Arquitecto de Integración - Ejercicio práctico BP

Resumen general

El objetivo de este documento es completar el ejercicio práctico para optar por el puesto de arquitecto de integración para el Banco Pichincha.

La problemática planteada es la siguiente:

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

Escenario:

Un banco tradicional busca modernizar su arquitectura tecnológica para mejorar sus servicios digitales y cumplir con las nuevas tendencias de la industria. Se requiere integrar:

- Core bancario tradicional existente, más un nuevo core bancario digital
- Nuevo sistema de banca web y banca móvil
- Plataforma de servicios de pago
- APIs para servicios de terceros (Openfinance)
- Sistemas de Gestión de Riesgos
- Sistema de Prevención de Fraudes

Consideraciones adicionales:

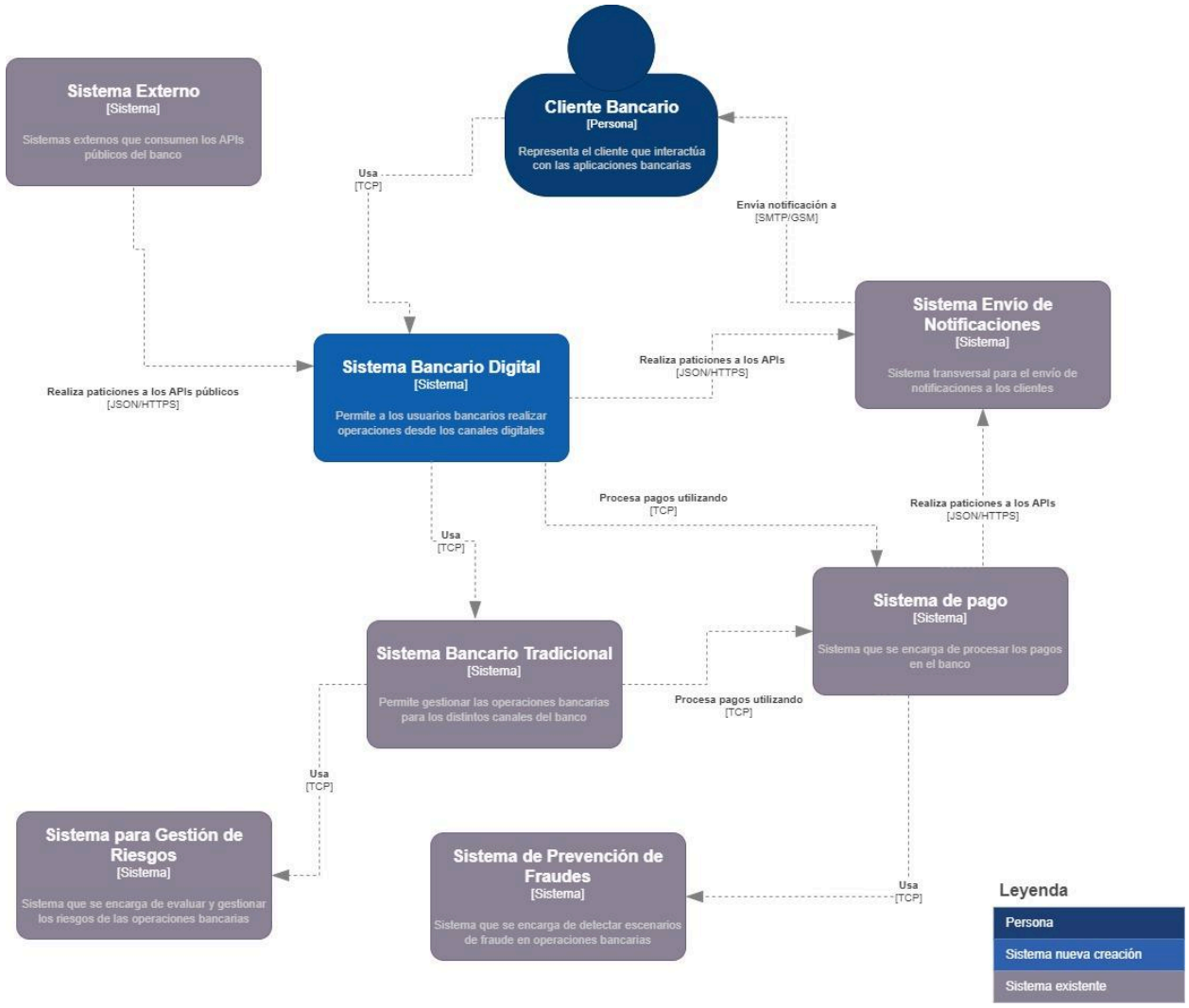
- Garantice la alta disponibilidad y tolerancia a fallos, seguridad y monitoreo
- Si lo considera necesario, su diseño de integración puede contener elementos de infraestructura en nube como Azure o AWS. Debe garantizar baja latencia.
- El diseño debe estar bajo el modelo C4
- Contemplar en el diseño el uso de eventos, siguiendo los lineamientos de la industria.

Diseño de arquitectura de integración de alto nivel

El diseño arquitectónico lo vamos a realizar apoyándonos del modelo C4, el cual es utilizado para diagramar los modelos de arquitectura de software. A partir de las especificaciones del ejercicio, solo se implementarán los siguientes diagramas: Contexto, Contenedores y Componentes. La componente de Código de este modelo quedaría fuera del alcance de este ejercicio técnico.

Nivel 1: Diagrama de Contexto del Sistema

Muestra una visión a alto nivel de la interacción entre las personas con los sistemas de software que componen el diseño arquitectónico.



Explicación general del modelo

Nota importante: Se intentará explicar el modelo en función de la información con que contamos, la información adicional que se propicia, se generará a partir de la inferencia conceptual de las entidades del negocio.

1. Un cliente bancario interactúa de manera digital con los servicios del banco a través de un nuevo sistema bancario digital.

2. Si el nuevo sistema bancario digital necesita enviar algún tipo de notificación a los clientes bancarios, utiliza un sistema externo que funciona transversal a todos las aplicaciones del banco (este componente no se encuentra descrito en la problemática, pero se adiciona por razones de brindar una solución más completa). Para razones demostrativas, asumimos que esta aplicación expone un conjunto de APIs para el envío de notificaciones vía correo electrónico o mensaje de texto. Una vez que la aplicación recibe la solicitud, entrega el mensaje al cliente destino utilizando el protocolo correspondiente (SMTP, en caso de los correos electrónicos y GSM para el caso de los mensajes de texto)

3. El nuevo sistema bancario digital utiliza la lógica de negocio que se encuentra almacenada en el Sistema Bancario Tradicional.

4. El sistema bancario tradicional funciona como el elemento que centraliza toda la lógica del negocio bancario. Cada sistema que necesita realizar algún tipo de operación bancaria, debe utilizar la lógica construida en esta aplicación.

5. Existe un sistema externo que se encarga normalmente de procesar todos los pagos que realiza el banco. Tanto el nuevo sistema de banca digital como el sistema de banca tradicional, cada vez que necesitan completar un pago, utilizan los servicios que se exponen desde esta aplicación.

