Arquitectura de Soluciones

CONFIDENCIAL

[Fecha 10/09/2024]

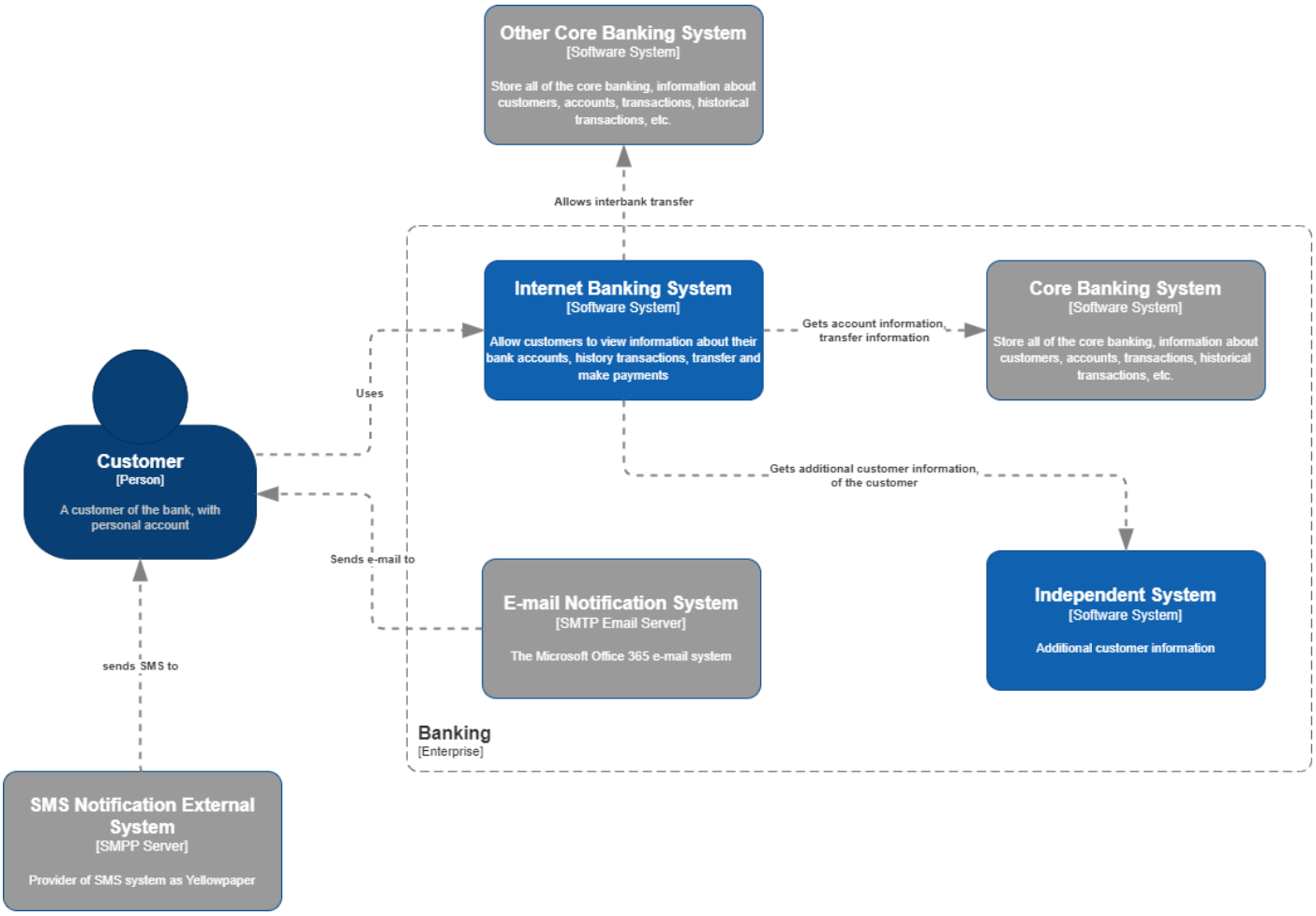
Versión [1.0]

