DOCUMENTO DE ARQUITECTURA DE SOLUCIONES DETALLADO - DASD – TEST DEVSU JUL2024

Ingeniero de Sistemas y Arquitecto de Soluciones Senior: Víctor Benites.

1 INTRODUCCIÓN

1.1. Objetivo

El presente documento de arquitectura de solución detallado (DASD) tiene como objetivo proporcionar una visión integral de la arquitectura propuesta para el desarrollo de este caso que le he llamado Proyecto de Nuevos Canales Digitales para Banco. La solución abarca diversas funcionalidades diseñadas para mejorar la experiencia del usuario y aumentar la eficiencia en las operaciones financieras. Para este proyecto los canales digitales a desarrollar incluyen según indica el caso Homebanking (Web) y Mobilebanking (Android e IOs), así como un robusto backend basado en microservicios utilizando Quarkus Framework.

Nota.- En Dic2023 hice una especialización en el MIT Masachussetts Institute of Technology en Liderazgo en la Innovación y mi caso de evaluación final fue justo Sistemas de Innovación en el Sistema Financiero.

Se adjuntan fuentes de Drawio y Ppt de los diagramas.

1.2. Descripción General del Proyecto

El proyecto se enfoca en la creación de canales digitales modernos y altamente funcionales que permitirán a los clientes del Banco acceder y gestionar sus cuentas y transacciones de manera conveniente y segura. Tengo experiencia en el Sistema Bancario por lo que voy a considerar los flujos que considero más importantes; entonces La Solución se compone de las siguientes funcionalidades claves:

- **Mi Perfil:** Los usuarios podrán obtener la información de su perfil personal, incluyendo información de contacto.
- **Consulta de Saldos y movimientos:** Los usuarios podrán verificar sus saldos de cuentas y revisar los movimientos de sus transacciones anteriores.
- **Desembolsos:** Facilitará la solicitud y aprobación de desembolsos financieros (URPI y largo), brindando una forma eficiente de acceso a fondos.

- **Pago de préstamos:** Los usuarios podrán realizar pagos de préstamos pendientes de manera rápida y sencilla.
- **Transferencias entre cuentas propias:** Permite la transferencia de fondos entre diferentes cuentas pertenecientes al mismo usuario.
- **Transferencias entre cuentas del Banco:** Posibilita la transferencia de fondos entre diferentes cuentas vinculadas al Banco.
- **Transferencias entre otros bancos:** Ofrece la capacidad de transferir fondos entre cuentas de diferentes bancos.
- **Interoperabilidad:** Garantiza la integración con las principales billeteras del mercado. En Perú: Yape, Plin.

1.3. Objetivos y Metas de la Arquitectura

La arquitectura propuesta tiene como objetivos principales:

- **Usabilidad:** Diseñar interfaces de usuario intuitivas y fáciles de usar para todas las funcionalidades, optimizando la experiencia del cliente.
- **Escalabilidad:** Construir una arquitectura escalable que pueda manejar un crecimiento significativo en la cantidad de usuarios y transacciones.
- **Seguridad:** Implementar medidas sólidas de seguridad para proteger los datos del usuario, las transacciones y la integridad del sistema en general.
- **Interoperabilidad:** Establecer conexiones efectivas con sistemas externos y plataformas de terceros para garantizar una integración perfecta. Estos sistemas y plataformas son ForgeRock (CIAM), Cloudflare, Biometría (Facephi), Notificaciones PUSH, Sistema de Análisis de Datoa, Sistema de Servicio de Email, Monitoreo, etc con el CORE Bancario.
- **Rendimiento:** Garantizar un rendimiento óptimo de las aplicaciones y el backend, minimizando los tiempos de respuesta y los cuellos de botella.
- **Mantenibilidad:** Desarrollar un sistema fácilmente mantenible y actualizable, con procedimientos claros para la gestión de cambios y mejoras.

2 CONTEXTO Y ALCANCE

2.1. Descripción del Problema y Necesidad

En un entorno financiero cada vez más digitalizado y competitivo, el negocio reconoce la importancia de ofrecer a sus clientes una experiencia moderna y conveniente para acceder y gestionar sus cuentas y transacciones. El sistema financiero actual exige soluciones ágiles que permitan a los usuarios realizar operaciones de manera eficiente y segura desde múltiples dispositivos, incluyendo navegadores web y dispositivos móviles.

2.2. Alcance Funcional

La solución abarca las siguientes funcionalidades principales:

- **Mi Perfil:** Obtención de información de perfil personal. Gestión de preferencias de notificaciones.
- **Consulta de Saldos y Movimientos:** Visualización de saldos en cuentas asociadas. Acceso a registros de transacciones.
- **Desembolsos:** Solicitud de desembolsos.
- **Pago de Préstamos:** Visualización de detalles de préstamos pendientes de pago.
- Realización de pagos de préstamos en línea.
- **Transferencias entre Cuentas Propias:** Transferencia de fondos entre cuentas bajo el mismo usuario.
- **Transferencias entre Cuentas del Banco:** Transferencia de fondos entre cuentas del Banco de diferentes usuarios.
- **Transferencias entre Otros Bancos:** Transferencia de fondos entre cuentas de diferentes bancos.
- **Interoperabilidad:** Integración con billeteras externas como son Yape o Plin.

2.3. Alcance No Funcional

Los aspectos no funcionales abordados por la solución incluyen:

- **Rendimiento:** Tiempos de respuesta aceptables incluso en momentos de alta carga.
- **Escalabilidad:** Capacidad para manejar aumentos significativos en el número de usuarios y transacciones.
- **Seguridad:** Implementación de prácticas de seguridad sólidas para proteger datos confidenciales adicionalmente al cumplimiento de los habilitadores de seguridad.
- **Usabilidad:** Diseño de interfaces intuitivas y amigables para los usuarios.
- **Mantenibilidad:** Creación de un sistema que sea fácilmente mantenible y actualizable según los lineamientos y estándares que maneja el Banco.

3 ARQUITECTURA DE ALTO NIVEL

La arquitectura propuesta para el desarrollo del CASO: Nuevos Canales Digitales se basa en una estructura modular y escalable que abarca diversas plataformas, incluyendo aplicaciones web, Android, iOS y un backend basado en microservicios. Cada componente ha sido diseñado para cumplir con los requisitos funcionales y no funcionales establecidos, garantizando una experiencia de usuario eficiente y segura.

3.1. Diagrama de arquitectura lógica de la solución

La Solución plantea una Arquitectura 100% Cloud en la Nube de Microsoft: Azure, llevando a cabo una implementación, compilación y ejecución de aplicaciones en entorno Cloud.

Todos los servicios podrán integrarse con múltiples plataformas utilizando componentes de Azure de modo que se pueda agrupar, compartir y escalar recursos.