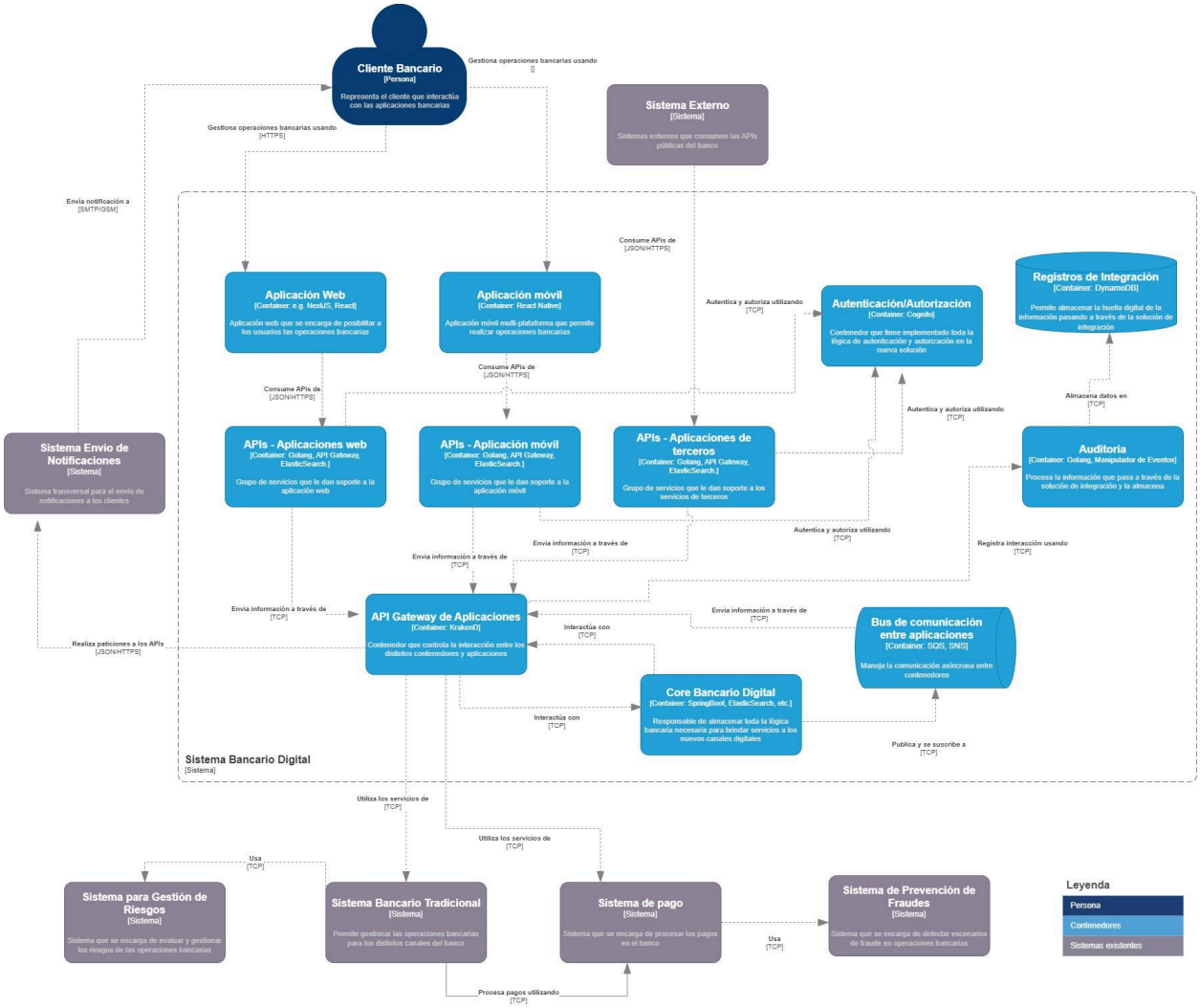
6. El sistema de pago, para su correcto funcionamiento utiliza el sistema de prevención de Fraudes.
7. El sistema bancario tradicional utiliza un sistema para la Gestión de Riesgos.
8. Los sistemas externos pueden consumir los APIs públicos que expondrá el nuevo sistema bancario digital.

Nivel 2: Diagrama de contenedores del sistema

Los diagramas de este nivel nos ayudan a descomponer el sistema en contenedores. Para este contexto, definimos como un contenedor, la representación de una aplicación, un almacén de datos, aplicaciones o servicios web, bases de datos, sistemas de ficheros, etc.

Nota importante: los sistemas ya construidos se tomarán como una unidad conceptual indivisible para este contexto. Los mecanismos de integración que se recomiendan para adaptar los sistemas existentes al nuevo diseño, se realizarán utilizando diagramas de componentes en las próximas secciones.

Nota importante: como arquitecto de integración no se encuentran dentro de mis responsabilidades la selección de las tecnologías con que se desarrollan las soluciones de software, sino más bien, todo lo que tiene que ver con la integración de las mismas. Es por ello que en el siguiente modelo, nos concentramos en la integración de los contenedores, siendo más específicos cuando el contenedor sea un elemento propio de la integración.



Explicación General del Modelo

Con el diagrama presentado, pretendemos visualizar la separación en contenedores de la nueva aplicación bancaria que se pretende diseñar.

Procedemos entonces a explicar la decisiones arquitectónicas:

- La nueva solución bancaria tendrá tres canales digitales para sus clientes: una aplicación web, una aplicación móvil y un grupo de APIs expuestos para que aplicaciones de terceros puedan consumir servicios bancarios (garantizando el Open Banking)
- Para el diseño arquitectónico, el primer patrón que implementamos es un **BFF (Backend For Frontend)**, ya que ponemos una solución dedicada para atender independientemente a cada canal de entrada. El motivo de la implementación de este patrón en este punto se debe a que para cada canal de entrada, se podrán personalizar

los mecanismos de gestión de las peticiones, así como aplicar reglas específicas de acuerdo a las características de los mismos.

- Para cada backend vamos a implementar un patrón **API Gateway** para crear un solo punto de entrada por backend, el cual se encargará de redirigir hacia el recurso correspondiente, en dependencia de la solicitud recibida.
- Todas las peticiones que lleguen a cada uno de los backends, deben ser autorizadas. Para realizar los procedimientos de autorización y autenticación (en caso de ser necesario) se utiliza el contenedor de Autenticación/Autorización.
- Todo el corazón del negocio de este nuevo sistema de banca digital se centralizará en el **core bancario digital**.
- Para la comunicación asíncrona de cada uno de los contenedores que componen esta aplicación se utilizará el contenedor **Bus de Comunicación entre Aplicaciones**. Para la interacción con este Bus se realizará utilizando los siguientes patrones de integración:
    - **Publicador / Suscriptor:** este patrón lo utilizaremos en los escenarios donde se crean tópicos para realizar cierta tarea y tenemos aplicaciones o contenedores suscritos a los mismos. Por otro lado, cada vez que necesitamos utilizar los servicios de estos elementos, enviamos un mensaje con el tópico en cuestión. Todos los elementos que se encuentren suscritos al tópico, recolectarán el mensaje y lo procesarán. En nuestro caso, podremos tener la aplicación de **Envío de Notificaciones** suscrita a uno o varios tópicos (dependiendo si hemos creado un tópico por tipo de notificación, lo cual me gustaría recomendar) y cada vez que necesitemos enviar una notificación, lo único que deberíamos hacer es enviar un mensaje al Bus de Comunicaciones con el tópico indicado.
    - **Hub and Spoke**(la traducción al español del nombre del patrón traería confusiones, ya que se le conoce por su nombre en inglés): para garantizar la alta disponibilidad, existen escenarios donde debemos tener capacidad computacional esperando para brindarle soporte a todas las solicitudes que se reciban de manera paralela. La utilización de este patrón nos permite poder suscribir un grupo de elementos de procesamiento computacional a eventos que ocurran dentro del sistema, utilizando para ello el **Bus de Comunicación entre Aplicaciones**.Para atender la capacidad computacional se recomienda una estrategia de **Round-robin**.
- Toda la información que pasa a través de los mecanismos de integración implementados debe almacenarse por razones de auditoría (teniendo en consideración los elementos que nunca deben guardarse o registrarse logs de ellos para cumplir con ciertas normas de regulación bancaria como son PCI DSS o Ley de Protección de Datos Personales). Para completar estas tareas de procesar la información que posteriormente será almacenada, se utiliza el contenedor de **Auditoría**, el cual estará suscrito a un evento dentro del **Bus de comunicación**.
- Una vez la información ha sido procesada, esta será almacenada en diferentes almacenes de datos (teniendo en consideración la información que necesitamos consultar a corto, mediano y largo plazo). En cualquier caso, se recomienda que la información que se almacena, se cifre en reposo.

