Nombre: Marlon Cedeño Panezo

Informe de Decisiones Arquitectónicas para Sistema de Integración Bancaria

1. Visión General de la Arquitectura

El sistema fue diseñado bajo una arquitectura de microservicios orientada a eventos, con una base en el patrón de arquitectura hexagonal. Esta decisión responde a la necesidad de integrar diversos servicios bancarios y sistemas externos de manera segura, escalable y desacoplada. Los microservicios permiten desplegar, escalar y mantener cada funcionalidad de manera independiente, mientras que el enfoque orientado a eventos asegura baja latencia, alta disponibilidad y facilidad para gestionar integraciones heterogéneas.

Desde el punto de vista funcional, el flujo inicia con un usuario accediendo desde Web o Mobile, autenticándose mediante OAuth 2.0, luego se enrutan las solicitudes al API Gateway y, posteriormente, a los microservicios correspondientes. La lógica de integración con servicios externos es gestionada por un Integration Service desacoplado, que se comunica mediante NATS y persiste trazabilidad en MongoDB.

2. Fortalecimiento del Modelo C4

2.1 Diagrama de Contexto

Este representa los siguientes elementos:

Actores Externos: Usuario (Web/Mobile), Core Bancario, Sistemas Externos.

Sistema: Sistema de Integración Bancaria.

Flujos: Comunicación HTTPS segura, autenticación OAuth 2.0.

Diagrama Contexto
https://s.icepanel.io/EIHIeEuC32s2vP/CwOf/landscape/diagrams/viewer?diagram=Fx1UKLaJj9&model=vQW0ej4nKB&overlay_tab=tags&x1=-2291.1&x2=1123.1&y1=-7.5&y2=1624.6

2.2 Diagrama de Contenedores

Se representan los siguientes contenedores:

- Aplicaciones cliente (Web, Mobile).
- API Gateway (Kong).
- Servidor OAuth 2.0.
- Microservicios (Go/Node).
- NATS para mensajería.
- MongoDB Atlas para persistencia.

Cada contenedor fue seleccionado con base en su capacidad para integrar y operar eficientemente en entornos financieros:

Kong (API Gateway): Elegido por su extensibilidad mediante plugins, control de acceso nativo, soporte para autenticación OAuth 2.0 y fácil integración con microservicios. Beneficio: gestión centralizada del tráfico y seguridad.

OAuth 2.0 Server: Estándar robusto para autenticación. Permite flujos avanzados como SSO y MFA. Beneficio: experiencia de usuario segura y conforme a normativas bancarias.

Go/Node.js (Microservicios): Go ofrece rendimiento y concurrencia eficiente; Node.js es útil para tareas I/O intensivas. Beneficio: rendimiento óptimo y adaptabilidad según la naturaleza del servicio.

NATS (mensajería): Sistema ligero y de alta disponibilidad, ideal para sistemas distribuidos. Beneficio: desacoplamiento efectivo y baja latencia.

MongoDB Atlas: Base de datos orientada a documentos, ideal para almacenar logs e integraciones. Beneficio: escalabilidad, esquema flexible, y multicloud.

Diagrama Contenedores:

https://s.icepanel.io/EIHIeEuC32s2vP/WR15/landscape/diagrams/viewer?diagram=87iu77uisf8&model=vQW0ej4nKB&overlay_tab=tags&x1=-3428.2&x2=1257.9&y1=-648.5&y2=1591.5

2.3 Diagrama de Componentes

Dentro del Integration Service se identifican componentes como:

- NATS Listener Adapter
- IntegrationUseCase
- MessageDispatcher
- CoreBankingClient
- MongoDBAdapter

Cada uno de estos componentes fue diseñado siguiendo el patrón de arquitectura Hexagonal (Ports & Adapters), cuyo objetivo es separar claramente la lógica de negocio de los detalles de infraestructura y de entrada/salida.

Importancia y Beneficios:

- Aislamiento de la lógica de negocio: Permite que el núcleo de la aplicación (IntegrationUseCase) no dependa de frameworks, bases de datos o servicios

externos. Esto garantiza mayor claridad, estabilidad y cohesión en la lógica de negocio.

