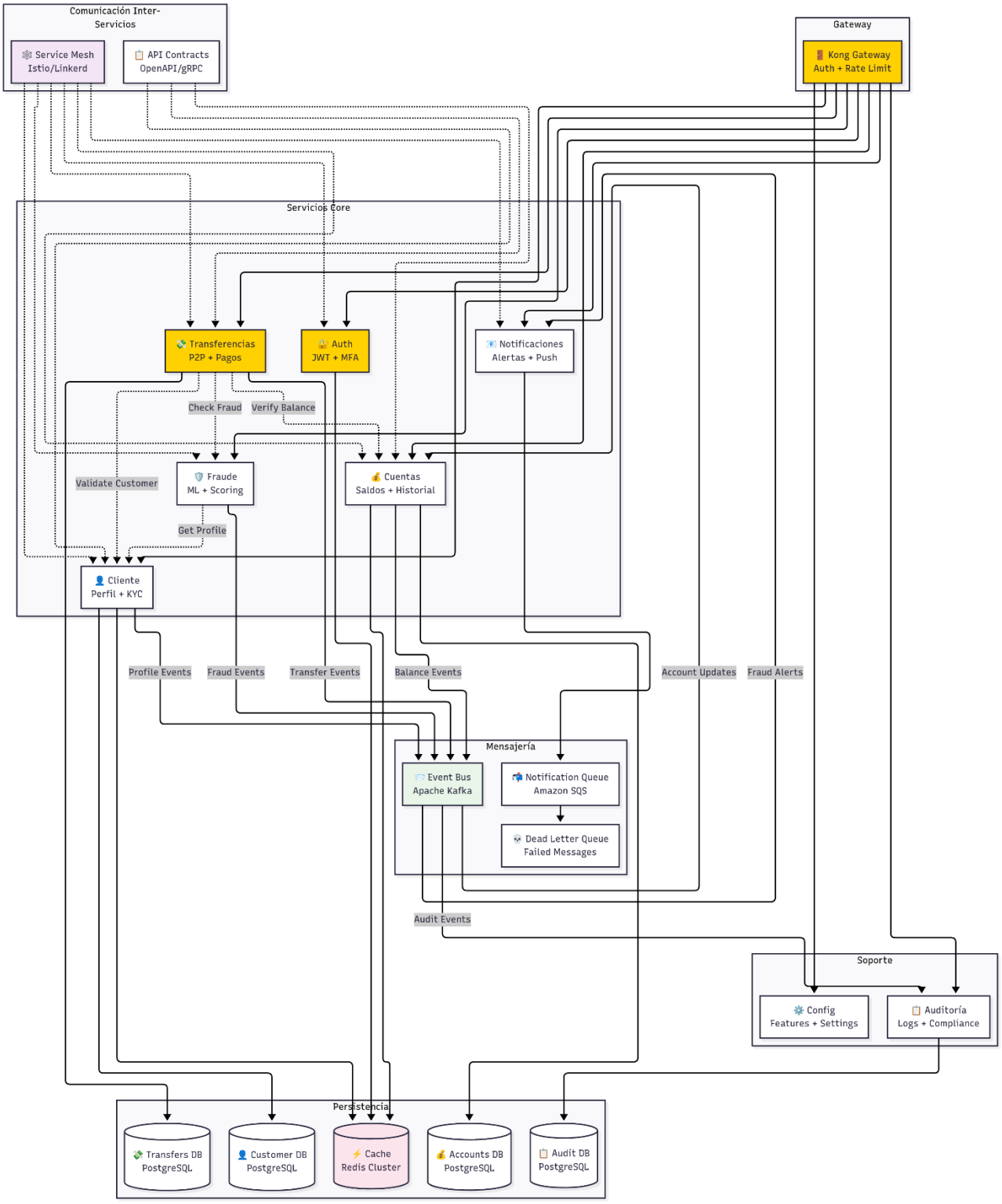
Recomendación: Microservicios Orientados a Eventos con Diseño Dirigido por el Dominio

Justificación:

1. **Escalabilidad y Aislamiento de Fallos:** Cada servicio puede escalarse de forma independiente basado en patrones de demanda (por ejemplo, los picos de transferencias ocurren durante el horario comercial), y los fallos en un servicio no se propagan a otros, lo que garantiza una resiliencia del sistema que es crucial para las operaciones bancarias 24/7.
2. **Arquitectura Orientada a Eventos:** El bus de eventos Apache Kafka facilita el acoplamiento flexible entre los servicios mientras mantiene la consistencia de datos mediante patrones de consistencia eventual. Este enfoque permite que los servicios reaccionen a los eventos de negocio sin dependencias rígidas, lo cual es vital para la resiliencia del sistema.

Alternativas Consideradas:

- **Arquitectura Monolítica:** Ofrece un despliegue más simple y consistencia de datos integrada, pero limita la escalabilidad y dificulta la adopción de nuevas tecnologías o el escalado de funciones específicas de forma independiente.