- El **nuevo core bancario**, para completar muchas de sus funciones, utilizará la lógica implementada en el **core bancario tradicional**.
- Si el nuevo core bancario necesita procesar pagos, realizará esta operación utilizando los servicios ya implementados en el **Sistema de Pagos**.
- Las aplicaciones no se comunican directamente entre ellas, para realizar estas tareas, se utiliza un API Gateway a nivel de aplicación, el cual tiene la responsabilidad de manejar la comunicación entre los distintos elementos que componen el sistema. Este API Gateway a nivel de aplicación, entre sus funciones más importantes se encuentran:
    - Autorizar la comunicación entre aplicaciones
    - Filtrar la información que se comparte entre elementos arquitectónicos
    - Desacoplar la comunicación asíncrona entre elementos de arquitectura

Nota adicional: En la descripción del ejercicio no se brinda información entre las aplicaciones ya existentes, por lo que las relaciones entre ellas se asumen a partir de la lógica del autor.

Nivel 3: Diagramas de componentes del sistema

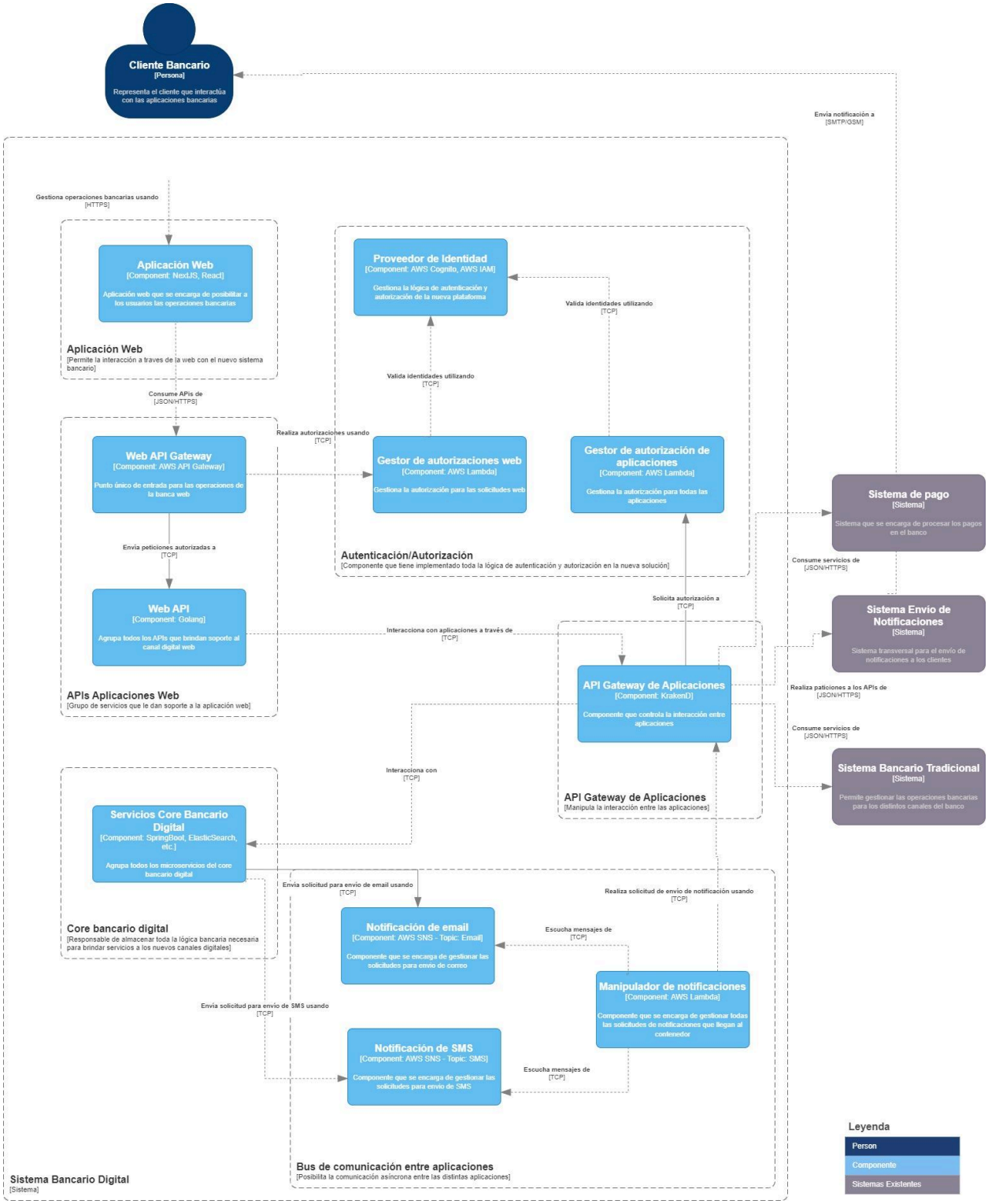
A continuación vamos a descomponer cada uno de los contenedores en su correspondiente diagrama de componente. En cada uno de estos diagramas modelamos la interacción de los componentes dentro del contenedor.

Nota Importante: los diagramas de componentes se enfocarán exclusivamente en las características de integración.

Nota Importante: la nube que se utilizará de base para la implementación de esta solución de integración será AWS (aunque el diseño arquitectónico es genérico y permite ser implementado en cualquier otra nube)

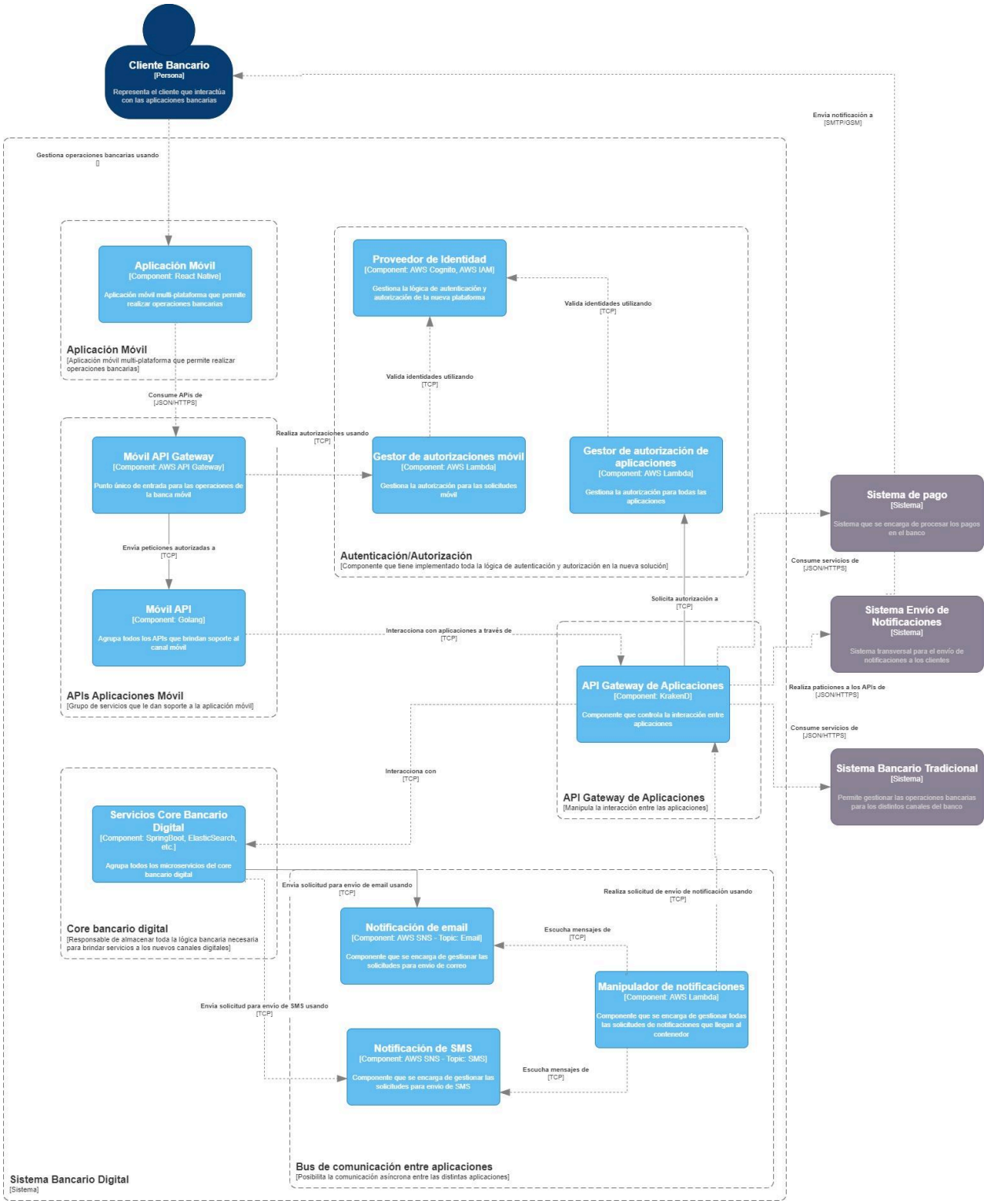
Canal Web

En el siguiente diagrama de componentes vamos a representar el flujo a través del canal web.
En el diagrama presentado, se representa la descomposición en componentes de varios de los contenedores que intervienen en dicho flujo.


