**1. Introducción**

**1.1 Propósito del documento**

El presente documento describe el diseño técnico del módulo Customer Position, un componente central dentro de la arquitectura de integración del banco para consolidar la posición global del cliente bajo el estándar BIAN. Este diseño forma parte del marco de referencia definido previamente, y tiene como finalidad guiar la construcción, despliegue y evolución del microservicio encargado de materializar, almacenar y exponer este DTO (Data Transfer Object) unificado.

A diferencia de un enfoque puramente de despliegue, este diseño técnico aborda aspectos estructurales y funcionales, incluyendo:

* La composición de módulos internos.
* La relación entre eventos CDC y atributos del DTO.
* La lógica de transformación y persistencia.
* El modelado de timings y mecanismos de actualización.
* La implementación de capacidades como caching, reconciliación, y trazabilidad.

**1.2 Alcance de la solución**

Este diseño abarca principalmente dos microservicios:

* Customer Position: microservicio responsable de recibir eventos, componer el objeto CustomerPosition, consultar balances, persistir el estado y servir como punto de acceso principal.
* Conciliador: microservicio encargado de validar, reprocesar y controlar los escenarios de error, complementando al componente principal y asegurando confiabilidad en flujos críticos.

El contexto de despliegue incluye integración con:

* Eventos CDC provenientes de Mainframe IBM Z (DB2 z/OS), bases de datos Oracle bajo GoldenGate, y plataformas adicionales como Plard (tarjetas) e Inversiones PGI.
* Sistemas distribuidos como Kafka, Redis JSON, Oracle Exadata, y APIs internas (ej. CardJWKS y CardSecurity vía API Connect).
* Mecanismos de auditoría para pistas y bitácoras, integrados con Kafka Connect y almacenamiento en Object Storage (Scality).

Este diseño también contempla la integración de módulos con Spring Cloud Stream y Spring Integration, permitiendo una arquitectura desacoplada, extensible y alineada a patrones EIP (Enterprise Integration Patterns).

**1.3 Stakeholders**

Los principales interesados en este documento incluyen:

* Equipos de desarrollo responsables de implementar y desplegar los microservicios Customer Position y Conciliador.
* Arquitectos de soluciones encargados de validar la alineación con los estándares corporativos, BIAN y patrones de integración.
* Equipos de infraestructura y operaciones, particularmente en plataformas como Kafka, Redis, Oracle, y Mainframe.
* Analistas funcionales que modelan los flujos de negocio relacionados con cuentas, tarjetas, seguros e inversiones.
* Equipos de auditoría y cumplimiento, debido al tratamiento de información sensible y trazabilidad.

**Objetivos del diseño**

* Consolidar la posición global del cliente a partir de múltiples fuentes de datos (cuentas, tarjetas, seguros, inversiones), con una estructura basada en el modelo CustomerPosition alineado a BIAN.
* Consumir y procesar eventos CDC de distintas plataformas tecnológicas (DB2, Oracle, Kafka, JSON), estableciendo la lógica de transformación hacia el DTO.
* Consultar saldos de manera directa desde Mainframe mediante DDF (Distributed Data Facility), garantizando precisión en los atributos de balance.
* Persistir y exponer el DTO resultante en una arquitectura de almacenamiento dual:
  + Redis JSON como cache L1 de acceso rápido.
  + Oracle Exadata como repositorio persistente indexado.
* Implementar mecanismos de fallback, conciliación y reintentos, asegurando confiabilidad y consistencia eventual.
* Brindar observabilidad y trazabilidad completa, registrando todos los eventos relevantes en un bucket de Object Storage vía Kafka Connect.
* Facilitar la evolución modular del sistema, permitiendo la inclusión de nuevos dominios o eventos con mínima intervención sobre el core técnico.

**2. Arquitectura de contexto**

* La presente sección define el entorno en el cual se integra el módulo Customer Position, incluyendo los sistemas fuente que generan datos relevantes, las plataformas de mensajería, las tecnologías de almacenamiento, y otros elementos que interactúan directa o indirectamente con el sistema.
* **2.1 Sistema en contexto**
* El módulo Customer Position forma parte de una plataforma de middleware orientada a eventos, cuyo objetivo es consolidar la posición financiera completa del cliente de forma materializada, estructurada y reutilizable para distintos canales o APIs internas.
* El sistema recibe y procesa eventos provenientes de distintos dominios del banco, como cuentas, tarjetas, inversiones y seguros, con el fin de construir un objeto JSON llamado CustomerPosition, alineado con el modelo BIAN. Esta información se pone a disposición de consumidores internos a través de un esquema desacoplado, con almacenamiento en memoria (acceso rápido) y persistente (confiabilidad y auditabilidad).
* El contexto operativo se ilustra en el siguiente diagrama:
* [Imagen de despliegue: “IMG\_843F32E6-F9E3-4810-9393-B94C38E2557F.jpeg”]
* **2.2 Actores externos y relaciones**

