PROPUESTA EJERCICIO PRACTICO DEVSUP

PROPUESTA DE TECNICA

Preparado por:

Christian Yépez
christyepez@gmail.com

TABLA DE CONTENIDO

ANTECEDENTES

Diseñar una arquitectura de integración para la modernización de sistemas bancarios.

Escenario:

Un banco tradicional busca modernizar su infraestructura tecnológica para mejorar sus servicios digitales y cumplir con nuevas tendencias de industria y regulaciones. se requiere integrar:

1. Core bancario tradicional existente, mas un nuevo core bancario digital.
2. Nuevo sistema de banca web y banca móvil.
3. Plataforma de servicios de pago
4. Apis para servicios de terceros (open Finance)
5. Sistema de gestión de riesgos
6. Sistema de prevención de fraudes.

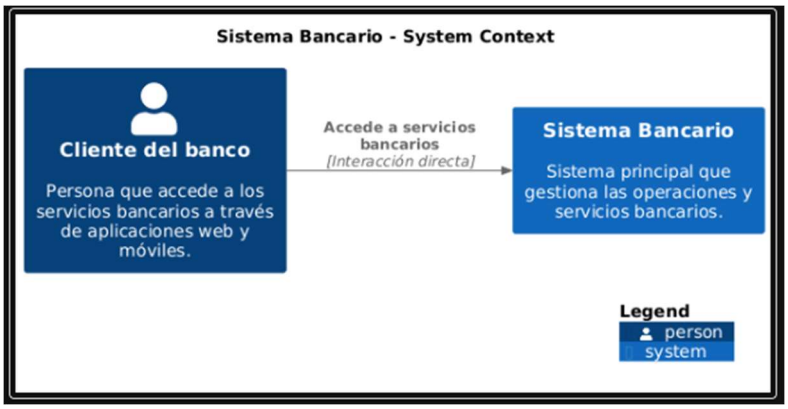
Tareas:

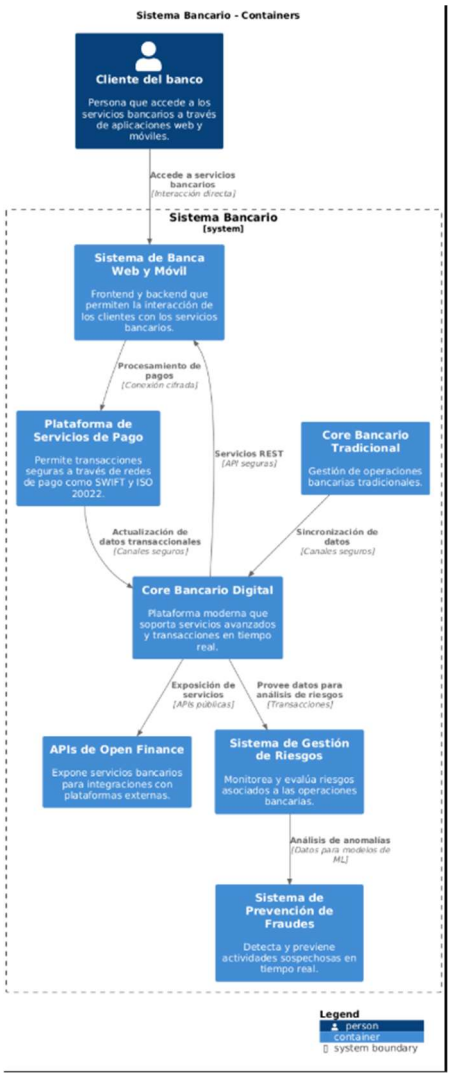
1. Diseñar una arquitectura de integración de alto nivel: Presentar una propuesta con diagramas claros y de alto nivel. Incluir el estándar BIAN en el diseño arquitectónico suma puntos frente al cliente final.
2. Especificar patrones de integración y tecnologías a utilizar. Justificar los patrones utilizados, explicando donde y porque se aplicaron en el diseño (no es solo mencionarlos).
3. Abordar requisitos de seguridad, cumpliendo normativa, ley orgánica de protección de datos personales: importante conocer los elementos que confirman este apartado, se debe argumentar y explicar como se encuentran estos presentes en la solución aportada.
4. Proponer estrategia para garantizar alta disponibilidad y recuperación ante desastres: presentar una estrategia basadas en la implementación de diseño propuesto. no se busca un repaso teórico de estrategias genéricas, sino un enfoque practico adaptado al diseño propuesto.
5. Proponer estrategia de integracion multicore: Explicar como y donde se reflea esta estrategia en el diseño.
6. Delinear un enfoque para la gestión de identidad y acceso en todos los sistemas: detallar los mecanismos propuestos para garantizar la seguridad en la integración de los sistemas. enfatizar los métodos utilizados para la autenticación, autorización y control de acceso en los distintos componentes del diseño.