Contenido

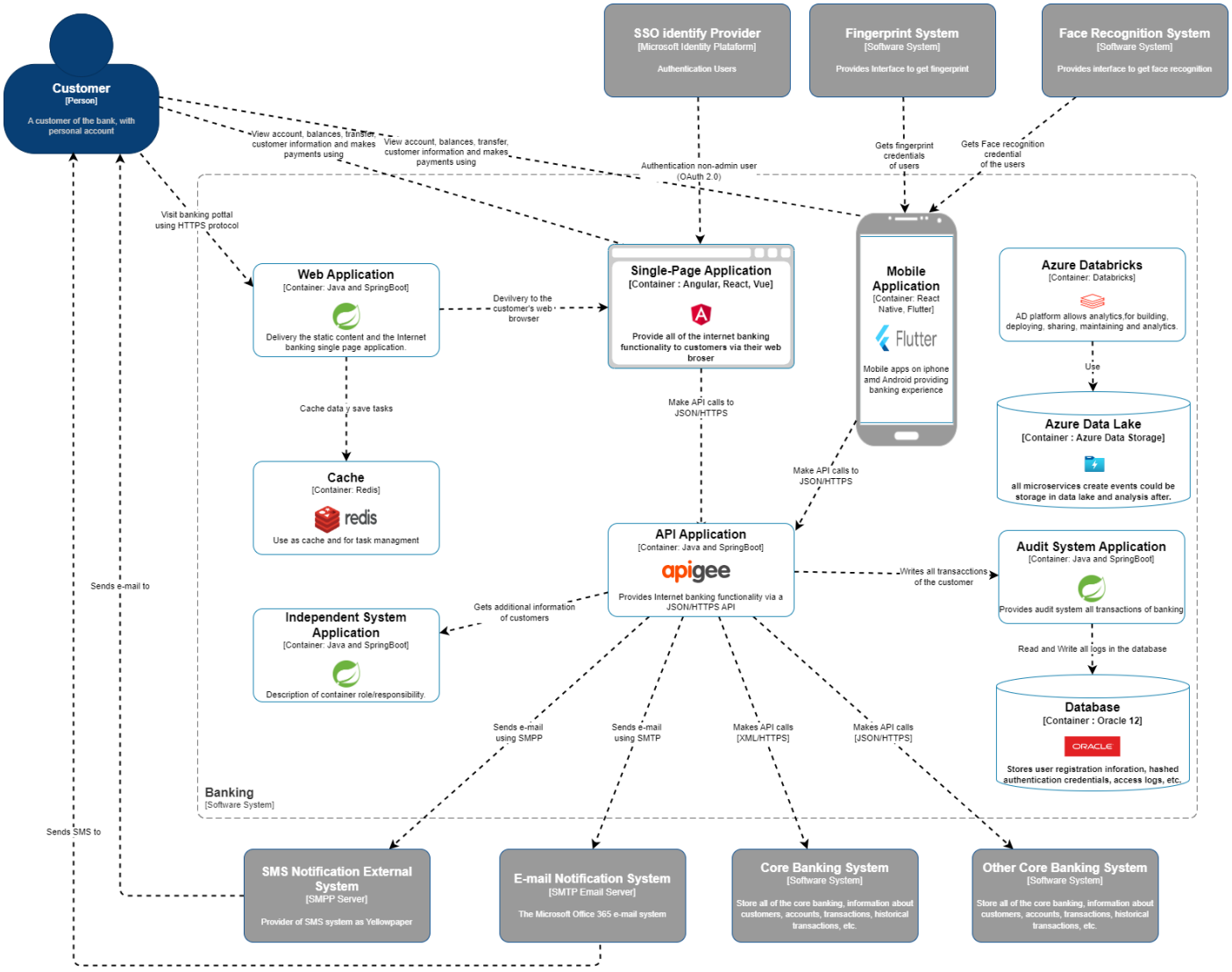
1. Diseñar una arquitectura de integración de alto nivel

Se realizo el modelado c4 en base al scope de la solución en base a los siguientes diagramas.