| * **Actor externo** | **Tipo** | **Descripción e interacción** |
| --- | --- | --- |
| Mainframe IBM z/OS (DB2) | Fuente de datos | Genera eventos CDC mediante replicación de tablas de movimientos y saldos (captación). Además, es accedido vía DDF mediante JDBC para consultas directas desde el componente de balances. |
| Oracle DBs (GoldenGate) | Fuente de datos | Incluye cuatro bases de datos que generan eventos vía GoldenGate: 1) Reactor, 2) Phoenix, 3) VisualTime (todas relacionadas con seguros), y 4) PGI (posición global de inversiones). Todos los eventos son enviados a Kafka. |
| Plard (AWS) | Fuente de datos | Plataforma para tarjetas. Genera eventos cuando una tarjeta migra desde Pampa, incluyendo un nuevo CardID. Estos eventos alimentan el CustomerPosition. |
| API Connect | Gateway de APIs | Gateway corporativo que expone APIs internas como CardSecurity y CardJWKS. Se accede mediante autenticación basada en JWS-ID. |
| Kafka | Middleware de eventos | Es el bus de eventos centralizado que recibe todos los eventos CDC y otros eventos funcionales. Es consumido por los microservicios Customer Position y Conciliador. |
| Redis con RedisJSON | Repositorio de acceso rápido | Almacena el DTO materializado CustomerPosition para consultas inmediatas. No se considera una caché sino una base de datos de acceso rápido. |
| Oracle Exadata (Converged DB) | Repositorio persistente | Almacena de forma estructurada y duradera el objeto CustomerPosition. Permite búsquedas avanzadas e indexación de campos JSON. Fuente única de verdad. |
| Object Storage (Scality) | Almacenamiento de trazabilidad | Recibe eventos de auditoría desde Kafka mediante un conector Sink de Kafka Connect. Su objetivo es el enfriamiento de eventos, no su explotación inmediata. |

**2.3 Interfaces y flujos clave**

A continuación se detallan los principales flujos de interacción entre el sistema y sus actores externos:

1. Eventos CDC desde Mainframe:
   * Eventos relacionados con movimientos y saldos en DB2 son replicados hacia Kafka.
   * También se realizan consultas JDBC usando DDF para obtener el saldo actualizado al momento de construir el DTO.
2. Eventos CDC desde Oracle GoldenGate:
   * Incluye eventos de seguros (Reactor, Phoenix, VisualTime) y de la posición global de inversiones (PGI).
   * Todos estos eventos son publicados en Kafka en formato estructurado para ser consumidos por Customer Position.
3. Eventos desde Plard:
   * Incluyen la migración de tarjetas y la asignación de un nuevo CardID.
   * Estos datos son integrados directamente al DTO del cliente para mantener trazabilidad.
4. Consulta a APIs internas mediante API Connect:
   * CardJWKS y CardSecurity permiten mapear datos sensibles como PAN a CardID y realizar validaciones adicionales.
   * La comunicación se realiza usando JWS-ID como método de autenticación.
5. Persistencia del DTO generado:
   * El objeto CustomerPosition es almacenado tanto en RedisJSON (acceso rápido) como en Oracle Exadata (persistencia).
   * Se mantienen sincronizados a través del componente DTO Persist.
6. Generación de trazabilidad y auditoría:
   * Todos los eventos entrantes, salientes o generados internamente (e.g., eventos derivados) son auditados.
   * Estos eventos se envían a un tópico Kafka y posteriormente a Object Storage mediante Kafka Connect.

Perfecto. A continuación te presento la Sección 3: Requisitos y restricciones del documento ARC42 para el microservicio Customer Position, considerando todos los elementos que has proporcionado:

**3. Requisitos y restricciones**

Esta sección detalla los requisitos funcionales y no funcionales que debe cumplir el sistema Customer Position, así como las restricciones impuestas por el entorno tecnológico, organizacional o normativo en el que se desplegará.