7. Diseñar una estrategia de API internas y externas, bajo estándares de la industria respecto a la mensajería.
8. Proponer un modelo de gobierno de API y microservicios.
9. Proponer un plan de migración gradual que minimice el riesgo operativo.

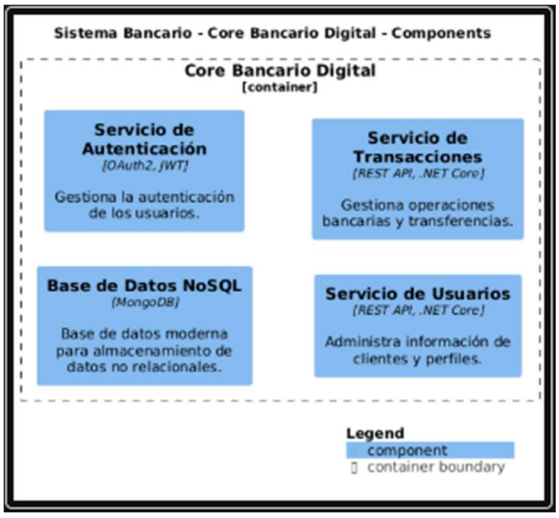
ARQUITECTURA PROPUESTA

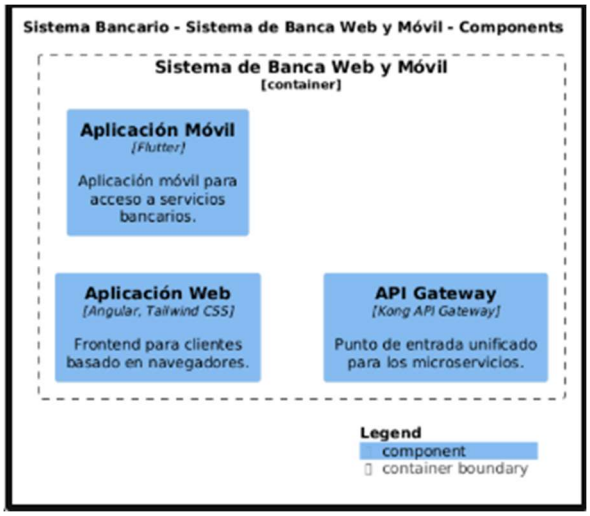
Apoyar a BANCO en el desarrollo e implementación de una arquitectura que permita integrar los sistemas del core bancario llevando a un esquema Nube.

