Diseño de Arquitectura de Integración para la Modernización de Sistemas Bancarios

Arquitectura de Integración

CONFIDENCIAL
Fecha 5-ago.-25
Versión 1.0

TABLA DE CONTENIDO

Control del Documento

REGISTROS DE CAMBIOS EN EL DOCUMENTO			
Versión	Motivo	Realizado por	Fecha
1.0	Creación del documento	Javier Andres Manyoma Navia	02/08/2025

Referencias

La siguiente lista corresponde a los documentos complementarios e insumos usados para en el diseño de la arquitectura:

LISTA DE DOCUMENTOS				
Nro.	Nombre	Versión	Descripción	Ruta/Enlace
1.	GitHub			https://github.com/manyoma3/EjercicioIntegracionBancariaDevsu/new/main
2.				
3.				
4.				
n.				

1. Contextualización y Alcance de la Solución

1.1. Diseño de arquitectura de integración de alto nivel

La industria bancaria enfrenta una transformación acelerada impulsada por la digitalización, la demanda de servicios más ágiles por parte de los clientes y la aparición de nuevas tecnologías y actores en el ecosistema financiero. En este contexto, los bancos tradicionales deben modernizar su infraestructura tecnológica para mantenerse competitivos, mejorar la experiencia del cliente, reducir costos operativos y cumplir con normativas emergentes como Open Finance, gestión de datos sensibles y estándares internacionales como BIAN (Banking Industry Architecture Network).

Actualmente, muchas entidades financieras operan sobre cores bancarios tradicionales, altamente acoplados y con limitaciones para integrarse con nuevas plataformas digitales. Esta rigidez dificulta la implementación de servicios innovadores, la apertura a terceros (APIs) y el cumplimiento de estándares de interoperabilidad y seguridad.

La modernización propuesta busca adoptar una arquitectura de integración escalable y desacoplada, que permita la convivencia entre sistemas legacy y nuevos sistemas digitales, mediante el uso de tecnologías modernas como microservicios, gestión de eventos, APIs estandarizadas y servicios en la nube (AWS).

La presente iniciativa pretende llevar un paso más allá esta experiencia y permitir que las APIS expuestas tengan un buen performance, tiempos de respuesta liberando la capacidad del equipo operativo tecnologico.

Objetivo: Diseñar una arquitectura que permita interoperabilidad entre sistemas antiguos y modernos, escalabilidad, seguridad, resiliencia y cumplimiento normativo, usando servicios de integración modernos (eventos, APIs, orquestación, IAM) y el modelo de negocio de BIAN.

Alcance:

La solución propuesta contempla el diseño de una arquitectura de integración bancaria moderna, orientada a servicios y basada en estándares de la industria. Esta arquitectura permitirá al banco:

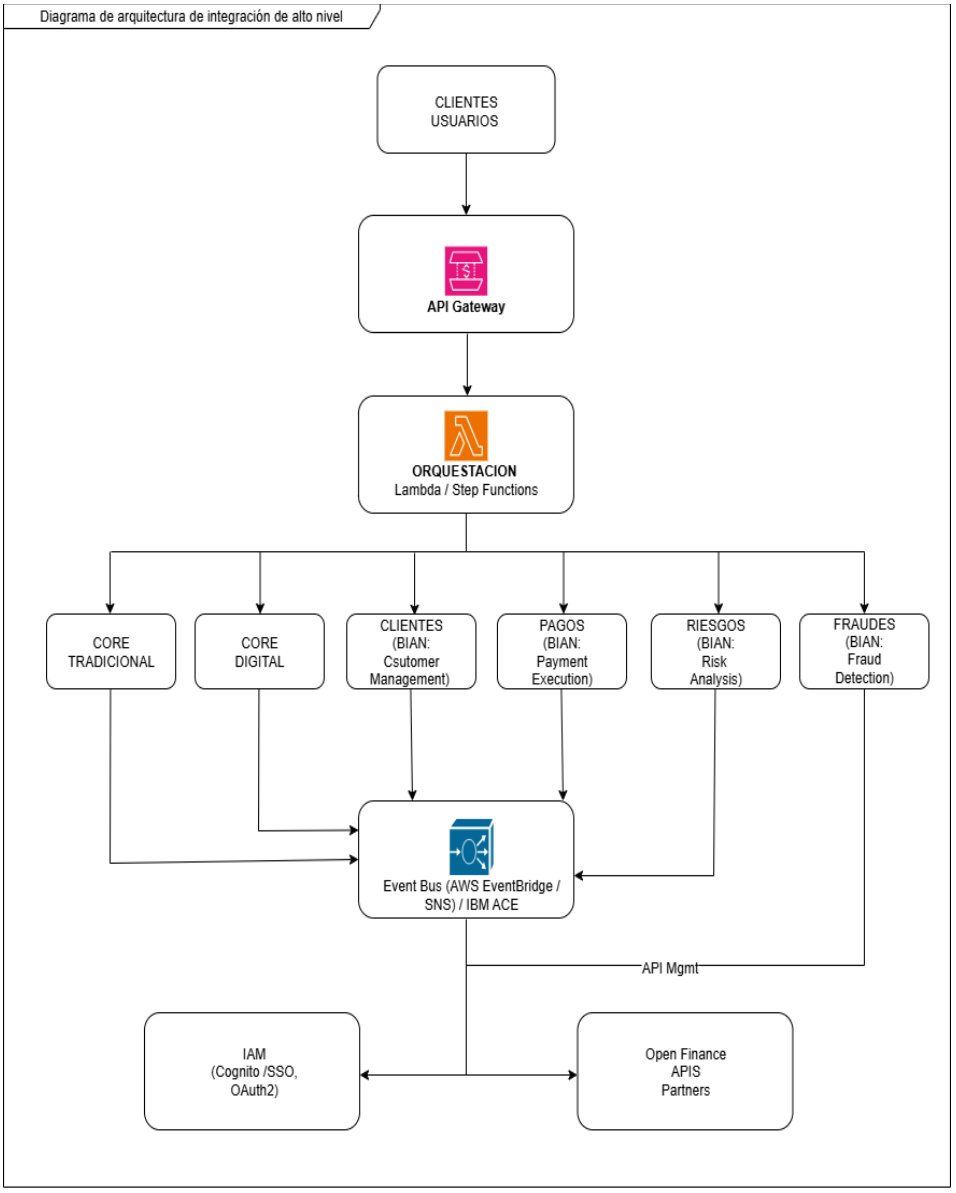
- Integrar el core bancario tradicional con un nuevo core digital, habilitando una transición gradual sin afectar la operación.
- Conectar canales digitales (banca web y móvil) a través de un API Gateway, garantizando seguridad,

monitoreo y control.

- Incorporar una plataforma de orquestación basada en flujos de negocio, permitiendo procesos complejos distribuidos entre varios sistemas.
- Se propone una arquitectura basada en *servicios bancarios desacoplados* (Service Domains). Implementar una plataforma de pagos moderna alineada con los dominios funcionales como Payment Execution, Customer Management, Loan Management, Fraud Detection, Risk Analysis del modelo BIAN.
- Integrar sistemas internos como gestión de riesgos y prevención de fraudes bajo un esquema basado en eventos y servicios desacoplados.
- Exponer y consumir APIs de terceros en cumplimiento con normativas de Open Finance.
- Garantizar la gestión de identidad y acceso centralizada mediante el uso de servicios como AWS Cognito y OAuth2.
- Aplicar principios de alta disponibilidad, recuperación ante desastres, monitoreo y trazabilidad, fundamentales en entornos bancarios.

El diseño se apoya en servicios de AWS como API Gateway, EventBridge, Lambda, Step Functions, Cognito, DynamoDB y KMS, entre otros, que ofrecen elasticidad, seguridad y bajo acoplamiento. Además, el uso del modelo BIAN como referencia funcional fortalece la alineación con estándares internacionales de arquitectura bancaria.