Cliente Bancario
[Persona]
Representa el cliente que interactúa con las aplicaciones bancarias
Envía notificación a
[SMTP/GSM]
Sistema Bancario Digital
[Sistema]
Gestiona operaciones bancarias usando
[HTTPS]
Aplicación Web
[Component: NextJS, React]
Aplicación web que se encarga de posibilitar a los usuarios las operaciones bancarias
Aplicación Web
[Permite la interacción a traves de la web con el nuevo sistema bancario]
Consume APIs de
[JSON/HTTPS]
Web API Gateway
[Component: AWS API Gateway]
Punto único de entrada para las operaciones de la banca web
Envía peticiones autorizadas a
[TCP]
Web API
[Component: Golang]
Agrupa todos los APIs que brindan soporte al canal digital web
APIs Aplicaciones Web
[Grupo de servicios que le dan soporte a la aplicación web]
Realiza autorizaciones usando
[TCP]
Proveedor de Identidad
[Component: AWS Cognito, AWS IAM]
Gestiona la lógica de autenticación y autorización de la nueva plataforma
Valida identidades utilizando
[TCP]
Valida identidades utilizando
[TCP]
Gestor de autorizaciones web
[Component: AWS Lambda]
Gestiona la autorización para las solicitudes web
Gestor de autorización de aplicaciones
[Component: AWS Lambda]
Gestiona la autorización para todas las aplicaciones
Autenticación/Autorización
[Componente que tiene implementado toda la lógica de autenticación y autorización en la nueva solución]
Solicita autorización a
[TCP]
Interacciona con aplicaciones a través de
[TCP]
API Gateway de Aplicaciones
[Component: KrakenD]
Componente que controla la interacción entre aplicaciones
API Gateway de Aplicaciones
[Manipula la interacción entre las aplicaciones]
Sistema de pago
[Sistema]
Sistema que se encarga de procesar los pagos en el banco
Consume servicios de
[JSON/HTTPS]
Sistema Envio de Notificaciones
[Sistema]
Sistema transversal para el envío de notificaciones a los clientes
Realiza paticiones a los APIs de
[JSON/HTTPS]
Consume servicios de
[JSON/HTTPS]
Sistema Bancario Tradicional
[Sistema]
Permite gestionar las operaciones bancarias para los distintos canales del banco
Interacciona con
[TCP]
Servicios Core Bancario Digital
[Component: SpringBoot, ElasticSearch, etc.]
Agrupa todos los microservicios del core bancario digital
Core bancario digital
[Responsable de almacenar toda la lógica bancaria necesaria para brindar servicios a los nuevos canales digitales]
Envía solicitud para envío de email usando
[TCP]
Realiza solicitud de envío de notificación usando
[TCP]
Notificación de email
[Component: AWS SNS - Topic: Email]
Componente que se encarga de gestionar las solicitudes para envío de correo
Escucha mensajes de
[TCP]
Manipulador de notificaciones
[Component: AWS Lambda]
Componente que se encarga de gestionar todas las solicitudes de notificaciones que llegan al contenedor
Envía solicitud para envío de SMS usando
[TCP]
Notificación de SMS
[Component: AWS SNS - Topic: SMS]
Componente que se encarga de gestionar las solicitudes para envío de SMS
Escucha mensajes de
[TCP]
Bus de comunicación entre aplicaciones
[Posibilita la comunicación asincrona entre las distintas aplicaciones]
Leyenda
Person
Componente
Sistemas Existentes

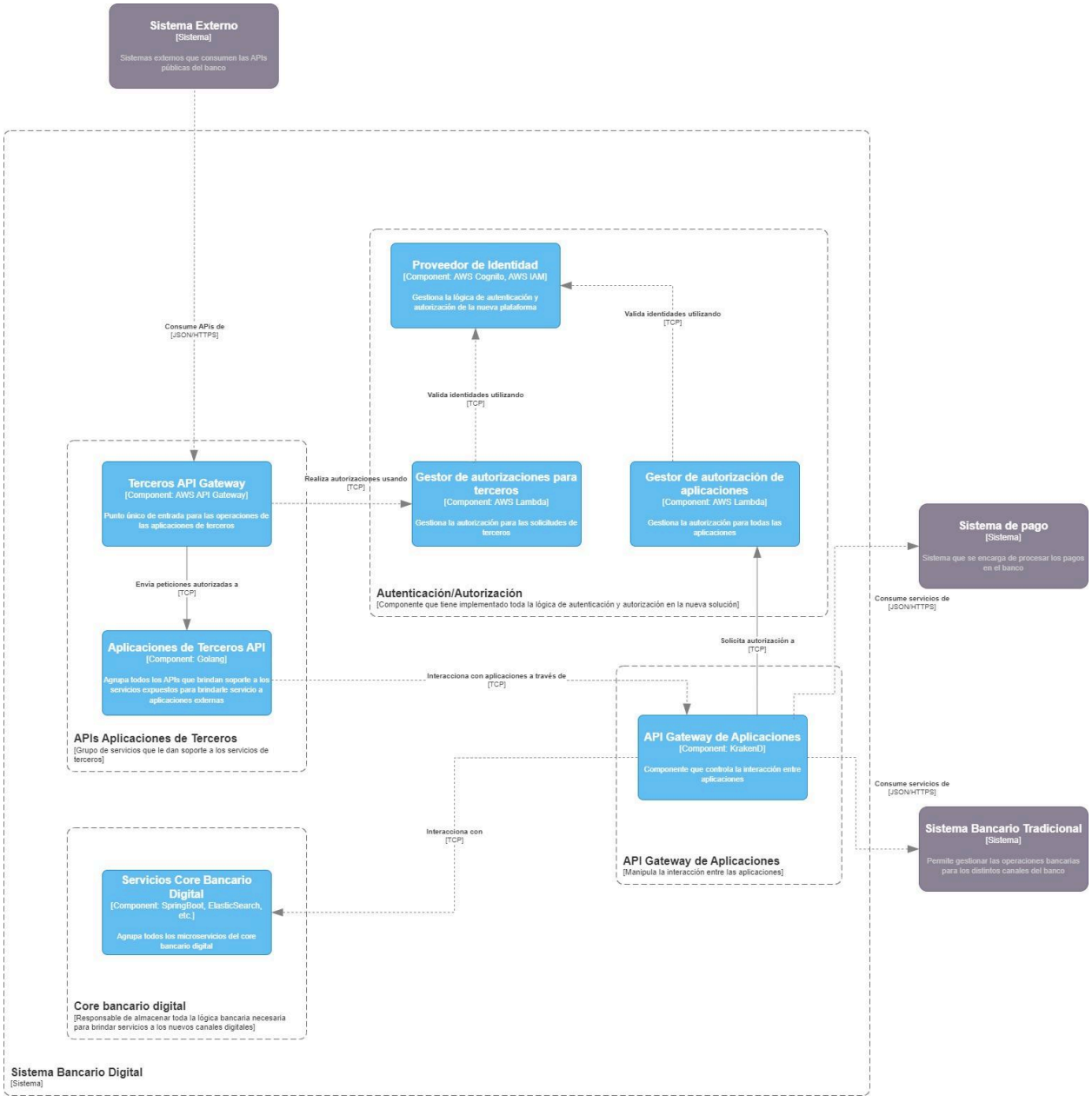
Canal Móvil

En el siguiente diagrama de componentes vamos a representar el flujo a través del canal móvil. En el diagrama presentado, se representa la descomposición en componentes de varios de los contenedores que intervienen en dicho flujo.