TENANT

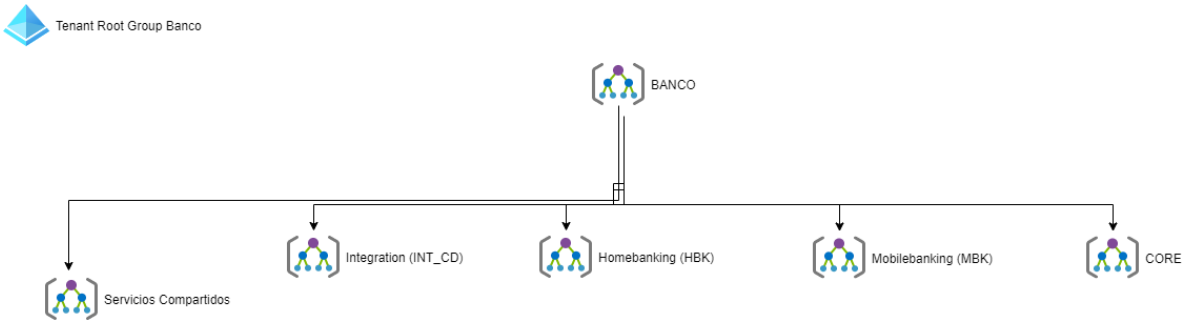


DIAGRAMA DE ARQUITECTURA DE MONITOREO

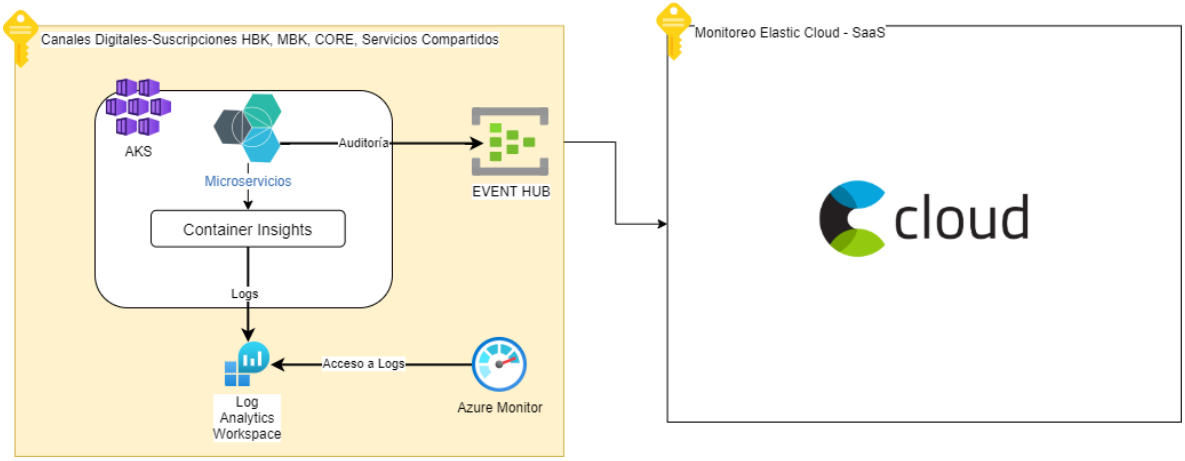
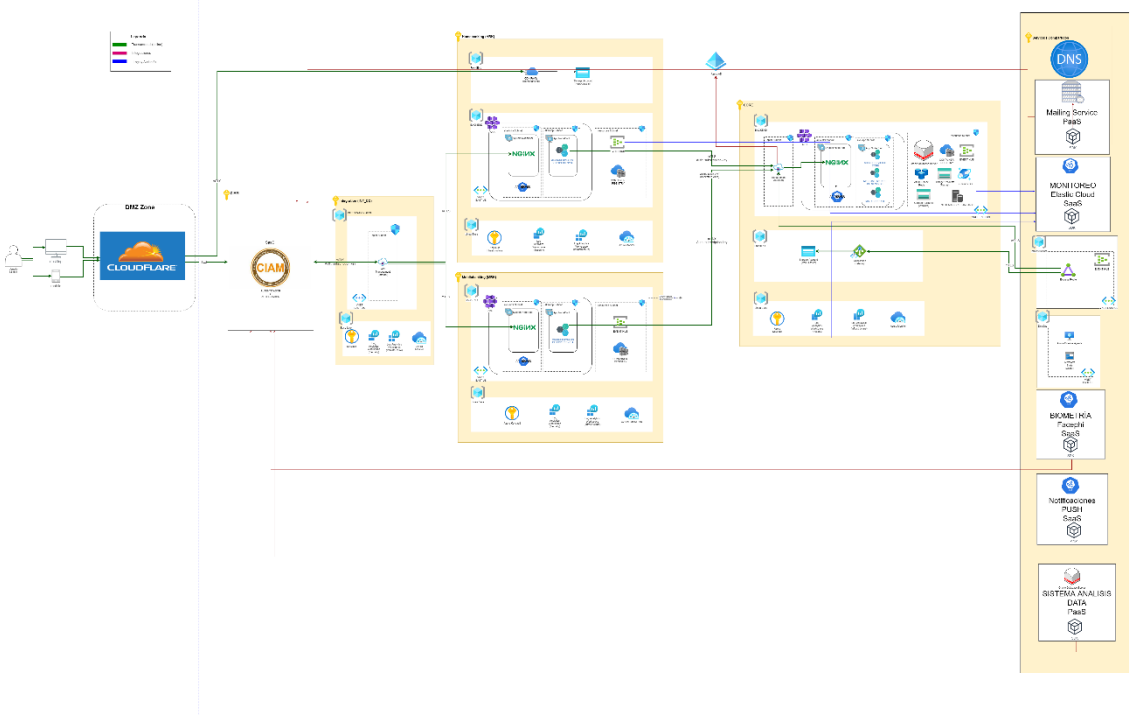


DIAGRAMA ARQUITECTURA LOGICA



Se plantea tener APIs de Experiencia integradas en la primera Suscripción (Integración) con un Api Management y éste será el nexo hacia otros componentes desplegados en los AKS (Azure Kubernetes Service) de HomeBanking y MobileBanking además de poder consumir las APIs de los servicios de la Suscripción Core.

La Web de Homebanking se integrará con los servicios de Cloudflare (WAF-Web Application Firewall) y hará uso de APIs que se expondrán mediante el CIAM (Customer Identity Access Management – El Gestor de Identidades del Banco- Autenticador y Autorizador). CIAM es un SaaS de la empresa ForgeRock.

La Web del CORE o BackOffice se integrará de forma privada mediante Azure AD para recursos internos del Banco y se expondrán APIs utilizando un API Management de servicios de la suscripción CORE.

El Registro de la Auditoría y Logs Funcionales se realizará de forma asíncrona en Event Hub; se habilitará un Event Hub en las suscripciones HMK, MBK, CORE y Servicios Compartidos; y va hacia un SaaS de Monitoreo: Elastic Cloud.

La Arquitectura tienen componentes externos que se detallan a continuación:

Cloudflare.- Producto que brinda servicios de red de entrega de contenido, ciberseguridad en la nube, mitigación de DDoS y servicios de registro de dominios. La administración y configuración de

Cloudflare está a cargo del Banco, por lo que estas no son tareas automatizadas. En Cloudflare se encuentra el WAF (Web Application Firewall) que supervisa, filtra o bloquea el tráfico HTTP hacia y desde una aplicación Web. También gestiona los dominios de la página web asociados al Banco, protegiéndola de múltiples ataques al servidor de aplicaciones web del BackEnd. Se tiene que hacer una integración entre el Cloudflare y el CIAM, para que se pueda llegar a la primera Suscripción: Integración donde se encuentra el APIM de Experiencia.

CIAM.- Plataforma que centraliza la gestión del acceso, autenticación y repositorio de identidades para los clientes externos gestionados por los nuevos canales digitales del Banco: HomeBanking y MobileBanking; cuya arquitectura será diseñada por el Arquitecto Víctor Benites.

La Arquitectura se descompone en 5 Suscripciones con sus respectivos componentes principales de acuerdo a la siguiente distribución:

1era. Suscripción.- Integración. Tiene como objetivo integrar la comunicación hacia los nuevos canales digitales desde fuentes externas que este caso son 2: CIAM y Cloudflare.