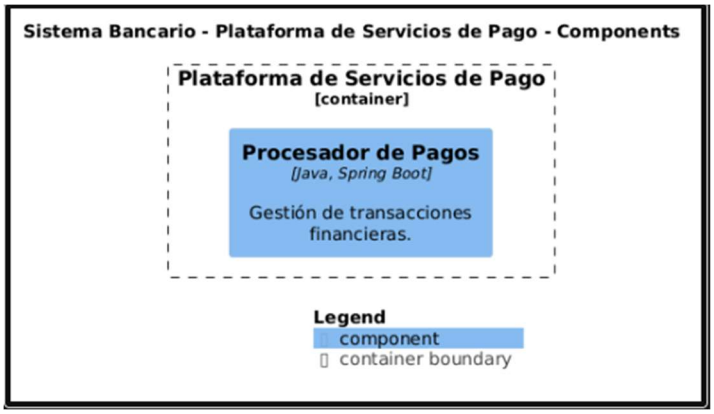


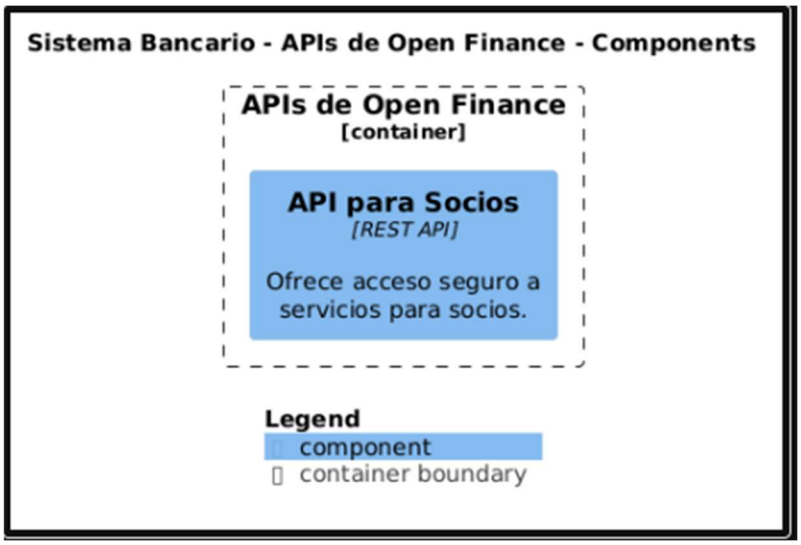


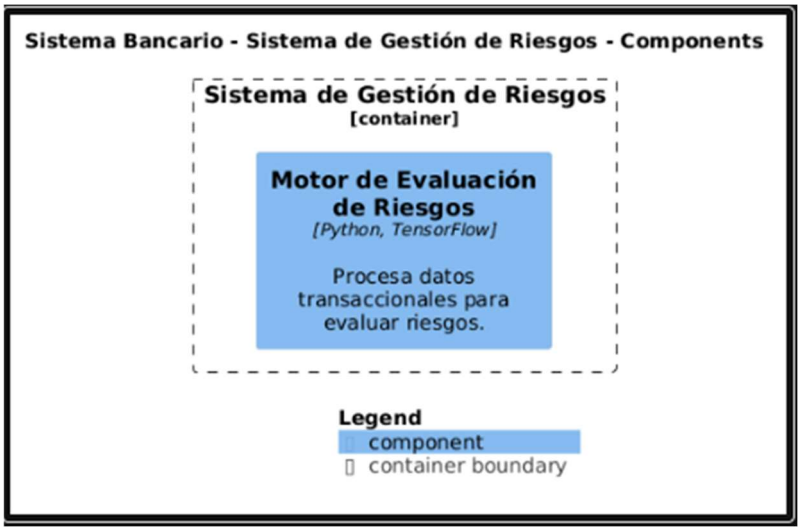

Sistema Bancario - Core Bancario Tradicional - Components
Core Bancario Tradicional
[container]
Servicios SOAP
[SOAP, XML]
Servicios legacy basados en SOAP para operaciones básicas.
Base de Datos SQL Server
[Microsoft SQL Server]
Almacena información crítica del sistema bancario tradicional.
Legend
component
container boundary

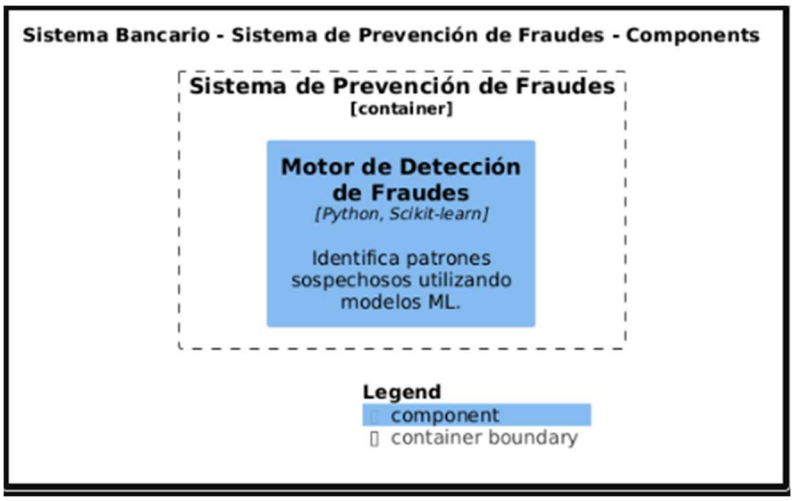

Sistema Bancario - Core Bancario Digital - Components
Core Bancario Digital
[container]
Servicio de Autenticación
[OAuth2, JWT]
Gestiona la autenticación de los usuarios.
Servicio de Transacciones
[REST API, .NET Core]
Gestiona operaciones bancarias y transferencias.
Base de Datos NoSQL
[MongoDB]
Base de datos moderna para almacenamiento de datos no relacionales.
Servicio de Usuarios
[REST API, .NET Core]
Administra información de clientes y perfiles.
Legend
component
container boundary


Sistema Bancario - Sistema de Banca Web y Móvil - Components
Sistema de Banca Web y Móvil
[container]
Aplicación Móvil
[Flutter]
Aplicación móvil para acceso a servicios bancarios.
Aplicación Web
[Angular, Tailwind CSS]
Frontend para clientes basado en navegadores.
API Gateway
[Kong API Gateway]
Punto de entrada unificado para los microservicios.
Legend
component
container boundary