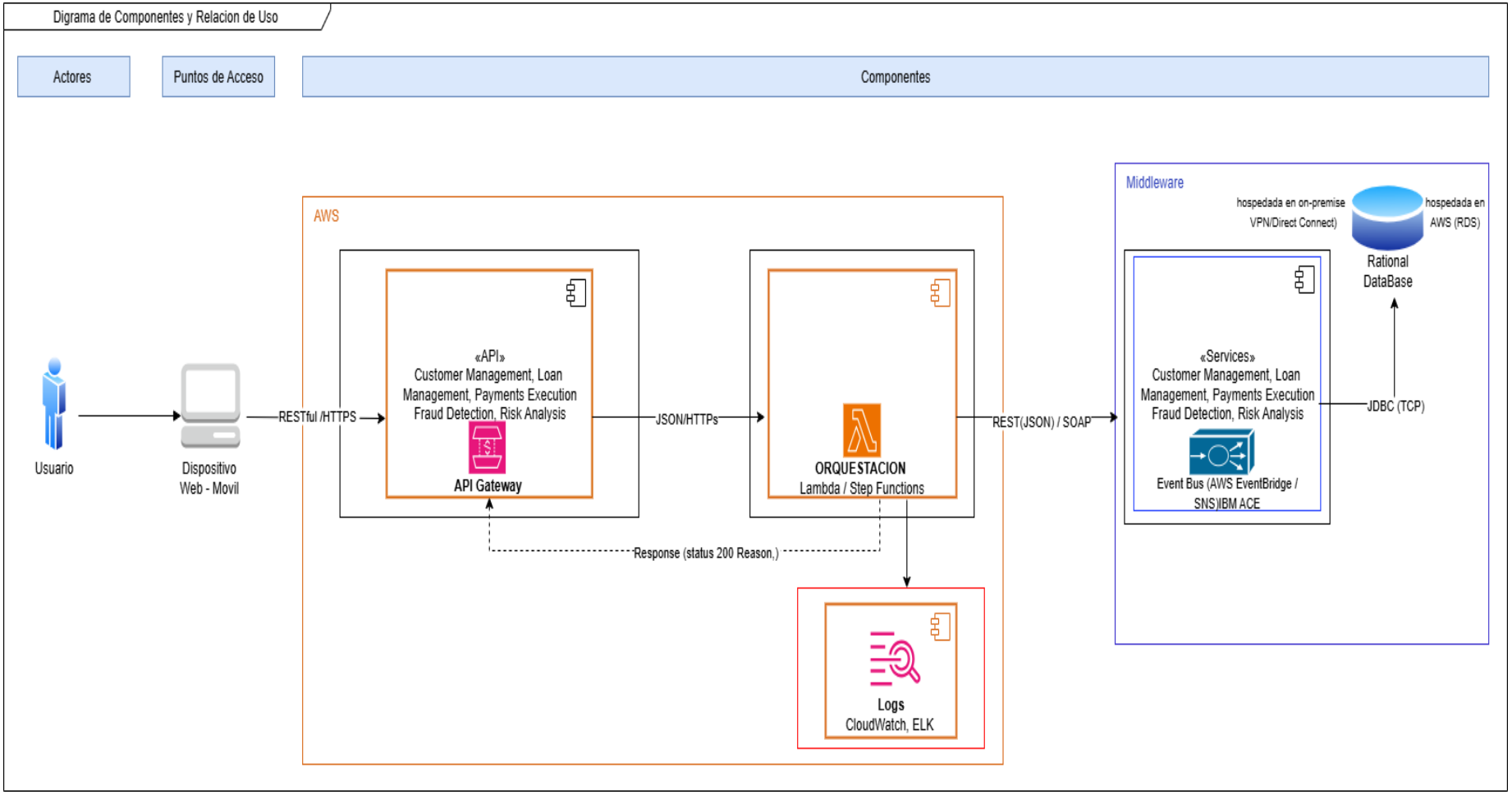
El flujo del proceso diseñado para el producto es:



2. Vista funcional de la solución

2.1. Patrones de Integración y Tecnologías Utilizadas

El siguiente diagrama presenta una vista estática de la solución en donde se muestran los componentes y sus relaciones de uso de acuerdo con los requisitos de negocio planteados para la solución


Digrama de Componentes y Relacion de Uso
Actores
Puntos de Acceso
Componentes
Usuario
Dispositivo
Web - Movil
RESTful /HTTPS
AWS
«API»
Customer Management, Loan
Management, Payments Execution
Fraud Detection, Risk Analysis
API Gateway
JSON/HTTPs
ORQUESTACION
Lambda / Step Functions
Response (status 200 Reason,)
Logs
CloudWatch, ELK
REST(JSON) / SOAP
Middleware
hospedada en on-premise
VPN/Direct Connect)
hospedada en
AWS (RDS)
Rational
DataBase
«Services»
Customer Management, Loan
Management, Payments Execution
Fraud Detection, Risk Analysis
Event Bus (AWS EventBridge /
SNS)IBM ACE
JDBC (TCP)

2.1.1. Descomposición funcional

Se especifican cada uno de los elementos involucrados en la solución teniendo en cuenta si son componentes o servicios, la función específica que realizará cada elemento con relación a la solución de negocio planteada.

A continuación, se describen los principales patrones de integración aplicados en el diseño, junto con las tecnologías utilizadas para su implementación. Estos patrones permiten cumplir con principios de arquitectura moderna como el desacoplamiento, escalabilidad, resiliencia y trazabilidad, especialmente
críticos en entornos bancarios.

Nombre del componente, Servicio o Patrón de integración	Descripción Funciones Requeridas del Componente y/o Servicio	Aplicación en la Arquitectura	Tecnologías Usadas	Justificación
API Gateway / Facade	Expone una interfaz unificada a múltiples sistemas backend.	Punto de entrada para banca web, móvil, y terceros.	Amazon API Gateway	Permite exponer APIs REST de forma segura, escalable y con bajo acoplamiento.

				Admite monitoreo, throttling, y versionamiento
Orquestación de Servicios	Coordina múltiples servicios y define flujos de negocio. El backend puede ser una Lambda function o un contenedor (ECS/Fargate) que recibe la solicitud y actúa como puente hacia el middleware.	Flujo de una transferencia entre canales, core, pagos y fraudes.	AWS Step Functions + AWS Lambda	Facilita la creación de flujos sin acoplar los servicios entre sí. Define lógica centralizada, visualizable y auditada. El backend puede ser una Lambda function o un contenedor (ECS/Fargate) que recibe la solicitud y actúa como puente hacia el middleware.

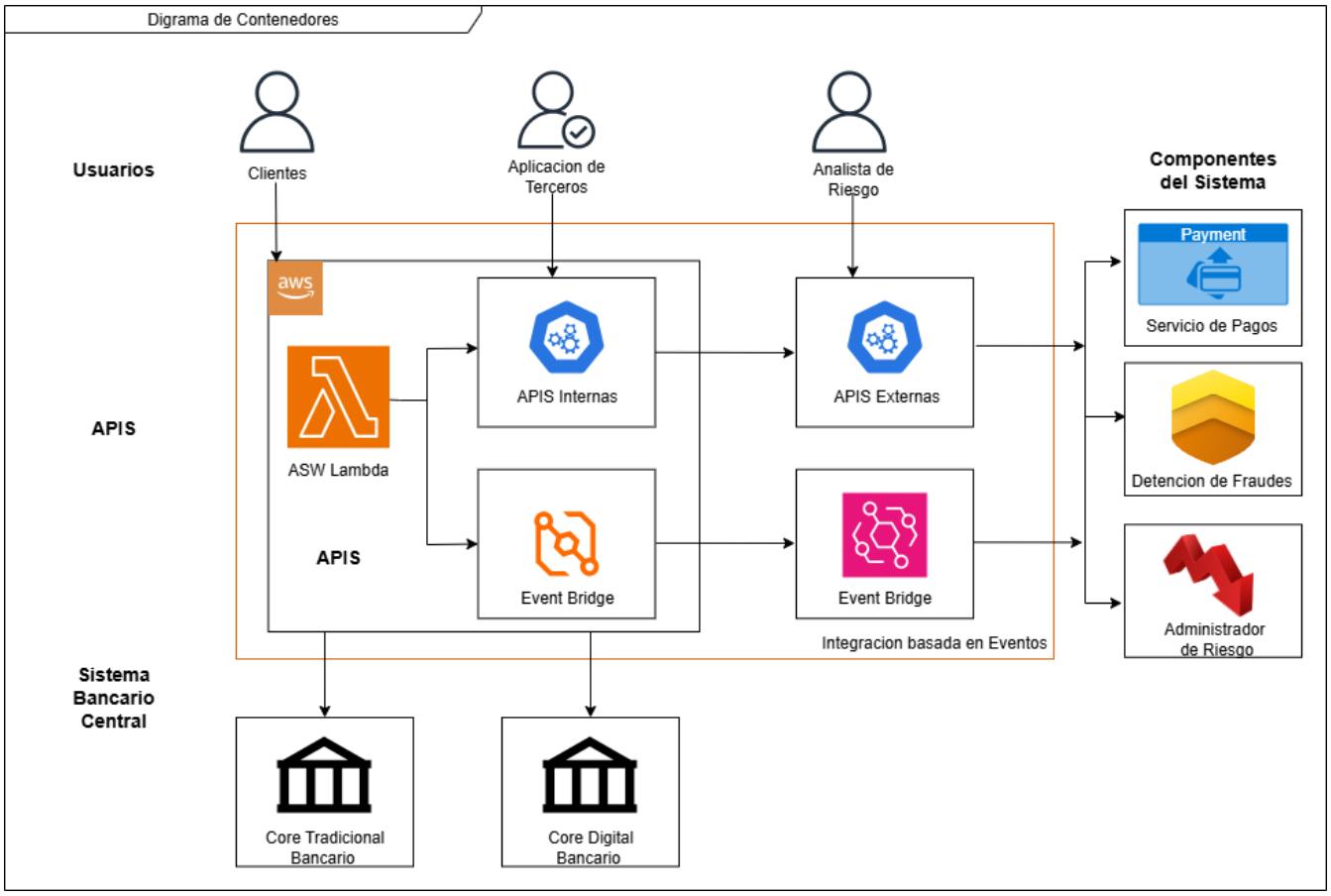
Mensajería Asíncrona (Event-Driven)	Intercambio de mensajes desacoplado mediante eventos.	Comunicación entre core digital, fraudes, pagos, riesgos.	Amazon EventBridge / SNS / SQS	Permite escalabilidad, resiliencia ante fallos y desacoplamiento temporal. Mejora trazabilidad de eventos e integración entre servicios.
IBM ACE	Funciona como middleware de integración (seguridad, transformación, enrutamiento).	El backend u orquestador se comunica con IBM ACE	IBM ACE	Funciona como middleware de integración (seguridad, transformación, enrutamiento).
Service Mesh / Microservicios	Divide funciones bancarias en unidades autónomas.	Core digital segmentado por dominios funcionales según BIAN.	ECS Fargate + App Mesh / EK	Habilita escalabilidad horizontal, despliegues

				independientes y alta resiliencia. Facilita la adopción progresiva del modelo BIAN.
Anticorrupción Layer (ACL)	Traduce y adapta interfaces entre sistemas legacy y modernos.	Conexión entre el Core Tradicional y el nuevo Core Digital.	Lambda + API Gateway / Glue / Mappings	Permite desacoplar el core tradicional mediante adaptadores y evitar impactos en sistemas nuevos por las limitaciones del legado.
Externalización de Identidad	Delegación de autenticación/autorización a un sistema externo.	Autenticación de usuarios en banca web/móvil y terceros.	Amazon Cognito + IAM + OAuth2	Simplifica el control de acceso con estándares seguros, federación de identidades y compatibilidad con