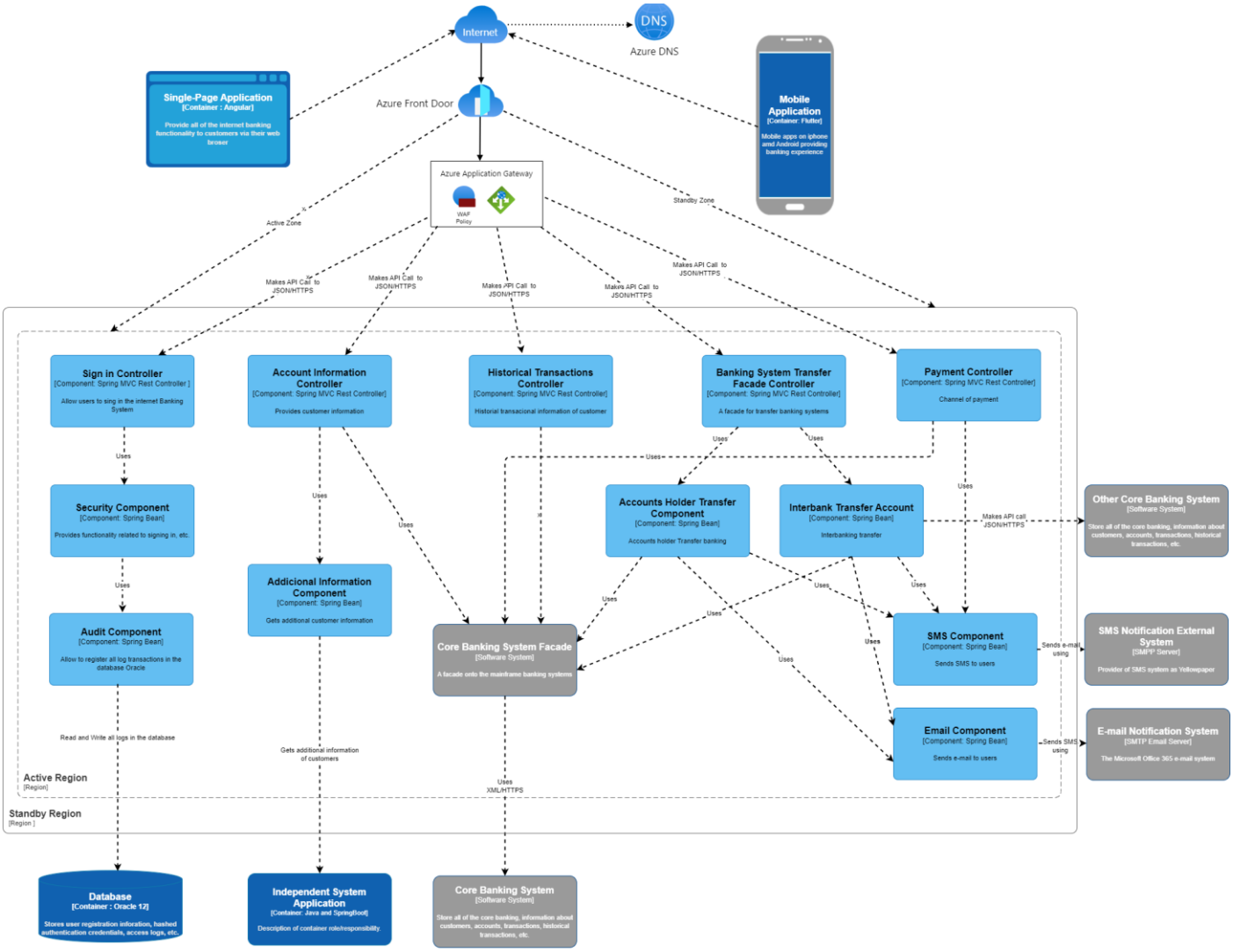
1.1. Contexto



1.2. Contenedores



1.3. Componentes



2. Uso de patrones de arquitectura

Se implementaron los siguientes patrones de arquitectura:

2.1. Data Managment

- **Index Table. -** Cree índices sobre los campos de los almacenes de datos a los que las consultas hacen referencia con frecuencia, Este patrón puede mejorar el rendimiento de las consultas al permitir que las aplicaciones localicen con mayor rapidez los datos que se recuperarán de un almacén de datos.
- **Materialized View**. - Generar vistas precargadas de los datos en uno o más almacenes de datos cuando los datos no tengan el formato ideal para las operaciones de consulta requeridas. Esto puede ayudar a respaldar la extracción de datos y consultas eficientes, y mejorar el rendimiento de la aplicación.