Se compone de lo siguientes recursos:

- ✓ Linea Base.- Grupo de recursos que van a servir para la administración de la infraestructura conformada por Azure Key Vault, Logs Analitycs WorkSpace que se encargan de los Logs de Infraestructura y Seguridad para su lectura en Azure Monitor.
- ✓ Api Management.- Es el componente que va a exponer todas las APIs (AKS MobileBanking y AKS HomeBanKing) de Experiencia del Banco. Esta capa de integración soporta ambos canales: APP+HMK, y va a exponer la capa de experiencia de los nuevos canales digitales del Banco.. Es decir, soporta la carga funcional de las aplicaciones para dale salida al BackEnd. Este API Management es de uso interno (red privada de azure)y sólo va a integrarse con CIAM mediante Peerings.

Los componentes de esta suscripción de integración serán alojados en la región EAST US.

2da. Suscripción.- HomeBanking (HBK). Se compone de los siguientes recursos.

- ✓ AKS HomeBanking.- Consta de 2 nodos principales: uno para los microservicios (applications user) donde se montan las APIS de experiencia y los servicios de experiencia, y el otro para la administración del AKS(nodo system), donde se encuentran los recursos del sistema y e Nginxs Ingress Controller.
- ✓ Nginx Ingress Controller.- Es la puerta de ingreso hacia los microservicios. Su función es ser un tipo de balanceador de carga especializado en entornos con contenedores.
- ✓ Linea Base.- Grupo de recursos que van a servir para la administración de la infraestructura conformada por Azure Key Vault, Logs Analitycs WorkSpace que se encargan de los Logs de Infraestructura y Seguridad para su lectura en Azure Monitor.
- ✓ Azure Key Vault.- Es parte de la Línea Base y se utiliza para el tema de secretos.
- ✓ Log Analytics Workspace.- Es parte de la Línea Base y está configurado en el AKS para obtener los Logs de los recursos (RAM, CPU,etc), este será el input del Azure Monitor.
- ✓ Container Registry.- Es el Repositorio de imágenes Docker versionado que se utilizará para desplegar los microservicios en el AKS.

- ✓ Event Hub.- Es el Centralizador de eventos utilizado para orquestar los logs de los microservicios desplegados en el AKS; en donde se genera el evento del log (PRODUCER) y donde existen 2 grupos de consumidores (CONSUMER) que alimentarán a Monitoreo dado por Elastic Cloud y a la Plataforma Analítica.(Si se llega a implementar-por aquí es un camino).

  Los Eventos de Auditoría se realizan a través de un registro en el Event Hub disponible en la Suscripción; los eventos serán recolectados y trasladados al repositorio del Elastic Cloud y/o Plataforma Analítica.

  También se tiene un Resource Group donde se va a implementar la web del banco, consu Storage Account y su CDN (Content Delivery Network). Esta web será desarrollada en Angular y es una SPA (Single Page Application) de contenido estático.

  Este Grupo de Recursos es la única diferencia con la 3era. Suscripción.

Los componentes de esta suscripción de HomeBanking serán alojados en la región EAST US.

3era. Suscripción.- MobileBanking (MBK). Se compone de los siguientes recursos.

- ✓ AKS HomeBanking.- Consta de 2 nodos principales: uno para los microservicios (applications user) donde se montan las APIS de experiencia y los servicios de experiencia, y el otro para la administración del AKS(nodo system), donde se encuentran los recursos del sistema y e Nginxs Ingress Controller.
- ✓ Nginx Ingress Controller.- Es la puerta de ingreso hacia los microservicios. Su función es ser un tipo de balanceador de carga especializado en entornos con contenedores.
- ✓ Linea Base.- Grupo de recursos que van a servir para la administración de la infraestructura conformada por Azure Key Vault, Logs Analitycs WorkSpace que se encargan de los Logs de Infraestructura y Seguridad para su lectura en Azure Monitor.
- ✓ Azure Key Vault.- Es parte de la Línea Base y se utiliza para el tema de secretos.
- ✓ Log Analytics Workspace.- Es parte de la Línea Base y está configurado en el AKS para obtener los Logs de los recursos (RAM, CPU,etc), este será el input del Azure Monitor.
- ✓ Container Registry.- Es el Repositorio de imágenes Docker versionado que se utilizará para desplegar los microservicios en el AKS.
- ✓ Event Hub.- Es el Centralizador de eventos utilizado para orquestar los logs de los microservicios desplegados en el AKS; en donde se genera el evento del log (PRODUCER) y donde existen 2 grupos de consumidores (CONSUMER) que alimentarán a Monitoreo dado por Elastic Cloud y a la Plataforma Analítica.(Si se llega a implementar-por aquí es un camino).

  Los Eventos de Auditoría se realizan a través de un registro en el Event Hub disponible en la Suscripción; los eventos serán recolectados y trasladados al repositorio del Elastic Cloud y/o Plataforma Analítica.

El Mobile es una app que se va a depositar en los proveedores: Google, iOs, Huawei.

Los componentes de esta suscripción de MobileBanking serán alojados en la región EAST US.

4ta. Suscripción.- CORE. Aquí se cumple con la Norma del Caso donde se pide que el cliente accese a una Plataforma CORE, y el otro acceso está en la 5ta. Suscripción de Servicios Compartidos donde se encuentra el Sistema del Cliente con sus datos en detalle.

Es la capa del BackEnd pues aloja el Core Bancario del Negocio. Sirve para hacer orquestaciones para lo que van a ser todas las funcionalidades, reglas de negocio, llamadas al CORE, etc. Es la capa que va a resolver las reglas de negocio de cada canal y también va a invocar y alojar los productos del CORE. Es un capa netamente BackEnd, es un AKS de línea de desarrollo de negocio, o de los Microservicios de Negocio. También se tiene aquí los Microservicios del BackOffice; en mi diseño es el administrador de los nuevos canales digitales, pues da sentido que viva en la capa CORE que por ahora sustenta 2 canales; es un administrador central, es la web interna de los nuevos canales digitales y tiene su conjunto de microservicios independiente, debe tener we y dominio dedicado interno.

Se convierte en un Hub de Integración. Tiene un grupo de recursos: CosmosDB, Oracle DB Server, Azure Cache Redis (es una BD pequeña, es un recurso para optimizar/performance de la aplicación mediante el uso temporal y reutilizable), Event Hub, entre otros.

- ✓ Application Gateway.- Es un balanceador de carga de tráfico web que administra el tráfico que llega a las aplicaciones web, es aquí donde se instala el certificado privado (algo así: https://backoffice.dominiobanco.com) y es de uso exclusivo para Intranet a través de VPN del Banco.
- ✓ Storage Account (website).- Donde se almacenan los estáticos de la web backoffice del tipo SPA (Single Page Application).
- ✓ Storage Account (firmas).- Donde se almacenan archivos e imágenes holográficas a utilizar por la aplicación web, estas son recuperadas mediante un microservicio backend desplegados en el AKS de la Suscripción CORE.
- ✓ API Management.- Dedicado a exponer las capacidades de esta capa CORE, cumpliendo con los principios de una arquitectura desacoplada, escalable y mantenible; es un activo cross(transversal) , un hub de integración hablando en el lenguaje de arquitectura de soluciones.
- ✓ Cosmos DB.- BD no relacional principal del Banco.
- ✓ Oracle Database Server.- BD relacional CORE del Banco.
- ✓ Sistema CORE Bancario.- Sistema Core del Banco, utiliza Oracle DB Server.
- ✓ Azure Cache Redis.- Es una memoria caché que almacena datos tipo hashes (clave valor), se convierte en una BD Temporal donde se guarda cierta información para optimizar la app. Es un caché centralizado.
- ✓ AKS CORE.- Azure Kubernetes Service CORE; se aprovisiona un nodo principal para los microservicios (applications user), este es el entorno donde se va a montar las APIS de Negocio, los servicios de Negocio y Operaciones propias del canal, es decir las parte orquestadora. Tiene un nodo principal para la administración del AKS (nodo system) en donde están los recursos del sistema y el NGINX Ingress Controller.
- ✓ Nginx Ingress Controller.-Es la puerta de ingreso hacia los microservicios, su función es ser un tipo de balanceador de carga especializado en entornos de contenedores.