				OpenID, SAML y OAuth2.
Event Sourcing (opcional)	Persistencia de eventos como fuente de verdad.	Transacciones financieras, conciliaciones, cambios de estado.	DynamoDB + EventBridge + S3	Permite trazabilidad completa de operaciones sensibles y cumplimiento normativo. Ideal para auditoría financiera y reconstrucción de estados.
Rational DataBase	IBM ACE realiza una llamada a una base de datos relacional o no relacional (SQL Server o PostgreSQL, mongoDB), ya sea hospedada en AWS (RDS) o en on-	Desde IBM ACE hace la llamada a la base de datos.	SQL Server o PostgreSQL, mongoDB	Para trazabilidad y auditoría de nuestras peticiones y respuestas.

	premise (con VPN/Direct Connect).			
Logs CloudWatch, ELK	Todas las transacciones se registran en logs estructurados para trazabilidad y auditoría (CloudWatch, ELK, etc.).	Desde el componente de orquestación se envían los logs	Logs CloudWatch, ELK	Para trazabilidad y auditoría de nuestras peticiones y respuestas.

2.1.2. Diagrama de Contenedores



3. Requisitos de Seguridad, Cumplimiento Normativo y Protección de Datos

3.1. Principios de Seguridad Aplicados

La arquitectura propuesta se basa en los principios de seguridad de la información:

- **Confidencialidad**: Protección del acceso no autorizado a los datos. Que la información sea accesible únicamente por personal autorizado
- **Integridad**: Prevención de modificaciones no autorizadas. Que la información que se transporte y almacene en el sistema sea válida y consistente

- **No repudio**: Que las acciones o eventos que se realicen en un sistema que ya se hayan completados no sean invalidados o negados posteriormente
- **Identificación y autenticación**: Hace referencia a que el sistema pueda diferenciar quien está teniendo interacción con él (identificación) y que este pueda confirmar que esta identificación es válida o autentica (autenticación).
-
- **Autorización**: Hace referencia a poder verificar que quien esté interactuando con el sistema tenga los permisos suficientes para realizar dicha interacción, ya sea acceder a un dato, una funcionalidad o un servicio.
- **Disponibilidad**: Acceso garantizado a los sistemas y datos cuando se requieran.
- **Trazabilidad**: Registro completo y auditable de las operaciones.
- **Seguridad por diseño**: Aplicación de medidas de protección desde la fase de diseño

3.2. Cumplimiento Normativo y Estándares

La solución está alineada con:

- **Ley 1581 de 2012 (Colombia):** Ley de protección de datos personales (similar a GDPR).
- **GDPR (si aplica):** Protección de datos personales en la Unión Europea.
- **PCI DSS:** Para el tratamiento seguro de datos de tarjetas en la plataforma de pagos.
- **ISO 27001:** Prácticas de gestión de seguridad de la información.
- **Open Finance / Open Banking:** Acceso regulado y seguro a información financiera.

3.2.1. Protección de Datos Personales

La arquitectura implementa controles específicos para asegurar la privacidad de los datos personales:

- **Cifrado en tránsito y en reposo:**
- TLS 1.2+ para las comunicaciones
- AWS KMS para cifrado en servicios como S3, RDS, DynamoDB
- **Gestión de consentimiento:**
- Registro de consentimiento explícito del titular de datos para uso y compartición.
- **Principio de minimización de datos:**
- Solo se procesan los datos necesarios para cada operación.
- **Anonimización o seudonimización:**
- En registros para pruebas, analítica o respaldo.

3.2.1.1. Seguridad en Servicios de Integración

La seguridad definida para éste proyecto tiene en cuenta dos escenarios. Un primer escenario es cuando los componentes se encuentran dentro de la intranet de la entidad; y el segundo escenario se da cuando los componentes a interconectar se encuentran por fuera de la intranet.

- **Escenario 1 - Componentes dentro de Intranet:** Los componentes que participan en la integración se comunican requieren de la habilitación de cifrado en el canal de comunicación TLS cuando el protocolo lo soporte, los protocolos soportados para comunicación a través de canal de comunicación seguro son: Web Services (SOAP y REST).

- **Escenario 2 - Componentes fuera de Intranet:** Para interconectar componentes que se encuentren por fuera de la intranet de la entidad (clientes externos, proveedores externos) el componente encargado de gestionar credenciales de seguridad entre ellos y el banco es el bus de servicios, no necesariamente se hacen a través de canal seguro.

Para escenarios de integración con Web Services se habilita el uso del estándar WS-Security para garantizar la integración, tanto desde clientes hacia el bus de servicios, como desde el bus de servicios hasta proveedores. En este escenario no importa si los clientes y proveedores se encuentran fuera o dentro de la intranet.

Para tener mayor claridad sobre la implementación de cifrado de comunicación TLS y WS-Security y que se necesita para su implementación se describen a continuación algunos aspectos a tener en cuenta.

3.2.1.2. Cifrado del Canal de Comunicación

El cifrado es el proceso mediante el cual un mensaje que es legible para el ser humano (texto plano) en convertido en formato que no es legible, es decir, texto cifrado. La razón principal de realizar cifrado es la de asegurar que únicamente el receptor autorizado sea quien pueda descifrar y leer el texto original, de esa manera se busca evitar que un tercero intercepte los mensajes y pueda leer la información que se intercambia.

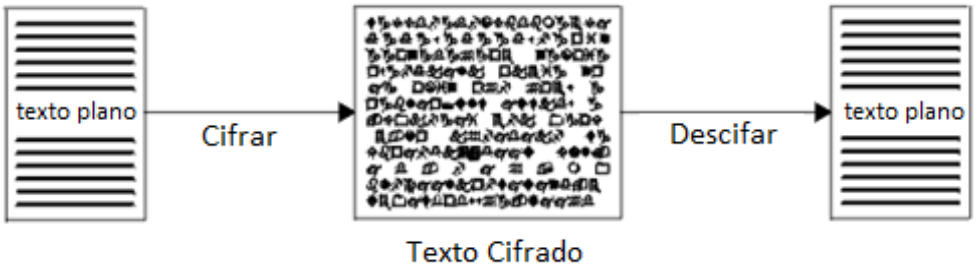


Figura 1. Ejemplo Proceso de Cifrado y Descifrado

Existen dos tipos de cifrado: Simétrico y Asimétrico

- **Cifrado Simétrico:** El cifrado simétrico es el proceso en el cual la misma llave es utilizada para cifrar y descifrar la información.

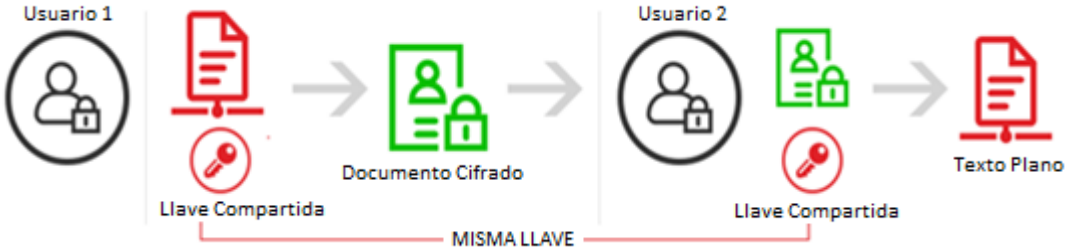


Figura 2. Cifrado Simétrico

Para la implementación de este tipo de cifrado es necesario emplear la misma llave para cifrar y descifrar el mensaje, lo que significa que todas las partes involucradas deben manejar una llave compartida, lo que conlleva a aumentar la posibilidad que un atacante obtenga la llave y pueda descifrar o cifrar información. Adicionalmente, el número de llaves a compartir hace que la gestión de éstas pueda llegar a ser muy difícil cuando el número de participantes aumente, ya que el número de llaves que se intercambiarían viene dado por la siguiente función.

Número de llaves= N*(N-1)/2

- **Cifrado Asimétrico:** El cifrado asimétrico también conocido como criptografía de llave pública emplea un par de llaves (llave pública y privada). Estas llaves criptográficas están relacionadas entre sí de tal manera que un mensaje es cifrado con una llave y descifrado con la otra. Las llaves públicas se comparten mientras que las llaves privadas sólo deben ser conocidas por su propietario o servidor.

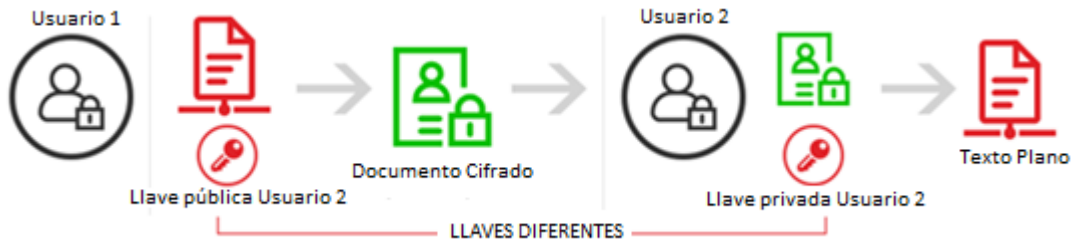


Figura 3. Cifrado Asimétrico

En cuanto al cifrado del canal de comunicación encontramos que existe la posibilidad de habilitar en ACE mediante TLS (v1.2), estos dos mecanismos emplean el modelo de cifrado asimétrico. TLS corresponde a las siglas Transport Layer Security. El propósito principal del protocolo es proveer: privacidad e integridad, identificación y confidencialidad directa (PFS-Perfect Forward Secrecy).