- **Ani-Corruption Layer** – Permite implementar una **facade** capa entre aplicaciones modernas y sistemas legados, que no comparten la misma semántica, Esta capa traduce las solicitudes que un subsistema realiza al otro subsistema. Utilice este patrón para garantizar que el diseño de una aplicación no esté limitado por dependencias de subsistemas externos. Este patrón fue descrito por primera vez por Eric Evans en Domain-Driven Design, por ejemplo, si se quiere consumir un microservicio a un sistema legado.
- **Valet Key**. - Utilice un token que proporcione a los clientes acceso directo restringido a un recurso específico, para descargar la transferencia de datos de la aplicación. Esto es particularmente útil en aplicaciones que utilizan sistemas de almacenamiento alojados en la nube o colas, y puede minimizar los costos y maximizar la escalabilidad y el rendimiento.
- **Event Sourcing. -** En lugar de almacenar solo el estado actual de los datos en un dominio, utilice un almacén de solo anexión para registrar la serie completa de acciones realizadas sobre esos datos. El almacén actúa como el sistema de registro y se puede utilizar para materializar los objetos del dominio. Esto puede simplificar las tareas en dominios complejos, al evitar la necesidad de sincronizar el modelo de datos y el dominio comercial, al tiempo que mejora el rendimiento, la escalabilidad y la capacidad de respuesta. También puede proporcionar coherencia a los datos transaccionales y mantener registros de auditoría completos e historial que pueden permitir acciones compensatorias.

2.2. Design and Implementation

- **Backends for Frontends**. – Cree servicios backend independientes para que los utilicen aplicaciones frontend o interfaces específicas. Este patrón es útil cuando desea evitar personalizar un solo backend para múltiples interfaces.
- **CQRS.** - CQRS significa Command and Query Responsibility Segregation, un patrón que separa las operaciones de lectura y actualización de un almacén de datos. Implementar CQRS en su aplicación puede maximizar su rendimiento, escalabilidad y seguridad. La flexibilidad creada al migrar a CQRS permite que un sistema evolucione mejor con el tiempo y evita que los comandos de actualización causen conflictos de fusión a nivel de dominio.

2.3. Messaging

- **Asynchronous Request-Reply**. – Desacoplar el procesamiento del backend de un host frontend, donde el procesamiento del backend debe ser asincrónico, pero el frontend aún necesita una respuesta clara, ejemplo de aplicación es la localización geográfica.
- **Choreography** . – Lógica de flujo de trabajo descentralizada y distribución de responsabilidades a otros componentes dentro de un sistema, una aplicación basada en la nube a menudo se divide en varios servicios pequeños que trabajan juntos para procesar una transacción comercial de end-to-end .
- **Priority Queue**. – El patrón de cola de prioridad permite que una carga de trabajo procese tareas de alta prioridad más rápidamente que las tareas de menor prioridad. Este patrón utiliza mensajes enviados a una o más colas y es útil en aplicaciones que ofrecen diferentes garantías de nivel de servicio a clientes individuales.

- **Publisher & Subscriber**. - Permite que una aplicación anuncie eventos a varios consumidores interesados de forma asincrónica, sin acoplar los remitentes con los receptores. También se denomina: mensajería de publicación/suscripción.
- **Saga distributed transactions**. - El patrón de diseño Saga es una forma de gestionar la coherencia de los datos entre microservicios en escenarios de transacciones distribuidas. Una saga es una secuencia de transacciones que actualiza cada servicio y publica un mensaje o evento para activar el siguiente paso de la transacción. Si un paso falla, la saga ejecuta transacciones compensatorias que contrarrestan las transacciones anteriores.