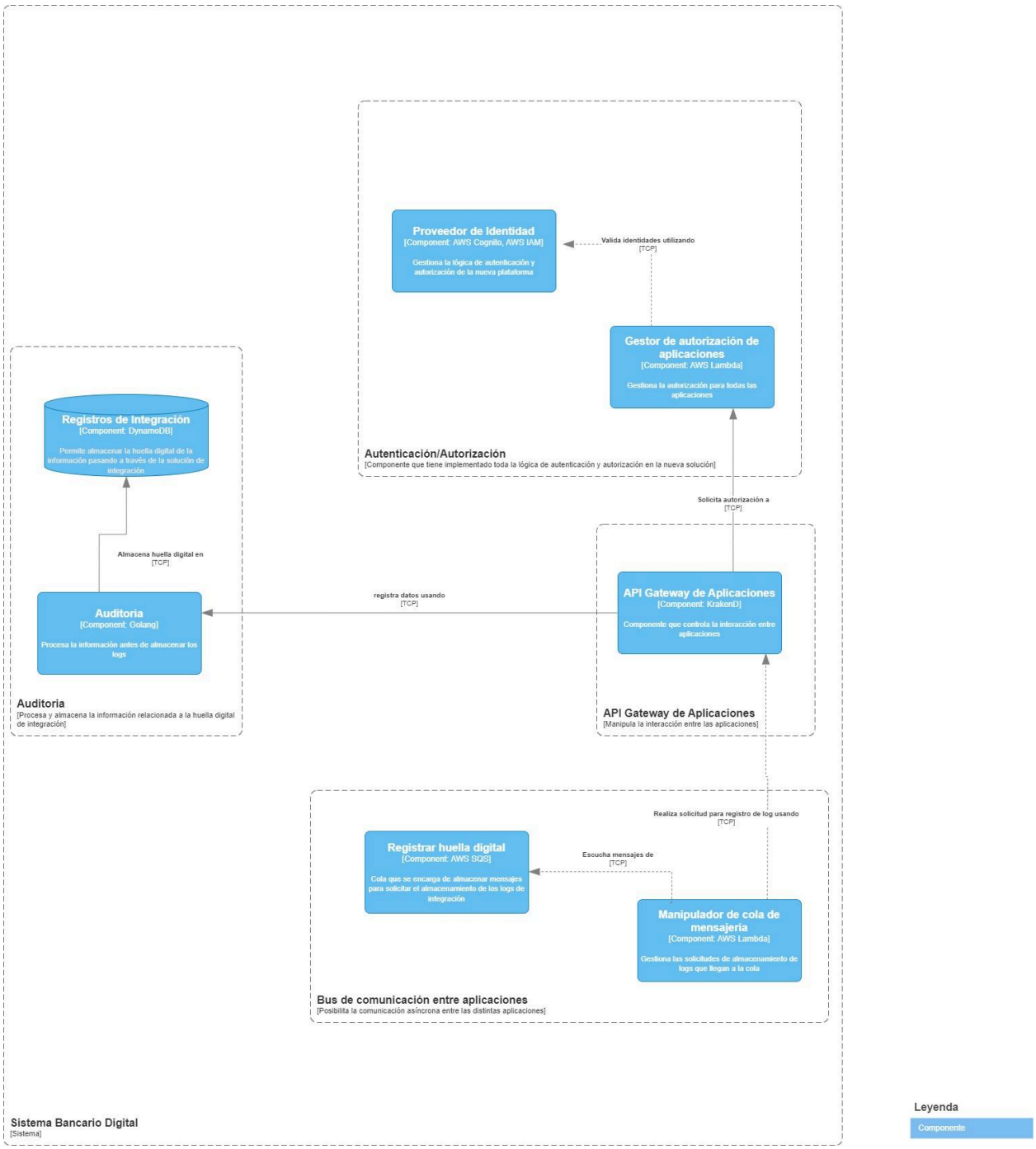
Aplicaciones de Terceros

Para brindar servicios de Open Banking, se diseñó una solución de integración que garantiza el desacople y la alta cohesión entre componentes. A continuación se describe el diseño a través de su respectivo diagrama de componentes:



Registros de huella digital

En el siguiente diagrama de componentes nos vamos a abstraer al registro de la huella digital de integración, la cual mantiene el registro de las entradas y salidas que viajan a través de los componentes de integración.



Patrones de integración y tecnologías utilizadas

Patrones de diseño

En la solución planteada hemos utilizado varios patrones de integración, los cuales describimos a de manera muy general a continuación:

1. El primer patrón que salta a la vista es el BFF (Backend For Frontend), el cual se ve evidenciado en el diseño a la hora de tener separado un API personalizado para cada canal digital. La implementación de este patrón en el diseño propuesto nos permite:
    a. Separar el tráfico de cada canal hacia unidades de procesamiento separadas
    b. Personalizar el procesamiento de las peticiones y las respuestas, mejorando así el rendimiento de las peticiones.
2. Para cada API externa, se implementó un patrón API Gateway, el cual nos permite tener un único punto de entrada que brinde soporte para varios backends. A nivel interno, para la comunicación entre aplicaciones se implementó un API Gateway empresarial. Dentro de las ventajas que contamos al implementar este patrón tenemos:
    a. Centramos la seguridad de la petición
    b. Facilidad de interacción con varios servicios utilizando un único punto de entrada.
    c. Gobernabilidad de APIs

3. En escenarios específicos, sobre todo dentro del flujo del patrón API Gateway, se utilizó un patrón de **Transformación de Datos**, ya que cuando se interactúa con varias aplicaciones, en muchas ocasiones es necesario transformar entradas y salidas por razones de seguridad, privacidad, etc.
4. **Para escenarios de comunicación asíncrona hemos utilizado los patrones Publicador / Subscriptor** y **Hub and Spoke**, lo cual puede evidenciarse en la implementación del Bus de Comunicación entre aplicaciones.

Tecnologías utilizadas

Aunque la solución arquitectónica busca ser lo más general posible, para la solución propuesta se tomaron como referencia los siguientes servicios de AWS:

- **AWS Cloud:** toda la solución está basada en la nube de AWS, ya que quería brindar un enfoque mono-nube para la misma.
- **Cloudfront:** se utiliza para distribuir contenido y establecer reglas de integración a nivel de frontera.
- **WAF:** Es un firewall que nos permite agregar otra capa de seguridad a nuestra solución de integración.
- **AWS API Gateway:** servicio de AWS que nos permite crear API Gateway a nivel de nube.
- **AWS Lambdas:** servicio de procesamiento sin servidor que nos permite procesar información sin tener ningún aprovisionamiento. En la solución se utilizan mayormente como handlers asociados a colas y como elementos de procesamiento para realizar tareas de autorización.
- **AWS Cognito:** Plataforma de identidad que utilizamos para autorizar usuarios, aplicaciones externas e internas.
- **ECS + EC2 + Fargate:** se utilizan en dos escenarios, primeramente para darle servicio a la aplicación de Frontend (Banca Web) y adicionalmente para desplegar el Aplication API Gateway que utiliza KrakenD. **KrakenD** es una solución API Gateway de alto rendimiento capaz de agregar servicios bajo un patrón **BFF (Backend For Frontend)**
- **AWS SQS:** servicio que se utiliza para enviar, almacenar y recibir mensajes entre componentes de software. Es una cola que se utiliza para desacoplar servicios.
- **AWS SNS:** servicio para gestionar procesos de mensajería bajo un enfoque de alto rendimiento.
- **DynamoDB**: base de datos de alto rendimiento, NoSQL, sin servidor y completamente administrada.
- **Secret Manager:** servicio para manejar de forma centralizada el ciclo de vida de los secretos.

