

Ejercicio Práctico - Arquitecto de Integración

Ejercicio práctico de arquitectura de integración

Crea un documento PDF con la respuesta al ejercicio.

El documento debe estar bien organizado y ser fácil de leer.

Cada uno de los diagramas C4 es un resultado importante. Asegúrate de hacerlos correctos y detallados. Asegúrese de agregar cualquier texto explicativo que considere necesario.

Cargue el documento en respuesta a este ejercicio. Asegúrate de subirlo dentro del tiempo de resolución.

Además, cree un repositorio público en Github y cargue el PDF en ese repositorio. Coloque la URL del repositorio en los comentarios de este ejercicio.

Descripción:

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

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

Núcleo bancario tradicional existente, más un nuevo núcleo bancario digital.

Nuevo sistema de banca web y banca móvil.

Plataforma de servicios de pago.

APIs para servicios de terceros (Open Finance).

Sistema de gestión de riesgos.

Sistema de prevención de fraude.

Tareas:

Diseñar una arquitectura de integración de alto nivel.

Especificar estándares de integración y tecnologías a utilizar.

Abordar los requisitos de seguridad, el cumplimiento normativo y la protección de datos personales.

Proponer una estrategia para garantizar la alta disponibilidad y la recuperación ante desastres.

Proponer una estrategia de integración multinúcleo.

Describir un enfoque para la gestión de identidades y accesos en todos los sistemas.

Diseñar una estrategia de API interna y externa, siguiendo los estándares de la industria en relación a la mensajería.

Proponer un modelo de gobernanza para APIs y microservicios.

Proponer un plan de migración gradual que minimice el riesgo operativo.

Consideraciones:

Garantizar alta disponibilidad (HA) y tolerancia a fallas (DR), seguridad, monitoreo.

Si es necesario, su diseño de integración puede contener elementos de infraestructura en la nube como Azure o AWS; Administrador de API. Debe garantizar una baja latencia.

El diseño debe estar bajo C4 (diagrama de contexto, contenedores, componentes). Se recomienda anotar al nivel de componente.

Considere el uso de eventos en el diseño, siguiendo las pautas de la industria.

Entregables:

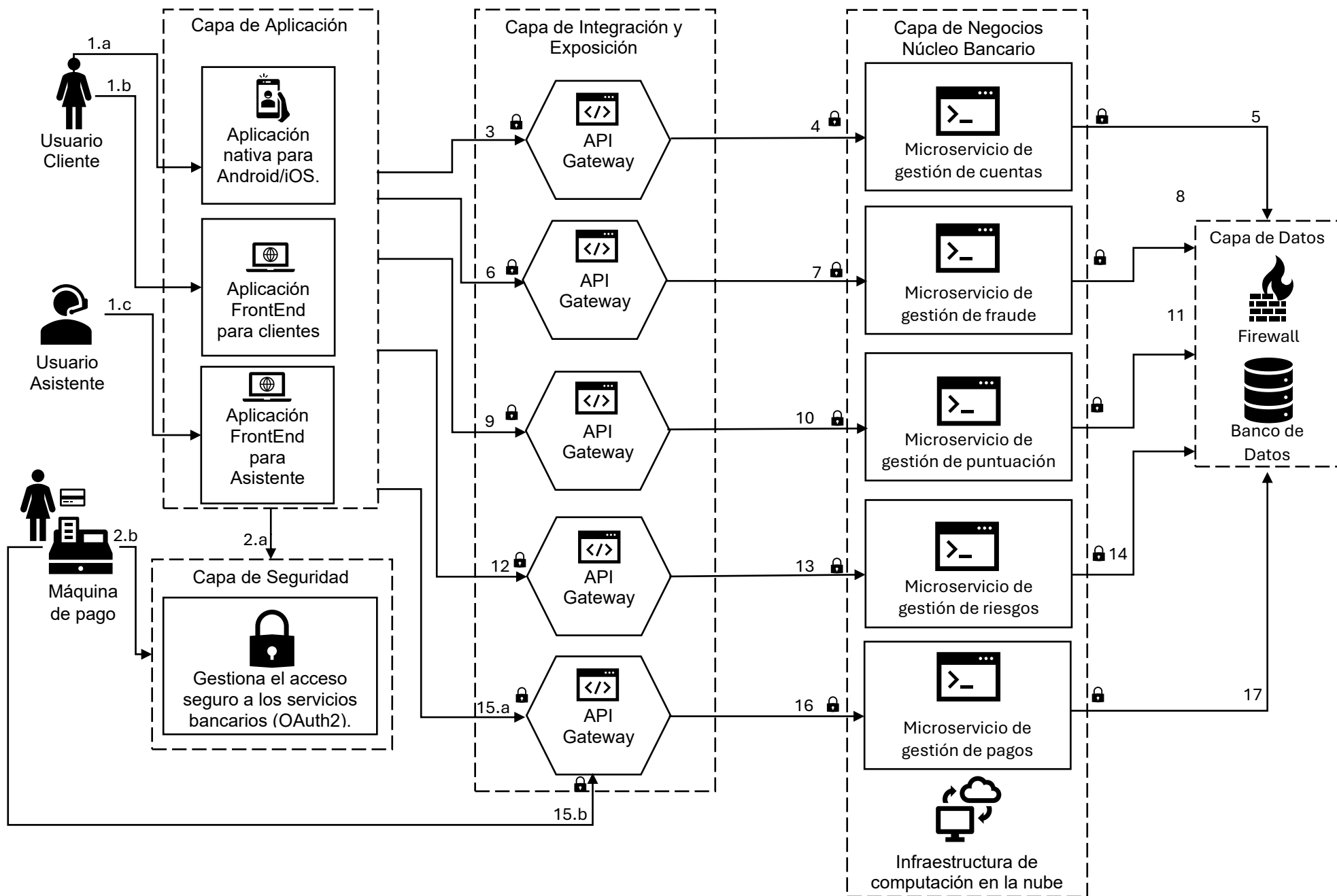
Diagramas C4

El aspirante podrá aportar los insumos necesarios para justificar la propuesta.

Repositorio GIT

Link: <https://github.com/manfrinva/devsu-challenge/tree/main>

Diagrama 4C – Solución de alto nivel



Descripción de los Elementos del Diagrama 4C – Solución Completa

El diagrama de la imagen representa una arquitectura de integración moderna para un sistema bancario, organizada en varias capas para garantizar escalabilidad, seguridad y eficiencia en la comunicación entre los diferentes componentes del sistema. El Diagrama está muy enfocado además de a la seguridad en la reutilización de aplicaciones.

Capas:

Capa de aplicación (responsable por la interacción con el usuario)

Aplicación Nativa para Android/iOS: Interfaz móvil que utilizan los clientes del banco para realizar operaciones bancarias, como consulta de saldo, transferencias, etc.

Aplicación FrontEnd para Clientes: Interfaz web accesible vía navegador, que permite a los clientes realizar operaciones bancarias en línea.

Aplicación FrontEnd para Asistente: Interfaz específica para asistentes u operadores de bancos, probablemente utilizada para atención y soporte al cliente.

Los usuarios interactúan: Diferentes tipos de usuarios (clientes, asistentes, máquinas de pago) interactúan con estas aplicaciones para acceder a los servicios bancarios.

Capa de seguridad (cobertura de seguridad)

Gestión de acceso seguro (OAuth): este componente es responsable de la autenticación y autorización del usuario, garantizando que solo los usuarios autorizados puedan acceder a los servicios bancarios. Proporciona seguridad para las transacciones y la comunicación entre aplicaciones y servicios BackEnd.

Capa de Integración y Exposición (API Gateway)

API Gateway: sirve como punto de entrada central para todas las solicitudes provenientes de aplicaciones. Controla la exposición de los servicios internos, enruta solicitudes, media protocolos y hace cumplir las políticas de seguridad. Se muestran varias puertas de enlace, cada una probablemente gestionando diferentes tipos de servicios o flujos de datos.

Capa de Negocios - Núcleo bancario (Microservicio)

Microservicio de Gestión de Cuentas: Maneja operaciones relacionadas con cuentas bancarias, como apertura, cierre, consultas de saldo y transacciones.