- **Privacidad e integridad:** La comunicación entre dos puntos (Cliente-servidor) se cifra de tal manera que ningún tercero pueda leer o alterar la información que está siendo intercambiada.

- **Identificación:** Mediante el uso de criptografía de llave pública, TLS provee identificación entre los participantes en una comunicación. Esto significa que el Servidor y el cliente pueden verificar el certificado del cliente para validar su identidad contra una entidad certificadora.

- **Confidencialidad Directa (PFS):** Con PFS se busca asegurar que, si se presenta el evento en donde la llave privada de un servidor este siendo comprometida, un atacante no pueda descifrar comunicaciones previas realizadas sobre TLS. Cada sesión involucra nuevas llaves y únicamente son válidas mientras la sesión se encuentre activa.

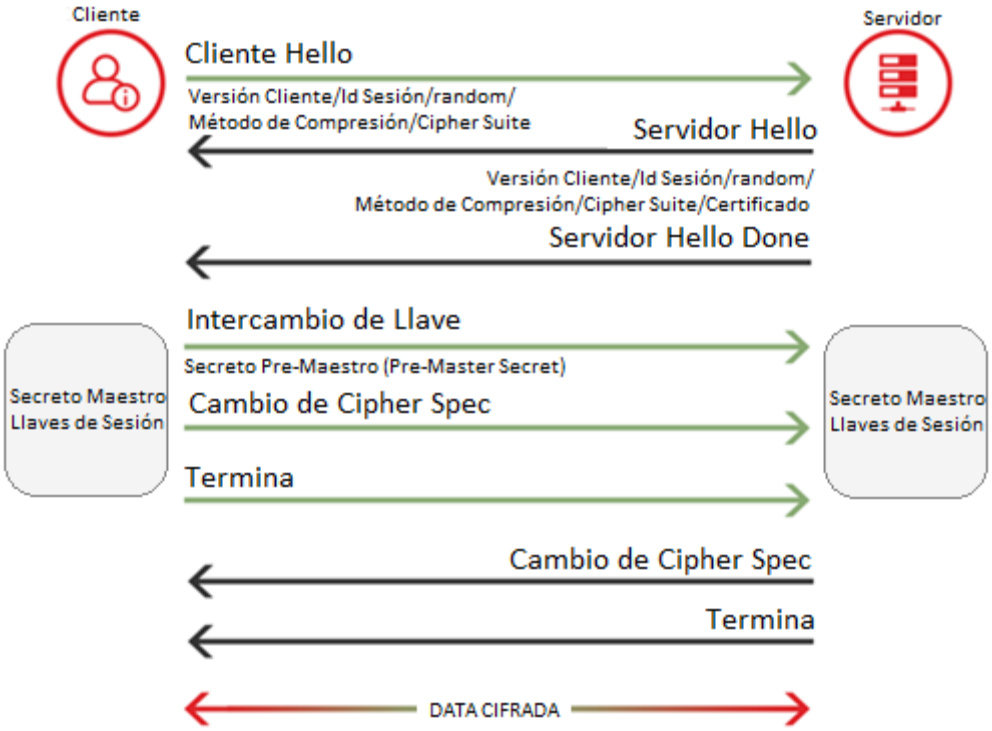


Figura 4. Fase de Negociación Creación Conexión Segura

3.2.2. Implementación en la Arquitectura

Implementación en la Arquitectura	
Requisito	Implementación en la Arquitectura
Autenticación segura	AWS Cognito: autenticación federada (SAML, OAuth2, OpenID) y gestión de sesiones.
Autorización granular	Políticas IAM para recursos AWS, y control RBAC en microservicios.
Cifrado de datos	AWS KMS + TLS + cifrado en S3, RDS y DynamoDB.
Registro y auditoría	AWS CloudTrail y CloudWatch para trazabilidad de accesos, eventos y errores.
Prevención de ataques	AWS WAF + Shield para protección ante amenazas web y DDoS.
Gestión de vulnerabilidades	Integración de herramientas de análisis estático y dinámico (por ejemplo, CodeGuru, Inspector).
Políticas de retención	Uso de lifecycle policies en almacenamiento y archivado seguro de logs.

4. Estrategia de Alta Disponibilidad y Recuperación ante Desastres

Proveer mecanismos de alta disponibilidad a nivel de capas de ejecución o servidores de integración en los cuales están desplegados los servicios. A nivel de infraestructura se dispone de una solución **Activo/StandBy** en un ambiente Productivo para los servidores de integración, esquemas de base de datos uno para auditoria, registro de errores, homologaciones y catálogo de servicios.

De otro lado, la construcción de los servicios se realiza bajo un modelo genérico, con el fin de permitir el acceso de forma estándar, reduciendo la complejidad de integración entre sistemas heterogéneos y logrando un alto grado de reutilización de los servicios tecnológicos existentes y de los nuevos servicios que serán creados.

4.1.1. Objetivo

Garantizar que los servicios bancarios estén siempre disponibles, incluso ante fallos regionales, errores humanos o desastres naturales, con mínimo impacto para los usuarios y cumplimiento de niveles de servicio (SLA).

4.1.2. Alta Disponibilidad Aplicada

Alta Disponibilidad Aplicada	
Componente	Estrategia de Alta Disponibilidad
API Gateway	Desplegado globalmente en múltiples zonas de disponibilidad (AZ) y regiones. Balanceo automático de carga.
Microservicios (ECS / Lambda)	Despliegue en múltiples AZs. ECS con autoescalado y health checks. Lambdas distribuidas con retry y fallback automático.
EventBridge / SNS / SQS	Servicios totalmente gestionados y tolerantes a fallos. Garantizan entrega duradera y desacoplada.
BBDD (DynamoDB, Aurora)	DynamoDB con replicación multi-AZ. Aurora con clústeres y réplicas automáticas.
Cognito e IAM	Alta disponibilidad inherente en la infraestructura de AWS.
CloudFront (si aplica)	CDN distribuida para minimizar latencia en canales web.

4.1.3. Recuperación Ante Desastres Aplicadas

Recuperación ante Desastres Aplicada	
Componente	Estrategia de Recuperación ante Desastres
Bases de Datos (Aurora / DynamoDB)	Backups automáticos diarios, replicación cruzada de región (Multi-Region).
APIs / Microservicios	IaC (Infraestructura como Código) para redeploy rápido (CloudFormation / Terraform).
Logs y auditoría	Centralizados en S3, con replicación en buckets cross-region y retención.
Eventos y mensajes	Almacenamiento persistente en SQS/EventBridge, con DLQ (Dead Letter Queue).
Plan de continuidad	Playbooks automatizados con AWS Systems Manager y monitoreo con CloudWatch + Alertas.

4.1.4. Ejemplo de Recuperación

En caso de caída de una región entera de AWS, el sistema puede ser reestablecido en una región secundaria mediante la infraestructura definida como código (IaC) y datos replicados (bases de datos, S3, colas). El tráfico se redireccionaría a través de DNS (Route53) y los servicios estarían disponibles en minutos.

4.1.5. Tabla resumen de HA/DR por servicio

Resumen de HA/DR por servicio		
Servicio	Alta Disponibilidad	Recuperación ante Desastre
API Gateway	Multi-AZ, balanceo y escalado automático	DNS failover con Route53
ECS (Microservicios)	Autoescalado, múltiples instancias	Redefinición con IaC
DynamoDB / Aurora	Multi-AZ + réplicas	Snapshots automáticos y replicación cross-region
EventBridge / SNS / SQS	Gestionados y tolerantes a fallos	DLQ + replay de eventos

S3 (Logs / Documentos)	Alta durabilidad y replicación regional	Versión + ciclo de vida + cifrado
Cognito / IAM	Tolerancia a fallos regional	Sincronización de identidades

5. Estrategia de Integración Multicore

La estrategia de integración multicore permitira que el banco Interopere con más de un sistema core bancario (legacy y/o moderno), habilitar transiciones graduales, evitando un “big bang”, exponer funcionalidades de ambos cores bajo una única interfaz (API) y aíslar al canal y al usuario final del backend real.

En la arquitectura de integración sobre AWS puede reflejar integración multicore de la siguiente manera:

5.1.1. Uso de API Gateway como punto único de entrada

Todos los canales digitales y externos consumiran las APIs sin saber qué core está detrás. Por ejemplo

POST /clientes -> puede consultar Core Legacy
POST /clientes (v2) -> puede consultar Core Moderno

5.1.2. Capa de orquestación (Lambdas o ECS) basada en reglas o metadatos

Las funciones Lambda o microservicios pueden decidir en tiempo de ejecución a cuál core dirigir la petición, con base en:

- Tipo de producto
- Segmento del cliente
- Geografía
- Estado de migración del cliente (ej. migrado o no)

Se pueden realizar reglas de negocio de la siguiente manera.

Ejemplo:

```
if (cliente.isMigrado()) {
```

```
    return coreModern.getCuenta(cliente.id);
} else {
    return coreLegacy.getCuenta(cliente.id);
}
```

5.1.3. Adopción de eventos desacoplados (EventBridge, SNS, SQS)

- Permite replicar transacciones o sincronizar datos entre cores de manera eventual.
- Se pueden tener listeners distintos por core.
- Ideal para coexistencia y migración progresiva.

5.1.4. Modelo BIAN como contrato común

Para identificar el dominio de BIAN al que pertenece una API se debe identificar la funcionalidad que va a exponer y compararla con las descripciones funcionales de los dominios que se definen en BIAN.