2.4. Network Security

- **Gateway routing pattern**: Dirigir solicitudes a múltiples servicios o instancias de servicio que pueden residir en diferentes regiones.
- **Gateway offloading pattern**: Descargue funcionalidades, como la mitigación de ataques, a un proxy de puerta de enlace.
- **Health endpoint monitoring pattern**: Exponer puntos finales que validen el estado de la carga de trabajo.

2.5. Reliability

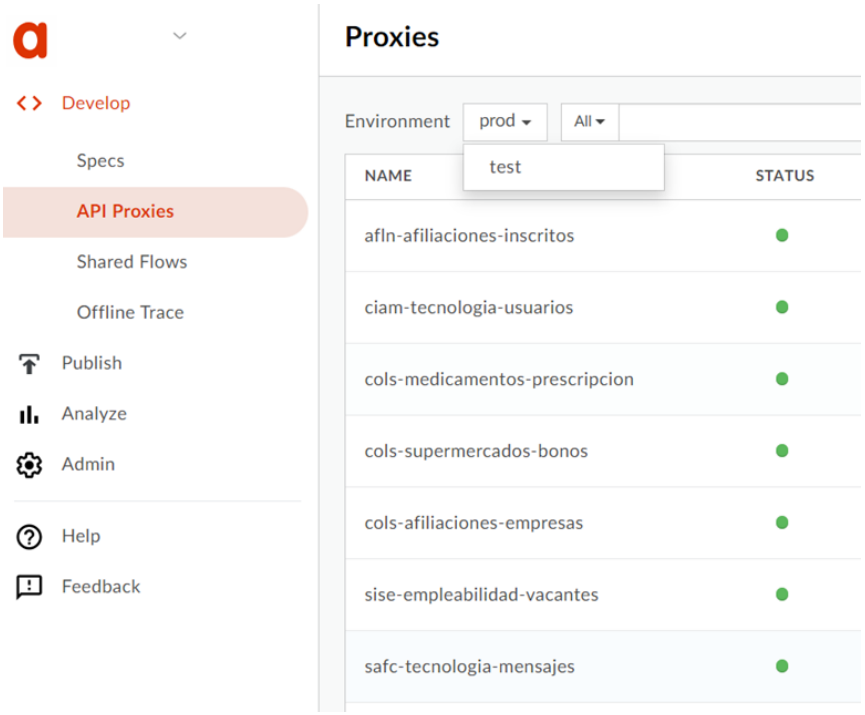
- **Circuit breaker**. – Gestionar errores que pueden tardar un tiempo variable en solucionarse al conectarse a un servicio o recurso remoto. Esto puede mejorar la estabilidad y la resiliencia de una aplicación.
- **Compensating Transaction**. - Cuando se utiliza una operación de consistencia eventual que consta de una serie de pasos, el patrón de transacción compensatoria puede resultar útil. En concreto, si uno o más de los pasos fallan, se puede utilizar el patrón de transacción compensatoria para deshacer el trabajo que los pasos realizaron. Normalmente, se encuentran operaciones que siguen el modelo de consistencia eventual en aplicaciones alojadas en la nube que implementan procesos y flujos de trabajo empresariales complejos.

2.6. Security

- **Federated Identity**. - Delegar la autenticación a un proveedor de identidad externo. Esto puede simplificar el desarrollo, minimizar los requisitos de administración de usuarios y mejorar la experiencia del usuario de la aplicación.

3. Integración con servicios externos

Con Apigee se puede establecer tanto API internas como externas, se requiere definir los ambientes en los cuales se va publicar las apis, casos de uso en escenarios para publicar.



Estándares

- **Políticas de Seguridad**: Aplicar autenticación basada en tokens (OAuth, JWT) o claves API para garantizar que solo los servicios autorizados puedan interactuar con las APIs.
- **Rate Limiting y Throttling**: Definir límites en las solicitudes de API para evitar la sobrecarga de servicios internos.
- **Monitorización y Análisis**: Utilizar Apigee para monitorizar el uso de las APIs, detectar problemas de rendimiento y realizar análisis de patrones de uso.
- **Orquestación y Transformación**: Apigee puede orquestar la comunicación entre varios servicios internos y transformar las solicitudes/respuestas para asegurar compatibilidad entre sistemas.