Microservicio de gestión de fraude: monitorea las transacciones en tiempo real para detectar y prevenir actividades fraudulentas.

Microservicio de gestión de puntajes: probablemente implique análisis de crédito y evaluación de riesgos para los clientes, asignando puntajes de crédito según el historial de transacciones.

Microservicio de Gestión de Riesgos: Evalúa y monitorea los riesgos en las operaciones bancarias, generando alertas e informes para su cumplimiento y mitigación.

Microservicio de gestión de pagos: procesa transacciones de pago, incluidas transferencias entre cuentas, pagos de facturas, etc.


Capa de datos:

Base de datos: almacena toda la información necesaria, incluidos detalles de la cuenta, transacciones, registros de actividad y datos de cumplimiento. Sirve como fuente central de datos para todos los microservicios bancarios centrales.

Flujo de datos y comunicación

Conexiones y Comunicación: Las flechas indican el flujo de datos y la comunicación entre las diferentes capas y componentes.

Todas las llamadas son sincrónicas.

El uso de íconos de candado () indica que las comunicaciones son seguras, como se espera en una arquitectura bancaria.

Explanación de las llamadas:

1.a - El Usuario/Cliente accede a la aplicación del banco a través de una aplicación de teléfono celular.

1.b - El Usuario/Cliente accede a la aplicación del banco a través del sitio web informático.

1.c - El Usuario/Asistente accede a la aplicación del banco a través del sistema informático.

2.a: la capa de aplicación se autentica en la capa de seguridad y recibe un token sistémico que se propagará.

2.b - La máquina de pago con tarjeta se autentica en la capa de seguridad y recibe un token del sistema que se propagará.

3 - Llamadas de aplicaciones FrontEnd autenticadas Gestión de cuentas API de puerta de enlace de exposición de microservicios

4 - API Gateway llama al microservicio de administración de cuentas para realizar operaciones (GET o POST)

5 - Microservicio de Gestión de Cuentas llama a la Base de Datos para ejecutar el procedimiento según una de las operaciones (GET o POST). Estar autorizado por el firewall.

6: La aplicación FrontEnd autenticada llama a la puerta de enlace API de exposición al microservicio de gestión de fraude.

7 - API Gateway llama al Microservicio de Gestión de Fraude para realizar operaciones (GET, PUT o POST).

8 - Microservicio de Fraude de Cuenta llama a la Base de Datos para ejecutar el procedimiento según una de las operaciones (GET, PUT o POST). Estar autorizado por el firewall.

9: la aplicación FrontEnd autenticada llama a la API de puerta de enlace de exposición del microservicio de Gestión de Puntuaciones.

10 - API Gateway llama al microservicio de gestión de puntajes para realizar operaciones (GET, PUT o POST).

11 - Microservicio Gestión de Puntuaciones llama a la Base de Datos para ejecutar el procedimiento según una de las operaciones (GET, PUT o POST). Estar autorizado por el firewall.

12: La aplicación FrontEnd autenticada llama a la API de puerta de enlace de exposición de microservicios de gestión de riesgos.

13 - API Gateway llama al Microservicio de Gestión de Riesgos para realizar operaciones (GET, PUT o POST).

14 - Microservicio de Riesgo de Cuenta llama a la Base de Datos para ejecutar el procedimiento según una de las operaciones (GET, PUT o POST). Estar autorizado por el firewall.

15.a - Llamadas de aplicaciones FrontEnd autenticadas API de puerta de enlace de exposición de microservicios de gestión de pagos

15.b: La máquina de pago con tarjeta, cuando se autentica, llama a la puerta de enlace API de exposición al microservicio de gestión de pagos (solo operación POST).

16 - API Gateway llama al Microservicio de Gestión de Pagos para realizar operaciones (GET o POST)

17 - Microservicio de Gestión de Pagos llama a la Base de Datos para ejecutar el procedimiento según una de las operaciones (GET o POST). Estar autorizado por el firewall.

Estrategias de Implementación:

Cifrado de datos: implemente el cifrado en reposo (datos almacenados) y en tránsito (datos transmitidos) utilizando estándares como AES-256 y TLS 1.3 para proteger datos personales y confidenciales.