En la página oficial BIAN se tienen dos herramientas muy útiles para la identificación del dominio, la primera es el poster con la representación gráfica de la jerarquía de los dominios y la otra es el repositorio digital donde se encuentran las descripciones de estos.

En la página oficial, en el menú

- Deliverables -> Standards
    - BIAN Service Landscape
        - **BIAN SERVICE LANDSCAPE 7.0** (print/poster format)
        - **DIGITAL REPOSITORY**

El poster muestra de forma gráfica la estructura de los diferentes dominios que tiene BIAN, y el repositorio digital contiene las descripciones y ejemplos de uso de cada uno de esos.
Se puede encontrar la última versión del “Service Landscape” en la sección de estándares[1] del BIAN.

La versión utilizada para la creación de este documento es “BIAN Service Landscape 7.0” Poster/pdf[2] y Repositorio Digital

Poster:

BIAN Standards https://www.bian.org/deliverables/bian-standards/

Poster/pdf http://bian.org/wp-content/uploads/2018/11/BIAN_Service_LandscapeV7_0.pdf
Digital Repository https://bian.org/servicelandscape/

El uso del poster permite realizar una búsqueda rápida de un dominio relacionando su término en inglés. (Ctrl+F y poniendo el termino en inglés se resaltan los términos encontrados)
Ejemplo, si estuviéramos buscando el dominio para una funcionalidad relacionada al crédito hipotecario, inicialmente buscaríamos las traducciones de los 2 términos, para este caso:

Hipotecario, Hipoteca -> Mortgage
Crédito -> Credit, Loan

En caso de no encontrar el término específico se pueden utilizar sinónimos en ambas direcciones, es decir, sinónimos de la palabra en inglés o en español. Una página útil para realizar la búsqueda de los términos y sus sinónimos en inglés es Linguee[3], que permite realizar traducción de términos y traducción de frases, y además presenta casos de ejemplo.
Hay que tener claras las definiciones de los términos en inglés ya que puede ocurrir que la traducción literal del término se use para un contexto diferente al deseado para el servicio.

Por ejemplo:

- Consulta -> Consultation, Inquiry, Enquiry, Query
- Consultar -> Consult, Inquire, Enquire, Query

Sustantivo	Verbo	Uso
Consultation	Consult	Se usa para referirse a procesos de consultoría o asesorías.

Inquiry	Inquire	Se usa generalmente para investigaciones formales, se define como el proceso mediante el cual se adquiere conocimiento, se da solución a problemas y dudas son resueltas.
Enquiry	Enquire	Significa lo mismo que “Inquiry” pero este se utiliza más en el reino unido.
Query	Query	Se usa para solicitar información, es el acto de realizar una pregunta, y se puede considerar como uno de los actos necesarios para realizar una investigación “Inquiry”

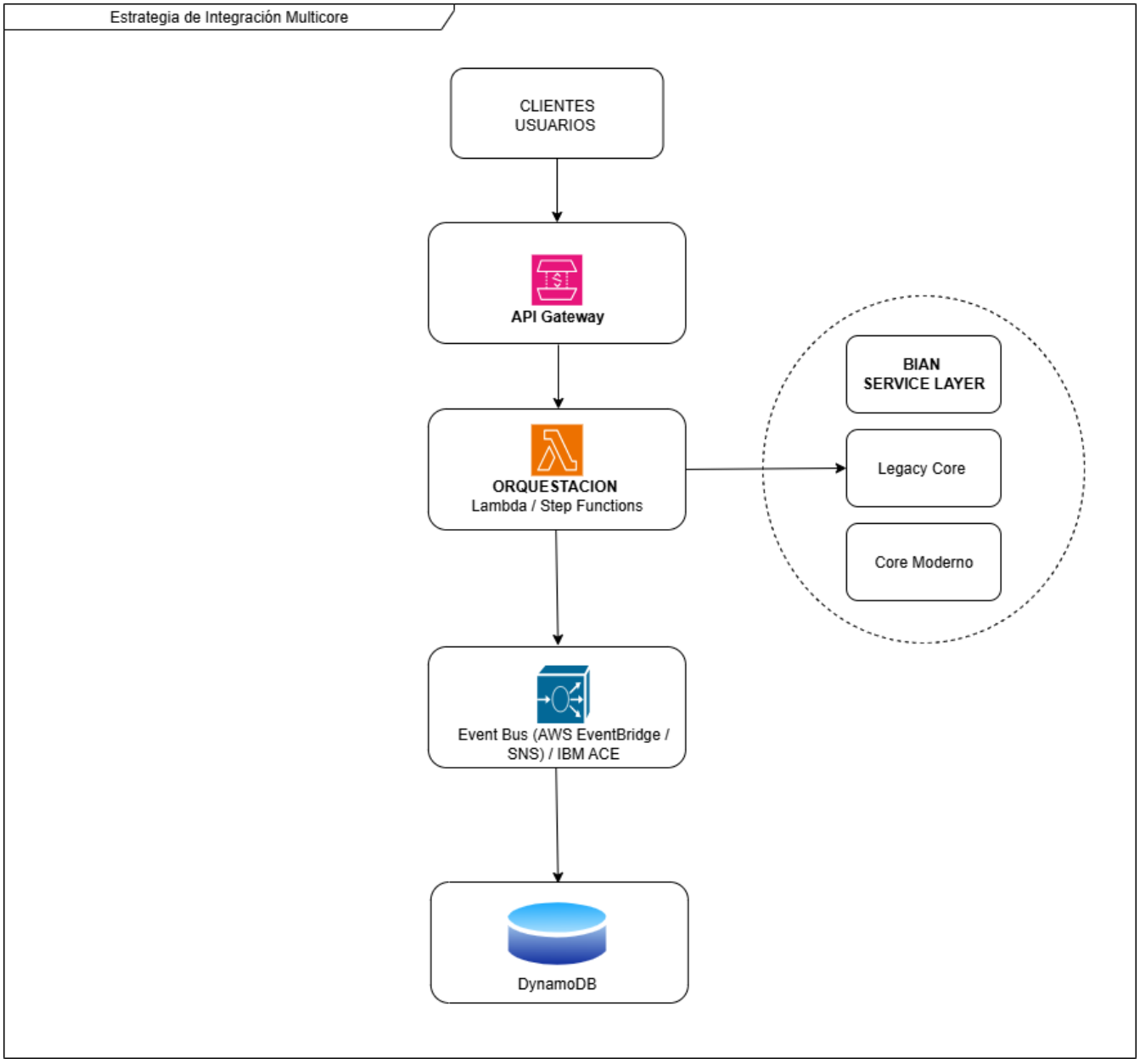
Con BIAN definiremos servicios estándar bancarios independientemente del core que los implemente:

- Customer Onboarding Service → se puede cumplir por el Core A o Core B.
- Los microservicios de integración implementan los BIAN Service Domains.
- El core queda oculto: se acopla al servicio BIAN, no a un canal.

5.1.5. Ejemplo de Componentes que reflejan integración multicore

Componentes que reflejan integración multicore	
Componente	Función en la estrategia
API Gateway	Exponer una única interfaz REST al cliente o canal
Lambda Orquestadora	Determinar en tiempo real a qué core llamar
EventBridge	Emitir eventos que ambos cores puedan consumir o replicar
DynamoDB / S3	Registro temporal de estados de migración
BIAN Service Layer	Contrato estándar que desacopla los canales del core real

5.1.6. Diagrama Estrategia de Integración Multicore



6. Gestión de Identidad y Acceso (IAM)

Teniendo en cuenta que la gestión de identidad y acceso es crítica en entornos bancarios, especialmente cuando se integran múltiples sistemas (core bancario tradicional, nuevo core digital, APIs, servicios móviles, etc.). proponemos garantizar **seguridad, cumplimiento normativo y control detallado de accesos**.

6.1.1. Principios Clave Aplicados

- **Principio de menor privilegio**: Los usuarios, servicios y sistemas solo tienen acceso a los recursos estrictamente necesarios.
- **Zero Trust Architecture**: Todo intento de acceso es verificado, independientemente del origen.
- **Segregación de responsabilidades (SoD)**: Las funciones críticas están distribuidas para evitar accesos indebidos.

6.1.2. Mecanismos de Autenticación

6.1.2.1. Usuarios o Clientes Finales.

- Autenticación multifactor (MFA) con OTP vía app o SMS.
- Integración con servicios como **Amazon Cognito** para manejar registro, inicio de sesión y tokens JWT.
- OAuth 2.0 / OpenID Connect como estándares de autenticación seguros para apps web y móviles.

6.1.2.2. Usuarios Internos y Administradores:

- Autenticación federada con proveedores de identidad corporativos (por ejemplo, AWS IAM Identity Center con Active Directory o Azure AD).
- MFA obligatoria para todos los accesos administrativos y de gestión.

6.1.2.3. Servicios entre sí (machine-to-machine):

- Uso de tokens OAuth2 con scopes definidos.
- Mutual TLS (mTLS) para autenticación fuerte entre servicios internos (ej., microservicios y APIs).

6.1.3. Mecanismos de Autenticación

- **Control de acceso basado en roles (RBAC)** para usuarios y servicios.
- **Control de acceso basado en atributos (ABAC)** en políticas de AWS IAM, permitiendo mayor granularidad.

- **Scopes y Claims** en tokens JWT para limitar accesos a recursos específicos por servicio.