Patrones Arquitectónicos Clave

Estrategia del Message Broker:

- **Event Bus Apache Kaka:** Gestiona los eventos de dominio (transferencias, actualizaciones de saldo, alertas de fraude) con entrega garantizada y capacidades de reproducción de eventos.
- **Amazon SQS para Notificaciones:** Administra la entrega de notificaciones de alto volumen con mecanismos de reintento incorporados.
- **Dead Letter Queues:** Captura mensajes fallidos para análisis e intervención manual.

Diseño de la Capa de Persistencia:

- **Bases de Datos Primarias PostgreSQL:** Cada microservicio utiliza PostgreSQL para el almacenamiento de datos transaccionales, elegido por su cumplimiento ACID, soporte robusto para JSON que permite esquemas flexibles, y fiabilidad probada en aplicaciones financieras con características como la recuperación a un punto en el tiempo (*point-in-time recovery*) y capacidades avanzadas de indexación.
- **Base de Datos por Servicio:** Cada servicio administra su propio esquema de base de datos y ciclo de vida de los datos, asegurando la autonomía del servicio y previniendo el acoplamiento de datos.
- **Redis Cluster:** Caching distribuido para datos de acceso frecuente (saldos de cuentas, perfiles de clientes) con tiempos de respuesta por debajo del milisegundo.
- **Base de Datos de Auditoría PostgreSQL:** Instancia dedicada de PostgreSQL para registros de auditoría inmutables utilizando el patrón de evento sourcing. Se aprovecha el soporte JSONB para un almacenamiento flexible de eventos y su indexación avanzada para consultas de cumplimiento normativo.

Comunicación Entre Servicios:

- **Llamadas Síncronas:** Llamadas directas REST/gRPC para requisitos de consistencia inmediata (validación de saldo, comprobaciones de fraude).
- **Eventos Asíncronos:** Eventos de Kafka para escenarios de consistencia eventual (notificaciones, registro de auditoría)
- **Service Mesh:** Istio/Linkerd proporciona descubrimiento de servicios, balanceo de carga, y observabilidad a través de todas las comunicaciones de servicio.
- **Contratos de API:** Las especificaciones OpenAPI garantizan la compatibilidad con versiones anteriores y la claridad en las interfaces de servicio.

Selección de Tecnología de Base de Datos

Recomendación: PostgreSQL para el Almacenamiento de Datos Primarios

Justificación

1. **Cumplimiento ACID e Integridad Financiera:** PostgreSQL proporciona garantías transaccionales ACID completas esenciales para las operaciones bancarias, con

modelos de consistencia robustos que previenen la corrupción de datos durante transacciones concurrentes. Su uso en instituciones financieras y mecanismos de bloqueo avanzados aseguran la integridad de los datos para operaciones bancarias críticas.

2. **Características Avanzadas y Rendimiento:** PostgreSQL ofrece indexación sofisticada (B-tree, Hash, GiST, GIN), soporte JSON/JSONB para almacenamiento flexible de documentos dentro de la estructura relacional, y optimización avanzada de consultas. Su soporte para procedimientos almacenados, triggers, y funciones de ventana permite cálculos financieros complejos directamente en la capa de base de datos.

Alternativas Consideradas:

- **MySQL:** Si bien es eficiente y ampliamente adoptado, su manejo de transacciones complejas y topos de datos avanzados es menos robusto que el de PostgreSQL, algo particularmente importante para los cálculos financieros y los requisitos de cumplimiento normativo.
- **Oracle Database:** Ofrece características de nivel empresarial, pero introduce costos de licencia significativos y dependencia del proveedor, mientras que PostgreSQL ofrece funcionalidad comparable aún siendo open source, teniendo gran soporte comunitario.

Estrategia de Caching de Datos

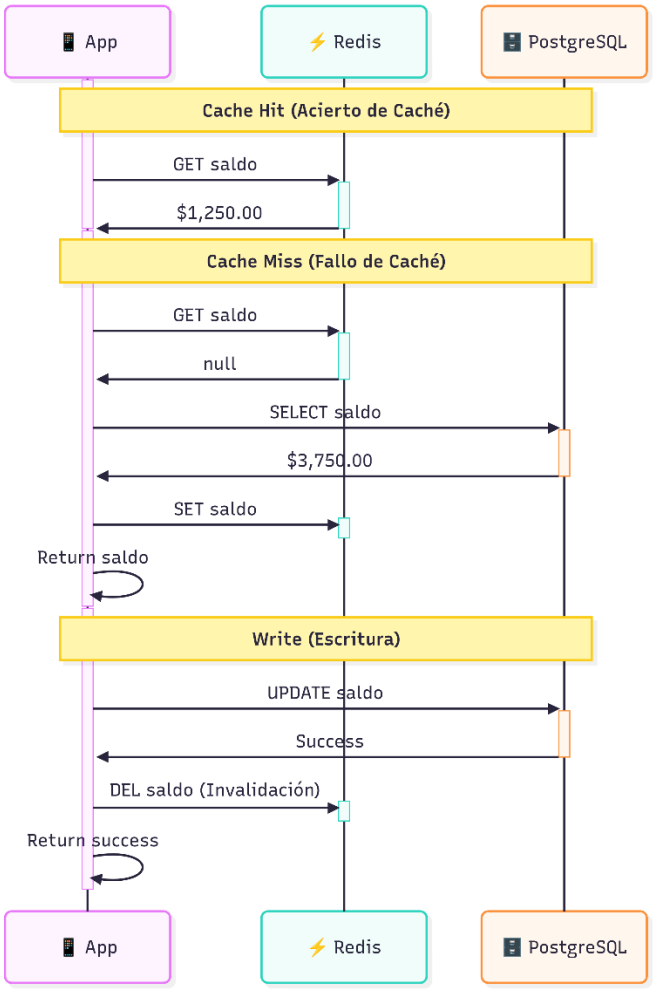
Recomendación: Patrón Cache-Aside con Redis