- Facilidad para realizar pruebas unitarias: Gracias al aislamiento, los componentes internos pueden ser testeados sin necesidad de levantar bases de datos ni servicios externos. Esto acelera el desarrollo y mejora la calidad del código.
- Flexibilidad tecnológica: Si en el futuro se desea cambiar MongoDB por PostgreSQL, o NATS por RabbitMQ, solo se deben modificar los adaptadores correspondientes (MongoDBAdapter, NATS Listener Adapter), sin tocar la lógica del negocio. Esto facilita la evolución del sistema sin riesgo de errores cruzados.
- Desacoplamiento y mantenibilidad: Cada adaptador es responsable de un canal o integración específica (mensajería, almacenamiento, API externa). Esto simplifica la detección de errores, la trazabilidad de fallos y permite realizar despliegues incrementales.
- Reusabilidad y consistencia: Este diseño promueve que varios componentes reutilicen lógica de negocio compartida a través del IntegrationUseCase, evitando duplicaciones.
- Cumplimiento con principios SOLID: Especialmente el Principio de Responsabilidad Única y el Principio de Inversión de Dependencias, que son cruciales para mantener una arquitectura robusta y sostenible a largo plazo.
- En conjunto, esta organización permite que el sistema sea más robusto, escalable, y fácil de mantener y evolucionar, lo cual es especialmente valioso en entornos financieros donde los cambios deben ser controlados, auditables y seguros.

Diagrama Componentes:

https://s.icepanel.io/EIHIeEuC32s2vP/3jn5/landscape/diagrams/viewer?diagram=YAb0AKf346&model=yihtdja23ps&object=ihY2XFmaio&object_tab=details&overlay_tab=tags&x1=-818&x2=2445.5&y1=-320&y2=1240

3. Consideraciones Económicas y de Costo

- Se optó por tecnologías open source (Go, NATS, MongoDB, Kong) para evitar costos de licenciamiento.
- MongoDB Atlas ofrece un modelo multicloud con escalado flexible, optimizando el uso según demanda.
- Kubernetes permite despliegues autoscalables, reduciendo costos en horarios de baja carga.
- El uso de herramientas como Prometheus y Grafana para monitoreo evita inversiones en plataformas cerradas.
- La arquitectura modular reduce los costos de mantenimiento correctivo y evolutivo.

- La elección de protocolos ligeros como gRPC y NATS reduce el uso de red y mejora la eficiencia del hardware**.**

4. Estándares del Sector Financiero y Cumplimiento

4.1 Seguridad y Autenticación

- Se implementó OAuth 2.0 y OIDC, conforme a los estándares de autenticación moderna, como exigido por regulaciones financieras.
- Uso de JWT firmados para garantizar integridad y trazabilidad.
- Cifrado TLS 1.2/1.3 en tránsito y cifrado en reposo (at-rest encryption) en MongoDB Atlas.
- Integración con proveedores de identidad permite cumplimiento de políticas de identidad federada.

4.2 Trazabilidad y Auditoría

Todos los eventos se almacenan estructuradamente en MongoDB, facilitando auditorías regulatorias.

Se implementan correlation IDs en cada interacción para trazabilidad completa entre sistemas.

4.3 Alta Disponibilidad y Resiliencia

Clustering de NATS y despliegue multi-zona de MongoDB Atlas aseguran redundancia.

Kubernetes asegura recuperación automática ante fallos, con probes y restart policies.

Uso de circuit breakers y retries programados en los microservicios mejora la resiliencia ante fallos de red o sistemas externos.

4.4 Interoperabilidad

Interfaces REST/gRPC cumplen con estándares industriales, garantizando compatibilidad con sistemas bancarios legados y proveedores de servicios externos.

5. Conclusiones

La arquitectura propuesta permite integrar de forma segura, escalable y mantenible servicios bancarios y externos:

- Usa patrones modernos: eventos, hexagonal, microservicios
- Se apoya en herramientas maduras: NATS, MongoDB, Api Gateay
- Optimiza resiliencia y trazabilidad: Circuit Breakers, Retrys.