- ✓ Linea Base.- Grupo de recursos que van a servir para la administración de la infraestructura conformada por Azure Key Vault, Logs Analitycs WorkSpace que se encargan de los Logs de Infraestructura y Seguridad para su lectura en Azure Monitor.
- ✓ Azure Key Vault.- Es parte de la Línea Base y se utiliza para el tema de secretos.
- ✓ Log Analytics Workspace.- Es parte de la Línea Base y está configurado en el AKS para obtener los Logs de los recursos (RAM, CPU,etc), este será el input del Azure Monitor.
- ✓ Container Registry.- Es el Repositorio de imágenes Docker versionado que se utilizará para desplegar los microservicios en el AKS.
- ✓ Event Hub.- Es el Centralizador de eventos utilizado para orquestar los logs de los microservicios desplegados en el AKS; en donde se genera el evento del log (PRODUCER) y donde existen 2 grupos de consumidores (CONSUMER) que alimentarán a Monitoreo dado por Elastic Cloud y a la Plataforma Analítica.(Si se llega a implementar-por aquí es un camino).

  Los Eventos de Auditoría se realizan a través de un registro en el Event Hub disponible en la Suscripción; los eventos serán recolectados y trasladados al repositorio del Elastic Cloud y/o Plataforma Analítica (Databrick + Data Factory + Datalake) en caso de habilitarse más adelante esta última.

Los componentes de esta suscripción de CORE serán alojados en la región EAST US.

5ta. Suscripción.- Suscripción de Servicios Compartidos.

Consta de un grupo de recursos transversales al Banco como por ejemplo: Servidores DNS, SaaS de Monitoreo Elastic Cloud, SaaS de Biometría FacePhi, Una Plataforma PaaS para envío de email y un SaaS para Notificaciones Push.

Cabe resaltar que la Biometría ingresa por CIAM, y si hubiera por ejemplo otro SaaS Transversal como por ejemplo Monitoreo de Fraudes también se colocaría en esta suscripción.

Aquí se cumple con varios requisitos del Caso: Norma de 2 sistemas de envío de notificaciones: Notificaciones PUSH y Envío de Email.

Aquí se cumple con el 2do Sistema de Datos de Cliente: Sistema del Cliente con sus datos en detalle.

El Sistema Facephi se encarga del cumplimiento del reconocimiento facial y se integra con el Sistema CIAM que es el encargado de la Autenticación y Autorización, aquí evidentemente es necesario una integración a través de los SDKs de ambos sistemas SaaS.

Aquí también se hace el Monitoreo en el Sistema SaaS Elastic Cloud, se utiliza un Patrón de Diseño en este caso de Integración que son los Event Hub en las Suscripciones, estos envían la data de Auditoría y Logs Funcionales al Elastic Cloud para su Análisis.

Los componentes de esta suscripción de CORE serán alojados en la región EAST US.

3.2. DECISIONES DE ARQUITECTURA

	FRENTE	DECISIÓN	DETALLE

1	Nube	Multisuscripción	Permite independencia en el aprovisionamiento de la IaC Infraestructura como Código, organización modular y escalable para las cargas de trabajo, permite aislar las cargas de trabajo específicas (landing zones)
2	Nube	Contingencia: Arquitectura Activa-Activa. Multiregión. Se debe realizar y aprobar un Plan DRP.	Segunda región con menor capacidad pero que soporte transaccionalidad; así se asegura la Continuidad del Negocio. Habilitar: Alta Disponibilidad 99.99% RTO: 4 Horas Max. en el Plan DRP para el peor escenario.
3	Nube	Nube Azure: Azure Availability Zones	Habilitar: Alta Disponibilidad 99.95%
	Seguridad	MTLS	Se debe asegurar la comunicación entre los componentes por autenticación mutua.
	Seguridad	OAuth	Autorización vía OAuth hacia CIAM desde la app y la web.
	Seguridad	Cloudflare-WAF	Se debe usar Cloudflare para la protección de las cargas de servicio web y asegurar las comunicaciones que provienen desde la app.
	Seguridad	JWE	Encriptación de datos de tarjeta para la generación del pinblock durante el enrolamiento viajará cifrada de forma asimétrica vía JWE.

3.3. Patrones de arquitectura Android y iOS.- Patrón Modular con Arquitectura Multicapas (Clean)

MVVM (Model-View-ViewModel) con Clean Architecture

MVVM es un patrón de arquitectura de software utilizado comúnmente en el desarrollo de aplicaciones de interfaz de usuario. Se compone de tres componentes principales:

- Model: Representa los datos y las reglas de negocio de la aplicación. Es responsable de manejar el acceso y la manipulación de datos, y no tiene conocimiento sobre la interfaz de usuario.
- View: Es la interfaz de usuario visible para el usuario. Se encarga de mostrar la información y recibir las interacciones del usuario. Idealmente, la vista debe ser lo más pasiva posible y no contener lógica de negocio compleja.
- ViewModel: Actúa como un intermediario entre el modelo y la vista. Se encarga de exponer los datos y comandos del modelo a la vista y de manejar las acciones del usuario. Además, se encarga de mantener el estado de la vista.

Características de MVVM:

- Desacopla la lógica de presentación de la lógica de negocio, lo que facilita la reutilización del código y el mantenimiento.
- Proporciona una separación clara entre la vista y el modelo, lo que permite cambios más sencillos y una mayor escalabilidad.
- Facilita las pruebas unitarias, ya que la lógica de negocio se encuentra en el ViewModel, que es independiente de la plataforma y no requiere la presencia de la interfaz de usuario.
- Permite un desarrollo más ágil y colaborativo, ya que los equipos pueden trabajar simultáneamente en la interfaz de usuario y la lógica de negocio sin afectar el trabajo del otro.

Características de Clean Architecture:

- Promueve la independencia de la tecnología, lo que facilita los cambios y las actualizaciones en el futuro.
- Mejora la testabilidad al permitir pruebas unitarias fáciles en cada capa sin la necesidad de interfaces de usuario o componentes externos.
- Facilita la reutilización del código, ya que las capas internas no dependen de las capas externas.
- Permite la escalabilidad, ya que la arquitectura se puede adaptar para satisfacer las necesidades cambiantes de una aplicación en crecimiento.

Beneficios de MVVM con Clean Architecture:

- Desacoplamiento y mantenibilidad: La combinación de MVVM con Clean Architecture facilita un desacoplamiento efectivo entre la lógica de presentación y la lógica de negocio, lo que mejora la mantenibilidad del código a medida que la aplicación crece y evoluciona.

- Testabilidad: La arquitectura limpia permite una mejor testabilidad en todos los niveles de la aplicación, ya que la lógica de negocio se encuentra en capas independientes y puede probarse sin la necesidad de interactuar con la interfaz de usuario.
- Escalabilidad: Al dividir la aplicación en capas bien definidas y dependencias unidireccionales, se facilita la escalabilidad, permitiendo que el equipo trabaje en diferentes capas simultáneamente sin afectar otras partes del sistema.
- Flexibilidad en la interfaz de usuario: MVVM separa la lógica de presentación de la interfaz de usuario, lo que facilita la creación de interfaces de usuario más flexibles y reutilizables.