Seguridad, cumplimiento normativo y ley orgánica de protección de datos

Elementos de seguridad implementados en la solución

La solución de integración presta especial interés a la hora de garantizar la seguridad a la hora de comunicar las distintas componentes.

A continuación se describen las estrategias diseñadas:

- **Cifrado de datos:** toda la información que circule a través de la infraestructura debe estar cifrada, para lo cual se recomienda utilizar las siguientes estrategias:

    - **Cifrado en tránsito:** Utilizar HTTPS/TLS para cifrar los datos en tránsito entre los usuarios y los servicios
    - **Cifrado en reposo:** : Se recomienda utilizar **AWS KMS** para cifrar los datos almacenados en bases de datos, volúmenes de almacenamiento y backups.

- **Configuración de herramientas de monitoreo:** es recomendable tener almacenado todas las acciones que se realizan sobre la cuenta de AWS, es por ello que se recomienda realizar las siguientes acciones:
    - Configurar el servicio de **AWS CloudTrail** para tener almacenadas todas las acciones que se realizan sobre la cuenta de AWS.
    - Utilizar el servicio de **AWS Config** para asegurarnos que los recursos cumplan con todas las políticas establecidas.
    - Se deberá mantener alertas en **AWS Health** para recibir notificaciones sobre posibles fallos en AWS que afecten nuestros servicios.

- **Seguridad a nivel de Red:** adicionalmente a lo visto con anterioridad, el otro elemento que debemos asegurar a la hora de garantizar el ambiente de integración es la seguridad a nivel de red. Dentro de las acciones que se diseñaron o se recomiendan realizar se encuentran:
    - Uso de **VPCs** para segmentar la red y tener aislados los componentes del sistema que no necesitan interactuar entre ellos.
    - Tener correctamente configurados los **Security Groups** y las **ACLs**, con lo cual podemos tener un control más certero del tráfico de entrada y salida.
    - Utilización de **AWS Shield** y **WAF** para proteger la infraestructura contra ataques DDoS

Cumplimiento normativo

Teniendo en cuenta las normativas bancarias, la solución de integración debe brindar los elementos necesarios para cumplir con cada una de ellas. Dentro de las principales normativas podemos mencionar:

- **PCI DSS**: orientada a establecer los procedimientos y reglas para el manejo de pagos con tarjetas.

  Para nuestro escenario de integración, se deben tener en consideración los siguientes elementos:
    - Toda la responsabilidad del proceso de pago (recolección de datos, procesamiento de transacciones y almacenamiento de datos) se asume que es responsabilidad de uno de los sistemas que se encuentra realizado con anterioridad (Sistema de Pago)
    - Nuestra responsabilidad para esta solución de integración es garantizar que la solución propuesta no incumpla con los pilares que sustentan la certificación.

  Para mantener una solución de integración que mantenga el entorno cumpliendo los requerimientos de este estándar, hemos implementado las siguientes consideraciones:

    - La solución de integración se encuentra protegida por firewalls perimetrales, los cuales se encuentran constantemente actualizados contra nuevas amenazas.
    - Toda la información que pasa entre los elementos de arquitectura es cifrada punto a punto mediante una política de llave pública y llave privada (utilizando políticas de rotación dinámica de claves)
    - La información en reposo es cifrada
    - La información sensible de pago que se mueve a través de la arquitectura no se persiste bajo ningún escenario.
    - Toda la información que se mueve entre los elementos arquitectónicos (si no es información sensible) se almacena y puede ser auditada en todo momento.
    -
- **GDPR**: Teniendo en cuenta que la institución bancaria en cuestión presenta clientes que son ciudadanos de la comunidad europea, es muy importante tener en cuenta este requisito para la solución de integración planteada.
  Este marco normativo establece el procedimiento a la hora de manipular datos personales, es aplicable para empresas que operan en la Unión Europea o con tiene clientes en ese territorio.
  De los elementos que deben cumplirse para garantizar la normativa y tenemos la responsabilidad de implementarlos en el diseño de integración propuesto tenemos:
    - Garantizar la privacidad a nivel de diseño de solución
    - Garantizar la privacidad en la transferencia de datos
    - Evitar las violaciones de la seguridad de datos personales
    -
- **SOC 2:** marco de trabajo para demostrar los controles de seguridad implementados con el objetivo de proteger los datos en la nube.

  El estándar se centra en cumplir 5 principios:
    - Seguridad
    - Disponibilidad
    - Confidencialidad
    - Integridad del procesamiento
    - Privacidad

  En nuestra solución de integración garantizamos estos puntos a través de la implementación de los siguientes elementos:
    - Hemos diseñado la implementación de Firewalls y sistemas de detección de intrusos, autenticación de 2 factores y autorizaciones de peticiones.
    - Hemos diseñado una solución de alta disponibilidad, con un procedimiento confiable para reponernos ante incidentes.
    - En el diseño propuesto nos hemos esforzado en mantener la confidencialidad de la información, para lo cual, entre otras cosas hemos recomendado el cifrado de datos en tránsito y en reposo, el filtrado de datos para evitar compartir información confidencial entre aplicaciones y los controles para garantizar que la información confidencial no se quede almacenada sin ser previamente cifrada.

- En el diseño propuesto se establecen mecanismos arquitectónicos que garantizan que el procesamiento de la información sea realizada por elementos autorizados, los cuales no deben en ningún caso distorsionar o cambiar la información.

Ley orgánica de protección de datos

Tanto la solución de integración como las distintas soluciones relacionadas a la plataforma deben cumplir con los elementos regulatorios que exige esta normativa. Los principales puntos son:

- **Consentimiento:** Implementar mecanismos para obtener y gestionar el consentimiento explícito de los usuarios para el tratamiento de sus datos personales.
- **Derecho de Acceso y Rectificación:** Proveer a los usuarios la capacidad de acceder y corregir sus datos personales.
- **Retención de Datos:** Definir y aplicar políticas de retención y eliminación de datos personales, asegurando que los datos no se conserven más tiempo del necesario.
- **Portabilidad de Datos:** Implementar funcionalidades que permitan a los usuarios exportar sus datos personales en un formato estructurado, comúnmente utilizado y legible por máquina.

Para el contexto de nuestra solución de integración, es importante señalar que en ningún escenario almacenamos datos personales, ya que nos aseguramos en todo momento, de que la información sensible no se almacene el los mecanismos que se encargan de persistir toda la información que se mueve a través de la solución planteada.

Alta disponibilidad y recuperación ante desastres

Es indudable que para sistemas de este tipo, garantizar la alta disponibilidad y contar con estrategias para poder recuperarse ante desastres, son elementos indispensables en cualquier diseño arquitectónico.

Alta disponibilidad

La alta disponibilidad es una característica que debe cumplir la implementación de soluciones arquitectónicas de esta naturaleza. Bajo esta premisa, como normas generales se deben seguir las siguientes recomendaciones:

- **Eliminar o reducir los puntos únicos de falla:** un punto único de fallo es un componente de tu pila de tecnología que causaría una interrupción del servicio si no estuviera disponible. En nuestra solución de integración tratamos de crear un diseño lo más desacoplado posible, evitando en todo momento esta situación. De manera adicional se recomienda a nivel de implementación del diseño que se trate de utilizar servicios de nube que su disponibilidad sea gestionada por la misma nube

(soluciones del tipo serverless como Lambdas, SQS, SNS, API Gateways, los cuales garantizan por diseño el auto escalado y la alta disponibilidad)

- **Implementar políticas de redundancia y replicación:** una característica importante para lograr alta disponibilidad en los diseños arquitectónicos es la redundancia y replicación de recursos computacionales (datos + infraestructura). Teniendo en cuenta el escenario donde se implantará la solución planteada (AWS como recomendación) debemos considerar que los sistemas que necesiten ser provisionados de capacidad computacional estática deben ser implementados en varias zonas de alta disponibilidad (AZ)
- **Implementar equilibrio de carga y gestión de tráfico:** para nuestra recomendación de implantación del diseño propuesto consideramos que el uso del servicio de AWS ELB (Elastic Load Balancing) nos puede ayudar a la distribución automática del tráfico que llega a las distintas aplicaciones que componen el diseño.
- **Automatizar los procesos de recuperación ante escenarios de error**
- **Planificación de fallas controladas para evaluar el plan de recuperación ante desastres.**

Adicionalmente, uno de los elementos que por diseño ayudan a la alta disponibilidad de las soluciones es el enfoque basado en eventos de la solución arquitectónica.