Monitoreo y auditoría: implemente un registro detallado y un monitoreo continuo de las actividades para la detección y auditoría de anomalías.

Firewalls y segmentación de red: Implemente firewalls para proteger contra el acceso no autorizado y segmente la red para aislar los sistemas críticos.

Abstracción de capa API: uso de capas de abstracción API para facilitar la comunicación entre el núcleo bancario tradicional y el nuevo núcleo digital, asegurando que ambos puedan coexistir y compartir información.

API RESTful para comunicación interna: utilice API RESTful para la comunicación entre microservicios, garantizando interoperabilidad y fácil integración.

Monitoreo constante de Microservicios para garantizar el mantenimiento de la estructura de la nube. Uso de agregar más recursos de memoria y PODS si es necesario para mantener la Capa de Negocios. Utilice contenedores (Docker) y orquestadores (Kubernetes) para garantizar alta disponibilidad, escalabilidad automática y conmutación rápida. Uso de Service Mesh: utilice una malla de servicios (por ejemplo, Istio) para

gestionar la comunicación entre microservicios en diferentes núcleos, proporcionando enrutamiento, equilibrio de carga y seguridad.

Modelo de Gobernanza de API y Microservicios:

Registro y catálogo de API: implemente un portal para desarrolladores y un catálogo de API para la documentación, administración y descubrimiento de API.

Políticas de control de versiones de API: adopte una política de control de versiones de API para garantizar la compatibilidad con versiones anteriores y facilitar las actualizaciones sin afectar a los consumidores.

Monitoreo y análisis de API: utilice herramientas de monitoreo para analizar el uso, el rendimiento y la detección de anomalías de API.

Políticas de limitación y limitación de tarifas: implemente controles y limitaciones de tarifas para proteger contra el abuso y garantizar la disponibilidad.

Lenguajes de programación para microservicios:

Elección basada en el contexto del proyecto: el mejor lenguaje y arquitectura dependen de las necesidades específicas del proyecto, el entorno de TI existente y la experiencia del equipo de desarrollo.

Adopción de estándares de la industria: independientemente del lenguaje elegido, es esencial adoptar estándares de la industria y las mejores prácticas de seguridad, escalabilidad y mantenimiento.

Observabilidad: independientemente de la elección de arquitectura y lenguaje, implementar soluciones de monitoreo y registro (como Prometheus, Grafana y ELK Stack) es crucial para el éxito operativo.

A continuación, tenemos algunas opciones:

Java

Marcos: Spring Boot, Micronaut.

Razones para la elección:

Amplia madurez y robustez para aplicaciones empresariales.

Fuerte soporte para la integración con bibliotecas y herramientas de seguridad, monitoreo y administración.

Comunidad activa y extensa documentación.

Uso ideal: ideal para servicios que requieren alta solidez, soporte a largo plazo e integración con sistemas heredados.

Python

Marcos: Flask, FastAPI.

Razones para la elección:

Sencillez y rapidez de desarrollo.

Excelente para la creación rápida de prototipos y el desarrollo de servicios con requisitos de rendimiento menos estrictos.

Soporte para aprendizaje automático y análisis de datos, integrándose bien con herramientas de inteligencia artificial.

Uso Ideal: Microservicios que requieren agilidad de desarrollo, procesamiento de datos o integración con modelos de aprendizaje automático.

GO (Golang)

Razones para la elección:

Rendimiento altamente eficiente y bajo consumo de recursos.

Simultaneidad nativa y robusta, beneficiosa para microservicios que requieren alto rendimiento y escalabilidad.

Sencillez en el lenguaje que facilita el mantenimiento y rápido desarrollo.

Uso ideal: Servicios de alta carga y baja latencia, como sistemas de pago o procesamiento de transacciones en tiempo real.

Nodo.js

Razones para la elección:

Excelente para aplicaciones intensivas de E/S y servicios en tiempo real.

Amplia comunidad de desarrolladores y una enorme variedad de bibliotecas disponibles.

Ideal para microservicios que requieren respuesta rápida y comunicación en tiempo real (por ejemplo, chat, notificaciones).