Desventajas de MVVM con Clean Architecture:

- Complejidad inicial: Implementar MVVM con Clean Architecture puede resultar en una mayor complejidad inicial debido a la necesidad de definir capas y responsabilidades claras. Sin embargo, esta inversión inicial generalmente se traduce en una mayor facilidad de mantenimiento y desarrollo a largo plazo.

Tiempo de desarrollo: Es posible que se requiera más tiempo para configurar y desarrollar una arquitectura limpia en comparación con un enfoque menos estructurado. Sin embargo, a largo plazo, la inversión de tiempo en una arquitectura limpia generalmente resulta en una aplicación más robusta y escalable.

3.4. Patrones de arquitectura Backend

Arquitectura de Microservicio

La arquitectura de microservicios es un enfoque para desarrollar aplicaciones en el que se construye un sistema distribuido compuesto por múltiples componentes independientes llamados microservicios. Cada microservicio representa una unidad de negocio o una funcionalidad específica de la aplicación y se ejecuta de forma autónoma.

Los microservicios se comunican entre sí a través de interfaces bien definidas, generalmente utilizando protocolos ligeros como HTTP/REST o protocolos de mensajería como AMQP o MQTT. Esta comunicación se basa en interacciones sin estado, lo que significa que cada solicitud de un cliente debe contener toda la información necesaria para ser procesada de forma independiente por el microservicio correspondiente.

Cada microservicio se desarrolla, despliega y escala de manera independiente. Esto significa que los equipos de desarrollo pueden trabajar de forma aislada en cada microservicio, utilizando tecnologías y lenguajes de programación que mejor se adapten a los requisitos de ese componente específico.

La descomposición de una aplicación en microservicios tiene varias ventajas. Primero, permite una mayor modularidad y reutilización de código, ya que cada microservicio se enfoca en una tarea específica y puede ser desarrollado y probado de manera independiente. Además, facilita la escalabilidad horizontal, ya que es posible ajustar la cantidad de instancias de cada microservicio según la demanda, sin afectar a los demás.

La arquitectura de microservicios también ofrece una mayor tolerancia a fallos y resistencia. Si un microservicio experimenta un problema o falla, los demás microservicios pueden continuar funcionando sin interrupciones, ya que no hay una dependencia fuerte entre ellos.

Patrón de arquitectura hexagonal

También conocido como arquitectura de puertos y adaptadores, es un enfoque para diseñar aplicaciones con una separación clara entre el núcleo de la lógica de negocio y las capas externas, como las interfaces de usuario, las bases de datos y servicios externos.
En este patrón el núcleo de la lógica de negocio se encuentra en el centro del hexágono y está aislado de las capas externas. El objetivo principal es desacoplar la lógica de negocio de las tecnologías y servicios externos, lo que brinda mayor flexibilidad y facilita las pruebas y modificaciones de la lógica sin afectar otras partes del sistema.

El patrón hexagonal se basa en los siguientes conceptos clave:
Núcleo de la lógica de negocio: Es el corazón de la aplicación y contiene las reglas y procesos de negocio. Aquí se definen los casos de uso y se implementa la lógica esencial de la aplicación.
Puertos: Son interfaces que definen la comunicación entre el núcleo y las capas externas. Los puertos definen los puntos de entrada y salida del núcleo y permiten la interacción con las capas externas de una manera independiente de la tecnología utilizada.
Adaptadores: Son las implementaciones concretas de los puertos. Los adaptadores se encargan de conectar el núcleo de la aplicación con las capas externas, como interfaces de usuario, bases de datos o servicios externos. Estos adaptadores se adaptan a las tecnologías y protocolos específicos utilizados en cada capa externa.

El flujo de datos en el patrón hexagonal sigue una dirección unidireccional. Las solicitudes ingresan a través de los puertos en el núcleo de la lógica de negocio, donde se procesan y se generan las respuestas correspondientes. Luego, las respuestas son enviadas de vuelta a través de los puertos hacia los adaptadores que se encargan de comunicarse con las capas externas.

Las ventajas del patrón hexagonal incluyen:
Desacoplamiento: Permite que el núcleo de la lógica de negocio sea independiente de las tecnologías externas, lo que facilita las pruebas unitarias y la modificación de la lógica sin afectar otras partes del sistema.

Flexibilidad: Permite cambiar las tecnologías o servicios externos sin afectar la lógica de negocio, ya que los adaptadores son responsables de la adaptación entre las interfaces internas y externas.
Testabilidad: Facilita las pruebas, ya que la lógica de negocio se puede probar de manera aislada utilizando implementaciones de puertos simulados o en memoria.
Modularidad: Permite una estructura modular y clara, con responsabilidades bien definidas para cada parte del sistema.

Patrón de mensajería asíncrona

Es un enfoque de diseño que permite la comunicación entre diferentes componentes de un sistema distribuido a través del intercambio de mensajes asincrónicos. En lugar de establecer una comunicación directa y sincrónica entre los componentes, se utilizan colas de mensajes para transmitir información de manera indirecta.
En este patrón, los componentes del sistema se comunican enviándose mensajes a través de un sistema de mensajería o una cola de mensajes. Estos son enviados de manera asíncrona, lo que significa que el emisor no espera una respuesta inmediata del receptor, en su lugar, el emisor continúa con su flujo de trabajo sin bloquearse mientras el receptor procesa el mensaje en su propio tiempo.

El patrón de mensajería asíncrona ofrece varias ventajas:
Desacoplamiento: Los componentes pueden comunicarse de manera indirecta a través de mensajes, lo que reduce el acoplamiento entre ellos. Cada componente solo necesita conocer la interfaz del mensaje, en lugar de depender directamente de la implementación y ubicación del receptor.
Escalabilidad: Al utilizar colas de mensajes, el sistema puede manejar grandes volúmenes de mensajes y distribuir la carga de trabajo de manera eficiente. Se pueden agregar más instancias de los receptores para procesar los mensajes en paralelo, lo que permite una mejor escalabilidad horizontal.
Tolerancia a fallos: Si un componente emisor no puede entregar un mensaje al receptor en un momento dado, el mensaje se almacena en la cola hasta que el receptor esté disponible para procesarlo. Esto brinda una mayor tolerancia a fallos y evita la pérdida de mensajes.
Integración flexible: Este patrón facilita la integración de diferentes sistemas y tecnologías. Los componentes pueden estar implementados en diferentes lenguajes de programación, utilizar diferentes plataformas o incluso estar ubicados en diferentes servidores, siempre que puedan interactuar a través del sistema de mensajería común.

Patrón de resiliencia

Es un enfoque de diseño que busca mejorar la capacidad de un sistema para resistir y recuperarse de fallos y perturbaciones. Este patrón se centra en garantizar que el sistema pueda mantener un nivel aceptable de funcionamiento incluso en presencia de errores, fallas en componentes o condiciones adversas.

El objetivo principal del patrón de resiliencia es lograr que el sistema sea resistente y capaz de adaptarse a las situaciones difíciles. Para lograr esto, se aplican diferentes técnicas y prácticas, como la redundancia, el aislamiento de fallos, la tolerancia a fallos y la capacidad de respuesta.

Algunas estrategias y prácticas comunes en el patrón de resiliencia incluyen:

Redundancia: Consiste en duplicar componentes o sistemas clave para garantizar que, si uno de ellos falla, haya otros disponibles para tomar su lugar. Esto puede incluir réplicas de servidores, bases de datos, servicios o incluso sistemas completos.

Aislamiento de fallos: Consiste en limitar el impacto de un fallo o error en un componente o sistema, aislando y conteniendo la falla para evitar que se propague al resto del sistema. Esto se logra mediante la separación de componentes en unidades más pequeñas y la aplicación de barreras de seguridad y protección.