Estrategia de Disaster Recovery (DRP)

Para implementar una DRP podemos utilizar las siguientes estrategias:

- Copia de seguridad y restauración (RPO horas, RTO horas)
- Luz Piloto (RPO minutos, RTO horas)
- Espera semi - activa (RPO segundos, RTO minutos)
- Activo-Activo en varios sitios (RPO segundo/instantáneo, RTO segundo)

RPO: Punto de recuperación objetivo, métrica que nos indica la cantidad de datos que una empresa puede tolerar perder durante un evento imprevisto.
RTO: Tiempo de recuperación objetivo, se refiere a la cantidad de tiempo que una aplicación, sistema y proceso pueden estar inactivos sin causar un daño significativo al negocio y el tiempo dedicado a restaurar la aplicación y sus datos para reanudar las operaciones comerciales normales después de un incidente significativo.

Copia de seguridad y restauración

Este enfoque es adecuado para mitigar la pérdida o la corrupción de información. Utilizando esta estrategia se podría mitigar un desastre regional mediante la replicación de datos en otras regiones de AWS o para mitigar la falta de redundancia de las cargas de trabajo implementadas en una sola zona de disponibilidad. Además de los datos, debe volver a implementar la infraestructura, la configuración y el código de la aplicación en la región de recuperación. Para permitir que la infraestructura se vuelva a desplegar rápidamente y sin errores, debe siempre implementar la infraestructura como código (IaC) .

Para poder poner en práctica este enfoque es necesario contar con una estrategia para copias de seguridad, la cual se ejecute periódicamente o que sea continua. La frecuencia con la que ejecute la copia de seguridad determinará el punto de recuperación alcanzable (que debe alinearse para cumplir con nuestro RPO). La copia de seguridad también debería ofrecer una forma de restaurarla al momento dado en el que se realizó

Si utilizamos AWS como nube para la implementación de esta solución, podríamos utilizar los siguientes servicios:

- Amazon EBS
- Copias de seguridad dentro de DynamoDB
- Copias de seguridad de Amazon EFS (si decidimos utilizar AWS Backups)

Luz piloto

Con el enfoque de *luz piloto* se replican los datos de una región a otra y se almacena una copia de la infraestructura de carga de trabajo principal. Los recursos necesarios para poder replicar y hacer una copia de seguridad de los datos, como las bases de datos y el almacenamiento de objetos, siempre están disponibles. Otros elementos, como los servidores de aplicaciones, se cargan con código de aplicación y configuraciones, pero se apagan y solo se usan durante las pruebas o cuando se invoca la conmutación por error de recuperación de desastres. A diferencia del enfoque de copia de seguridad y restauración, la infraestructura principal siempre está disponible y siempre tiene la opción de aprovisionar rápidamente un entorno de producción a gran escala al encender y escalar los servidores de aplicaciones horizontalmente. Teniendo en consideración que se recomienda tener todo implementado utilizando IaC, los entornos de respaldo se pueden poner en ejecución rápidamente.

El enfoque de luz piloto minimiza el coste continuo de la recuperación de desastres pues minimiza los recursos activos y simplifica la recuperación en el momento de un desastre porque todos los requisitos de la infraestructura central están en su lugar.

Espera semi activa

El enfoque de espera semiactiva implica garantizar que haya una reducción vertical, pero totalmente funcional, del entorno de producción en otra región. Este enfoque amplía el concepto de luz piloto y reduce el tiempo de recuperación porque la carga de trabajo siempre está activa en otra región. Este enfoque también permite realizar pruebas más fácilmente o implementar pruebas continuas para aumentar la confianza en la capacidad para recuperarse de un desastre.

Teniendo en cuenta la recomendación de utilizar un enfoque serverless (el cual nos cobra sólo por lo que utilizamos), este enfoque sería muy recomendado.

Estrategia Activa/Activa

Puede ejecutar la carga de trabajo simultáneamente en varias regiones como parte de una estrategia *activa/activa en varios sitios* o *espera activa/pasiva*. La estrategia activa/activa sirve el tráfico desde todas las regiones en las que se implementa, mientras que la estrategia de espera activa solo sirve el tráfico de una sola región. Las demás regiones solo se utilizan para la recuperación de desastres. Con una estrategia activa/activa en varios sitios, los usuarios pueden acceder a la carga de trabajo en cualquiera de las regiones en las que se implemente. Este enfoque es el más complejo y caro para la recuperación de desastres, pero puede reducir el tiempo de recuperación a casi cero en la mayoría de desastres con las opciones de tecnología e implementación correctas (sin embargo, la corrupción de datos puede necesitar depender de copias de seguridad, lo que generalmente resulta en un punto de recuperación distinto de cero).

Resumen final

A partir de todas las estrategias explicadas y teniendo en cuenta que en el ejercicio no se detallan elementos restrictivos podríamos concluir lo siguiente:

- Si hacemos un equilibrio entre costo y beneficio, las estrategia de **Espera semi activa** considero que es la más adecuada.
- Si tenemos en consideración el tipo de negocio y sus requerimientos no funcionales, considero que la estrategia **activa/activa** se ajusta mucho más a las necesidades del negocio.

Gestión de identidad y acceso

Para la gestión de la identidad y acceso diseñamos un componente centralizado, el cual tiene la responsabilidad de realizar todas las tareas de este tipo.

Teniendo en consideración la recomendación de utilizar la nube de AWS para la implementación de la solución arquitectónica propuesta, proponemos utilizar los siguientes servicios:

- **IAM**: servicio que nos ayuda a gestionar de manera segura el acceso a los recursos de AWS.

- **AWS Cognito:** Es una plataforma de identidad para aplicaciones web y móviles. Es un directorio de usuarios, un servidor de autenticación y un servicio de autorización para credenciales.

IAM

Con este servicio vamos a garantizar que cada servicio solo tenga el menor acceso posible para poder realizar sus tareas. Para lograr este objetivo, se recomiendan realizar las siguientes tareas a la hora de implementar la solución diseñada:

- Se debe realizar un mapa de accesos para identificar qué servicios debe interactuar con cual.
- Se deben realizar grupos y convertirlos en roles para poder asociarlos a permisos.
- Toda la configuración de permisos debe realizarse en la herramienta de IAC (en nuestro caso recomendamos serverless framework) ya que si se necesitan desplegar los servicios en otras zonas y demás, una vez desplegados los servicios, ya contarían con los permisos adecuados.
- Se debe evitar siempre que se pueda, la creación de permisos manuales en la nube.
- La gestión de usuarios se debe realizar federados.

AWS Cognito

De manera general vamos a utilizar AWS Cognito para autorizar usuarios y aplicaciones a través de los componentes de arquitectura. Para poder separar las responsabilidades y permisos de cada grupo de usuarios, en el diseño arquitectónico se definieron 3 pools:

- **Bank Customers:** son los clientes del banco que acceden normalmente a los servicios a través de los canales digitales (Banca Web y Banca móvil)
- **Third Party App:** aplicaciones de terceros que consumen los APIs externos que expone el banco.
- **Internal Apps:** aplicaciones internas que tienen permiso para interactuar entre sí.

El utilizar el servicio de AWS Cognito para manejar el acceso de los usuarios y las aplicaciones a los diferentes servicios de la nueva arquitectura, nos hace reflexionar en un problema que se presenta de manera inmediata: **debemos poder hacer convivir los usuarios y aplicaciones que ya forman parte de la solución bancaria, con este nuevo diseño propuesto**.