Justificación:

1. **Optimización de Rendimiento:** El patrón Cache-Aside reduce la carga de la base de datos en un 60-80% para datos de clientes accedidos frecuentemente (saldos de cuentas, historial de transacciones), mejorando los tiempos de respuesta de 200ms a 20ms para los datos en caché, lo cual es crucial para la experiencia del cliente.
2. **Control de Consistencia de Datos:** A diferencia del write-through caching, el Cache-Aside otorga a las aplicaciones un control explícito sobre cuando se almacena e invalida el dato, esencial para los datos financieros donde la información de saldo obsoleta podría conducir a sobregiros o problemas de cumplimiento normativo.

Alternativas Consideradas

- **Write-Through Caching:** Garantiza la consistencia del caché, pero añade latencia a las operaciones de escritura, lo cual es problemático para el procesamiento de transacciones en tiempo real donde se requiere una confirmación inmediata.



Estrategia de las Claves de Caché:

- **Datos del Cliente:** `customer:{id}:{tipo_dato}` (TTL: 5 minutos)
- **Saldos de Cuentas:** `account:{id}:balance` (TTL: 2 minutos)
- **Historial de Transacciones:** `historial_txn:{id_cuenta}:{página}` (TTL: 10 minutos)
- **Datos de Configuración:** `config:{servicio}:{clave}` (TTL: 1 hora)

Implementación de la Base de Datos de Auditoría

Implementación del Event Sourcing:

Instancia dedicada de PostgreSQL para registros de auditoría inmutables utilizando el patrón de evento sourcing. Se aprovecha el soporte JSONB de PostgreSQL para un almacenamiento flexible de eventos y su indexación avanzada para consultas de cumplimiento normativo.

Esquema de Event Store para Auditoría:

Este esquema lógico se basa en la tabla `audit_events` y garantiza la no repudiación y la trazabilidad completa necesarias para el cumplimiento normativo bancario.

Columna Clave	Tipo de Dato Lógico	Propósito y Función Arquitectónica
event_id	**UUID**	Identificador único e inmutable del registro (clave primaria).
event_sequence	**BIGSERIAL**	Secuencia global de eventos. Asegura el orden cronológico estricto de todos los eventos del sistema.
aggregate_id	**VARCHAR**	Identificador de la entidad de negocio afectada (ej. account_id).
event_data	**JSONB**	Contenido completo, inmutable y flexible del evento. Permite consultar contenido estructurado de manera eficiente.
event_hash	**VARCHAR(64)**	Huella criptográfica (hash) del evento. Garantiza la integridad del registro y la no repudiación (prueba de que el dato no fue alterado).
previous_event_hash	**VARCHAR(64)**	Hash del evento inmediatamente anterior. Crea una cadena de integridad para la auditoría.
user_id, session_id	**VARCHAR**	Metadatos de Trazabilidad. Identifica de forma única al usuario y la sesión para el cumplimiento de KYC/AML.
event_timestamp	**TIMESTAMP Z/T**	Marca de tiempo precisa de cuando ocurrió el evento.

Índices Clave para Cumplimiento Normativo y Rendimiento

Para garantizar la velocidad y la eficiencia de las consultas de auditoría y cumplimiento, se han definido los siguientes índices sobre la tabla principal `audit_events`:

Índice Clave	Propósito Estratégico
Idx_audit_events_aggregate (aggregate_id, aggregate_type)	Permite reconstruir rápidamente el estado de una entidad específica (ej. obtener todos los eventos de una cuenta). Crucial para la reconstrucción de estado.
idx_audit_events_timestamp (event_timestamp)	Acelera las consultas basadas en períodos de tiempo (ej. “todos los eventos de la última semana”) Crucial para la generación de informes.
idx_audit_events_user (user_id)	Optimiza las búsquedas basadas en el actor que inició el evento. Esencial para la trazabilidad de usuarios.
idx_audit_events_type (event_type)	Permite filtrar rápidamente por tipo de evento (ej. “sólo transferencias”)

Componente de Optimización: Snapshots

Para mejorar el rendimiento en la reconstrucción del estado de las entidades, se utilizará una tabla secundaria (audit_snapshots). Esta tabla almacenará estados consolidados del aggregate períodicamente. Esto es una técnica de optimización de lectura que evita tener que procesar miles de eventos históricos cada vez que se necesita el estado actual de una entidad de negocio.