Tolerancia a fallos: Consiste en diseñar el sistema de manera que pueda continuar funcionando a pesar de la presencia de fallos. Esto puede implicar el uso de mecanismos de recuperación automática, como el reintento de operaciones fallidas o la conmutación por error a componentes de respaldo.

Capacidad de respuesta: Consiste en diseñar el sistema para que pueda adaptarse y responder de manera adecuada frente a condiciones adversas. Esto puede incluir el ajuste de la capacidad de procesamiento, la gestión de recursos y la priorización de tareas críticas durante situaciones de alta demanda o estrés.

Supervisión y gestión de fallas: Implica la implementación de mecanismos de monitoreo continuo del sistema para detectar y registrar fallos, errores o comportamientos anómalos. Esto permite identificar problemas rápidamente y tomar medidas para mitigarlos.

La aplicación de este patrón implica un enfoque proactivo en la identificación de posibles puntos de falla y la implementación de estrategias adecuadas para mitigarlos. Además, la resiliencia debe ser considerada en todas las etapas del ciclo de vida del sistema, desde el diseño y desarrollo hasta la implementación y operación.

Patrón de DevOps y Despliegue Continuo

Es un enfoque de desarrollo y operaciones que busca mejorar la colaboración, la eficiencia y la calidad en el ciclo de vida de desarrollo de software. Este patrón se centra en la integración continua, la entrega y el despliegue continuos, y promueve la automatización de tareas, la comunicación constante y la mejora continua del proceso de desarrollo.

El objetivo principal del patrón de DevOps y despliegue continuo es lograr una entrega de software rápida y confiable, garantizando la calidad del producto final y reduciendo los errores y problemas en producción. Esto se logra mediante la integración estrecha de los equipos de desarrollo y operaciones, y la automatización de tareas repetitivas y propensas a errores.

Algunos conceptos clave del patrón de DevOps y despliegue continuo incluyen:

Integración continua (CI): Es el proceso de combinar y probar regularmente los cambios realizados por los desarrolladores en un repositorio compartido. La integración continua ayuda a detectar errores y conflictos tempranamente, evitando la acumulación de problemas a medida que avanza el desarrollo.

Entrega continua (CD): Es la capacidad de entregar software listo para producción de manera rápida y confiable. Implica la automatización de pruebas, construcción, empaquetado y liberación del software para asegurar que esté en un estado adecuado para ser desplegado en cualquier momento.

Despliegue continuo: Es la práctica de desplegar automáticamente el software en entornos de producción o de prueba una vez que pasa exitosamente por el proceso de entrega continua. Esto permite una entrega inmediata de las nuevas características o correcciones a los usuarios finales, reduciendo los tiempos de espera y acelerando la retroalimentación.

Automatización: Es un elemento fundamental del patrón de DevOps y despliegue continuo. Se refiere a la utilización de herramientas y scripts para automatizar tareas como compilación, pruebas, despliegue y configuración de infraestructura. La automatización permite ahorrar tiempo, minimizar errores y mejorar la consistencia en el proceso de desarrollo.

Monitoreo y retroalimentación: Es necesario tener un enfoque continuo de monitoreo y recopilación de información sobre el desempeño del software en producción. Esto permite detectar problemas o cuellos de botella, y brinda retroalimentación valiosa para mejorar el proceso de desarrollo y tomar decisiones informadas.

Las ventajas del patrón de DevOps y despliegue continuo incluyen:

Mayor velocidad y agilidad: Permite una entrega rápida y frecuente de nuevas funcionalidades y correcciones de errores a los usuarios finales.

Reducción de errores: La automatización y las pruebas continuas ayudan a identificar y solucionar problemas tempranamente, reduciendo errores en producción.

Mejora de la calidad: La integración continua y las pruebas automatizadas garantizan la calidad del software entregado.

Colaboración mejorada: La estrecha colaboración entre los equipos de desarrollo y operaciones fomenta una comunicación constante y una comprensión compartida de los requisitos y objetivos.

Retroalimentación rápida: El monitoreo en tiempo real y la retroalimentación continua permiten realizar mejoras y ajustes de forma rápida y eficiente.

3.5. Frameworks:

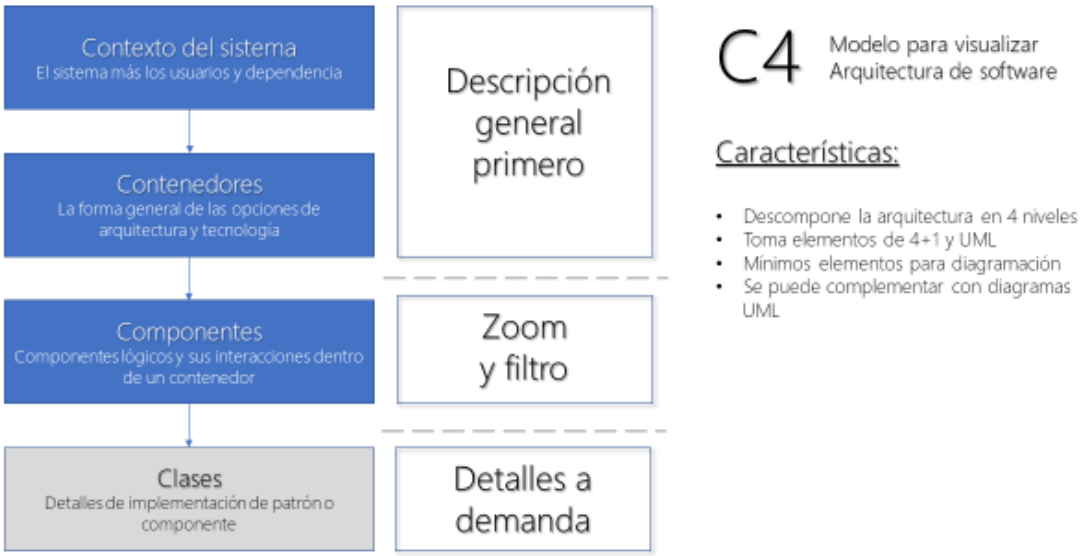
Frente Android / Huawei: Arquitectura Clean – MVVM – ROCKET.- Se compone de micro-cores en el que se delegan partes del desarrollo en las distintas capas del modelo Clean.

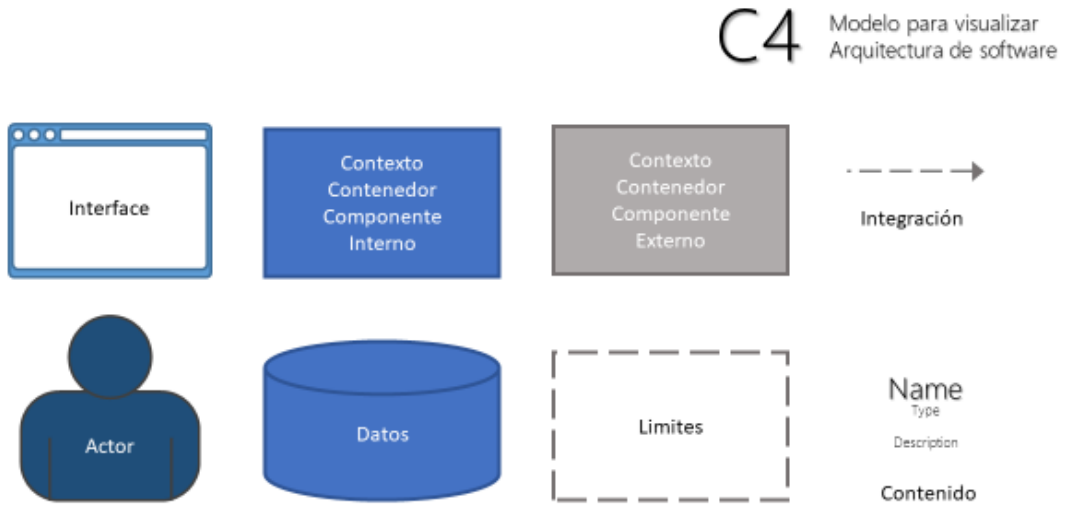
Frente iOS.- SwiftUI+, basada en Arquitectura CLEAN.