Sistema Bancario - Plataforma de Servicios de Pago - Components
Plataforma de Servicios de Pago
[container]
Procesador de Pagos
[Java, Spring Boot]
Gestión de transacciones financieras.
Legend
component
container boundary


Sistema Bancario - APIs de Open Finance - Components
APIs de Open Finance
[container]
API para Socios
[REST API]
Ofrece acceso seguro a servicios para socios.
Legend
component
container boundary


Sistema Bancario - Sistema de Gestión de Riesgos - Components
Sistema de Gestión de Riesgos
[container]
Motor de Evaluación de Riesgos
[Python, TensorFlow]
Procesa datos transaccionales para evaluar riesgos.
Legend
component
container boundary


Sistema Bancario - Sistema de Prevención de Fraudes - Components
Sistema de Prevención de Fraudes
[container]
Motor de Detección de Fraudes
[Python, Scikit-learn]
Identifica patrones sospechosos utilizando modelos ML.
Legend
component
container boundary

EJERCICIO PRÁCTICO DE ARQUITECTURA DE INTEGRACIÓN

1. INTRODUCCIÓN

Este documento presenta el diseño de una arquitectura de integración para la modernización de sistemas bancarios. La solución propuesta está alineada con estándares de la industria, incluyendo BIAN, y abarca integración de sistemas internos y externos, seguridad, alta disponibilidad, y estrategia de migración.

2. DISEÑO DE ARQUITECTURA DE INTEGRACIÓN

2.1 Diagrama de Contexto (Nivel C4: Contexto)

Este diagrama muestra los sistemas clave involucrados en la arquitectura:

- Core bancario tradicional existente
- Nuevo core bancario digital
- Nuevo sistema de banca web y móvil
- Plataforma de servicios de pago
- APIs para servicios de terceros (Open Finance)
- Sistema de gestión de riesgos
- Sistema de prevención de fraudes

Explicación: El contexto general resalta las interacciones entre el banco, clientes, socios de terceros, y reguladores.

Script PlantUML:

```
@startuml
actor Cliente
package Banco {
  rectangle "Core Bancario Tradicional" as CoreTradicional
  rectangle "Core Bancario Digital" as CoreDigital
  rectangle "Banca Web y Móvil" as BancaWebMovil
  rectangle "Plataforma de Servicios de Pago" as ServiciosPago
  rectangle "APIs de Open Finance" as OpenFinance
  rectangle "Gestión de Riesgos" as GestionRiesgos
  rectangle "Prevención de Fraudes" as PrevFraudes
}
Cliente --> BancaWebMovil : Acceso
BancaWebMovil --> CoreDigital : Solicitudes
CoreDigital --> CoreTradicional : Sincronización
ServiciosPago --> CoreDigital : Procesos de Pago
OpenFinance --> CoreDigital : Consumo de APIs
GestionRiesgos --> CoreDigital : Análisis
PrevFraudes --> CoreDigital : Validación
@enduml
```

2.2 Diagrama de Contenedores (Nivel C4: Contenedores)

Este diagrama detalla las aplicaciones y sistemas en contenedores:

- **Core Tradicional:**
    - Base de datos relacional (SQL Server).
    - Servicios SOAP.
- **Core Digital:**
    - Microservicios (Spring Boot o .NET Core).
    - Base de datos no relacional (MongoDB).
- **Sistema de Banca Web/Móvil:**
    - Frontend (Angular o React).
    - Backend API Gateway (Kong o Azure API Management).
- **Plataforma de Servicios de Pago:**
    - Integración con redes de pagos (ISO 20022).
- **Gestión de riesgos:**
    - Análisis de datos (Apache Kafka, Spark).
- **Prevención de fraudes:**
    - Modelos de machine learning (Python/MLFlow).

Explicación: Cada contenedor representa un sistema que se comunica a través de APIs internas y eventos asincrónicos.

Script PlantUML:

```
@startuml
package CoreTradicional {
  component SQLServer
  component SOAPServices
}
package CoreDigital {
  component Microservices
  component MongoDB
}
package BancaWebMovil {
  component Frontend
  component APIGateway
}
package ServiciosPago {
  component ISO20022Integration
}
package GestionRiesgos {
  component Kafka
  component Spark
}
package PrevFraudes {
  component MLFlow
}
Frontend --> APIGateway
APIGateway --> Microservices
Microservices --> MongoDB
Microservices --> SQLServer
ISO20022Integration --> Microservices
Kafka --> Spark
MLFlow --> Spark
@enduml
```

2.3 Diagrama de Componentes (Nivel C4: Componentes)

Este diagrama especifica componentes clave dentro de cada contenedor, incluyendo:

- **Core Tradicional:** Adaptadores de integración para interoperabilidad con nuevos sistemas.
- **Core Digital:** Microservicios especializados (Cuentas, Pagos, Usuarios).
- **Banca Web/Móvil:** Módulos de autenticación, transacciones y consulta de saldos.
- **Prevención de Fraudes:** Detector de anomalías, módulo de alertas en tiempo real.