Puntos de Datos Clave de Auditoría

- **Acciones de Usuario:** Inicio/cierre de sesión, cambios de perfil, inicio de transacciones.
- **Ciclo de Vida de Transacciones:** Autorización, validación, ejecución y liquidación.
- **Eventos de Seguridad:** Intentos de inicio de sesión fallidos, desafíos de MFS, actividad sospechosa.
- **Eventos del Sistema:** Despliegues de servicios, cambios de configuración, errores del sistema.
- **Acceso a Datos:** Qué datos fueron accedidos, cuándo, por quién, y desde dónde.

---

6. Infraestructura en la Nube y Requisitos No Funcionales

Selección de Proveedor Cloud

Recomendación: Amazon Web Services (AWS)

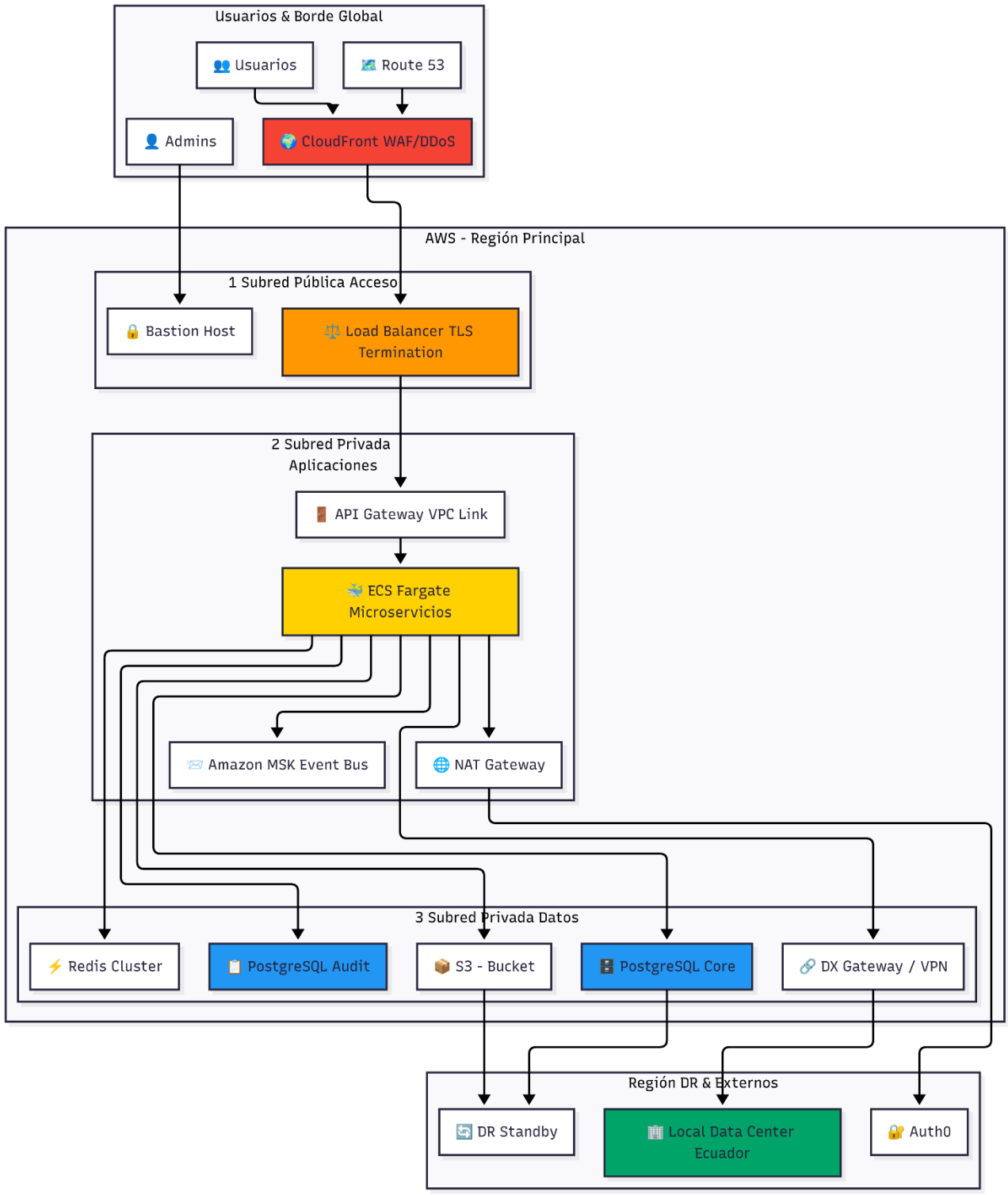
Justificación:

1. **Experiencia en Servicios Financieros:** AWS posee el portafolio más profundo y las certificaciones de cumplimiento más robustas validadas para cargas de trabajo bancarias.
2. **Infraestructura Global y Capacidades de DR:** La extensa red de regiones y zonas de disponibilidad de AWS permite implementar estrategias de Disaster Recovery con RTO y RPO que exceden los SLAs bancarios

Alternativas Consideradas:

- **Microsoft Azure:** Ofrece una sólida integración empresarial y excelentes capacidades de nube híbrida. Sin embargo, cuenta con menos servicios especializados para el sector financiero y su infraestructura global es menos madura en comparación.

Arquitectura AWS de Alto Nivel



Implentación de Requerimientos No Funcionales (NFRs)

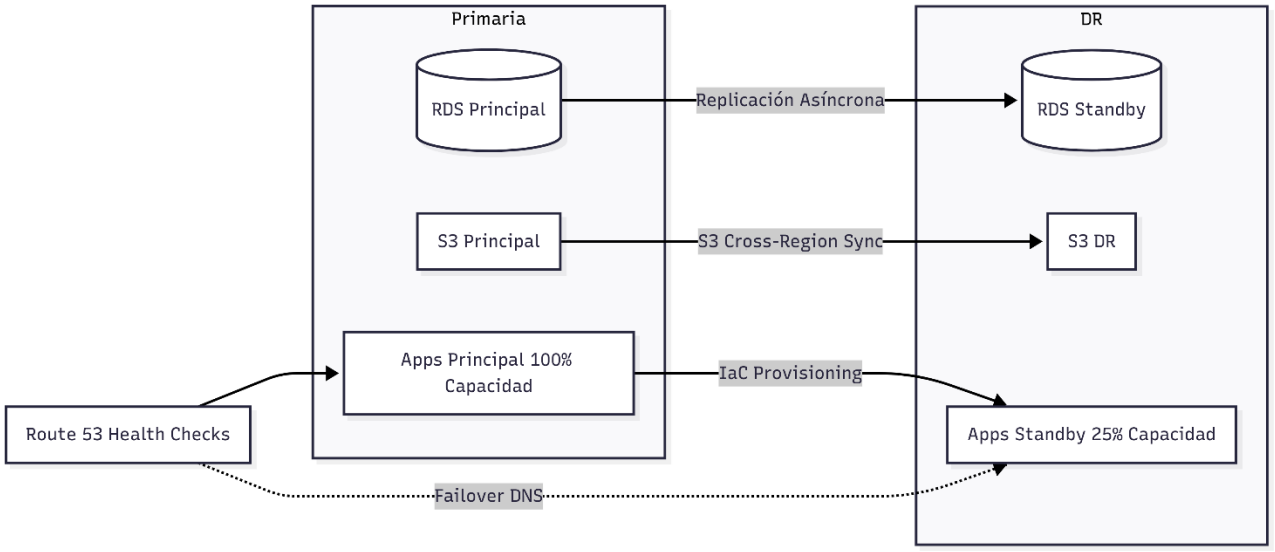
1. **Alta Disponibilidad y Tolerancia a Fallos**
   Objetivo: 99,95% de disponibilidad (4,38 horas de inactividad/año)
   Implementación:

- **Despliegue Multi-AZ:** Todos los componentes críticos desplegados en 3 zonas de disponibilidad.
- **Grupos de Autoescalado:** Servicios ECS con escalado basado en seguimiento de objetivo (CPU 70%, Memoria 80%)
- **Chequeos de Salud:** Chequeos de salud al balanceador de carga de aplicación con intervalos de 30 segundos.
- **Interruptores de Circuito:** Implementación del patrón Hystrix con timeout de 5 segundos y umbral de fallo del 50%.

Servicios AWS:

- **Application Load Balancer:** Balanceo de carga inter-zona con chequeos de salud.
- **ECS Fargate:** Orquestación de contenedores con autoescalado de servicios.
- **RDS Multi-AZ:** Conmutación por error automática (failover) con un RTO de 1-2 minutos.
- **ElastiCache Cluster Mode:** Replicación de Redis con conmutación por error automática.