Frente FrontEndWeb.- Arquitectura de Microfrontends, usando Angular V.14.10

Arquitectura BackEnd.- Quarkus – Java. Arquitectura de Microservicios y serverless. Java 17 y Quarkus 3.

4 ARQUITECTURA C4





C4 : Notación del modelo

Dentro de los Diagramas C4, en C3 los Componentes (apis y microservicios) se presentarán organizados en las diferentes capas definidas para los NCD, estas son:

Nro	Capa	Componente
1	**API EXPERIENCIA**	Es la capa más expuesta de nuestra implementación de backend. Aquí se encuentran los distintos endpoints que se están publicando en el API Management de Experiencia. Tiene 2 subcapas: Endpoints que se publican en el API Management Microservicios de experiencia o BFF(back for front); por cada uno de los endpoints se tiene 2 versiones: app y web
2	**API CORE**	En esta capa se encuentran los endpoints de CORE expuestos a través del API Management de CORE. Tiene 3 subcapas: Apis CORE.- En esta etapa se encuentran los endpoints del CORE expuestos a través del API Management de CORE Microservicios de Negocios: MS BS.- Aquí están los microservicios del negocio o BS(business), donde se aplica toda la lógica de negocio que se indica en los requerimientos funcionales. Microservicios de Integración: MS IS.- Aquí se encuentran los microservicios de integración o IS.

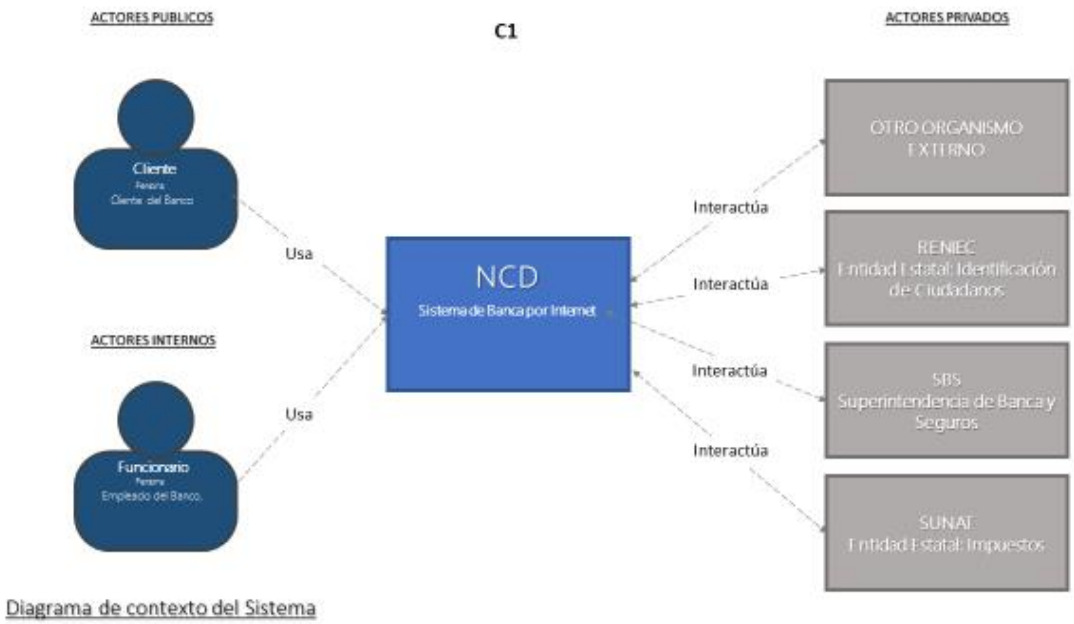
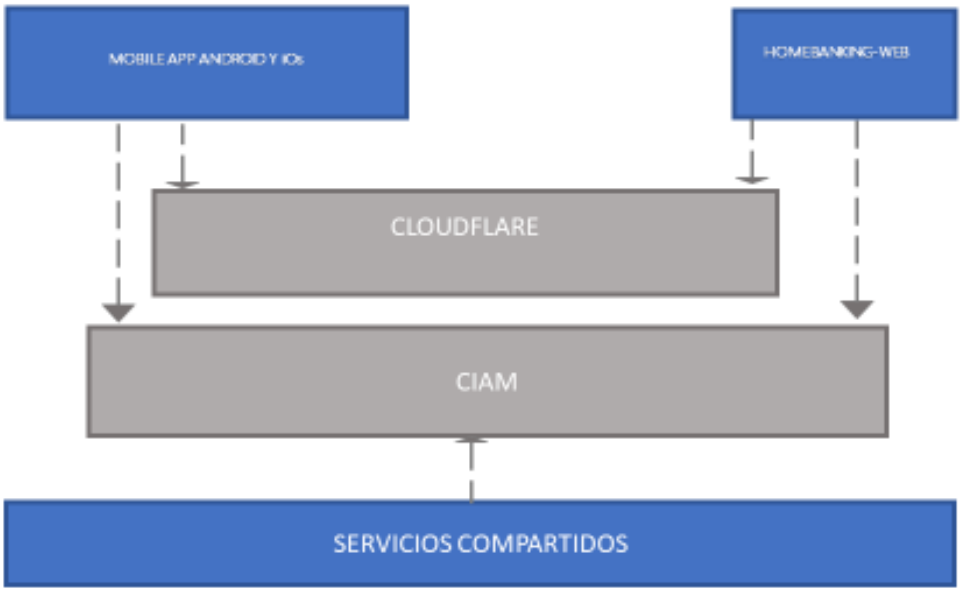
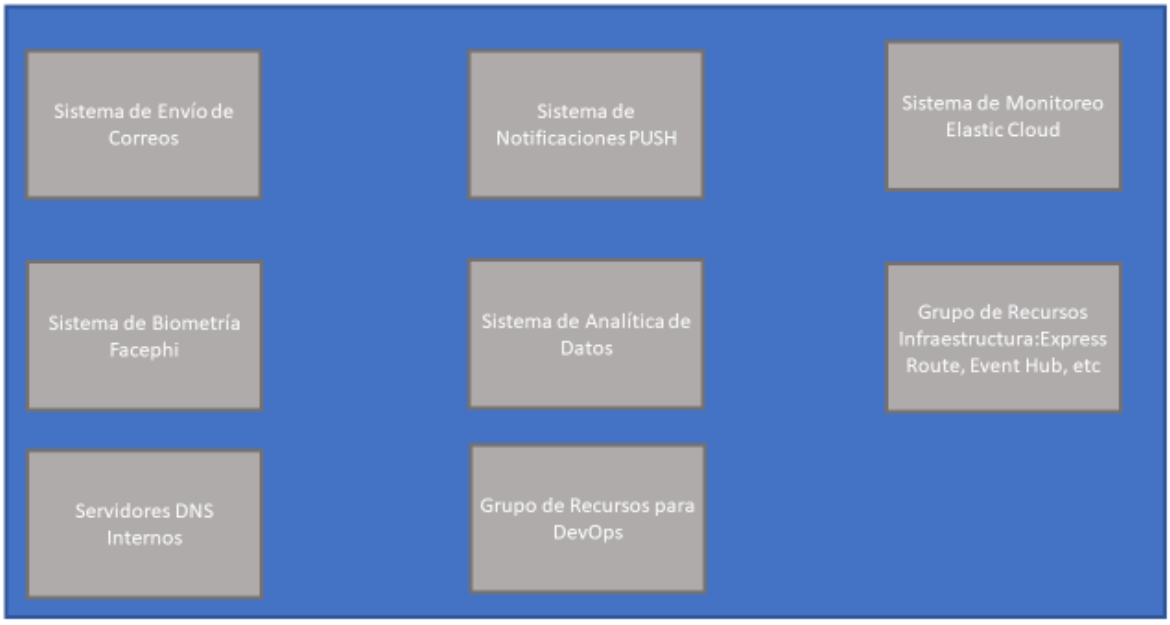