6.1.4. Gestión Centralizada de Identidades y Accesos

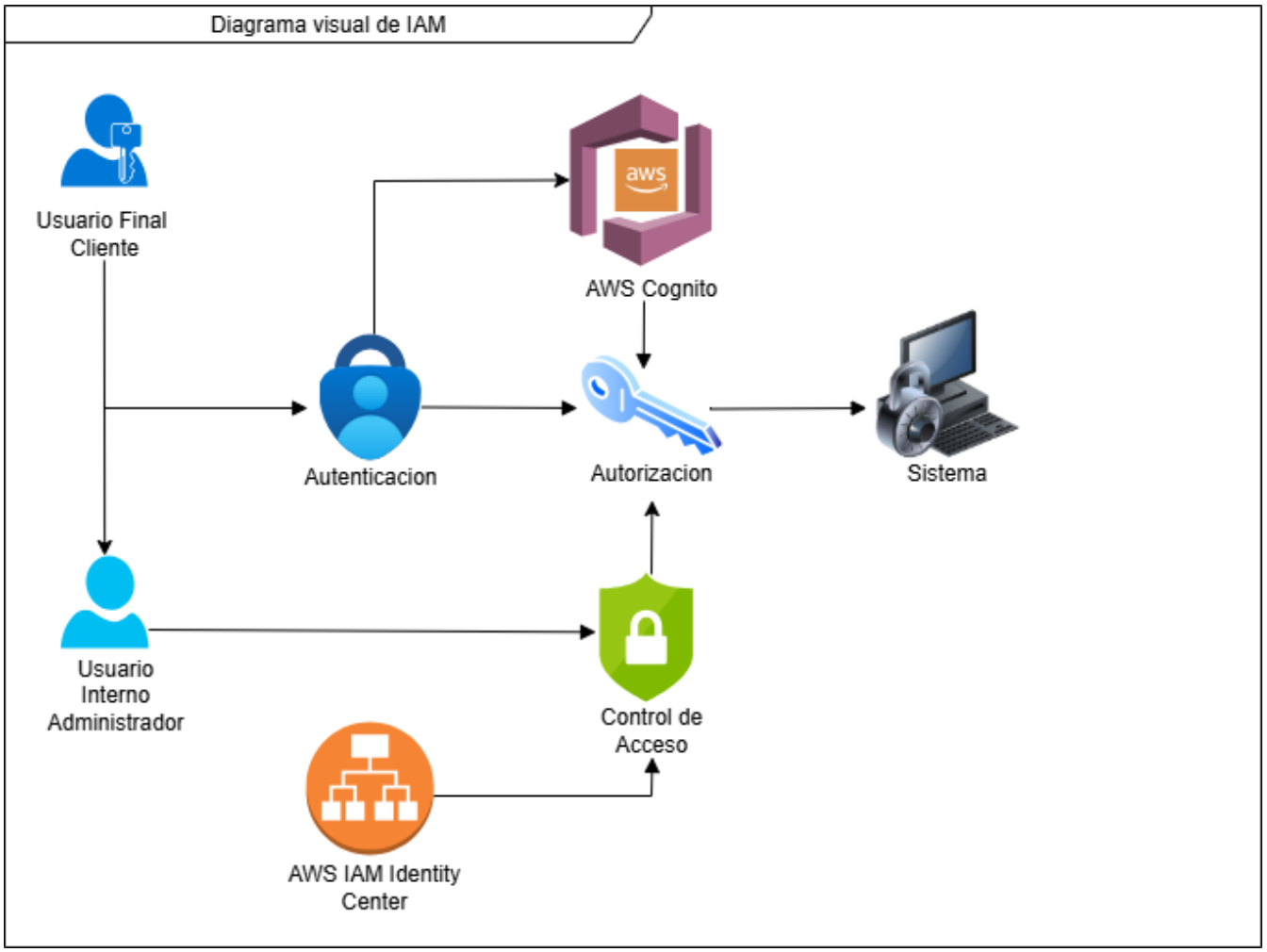
- **Amazon Cognito** gestiona identidades externas (clientes).
- **IAM Identity Center (anteriormente AWS SSO)** para empleados e integraciones con directorios corporativos.
- **Secrets Manager y Parameter Store** para manejar contraseñas, tokens y claves API de forma segura y centralizada.

6.1.5. Auditoría y Monitoreo de Accesos

- **AWS CloudTrail** habilitado para registrar y auditar todas las acciones de IAM, accesos a recursos, cambios en roles y políticas.
- **Amazon GuardDuty** y **AWS Security Hub** para detectar accesos inusuales o potencialmente maliciosos.
- Generación de alertas ante accesos fallidos, cambios en credenciales o uso de tokens expirados.

6.1.6. Aplicación en el Diseño de Integración

Aplicación en el Diseño de Integración	
Componente	IAM Aplicado
Core Bancario Digital	Mecanismos de mTLS y JWT para consumo seguro de APIs
APIs Externas (Open Finance)	OAuth2.0 con scopes personalizados, protección por API Gateway
Aplicaciones móviles	Cognito con MFA, token refresh seguro
Backend de Microservicios	IAM Roles, políticas ABAC, autenticación federada
Administración y monitoreo	IAM Identity Center con MFA y acceso granular



7. Estrategia de APIs Internas y Externas para Integración Bancaria

Con esta esta estrategia de APIS busca facilitar la interoperabilidad segura, escalable y reutilizable entre los sistemas del banco (cores, canales digitales, prevención de fraudes, etc.) y con terceros (Open Finance, fintechs, proveedores, etc.) mediante una **arquitectura de APIs robusta y moderna**, soportada en tecnologías cloud-native como **Amazon API Gateway**, **AWS App Mesh**, **Amazon EventBridge** y **mensajería asíncrona basada en eventos**.

7.1.1. Clasificación de APIs

Se tiene una clasificación de las APIs basada en un modelo de industria, categorizándolas como APIs Internas, Externas y Privadas. Esta categorización se realiza por tipo de consumidor y es importante de cara a la seguridad porque permite tener los controles suficientes dependiendo del público objetivo al que se le exponga la información.

Clasificación de APIS			
Tipo de API	**Descripción**	**Audiencia**	**Ejemplo**
Internas	Para comunicación entre microservicios internos, canales digitales y el core bancario	Equipos internos, microservicios	/api/v1/core-clientes/consultaSaldo
Externas	Para integrarse con terceros, plataformas Open Finance y aliados estratégicos	Fintechs, apps aliadas, clientes	/openbanking/v1/cuentas
Privadas	Para el uso de un grupo restringido o determinado y con extremada seguridad	Equipos restringidos	/api/v1/privates/consultaSaldo

7.1.1.1. Clasificación de los consumidores

Los consumidores de APIs se clasifican dependiendo de la interacción con el consumidor. El consumidor puede ser:

- Un aplicativo que tiene interacción directa con **usuarios externos (No clientes)** al Banco. **Ejemplo:** Aplicativo “Home Banco” (https://www.banco.com/) que permite conocer a cualquier tipo de usuario la información de servicios y productos que ofrece la organización

- Un aplicativo que tiene interacción directa con **clientes** del Banco. **Ejemplo:** Aplicativo “Sucursal Virtual Personas” que permite a los clientes consultar su información y sus

productos (Datos de contacto, productos que posee, saldos de productos, movimientos, etc.)

- Un aplicativo que tiene interacción directa con **empleados** del Banco. **Ejemplo:** Aplicativo “Conectados” que permite a los colaboradores del Banco ver información de cursos de aprendizaje, desempeño, vacantes de la organización, etc.

- Un aplicativo que tiene interacción directa con **componentes de la capa de mediación (APIs gestionadas, integración empresarial o transaccional)** del Banco. **Ejemplo:** Cuando el “Core bancario Depósitos” quiere consumir una funcionalidad expuesta mediante API, por restricciones técnicas y de seguridad, requiere una capa intermediaria, como lo es la capa de integración empresarial o transaccional para hacerlo.

7.1.1.2. Clasificación de la informacion

Podemos utilizr tres criterios por los que se clasifica la información en la organización: la confidencialidad, la Integridad y la disponibilidad. Estos tres criterios son muy importantes para los lineamientos de seguridad de las APIs Gestionadas, específicamente por las clasificaciones de la confidencialidad de la información y por el nivel de impacto en términos de la integridad y la disponibilidad.
Las clasificaciones de confidencialidad son:

- Pública
- Interna
- Confidencial
- Restringida

Los tipos de impacto en la integridad y disponibilidad de la información son:

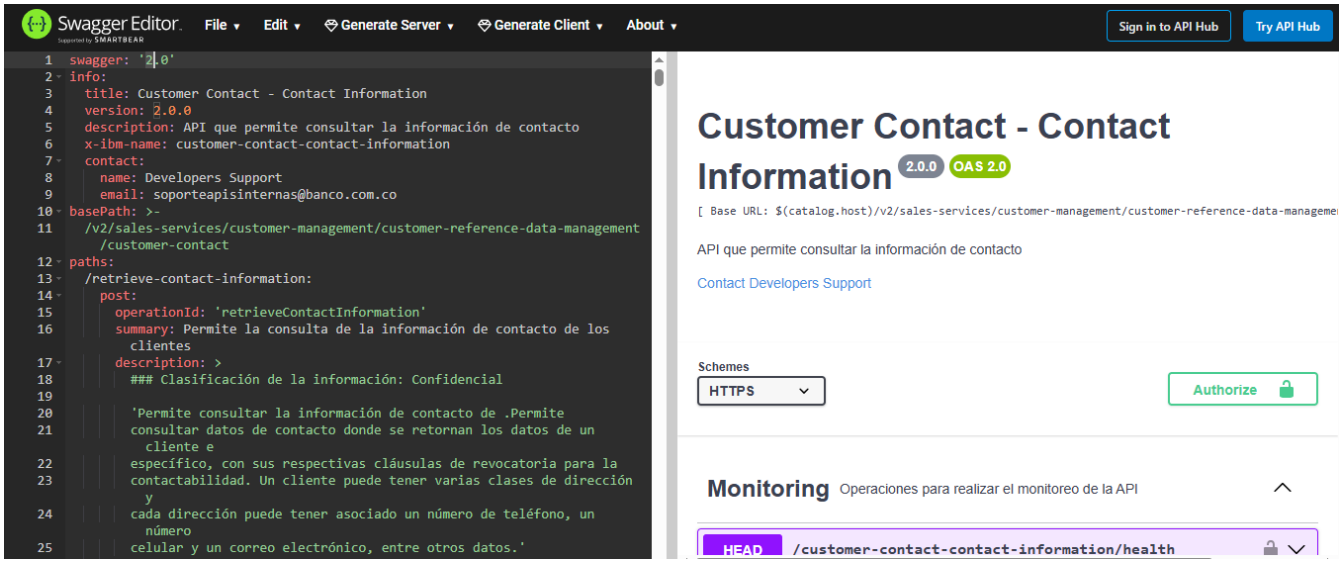
- Impacto crítico
- Impacto moderado
- Impacto tolerable
- Sin impacto