**3.1 Requisitos funcionales**

| **ID** | **Descripción del requisito funcional** |
| --- | --- |
| RF-01 | El sistema debe construir el objeto CustomerPosition a partir de eventos provenientes de múltiples fuentes (cuentas, tarjetas, inversiones, seguros, balances). |
| RF-02 | El sistema debe mantener el objeto CustomerPosition en una base de datos de acceso rápido (Redis con RedisJSON) para exposición inmediata a consumidores internos. |
| RF-03 | El sistema debe persistir de forma estructurada el objeto CustomerPosition en una base de datos relacional con soporte JSON (Oracle Converged DB) para trazabilidad y auditabilidad. |
| RF-04 | El sistema debe consumir eventos desde Kafka provenientes de replicación CDC de DB2 (Mainframe) y Oracle GoldenGate. |
| RF-05 | El sistema debe ser capaz de integrar nuevos eventos funcionales provenientes de otras plataformas (por ejemplo, AWS Plard para tarjetas). |
| RF-06 | El sistema debe exponer la información contenida en el CustomerPosition para su consulta por otras APIs sin necesidad de orquestación en línea. |
| RF-07 | El sistema debe emitir eventos derivados cuando se detecten cambios relevantes en el CustomerPosition. |
| RF-08 | El sistema debe registrar todos los eventos recibidos, procesados y emitidos en un canal de auditoría para trazabilidad. |
| RF-09 | El sistema debe consultar directamente a DB2 (Mainframe) vía DDF para obtener balances actualizados cuando reciba eventos de cambio. |
| RF-10 | El sistema debe gestionar la lógica de consolidación de eventos y tolerancia a fallos mediante un conciliador desacoplado. |

**3.2 Requisitos no funcionales**

| **ID** | **Descripción del requisito no funcional** |
| --- | --- |
| RNF-01 | Alta disponibilidad y tolerancia a fallos: el microservicio debe funcionar en ambientes de producción distribuidos con réplicas. |
| RNF-02 | Tiempo de respuesta bajo para lectura del DTO en Redis (esperado < 100ms por consulta). |
| RNF-03 | Persistencia transaccional del objeto CustomerPosition en Oracle, garantizando consistencia eventual en escenarios de concurrencia. |
| RNF-04 | Escalabilidad horizontal en los componentes de consumo y generación de DTOs. |
| RNF-05 | Tiempos de recuperación y resiliencia: el sistema debe poder recuperarse de caídas parciales sin pérdida de datos. |
| RNF-06 | Soporte a particionamiento lógico por customerId para facilitar consolidación, enrutamiento y backpressure. |
| RNF-07 | Compatibilidad con despliegue en plataformas OpenShift bajo estándares corporativos de Banco Santander. |
| RNF-08 | Cumplimiento de estándares de seguridad y cifrado para datos sensibles, especialmente en APIs y bases de datos. |
| RNF-09 | Uso de módulos Redis JSON y capacidades de indexado JSON en Oracle para optimizar consultas. |
| RNF-10 | Compatibilidad con el esquema de integración de pistas y bitácoras corporativas mediante Kafka Connect + Scality. |

**3.3 Restricciones tecnológicas**

| **ID** | **Restricción** |
| --- | --- |
| RTEC-01 | Debe utilizar Kafka como bus de eventos corporativo. |
| RTEC-02 | Debe utilizar Oracle GoldenGate y replicación CDC como mecanismos de ingesta de eventos desde Mainframe y Oracle. |
| RTEC-03 | Debe utilizar Spring Boot con Spring Cloud Stream y Spring Integration como frameworks de implementación. |
| RTEC-04 | Debe utilizar Redis con soporte RedisJSON como base de datos de acceso rápido. |
| RTEC-05 | Debe utilizar Oracle Converged Database como base de datos persistente con capacidades JSON. |
| RTEC-06 | Debe integrarse con el sistema de almacenamiento de auditoría basado en Kafka Connect + Scality. |
| RTEC-07 | Debe exponer APIs internas únicamente vía API Connect y autenticadas con JWS-ID. |
| RTEC-08 | El microservicio debe ser desplegado dentro del namespace dto-middleware en OpenShift. |

Perfecto. A continuación, te presento la Sección 4 – Visión de la solución (Estrategia de solución) del documento ARC42, redactada según el contexto que has proporcionado:

**4. Visión de la solución**

**4.1 Descripción general de la estrategia**

La solución técnica propuesta para la construcción del módulo Customer Position dentro de la arquitectura del DTO Middleware se basa en una estrategia event-driven distribuida, diseñada para operar en una plataforma de microservicios desplegada en OpenShift. La solución contempla la implementación de dos microservicios principales:

* customer-position-ms
* conciliador-ms

Cada uno incorpora patrones definidos en la arquitectura de referencia del banco y tiene una responsabilidad técnica claramente delimitada, favoreciendo la modularidad, trazabilidad y evolución del sistema.