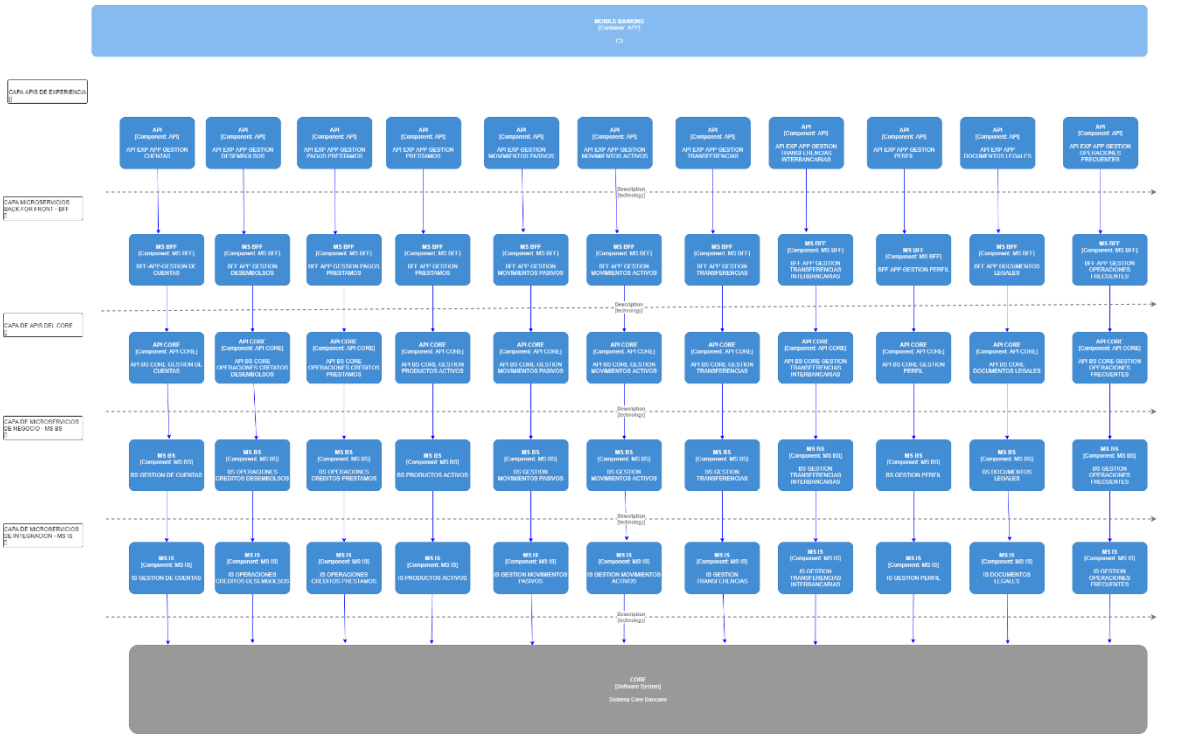
Diagrama de contexto del Sistema

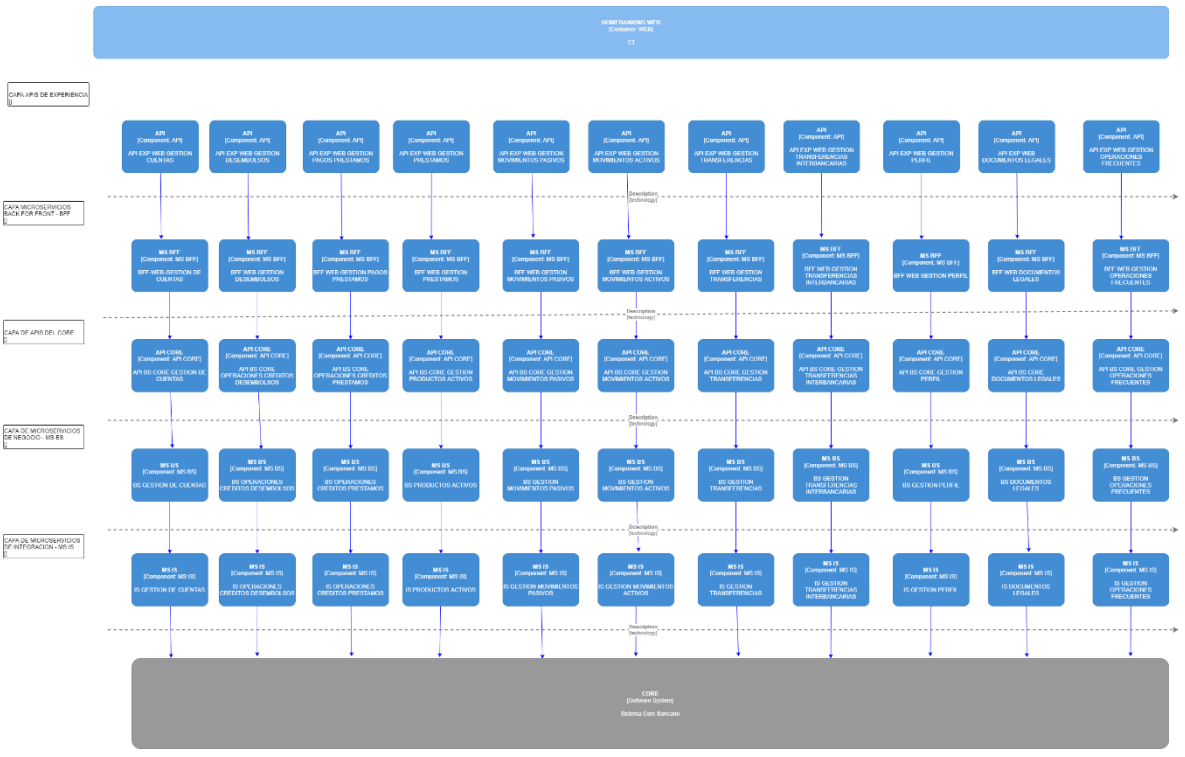
C2



C2 SERVICIOS COMPARTIDOS




MOBILE BANKING
[Container: APP]
C3
CAPA APIS DE EXPERIENCIA
CAPA MICROSERVICIOS BACK FOR FRONT - BFF
CAPA DE APIS DEL CORE
CAPA DE MICROSERVICIOS DE NEGOCIO - MS BS
CAPA DE MICROSERVICIOS DE INTEGRACION - MS IS
CORE
[Software System]
Sistema Core Bancario


HOMITBANKING WEB
[Container: WEB]
C3
CAPA APIS DE EXPERIENCIA
CAPA MICROSERVICIOS BACK FOR FRONT - BFF
CAPA DE APIS DEL CORE
CAPA DE MICROSERVICIOS DE NEGOCIO - MS BS
CAPA DE MICROSERVICIOS DE INTEGRACION - MS IS
CORE
[Software System]
Sistema Core Bancario