7.1.2. Estándares y Buenas Prácticas Aplicadas

- **Diseño basado en OpenAPI 3.0 / Swagger**
  Documentación clara y contratos de servicios definidos.
- **RESTful APIs para operaciones CRUD**
  Estandarización de verbos HTTP (GET, POST, PUT, DELETE).

- **GraphQL en canales digitales (opcional)**
  Optimiza las consultas para apps móviles/web.
- **Versionamiento de APIs**
  Por ejemplo: /api/v1/, /api/v2/.
- **HATEOAS (Hypermedia As The Engine Of Application State)**
  APIs que autoexplican las acciones disponibles.
- **Pagos, notificaciones o eventos críticos**
  Usan mensajería basada en eventos con Amazon EventBridge o Kafka.

Ejemplo:



7.1.3. Mensajería: Estándares y Aplicación

7.1.3.1. Eventos (Event-Driven Architecture - EDA)

- Utilizados para procesos como:
  - Transacciones detectadas por el motor antifraude.
  - Confirmaciones de pagos entre el core y la pasarela de pagos.
- Tecnología:
  - **Amazon EventBridge** o **Apache Kafka** para el bus de eventos.
  - Mensajes en formato **JSON** o **Avro**.
  - Enrutamiento basado en temas/topics.

7.1.3.2. Mensajería Asíncrona

- Para desacoplar los sistemas:
  - Core - Prevención de fraudes.

    - Canales digitales - Sistema de riesgos.
- Uso de colas:
    - **Amazon SQS**, **RabbitMQ, KafKa**.
- Patrones aplicados: **Pub/Sub**, **Queue-based Load Leveling**.

7.1.4. Seguridad de las APIs

Existe diferentes escenarios de seguridad de las APIs; estos escenarios se construyen teniendo presentes los diferentes criterios de clasificación para la definición de la seguridad de las APIs (Tipo de API, Tipo de Consumidor, Clasificación de la Información).

- **Autorización**: Scopes por tipo de consumidor y rol
- **Authorization Code (Código de Autorización):** Es usado en aplicaciones que tienen interacción directa con el cliente, donde se utiliza un código de autorización que se obtiene cuando un cliente se autentica contra el servidor de autorización; este código de autorización se utiliza para obtener el Token de consumo (Access Token).
- **Implicit (Implícito):** Es usado en aplicaciones de una sola página (Single-Page Applications)
- **Resource Owner Password Credentials (Flujo Password**): Es usado cuando la aplicación consumidora es de confianza, debido que, para ejecutar este flujo, se envían las credenciales del usuario a través de una de la aplicación consumidora, permitiendo a la misma, conocer las credenciales del usuario.
- **Client Credentials (Flujo de Aplicación):** Es usado cuando la aplicación consumidora no tiene interacción directa con el cliente y es una aplicación de confianza, debido que, para ejecutar este flujo, no se requiere de la autorización del usuario para acceder a su información. Los únicos datos que se validan en el servidor de OAuth son las credenciales de la aplicación consumidora.
- **Autenticación**: OAuth 2.0 + JWT
- **Autenticación básica:** Este método de autenticación se basa en un nombre de usuario y una contraseña para identificarte.
- **API Key (ClientID):** Este método de autenticación se basa en un identificador único de consumidor. Este identificador único es usado para autenticar las peticiones asociadas a un consumidor específico para las APIs, donde el API Key nos permite autorizarlo, revisar la trazabilidad, conocer el consumo y tomar decisiones al respecto**.**
- **API key y Secret**: Este método de autenticación se basa en el API Key (ClientID) más la utilización de un secreto que se envía en la petición que sólo se conoce entre el consumidor específico (Cliente) y el servidor (Servidor que expone la API).
- **JWT/JWS (JSON Web Token – JSON Web Signature):** Este método de autenticación se basa en la utilización de un Token con una validación de firma digital (JWS). JSON Web Token (JWT) es un estándar abierto RFC 7519 que define una

manera compacta y autocontenida para transmitir de forma segura información entre dos partes usando un objeto JSON.

- **mTLS (Mutual TLS Authentication):** Este método de autenticación se basa en el intercambio de certificados digitales de parte del aplicativo consumidor (Cliente) y el servidor (Servidor que expone la API). Con mTLS, el aplicativo consumidor y el servidor presentan un certificado durante un protocolo de enlace TLS que demuestra la identidad de forma mutua
- **Throttling** y **Rate Limiting**: Evita abusos y ataques
- **API Gateway (Amazon)**: Proxy seguro con WAF, logging y control de acceso
- **API Keys o Tokens firmados** para terceros

7.1.5. Gobierno y gestión de APIs

- **API Manager** (por ejemplo Amazon API Gateway + AWS Service Catalog o Kong, API Connect IBM).

- **Ciclo de vida:**

    - Creación
    - Desarrollo
    - Certificación
    - Publicación en Producción
    - Obsolescencia/Deprecación
    - Jubilación y Retiro

- **Observabilidad**: CloudWatch, X-Ray para trazabilidad.

- **Catálogo y versionado centralizado.**

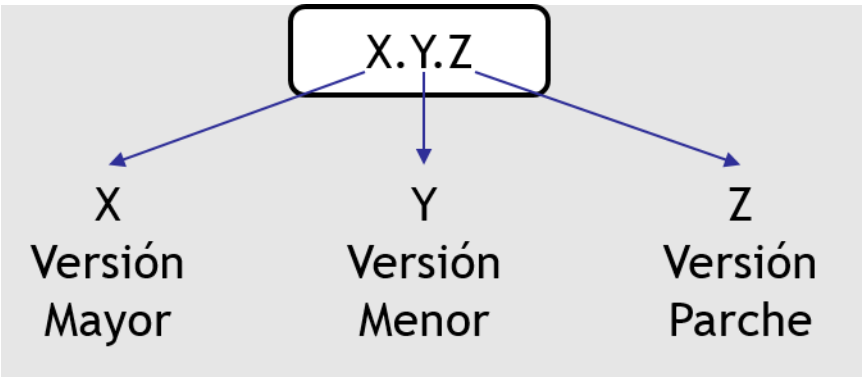
El **catálogo o gobierno** lo podemos crear para consulta de información que permite a los usuarios sean técnicos o funcionales, obtener información de los servicios y políticas más detalladas, con una interfaz amigable mejorando su usabilidad. Este puede estar en una base de datos, en un service registry (wsrr) etc.

El versionamiento dentro del ciclo de vida comprende una serie de etapas desde el momento de la identificación de la versión inicial, hasta su obsolescencia y retiro en la versión final, en donde la versión tiene cambios que pueden afectar su estructura.

Al actualizar la versión en ambiente de producción se puede afectar a los potenciales consumidores, debido a esto deben convivir diferentes versiones (máximo 2 [obsoleta,

Superior]), dándoles tiempo a los consumidores para adaptarse a la nueva versión.
Los números de versión y la forma en que cambian entregan significado del código y lo que fue modificado de una versión a otra.

Las versiones pueden clasificarse según su impacto:



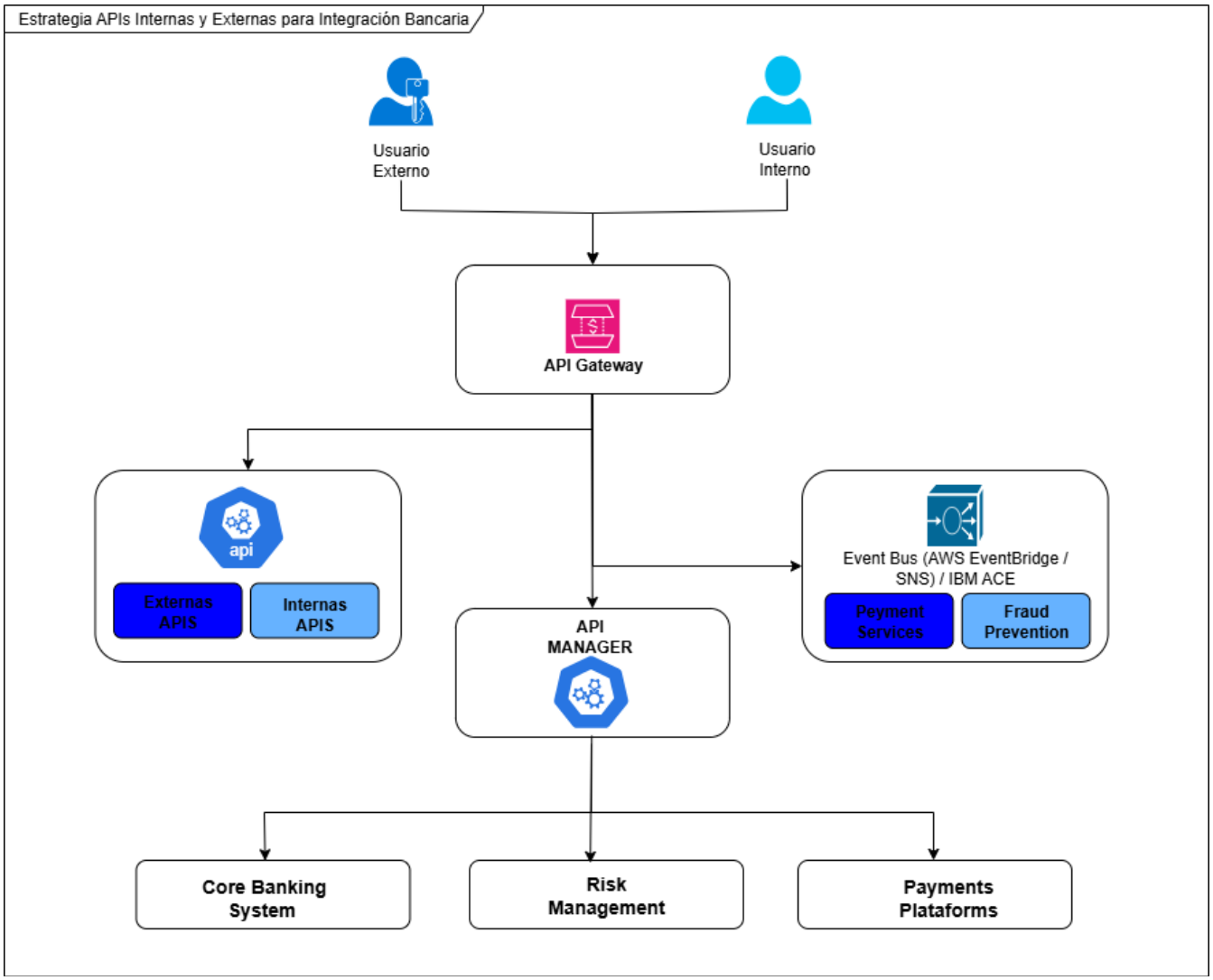
Donde X, Y, Z son enteros no negativos, donde X es la versión Mayor, Y la versión Menor, y Z la versión Parche.

Versión	Descripción
Versión Mayor (X)	Introducen modificaciones importantes en el contrato del servicio o en la funcionalidad (Interna o Externa) del API o Producto por lo que rompen la compatibilidad de las dependencias, y que tiene impacto en los consumidores. Ejemplo: 1.0.0, 2.0.0, 3.0.0
Versión Menor (Y)	Introduce modificaciones que NO tienen impacto en los consumidores actuales del servicio. Implementan adiciones de funcionalidades que no rompen la compatibilidad de las dependencias. Ejemplo: 1.1.0, 1.2.0, 1.3.0
Versión Parche (Z)	Cambios internos, los cuales corrigen comportamientos incorrectos en el servicio y NO tienen ningún impacto en los consumidores actuales del servicio. Ejemplo: 1.0.0, 2.1.1. 3.2.2

- Sandbox para pruebas de terceros.

7.1.6. Cómo se refleja en el diseño propuesto

- **API Gateway centralizado** conecta canales, backend, y servicios de terceros.
- **EventBridge** como canal de eventos desacoplado.
- **Contenedor API Manager** en el Diagrama C4 actúa como mediador y orquestador.
- **Mensajería desacoplada** permite resiliencia y escalabilidad.



8. Modelo de gestión de APIs

En una arquitectura basada en microservicios y APIs, una correcta gestión asegura interoperabilidad, control de versiones, seguridad, trazabilidad y escalabilidad. En el contexto bancario, esto es vital para cumplir regulaciones, garantizar experiencia de usuario y facilitar

la integración con terceros por ello se propone de siguiente manera aunque algunos de los puntos ya se plasmaron fases anteriores del documento.

8.1. Define el modelo de gestión de APIs

Lo vamos a estructurar de la siguiente manera resumidamente ya que ya lo plasmamos en fases anteriores de este documento.

8.1.1. Plataforma de Gestión de APIs

- **AWS API Gateway**: Para exponer APIs internas y externas, aplicar políticas de throttling, seguridad (JWT, OAuth2) y monitoreo.
- **Amazon App Mesh** o **Service Mesh (como Istio en EKS)**: Para gestionar la comunicación entre microservicios.

8.1.2. Ciclo de Vida de APIs

Incluye fases como:

- Diseño (con OpenAPI/Swagger)
- Implementación
- Pruebas
- Versionamiento (v1, v2…)
- Publicación (por entorno: dev, QA, prod)
- Monitoreo (CloudWatch, X-Ray)
- Retiro/Deprecación

8.1.3. Gestión del catálogo

- **AWS API Gateway Developer Portal** (o SwaggerHub) como portal de descubrimiento para desarrolladores internos y externos.

8.1.4. Seguridad de APIs

- **IAM Roles**, **Cognito**, **OAuth2/JWT**
- **API Key + Usage Plans** para control de acceso externo
- **WAF** para filtrar amenazas y limitar accesos sospechosos

8.2. Define el modelo de gestión de microservicios

Lo vamos a estructurar de la siguiente manera resumidamente ya que ya lo plasmamos en fases anteriores de este documento.

8.2.1. Diseño y Despliegue

- Cada microservicio es autónomo, pequeño y especializado
- Desplegados sobre **AWS Lambda**, **ECS Fargate**, o **EKS**
- Comunicación vía eventos (EventBridge, SQS, SNS) o API REST/GraphQL

8.2.2. Herramientas para Gestión

- **ECS / EKS**: Orquestación de contenedores
- **CloudFormation / CDK / Terraform**: Infraestructura como código
- **AWS X-Ray**: Trazabilidad distribuida
- **CloudWatch + Prometheus/Grafana**: Monitoreo

8.2.3. Observabilidad y Logging

- Logs estructurados con correlación entre servicios
- Monitoreo de SLIs/SLOs

8.3. Gobierno y versionamiento

Lo vamos a estructurar de la siguiente manera resumidamente ya que ya lo plasmamos en fases anteriores de este documento.

- Usa **versionamiento explícito** en las rutas (/api/v1/clientes)
- Crea una **política de deprecación** (por ejemplo: preavisar 6 meses antes del retiro de una versión)
- Políticas de naming, control de cambios, pruebas contractuales

8.4. Automatización del ciclo DevSecOps

- CI/CD con **CodePipeline**, **CodeBuild**, **GitHub Actions** o **Jenkins**
- Escaneos de seguridad: Snyk, SonarQube, CodeQL
- Automatización de pruebas de contrato (con Pact o Postman/Newman)

Este modelo de gestión garantiza una gobernanza efectiva sobre las APIs y microservicios, promueve la escalabilidad, resiliencia, trazabilidad y cumplimiento normativo exigido por el sector financiero.

9. Plan de Migración

La modernización de sistemas bancarios críticos requiere una estrategia de migración que reduzca el riesgo operativo, asegure la integridad de los datos, y permita la validación progresiva de cada componente antes de su adopción total. El enfoque propuesto se basa en una migración gradual por fases y en paralelo, con monitoreo y pruebas constantes.

9.1. Principios clave del plan de migración

Los principios calves pueden ser os siguientes.

- **Minimizar el tiempo de indisponibilidad**
- **Asegurar compatibilidad entre sistemas legados y nuevos**
- **Usar técnicas como estrangulamiento (Strangler Pattern)**
- **Trazabilidad y reversibilidad**
- **Automatización y monitoreo**
- **Backups constantes y pruebas de recuperación**

9.2. Fases del plan de migración gradual

Las fases se pueden dividir en las siguientes:

Fase 1: Evaluación y planificación

- Identificación de sistemas críticos y dependencias
- Análisis de riesgo de cada componente
- Diseño de arquitectura moderna paralela
- Definición de KPIs y métricas de éxito

Fase 2: Preparación de entorno híbrido

- Habilitación de conectores (APIs/ESB) entre legacy y moderno
- Despliegue de infraestructura en la nube (ej. AWS)
- Pruebas de interoperabilidad

Fase 3: Migración por módulos o dominios

- Migrar funcionalidades **por dominio** (ej. clientes, cuentas, pagos)
- Aplicar el patrón **Strangler Fig**: El nuevo sistema toma el control progresivamente
- Rutas de API Gateway redirigen tráfico por funcionalidad

Fase 4: Validación paralela (shadowing)

- Operación en paralelo durante X semanas
- Comparación de outputs (legacy vs moderno)
- Ajustes finos en microservicios

Fase 5: Redirección progresiva de tráfico

- Primero a usuarios internos o testers
- Luego a segmentos controlados de clientes reales
- Soporte de rollback inmediato

Fase 6: Retiro del sistema legado

- Solo cuando el 100% del tráfico está en el sistema moderno
- Documentación de lecciones aprendidas
- Eliminación segura del sistema anterior

9.3. Mecanismos de reducción de riesgo

- **Monitoreo proactivo** con CloudWatch, Prometheus
- **Logs centralizados** con ELK o AWS CloudWatch Logs
- **Alertas automáticas**
- **Pruebas automatizadas** (unitarias, integración, regresión)
- **Plan de rollback documentado**

Esta estrategia de migración gradual reduce los riesgos técnicos y operativos, permite validar cada paso con seguridad, y ofrece capacidad de revertir cambios si es necesario, lo cual es clave para un entorno bancario altamente regulado y sensible como el propuesto.