Teniendo en cuenta que el ejercicio no brinda información sobre cómo se encuentran gestionados los permisos de usuarios y aplicaciones, intentaremos brindar una solución genérica a este problema.

Para resolver esta problemática, podemos recomendar la siguiente heurística:

- Si el usuario es nuevo, se registra directamente en **AWS Cognito** y todo debiera funcionar tal y como se encuentra diseñado.
- Si el usuario ya es parte del sistema podremos utilizar el siguiente enfoque:
    - Migrar los usuarios antiguos a **AWS Cognito**, pero esto llevaría a tener en consideración los costos y el esfuerzo necesario para construir un plan de migración que nos permita mover de una manera segura todos estos recursos.
    - Transformar el componente de **Autenticación/Autorización** para que utilice varias fuentes de verdad (podría traer ciertos problemas de rendimiento que van a depender de la cantidad de fuentes de verdad y los mecanismos de acceso a las mismas).

Para determinar el mejor enfoque a la hora de atacar este problema, sería necesario contar con más contexto del que tenemos actualmente.

El componente de **Autenticación/Autorización** no solo valida clientes bancarios, sino también el acceso a las aplicaciones.Es por ello que como estrategia general, cada aplicación debe ser representada con usuarios de Cognito, los cuales tengan claramente definidos sus permisos a la hora de interactuar con los demás elementos de la arquitectura.

Estrategias de APIs internas y externas

A la hora de definir una estrategia para implementar la comunicación sobre las API, es importante poder segmentar en dos categorías: **Internas** y **Externas**. La manera de gestionar la integración depende claramente de qué tipo de API estamos manejando.

Para las **APIs internas**, se debe tener en cuenta lo siguiente:
- Debe estar implementadas normalmente bajo un patrón API Gateway, ya que normalmente va a existir la necesidad de agregar muchos servicios a través de un único punto de entrada
- Debe contar con una latencia super baja
- Debe tener la posibilidad de contar con un patrón de transformación de datos para evitar mostrar información que no le competen a otros dominios
- Deben contar con herramientas de autorización internas

En otro caso, para las **APIs externas**, se deben tener en consideración los siguientes elementos:
- Se debe conocer el origen de la petición y esta debe venir desde un origen autorizado.
- Se debe establecer rate limits.
- Se debe contar con estrategias de autenticación y autorización basados en OAuth2 y JWT, entre otros.
- Si es posible, agregar otra capa de autenticación basados en API Keys que firmen la petición.

Gobierno de APIs y Microservicios

Una de las responsabilidades fundamentales del Arquitecto de Integración es llevar un estricto control sobre los servicios con que cuenta su arquitectura. Es por ello que llevar una estrategia eficiente del gobierno de APIs y Microservicios es un elemento imprescindible para desempeñar este rol satisfactoriamente.

Lo primero que debemos realizar es seleccionar el tipo de gobierno que debemos implementar. Dentro de los principales modelos podemos señalar:

- **Gobierno centralizado:** un equipo centraliza todas las revisiones y aprobaciones de los cambios de la arquitectura.
- **Gobierno híbrido:** muy similar al gobierno centralizado, pero en este caso, existen subdivisiones que se encargan de ciertos temas.
- **Gobierno distribuido:** se tienen varios equipos para productos específicos, cada equipo es responsable de su producto y los cambios de arquitectura se hacen bajo este dominio.

Por un tema de flexibilidad, sin lugar a dudas el enfoque distribuido aporta mayor agilidad, pero hay que tener en cuenta que para entornos bancarios, donde los procesos deben venir definidos bajo un grupo de estándares y normas que se aplican a todos por igual, podría ser muy recomendable utilizar el enfoque de **gobernanza centralizada**.

En cualquier caso, todos estos enfoques deben seguir los mismos objetivos:

- **Garantizar que las APIs se encuentren siempre en funcionamiento**, gestionando correctamente sus versiones, las cuales debieran coexistir bajo un entorno de estabilidad.
- **Gestionar la complejidad:** mantener un entorno donde los equipos tengan plena comprensión de las APIs disponibles, asegurando que los desarrolladores tengan plena claridad sobre la complejidad de sus entornos.
- **Seguridad y cumplimiento:** garantizar el acceso correcto al API, asegurándonos de que solo las personas adecuadas tienen acceso a las APIs.
- **Alineación valor, coste y negocio:** las APIs tiene que aportar valor, por lo tanto hay que asegurarse que cada una de las APIs cumplen con su labor y van alineadas a la visión con la que fueron creadas.

Otro de los puntos trascendentales en el gobierno de las APIs y microservicios es contar con un eficiente inventario. Para alcanzar estos objetivos nos podemos ayudar de algunas herramientas como:

- Servicio de API Gateway de AWS
- Postman
- Swagger
- MuleSoft AnyPoint Platform

Plan de Migración

Descripción general

Para el diseño del plan de migración es necesario tener bien claro los objetivos y el alcance del mismo, lo cual nos va a permitir trazar la estrategia adecuada para completar esta tarea satisfactoriamente.

Detalle de contexto
Para nuestro escenario, no queda claro la realidad actual de los sistemas que ya se encuentran en funcionamiento, por lo que se asumirá que se migrarán para que coexistan todos en una misma nube.

En nuestro caso podríamos definir, de manera muy general, los siguientes objetivos:

- Migrar los sistemas en funcionamiento a la nueva nube
- Realizar los cambios necesarios a los sistemas en funcionamiento para que se adapten a los nuevos patrones de integración.
- Configurar los servicios sobre AWS que le brindis el soporte a los sistemas que se migrarán

Nota: esto es solo una muestra muy general de los mismos

Una vez que los objetivos se encuentren correctamente descritos, se deben identificar y priorizar las aplicaciones y los datos críticos

En nuestro caso tendríamos que describir las aplicaciones que ya se encuentran en funcionamiento y se migrarán a este nuevo enfoque de integración. De manera adicional se debe tener en consideración la criticidad a la hora de migrar los datos (en caso de ser necesario). Para escenarios de este tipo, se recomienda no tocar los datos y solo mover las unidades de procesamiento.

Cuando la fase de identificación culmine satisfactoriamente, se debe comenzar a crear una planificación, la cual podríamos dividir en las siguientes etapas:

- Crear un cronograma con fechas claves y pasos para cada fase de la migración
- Asignar de manera eficiente los recursos, tanto humanos como tecnológicos
- Evaluar la infraestructura actual para evaluar qué necesita cambios y que se puede migrar directamente.

Seguidamente debemos seleccionar las estrategias de migración que queremos llevar a cabo. Dentro de las más comunes podemos encontrar:

- **Rehost**: mover aplicaciones sin cambios significativos

- **Refactor**: Adaptar y optimizar aplicaciones para el nuevo entorno
- **Replatform**: Realizar ajustes adicionales para optimizar las aplicaciones para el nuevo entorno.
- **Rebuild**: reconstruir aplicaciones desde cero
- **Replace**: sustituir soluciones existentes por aplicaciones SaaS

Antes de realizar el proceso de migración se deben realizar pruebas exhaustivas para asegurarnos que las aplicaciones funcionan correctamente. Adicionalmente es muy importante realizar validaciones de seguridad.

Una vez que ya tenemos probado todo, nos encontramos listos para proceder con la migración, para lo cual, se recomienda tener en cuenta los siguientes elementos:

- La migración debe realizarse por fases para minimizar el impacto en las operaciones diarias.
- Se debe monitorear el rendimiento y la seguridad durante y después de la migración

Cuando la migración se encuentre completada, es importante planificar procesos de optimización post-migración. Dentro de los elementos a tener en cuenta tenemos:

- Realización de ajustes y optimizaciones basados en el rendimiento y el feedback recibido.
- Capacitar al equipo para gestionar el nuevo entorno.

De esta manera quedaría completado el plan de migración para el nuevo enfoque propuesto a la hora de implementar el diseño arquitectónico propuesto.

Github url: https://github.com/rafaelbomate/bp-arquitecto-integracion