2. **Recuperación Ante Desastres**
   Objetivo: RTO ≤ 4 horas, RPO ≤ 15 minutos.
   Implementación:



Estrategia de Disaster Recovery:

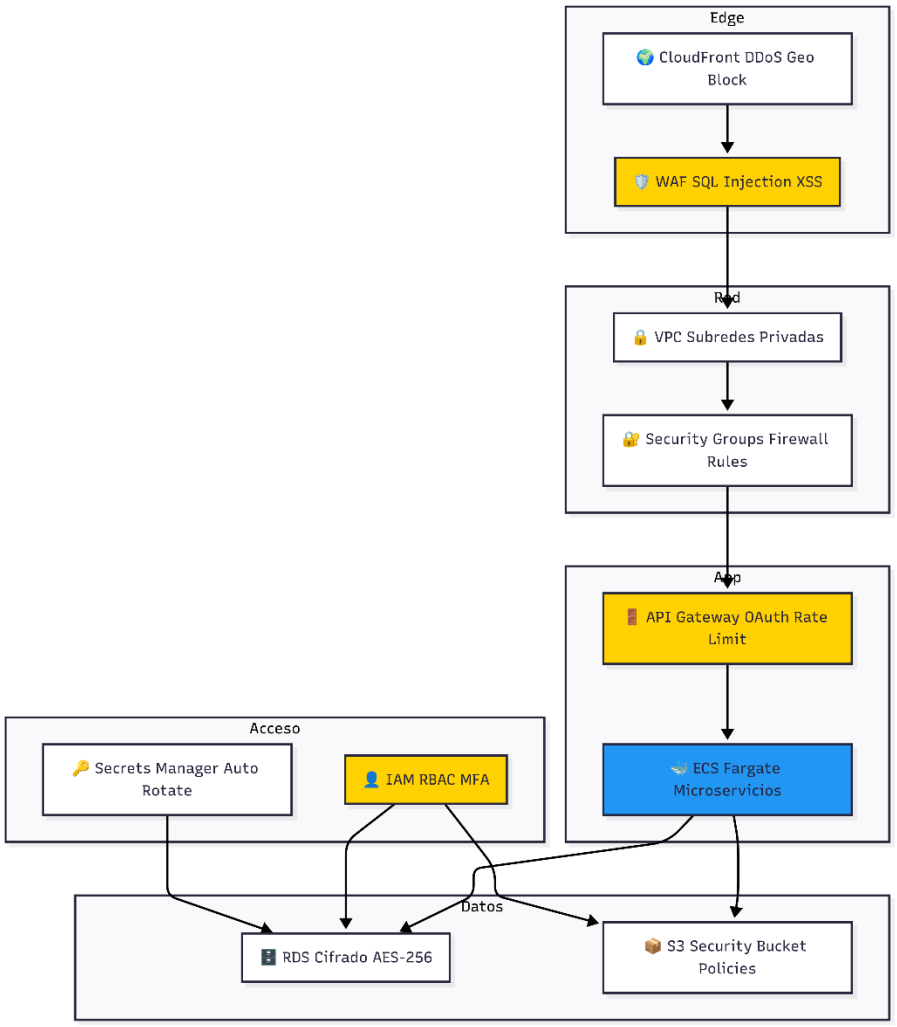
- **Warm Standby:** La infraestructura de aplicación (AWS) opera en Espera Caliente (*Warm Standby* a 25%) en la Región AWS de DR. El Core Data sensible utiliza infraestructura de Standby dentro de la jurisdicción de Ecuador (LocalDC).
- **Conmutación por Error Automatizada:** La conmutación por error de DNS vía Route 53 se activa en 2 minutos, dirigiendo el tráfico al Standby en el LocalDC para

el Core Data, o a la infraestructura de DR en AWS para los microservicios y la aplicación.

- **Replicación de Datos:** Doble replicación: 1. Replicas de lectura de AWS interregionales (15 min. lag) para el AuditDB y la capa de aplicación. 2. Replicación en país para la base de datos de Core Data sensible, asegurando la soberanía.
- **Procedimientos de Recuperación:** Recuperación prioritaria del Core Data en el LocalDC. Autoescalado de infraestructura de aplicaciones de AWS a plena capacidad en 30 minutos una vez que el Core Data esté en línea.

3. **Implementación de Seguridad**

Estrategia de Defensa en Profundidad:



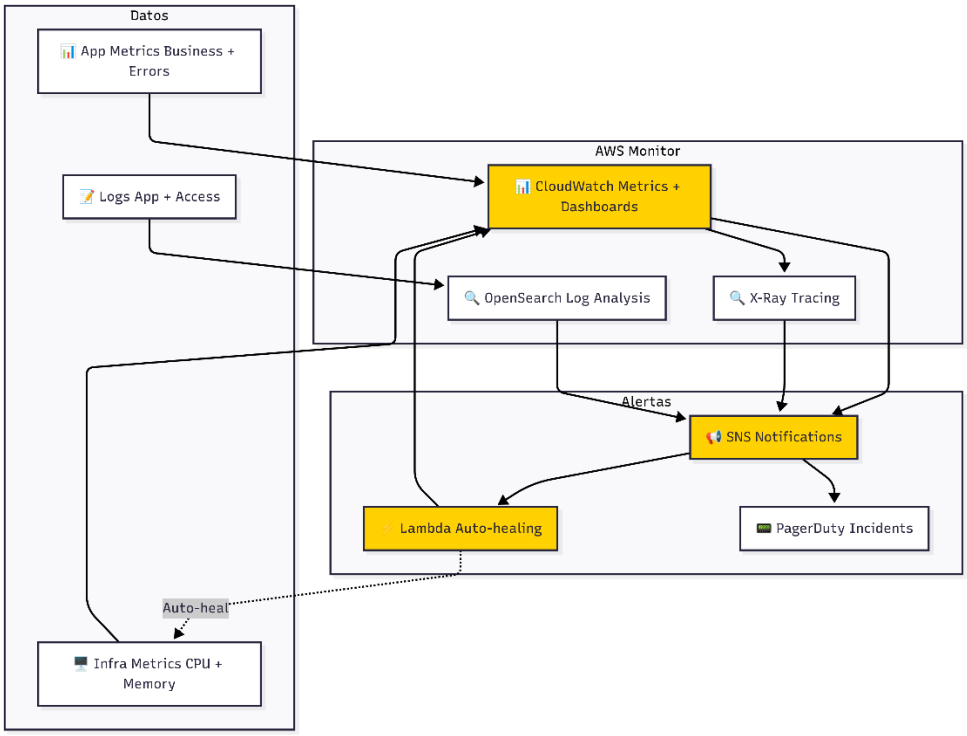
Controles de Seguridad Clave

- **Aislamiento de Red:** Todas las capas de aplicación, bases de datos y message brokers residen en subredes privadas sin acceso directo desde Internet, limitando la superficie de ataque.

- **Cifrado Total:** TLS 1.3 y mTLS son obligatorios para los datos en tránsito (entre el Edge, el APIGateway, y los ECS Microservicios). Se utiliza AES-256 para el cifrado de datos en reposo (bases de datos y S3).
- **Gestión de Claves:** Se utiliza AWS KMS con claves gestionadas por el cliente, aplicando políticas de rotación automática para los secretos críticos.
- **Control de Acceso:** Se implementa un modelo RBAC (Role-Based Access Control) mediante roles IAM con el principio de menor privilegio y credenciales temporales.
- **Monitorización:** Se utilizan CloudTrail, GuardDuty y Config para el registro de auditoría, la detección de amenazas, y la supervisión de la conformidad de la infraestructura con las políticas de seguridad.

4. **Monitoreo y Auto-Healing**

Estrategia Integral de Monitoreo:



Capacidades de Auto-Healing:

- **Recuperación de Servicio:** Reinicio automático de contenedores ante fallos en el chequeo de salud.
- **Respuesta de Escalado:** Activación de autoescalado basada en umbrales de CPU/memoria.
- **Conmutación por Error de Base de Datos:** Failover automático de RDS con una recuperación de 1-2 minutos.
- **Reconstrucción de Caché (Adaptado):** Reemplazo automático de nodos de clúster Redis y pre-calentamiento de datos estáticos/configuración. El

poblamiento de datos dinámicos se realiza mediante la lógice de Cache-Aside de la aplicación.

Métricas y Alertas Clave:

- **Métricas de SLA:** Objetivo de disponibilidad del 99,95% con monitorización en tiempo real.
- **Métricas de Rendimiento:** Tiempo de respuesta de API < 200 ms (P95), monitoreo de rendimiento.
- **Métricas de Negocio:** Tasa de éxito de transacciones > 99,9%, tasa de detección de fraude.
- **Métricas de Seguridad:** Intentos de autenticación fallidos, patrones de actividad sospechosa.

---

7. Cumplimiento Regulatorio

Marco Regulatorio Bancario de Ecuador

Recomendación: Arquitectura de Nube Híbrida con Procesamiento Local de Datos (Ecuador Data Residency).

Justificación:

1. **Cumplimiento Legal y Soberanía de Datos:** Garantiza el cumplimiento de los requisitos de la Superintendencia de Bancos, manteniendo el procesamiento y almacenamiento de datos sensibles dentro del territorio nacional mediante la integración con un Centro de Datos Local.
2. **Eficiencia Operativa:** Se aprovecha la Región AWS más cercana para la infraestructura de Disaster Recovery y para ejecutar cargas de trabajo no sensibles o escalables, asegurando la confiabilidad y reduciendo los costos operativos del CDL.

Alternativas Consideradas:

- **Infraestructura Totalmente Local (On-Premises):** Garantiza soberanía completa, pero resultaría en costos operativos significativamente más altos y una escalabilidad limitada, dificultando la competitividad.

Estándares Regulatorios Clave e Implementación

Regulación	**Requisitos Clave**	**Implementación Arquitectónica**
Superintendencia de Bancos	Soberanía de datos, gestión de riesgos operativos,	Despliegue Multi-AZ en CDL, registros de auditoría

	estándares de ciberseguridad.	inmutables, implementación de MFA
Ley de Protección de Datos	Gestión de consentimiento, minimización de datos, procesamiento local	Controles de datos a nivel de microservicios, seguimiento explícito de consentimiento, procesamiento en jurisdicción de Ecuador.
PCI DSS	Cifrado de datos de tarjeta, segmentación de red, controles de acceso	Cifrado AES-256, aislamiento de VPC, RBAC con registro de auditoría.
AMI/CFT Compliance	Monitorización en tiempo real, reportes de actividad sospechosa	Detección de patrones basada en ML, reportes regulatorios automatizados.