Uso ideal: microservicios centrados en la interacción en tiempo real, aplicaciones web y API ligeras.

C# (.NET Core)

Razones para la elección:

Integrado con el ecosistema de Microsoft, ideal para organizaciones que ya utilizan tecnologías de Microsoft.

Excelente rendimiento y soporte para desarrollo multiplataforma con .NET Core.

Sólido soporte para seguridad, integración de servicios y API.

Uso ideal: Servicios de misión crítica en entornos corporativos que utilizan tecnologías de Microsoft.

Plan de Migración Gradual que Minimiza el Riesgo Operacional:

Fase de prueba de concepto (PoC): comience con una PoC para probar la integración del nuevo núcleo digital con sistemas heredados en un entorno controlado.

Migración por Módulos: Migrar funcionalidades a pequeños módulos o servicios, comenzando por aquellos de menor impacto, para facilitar la identificación y resolución de problemas.

Entorno de prueba paralelo: cree un entorno de prueba paralelo para realizar pruebas completas del nuevo sistema mientras el sistema heredado aún está en funcionamiento.

Reversión automática: implemente mecanismos de reversión automática para volver rápidamente al sistema heredado en caso de fallas durante la migración.

Capacitación y comunicación: capacite a los equipos de TI y de soporte, y comuníquese con los usuarios finales sobre los cambios, garantizando una transición sin problemas.

Consideraciones finales:

Con base en el análisis visual del diagrama C4 presentado, parece abordar adecuadamente los requisitos de la arquitectura de integración para la modernización de los sistemas bancarios, tal como se solicita en el proyecto. A continuación, se detallan algunos puntos clave que demuestran el cumplimiento de los requisitos:

Arquitectura de alto nivel y patrones de integración: el diagrama está organizado en capas, lo cual es una buena práctica para la modularidad y la comprensión. Las capas incluyen Aplicación, Integración y Exposición, Negocios (Núcleo Bancario) y Datos. Esta estructura facilita la integración de diferentes servicios y tecnologías.

Gestión de seguridad y acceso: la presencia de una "Capa de seguridad" para gestionar el acceso seguro a los servicios bancarios (utilizando OAuth2) indica que se han considerado aspectos de seguridad y cumplimiento normativo. Además, los íconos de candados se utilizan para indicar la seguridad en puntos de integración críticos.

Alta disponibilidad y recuperación ante desastres: la arquitectura sugiere redundancia y distribución de servicios (como el uso de múltiples puertas de enlace API), lo que puede contribuir a una alta disponibilidad. Sin embargo, los detalles específicos sobre la tolerancia a fallos y la recuperación ante desastres no se muestran explícitamente en el diagrama.

Integración multinúcleo y microservicios: el diagrama hace uso de microservicios para diferentes funciones bancarias, como gestión de cuentas, fraude, puntaje, riesgos y pagos, lo que facilita la escalabilidad y el mantenimiento. Este enfoque está en línea con la tendencia hacia la integración de sistemas modernos con capacidades de procesamiento multinúcleo.

Gobernanza de API y microservicios: la inclusión de múltiples puertas de enlace API sugiere un marco para gestionar la exposición y la integración de API de forma controlada, un aspecto importante para la gobernanza y la seguridad.

Migración gradual y minimización de riesgos operativos: aunque el diagrama muestra la integración de sistemas nuevos y existentes, no hay detalles explícitos sobre el plan de migración gradual o cómo se minimizarán los riesgos operativos durante la transición. Este aspecto puede requerir documentación adicional o planes de migración detallados.

Uso de infraestructura en la nube: la presencia de un ícono de infraestructura de computación en la nube sugiere que el diseño considera el uso de soluciones en la nube como Azure o AWS, lo cual es beneficioso para la escalabilidad y la flexibilidad.

Estas estrategias y enfoques deben incorporarse al diseño arquitectónico, utilizando componentes y servicios modernos que respalden las mejores prácticas de seguridad, alta disponibilidad y un enfoque escalonado de la migración. La implementación cuidadosa de estos principios garantizará que la modernización del sistema bancario se produzca de manera segura, eficiente y con un riesgo operativo mínimo.