**4.2 Fundamentos arquitectónicos**

La arquitectura se fundamenta en los siguientes principios y tecnologías:

| **Principio / Tecnología** | **Aplicación en la solución** |
| --- | --- |
| Event-driven architecture | Los cambios en los datos fuente (Mainframe, seguros, inversiones) se reflejan mediante eventos CDC publicados en Kafka. |
| Spring Cloud Stream | Se emplea para facilitar la conexión entre microservicios y Kafka, permitiendo configuración declarativa de bindings. |
| Spring Integration | Facilita la construcción de pipelines internos dentro de los microservicios para orquestar la lógica de composición de DTOs, fallback, conciliación y publicación de nuevos eventos. |
| Redis JSON | Se utiliza como motor de almacenamiento de acceso rápido, manteniendo representaciones JSON de los objetos CustomerPosition. No se trata de una caché tradicional, sino de un repositorio primario de bajo tiempo de respuesta. |
| Oracle Converged DB | Motor de persistencia secundaria. Almacena de forma estructurada los DTOs generados, permitiendo indexación de campos JSON, búsquedas complejas y durabilidad. |
| Kafka y Kafka Connect | Actúan como bus de eventos central. Kafka Connect se utiliza para recibir eventos CDC (desde DB2, Oracle, PGI) y para emitir eventos de trazabilidad hacia Object Storage. |
| Scality S3 (Object Storage) | Destino final de eventos de trazabilidad mediante un conector Sink de Kafka. No contempla explotación de pistas, solo su almacenamiento. |
| OpenShift Secrets | Las llaves privadas necesarias para la generación de JWS ID en llamadas hacia las APIs de seguridad (CarJWKS y CarSecurity) se almacenan como secretos seguros dentro de OpenShift. |

**4.3 Separación de responsabilidades y alineación con patrones**

Cada microservicio implementa patrones específicos:

* customer-position-ms:
  + Composición y persistencia del DTO CustomerPosition.
  + Obtención de saldos mediante queries DDF hacia el Mainframe.
  + Manejo de eventos funcionales (cuentas, tarjetas, seguros, inversiones).
  + Generación y publicación de eventos derivados a partir de los datos procesados.
* conciliador-ms:
  + Recepción de comandos en caso de errores o datos incompletos.
  + Integración con canales durables para trazabilidad.
  + Delega la reconstrucción del DTO mediante eventos corregidos hacia customer-position-ms.

Este diseño garantiza desacoplamiento completo entre productores de eventos (ej. sistemas core bancarios), consumidores (ej. APIs internas), y la lógica de composición y conciliación, permitiendo escalar y mantener cada componente de forma independiente.

¿Deseas que te lo entregue ahora en formato Word o seguimos con la siguiente sección del ARC42 (Sección 5: Building Blocks)?

* RNF-01 (Modelo anidado y rendimiento): El objeto CustomerPosition se basa en un modelo JSON anidado que sigue la estructura del estándar BIAN (por ejemplo: CustomerPosition → Accounts → Contracts → Balances).

Aunque este diseño promueve una alineación semántica sólida y una extensión estructurada, su complejidad puede afectar el rendimiento de lectura, serialización y manipulación en memoria, especialmente al realizar proyecciones parciales o filtrado por atributos internos.

Como medida mitigante:

* + Se requiere que las consultas de acceso común estén respaldadas por una estrategia de proyección e indexación eficiente (por ejemplo, usando Redis JSON con path queries optimizadas).
  + Las APIs consumidoras deben diseñarse para consumir secciones específicas del DTO, sin necesidad de parsear el documento completo.
  + El modelo persistido debe permitir evolución incremental del esquema (por ejemplo, mediante versionamiento o campos opcionales) sin afectar retrocompatibilidad.
* RNF-02 (Modularidad y desacoplamiento): A pesar de que el DTO CustomerPosition representa una vista consolidada, los componentes internos deben operar de forma desacoplada para permitir la reutilización de partes del modelo (ej. solo tarjetas o solo inversiones), sin requerir la construcción completa del objeto completo si no es necesario.
* RNF-03 (Uso de cache y persistencia): El objeto completo será almacenado en:
  + Redis (para acceso rápido en formato JSON).
  + Oracle Exadata (como persistencia primaria con capacidades de indexación JSON nativa).
* Restricción técnica: El diseño JSON debe ser compatible con los motores de almacenamiento seleccionados (Redis JSON y Oracle Converged Database), garantizando:
  + Navegación por path.
  + Proyecciones parciales eficientes.
  + Capacidad de evolucionar el modelo sin pérdida de datos previos.

NFR-CP-01 – Representación desacoplada del modelo BIAN en el JSON de CustomerPosition:

El objeto JSON que representa la CustomerPosition, tanto en su persistencia en Oracle como en su almacenamiento en Redis, debe estar alineado conceptualmente con el estándar BIAN. No obstante, no se requiere que siga de forma estricta la estructura anidada y compleja propuesta por dicho modelo.

Para mejorar el rendimiento, facilitar la serialización/deserialización, y simplificar el mantenimiento, se permitirá aplicar estrategias de diseño que reduzcan la profundidad de anidamiento o que agrupen entidades lógicamente sin perder semántica funcional.

Estas decisiones deberán garantizar la compatibilidad con el contrato expuesto por el API de CustomerPosition, asegurando que el objeto pueda alimentar correctamente dicha API, incluso si internamente se modela de forma más simple.