Script PlantUML:

```
@startuml
package CoreDigital {
  component "Microservicio Cuentas" as Cuentas
  component "Microservicio Pagos" as Pagos
  component "Microservicio Usuarios" as Usuarios
}
package BancaWebMovil {
  component "Módulo Autenticación" as Autenticacion
  component "Módulo Transacciones" as Transacciones
  component "Módulo Consulta Saldos" as ConsultaSaldos
}
Autenticacion --> Usuarios
Transacciones --> Cuentas
Transacciones --> Pagos
ConsultaSaldos --> Cuentas
@enduml
```

3. Patrones de Integración y Tecnologías

3.1 Patrones de Integración

- **Orquestación:** Para coordinar procesos entre el Core Digital y Tradicional.
- **Event-Driven Architecture:** Utilizando Kafka para manejar eventos entre sistemas.
- **API Gateway:** Para centralizar y asegurar el acceso a las APIs.
- **Adapter Pattern:** En el Core Tradicional para traducir entre protocolos legacy y modernos.

Justificación:

- **Orquestación:** Ideal para procesos secuenciales críticos como pagos.
- **EDA:** Minimiza la latencia y desacopla sistemas.
- **API Gateway:** Mejora la seguridad y facilita la exposición de servicios.
- **Adapter Pattern:** Extiende la vida útil de sistemas legacy.

3.2 Tecnologías Seleccionadas

- **Kafka:** Broker de mensajería para manejar grandes volúmenes de eventos.
- **Azure API Management:** API Gateway con capacidad de escalabilidad y monitoreo.

- **OAuth2/OpenID Connect:** Gestión de identidad y acceso segura.
- **BIAN:** Facilita interoperabilidad entre sistemas bancarios. Por medio de los estandares que nos permite tener.

4. Seguridad

- **Encriptación:** Uso de AES256 para datos en tránsito y en reposo.
- **Autenticación:** MFA y OAuth 2.0.
- **Normativa:** Cumplimiento de GDPR/CCPA.
- **Auditorías:** Monitoreo continuo con Azure Monitor y Logs.

Justificación: Cumple requisitos normativos y protege contra accesos no autorizados. Teniendo un monitoreo constante, alertas inmediatas y disponibilidad de auditorias.

5. Alta Disponibilidad y Recuperación ante Desastres

- **HA:** Replicación activa-activa en la nube.
- **DR:** Backup incremental con Azure Backup.
- **Load Balancing:** Uso de Azure Front Door para baja latencia.

Justificación: La replicación y balanceo garantizan la continuidad operativa. Manejando respaldos invrementales y replicacion de la informacion en regiones seguras.

6. Gestión de Identidad y Acceso

- **Centralización:** Implementación de un IAM.
- **Roles y Perfiles:** Basados en RBAC.
- **SSO:** Integración con IdentityServer.

Justificación: Simplifica la gestión de accesos y asegura políticas consistentes. Por medio de herraamientas que controlan eficazmente los accesos por roles.

7. Estrategia de API Internas y Externas

- **Internas:** JSON sobre HTTP/REST con control de versiones.
- **Externas:** OpenAPI Specification (Swagger).
- **Mensajería:** Utilización de eventos (Kafka) para minimizar latencia.

Justificación: Garantiza interoperabilidad y facilita la integración con terceros.

8. Modelo de Gobierno de API y Microservicios

- **Documentación:** Utilizar Swagger para APIs.
- **Estandares :** Manejando esquemas alineados a las buenas practicas en el desarrollo
- **Control de Calidad:** CI/CD con GitHub Actions.
- **Supervisión:** Alertas configuradas en Prometheus y Grafana.

Justificación: Asegura calidad en el desarrollo y operación de servicios.

9. Plan de Migración Gradual

- **Fase 1:** Implementar capa de integración para coexistencia del Core Tradicional y Digital.
- **Fase 2:** Migrar servicios prioritarios (banca web y móvil).
- **Fase 3:** Extender integración a terceros (Open Finance).

Justificación: Reduce riesgos operativos y permite iteraciones controladas.

10. Conclusión

La solución propuesta ofrece un enfoque modular y escalable, permitiendo al banco modernizar sus sistemas sin interrumpir operaciones críticas.