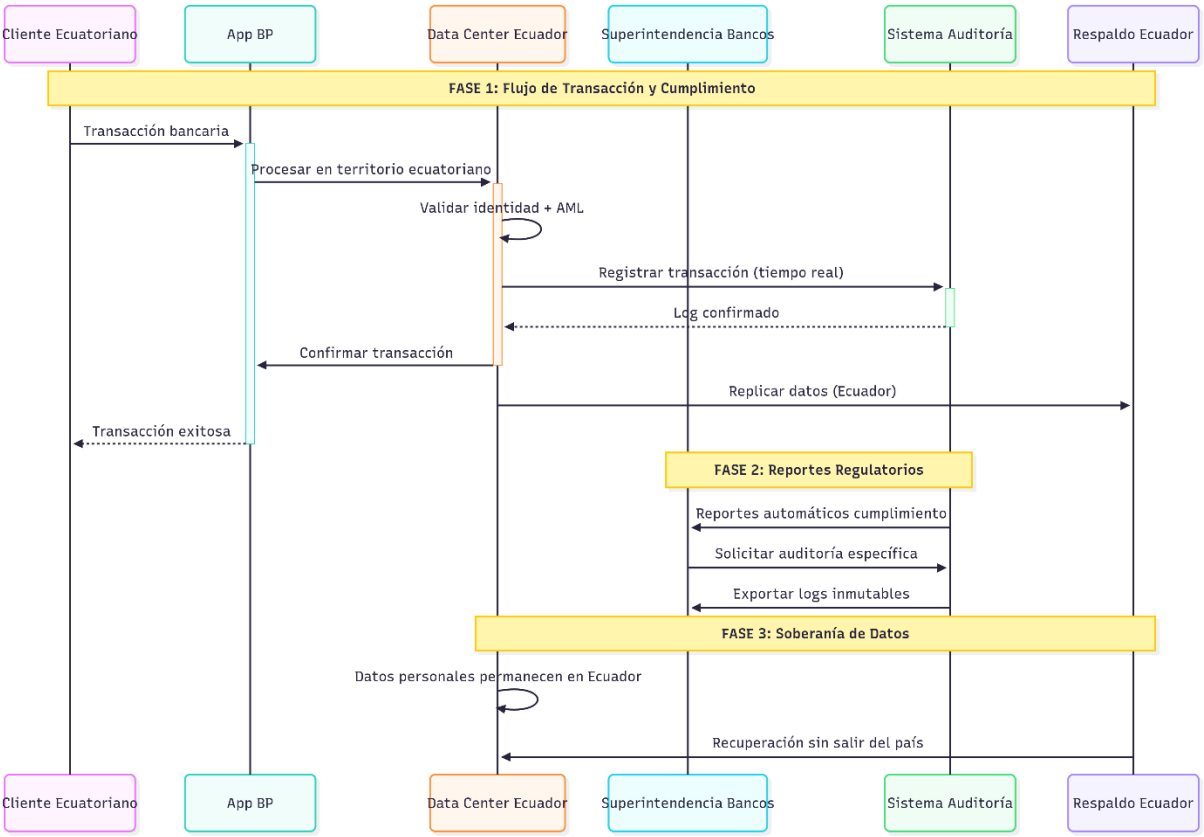
PCI DSS (Estándar de Seguridad de Datos de la Industria de Tarjetas de Pago)

Requisito: Manejo seguro de datos de tarjetas de pago para transacciones internacionales (Visa, Mastercard) que operan en Ecuador.

Conformidad Arquitectónica:

- **Cifrado de Datos:** Todos los datos de tarjetas de pago cifrados usando AES-256 tanto en tránsito como en reposo.
- **Segmentación de Red:** El Procesamiento de datos de tarjetas aislado de subredes VPC dedicadas con grupos de seguridad estrictos.
- **Controles de Acceso:** Acceso basado en roles con autenticación multifactor y registro completo de auditoría.
- **Pruebas de Seguridad:** Escaneo de vulnerabilidades automatizado y pruebas de penetración anuales realizadas por proveedores certificados.

Arquitectura de Cumplimiento en Ecuador:



Mapeo de Cumplimiento para Ecuador:

- **Superintendencia de Bancos:** Autenticación multifactor, controles de riesgo operativo, procesamiento de datos locales.
- **Ley de Protección de Datos:** Gestión explícita de consentimiento, minimización de datos, requisitos de almacenamiento local.
- **AML/CFT (Antilavado de Dinero):** Monitorización de transacciones en tiempo real con detección de patrones basada en ML para el reporte de actividad sospechosa.
- **Código Monetario y Financiero:** Conformidad con las regulaciones de sistemas de pago electrónico del Banco Central del Ecuador.