APIs Internas

- **Conexión de microservicios**: Apigee puede ayudar a gestionar la comunicación entre microservicios internos, proporcionando políticas para el control de tráfico, caching, y control de errores.
- **Automatización de procesos internos**: Por ejemplo, APIs que exponen servicios internos de recursos humanos, gestión de inventario, o sistemas de facturación que son utilizados por las aplicaciones internas.

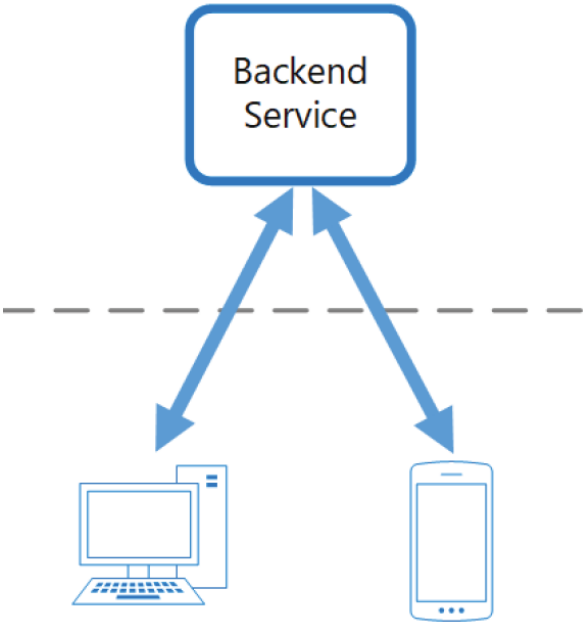
APIs Externas

- **Exposición pública**: Las APIs externas están accesibles desde Internet y generalmente requieren controles de seguridad más estrictos para proteger los datos y sistemas.
- **Autenticación y autorización robustas**: Es fundamental implementar autenticación sólida (OAuth 2.0, OpenID Connect) para identificar y controlar el acceso de los usuarios externos.

- **Escalabilidad**: Las APIs externas suelen tener una alta demanda, por lo que deben estar diseñadas para escalar automáticamente y manejar grandes volúmenes de tráfico.
- **Documentación pública**: Al estar dirigidas a desarrolladores externos, es importante proporcionar una documentación clara y herramientas de soporte (como portales de desarrolladores) para facilitar su adopción.
- **Control y gobernanza**: Las APIs públicas requieren políticas más detalladas para la limitación de tasas, acuerdos de nivel de servicio (SLAs), y monitoreo para evitar abusos o mal uso.

4. Arquitectura de aplicación fron-end móvil

Arquitectura backend-frontend, Cree servicios de backend independientes para que los utilicen aplicaciones frontend o interfaces específicas. Este patrón es útil cuando desea evitar personalizar un solo backend para múltiples interfaces. Este patrón fue descrito por primera vez por Sam Newman



A nivel de front-end utilizar un SPA en Angular, React, Vue que permite ser responsivo para cualquier dispositivo

5. Arquitectura de acceso a datos

Un ORM es una técnica que mapea objetos del lenguaje de programación a tablas en la base de datos. En .NET se utiliza el Entity Framework en el caso de Java se utiliza Hibernate, ventajas:

- Permite la abstracción de SQL
- Código más limpio y fácil de mantener.
- Soporte para transacciones, migraciones de bases de datos y cambios de esquema.

6. Conocimiento de nubes

Los componentes que se utilizan en la solución son los siguientes:

- **Azure App Services**. – Microservicios con un enfoque DDD de modelado de software y la utilización de contendedores, Creación e implementación de aplicaciones web y API.
- **Azure SQL Database**. - Almacenamiento de datos estructurados y soporte de aplicaciones web y móviles.
- **Azure Blob Storage**. – Servicio de almacenamiento de objetos para datos no estructurados.
- **Event Grid Service Bus**. – Permite el manejo de mensajería en Azure
- **Azure Functions. -** Funciones serveless que es una IaC
- **Azure Front Door. -** Es un CDN que ofrece alto rendimiento, escalabilidad y experiencias de usuario seguras para su contenido y aplicaciones, ya que es un servicio global que permite distribuir requerimientos entre regiones, como se detalla en el diagrama de componentes.
- **Azure Active Directory**. - Gestionar el acceso y la autenticación de usuarios.
- **Azure Application Gateway**. - Proporciona enrutamiento a nivel de aplicación, terminación SSL y capacidades de firewall de aplicaciones web.
- **Azure Data Lakehouse**. – El data lakehouse combina los beneficios de los data lakes y los data warehouses y proporciona: Acceso abierto y directo a los datos almacenados en formatos de datos estándar. Protocolos de indexación optimizados para el aprendizaje automático y la ciencia de datos. Baja latencia de consultas y alta confiabilidad para inteligencia empresarial y análisis avanzados.
- **Azure Databricks**. - Azure Databricks combina la potencia de Apache Spark con Delta Lake y herramientas personalizadas para brindar una experiencia ETL (extracción, transformación y carga) inigualable. Puede usar SQL, Python y Scala para crear lógica ETL y luego organizar la implementación de trabajos programados.

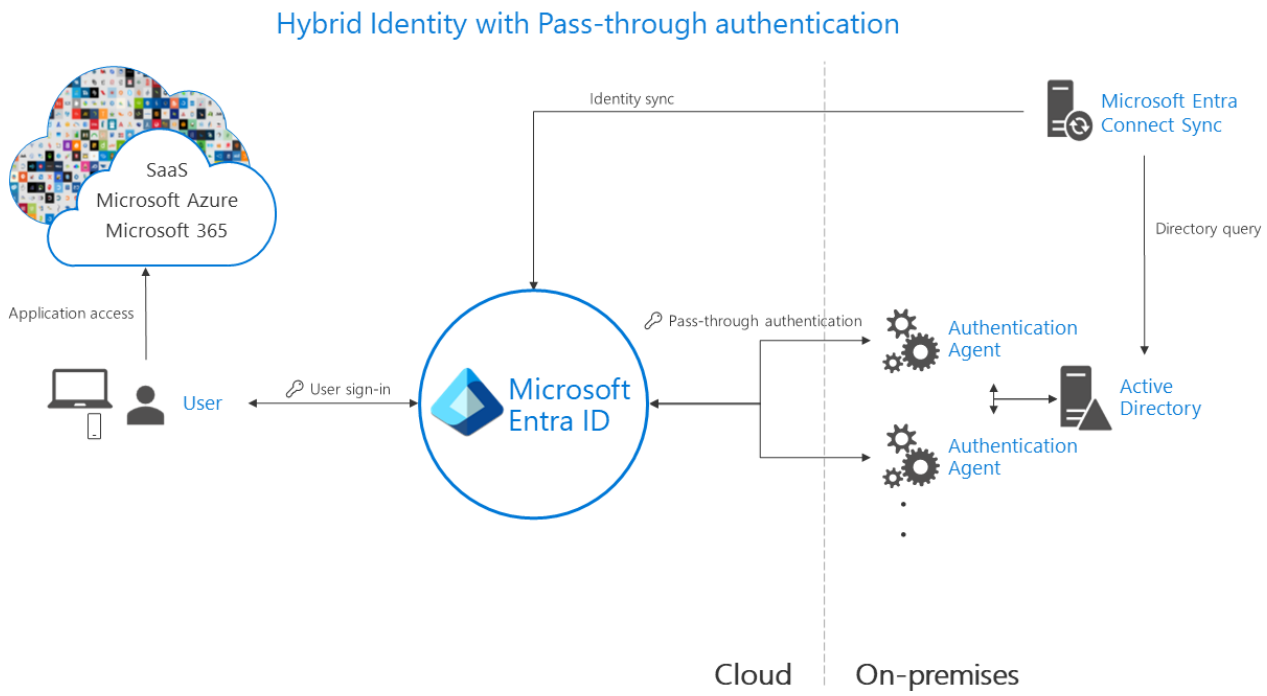
7. Arquitectura de autenticación

Se lo puede realizar con Hybrid identity with Pass-trough authentication, utilizando dos agentes para redundancia

Con Microsoft Entra ID podemos tener:

- **Autenticación y Autorización** con Single Sign-On (SSO), Autenticación multifactor (MFA), Autenticación basada en riesgos.
- **Gestión de Identidades** de usuarios y grupos, Federación de identidades, Sincronización con Active Directory On-Premise.
- **Gobierno de Identidades y Accesos** con políticas de acceso condicional, Privileged Identity Management (PIM)
- **Aplicaciones Empresariales** acceso a la Galería de aplicaciones preintegradas con SaaS, autenticación con OpenID Connect y OAuth 2.0, Acceso seguro a aplicaciones locales.
- **Protección contra Amenazas** gracias a la Identidad protegida, Supervisión continua.

Un esquema sería el siguiente.



8. Arquitectura de integración con Onboarding

Nos referimos a cómo una organización implementa y gestiona el proceso de incorporación de nuevos usuarios, empleados, clientes o proveedores dentro de su ecosistema tecnológico. Este tipo de arquitectura permite la automatización y optimización del proceso de onboarding, asegurando una experiencia fluida y eficiente para todas las partes involucradas, se debe tener en cuenta las siguientes capas:

- **Capa de Presentación (UI/UX)**. - Esta capa es la interfaz visual a través de la cual los usuarios interactúan con el sistema de onboarding. Es esencial para proporcionar una experiencia de usuario intuitiva y fluida, se puede desarrollar en Angular, Vue, React, Blazor.
- **Capa de Servicios (API Gateway)**. - Esta capa actúa como intermediario entre el front-end y los servicios de back-end. Un API Gateway gestiona las solicitudes del front-end y distribuye estas solicitudes a los microservicios adecuados o sistemas internos, se puede implementar en GCP ApiGee, Amazon API Gateway, en azure - API Managment Azure Functions Proxies
- **Microservicios para Procesos de Onboarding**. - Estos microservicios manejan diferentes aspectos del proceso de onboarding, permitiendo una modularización y desacoplamiento de las funciones, se puede implementar los siguientes patrones Event-Driven Architecture, Orquestación vs Coreografía.
- **Capa de Integración con Sistemas Externos**. - La integración con servicios de terceros es esencial para automatizar el proceso de verificación, validación y comunicación con otros sistemas externos.

- **Capa de Gestión de Procesos (BPM)** - Los sistemas de Business Process Management (BPM) permiten orquestar y gestionar los diferentes flujos del proceso de onboarding, desde la recolección de datos hasta la verificación y la activación final.
- **Capa de Gestión de Datos.** - Esta capa es responsable de almacenar y gestionar toda la información recopilada durante el proceso de onboarding, garantizando la seguridad y privacidad de los datos.
- **Capa de Seguridad.** - Garantizar que todo el proceso de onboarding cumpla con los estándares de seguridad y las normativas de privacidad (por ejemplo, GDPR, CCPA).
- **Capa de Comunicación y Notificaciones** - Maneja las interacciones con los usuarios a lo largo del proceso de onboarding, desde la bienvenida inicial hasta los recordatorios o avisos de finalización.

9. Diseño de la solución de auditoria

Se debe considerar las siguientes capas:

- **Capa de Generación de Eventos**. - Esta capa es responsable de generar eventos de auditoría en tiempo real desde las diferentes partes del sistema. Los eventos pueden ser generados tanto por los usuarios como por procesos automáticos dentro del sistema, se puede implementar rabbitmq o Azure Event Hubs.
- **Capa de Almacenamiento de Eventos (Log Store)**. - Los eventos de auditoría se deben almacenar en un sistema que asegure la integridad de los datos y permita realizar búsquedas eficientes, en bases de datos como SQL Server, Oracle, NoSQL como MongoDB, Elasticsearch, Azure Blob Storage.
- **Capa de Procesamiento de Eventos**. - Esta capa gestiona el procesamiento de los eventos registrados y realiza tareas como el enriquecimiento de datos, la normalización o incluso la detección de patrones sospechosos, se puede utilizar Azure Event Grid, Apache Kafka, Azure Data Factory.
- **Capa de Seguridad y Cifrado**. - La seguridad es clave para una solución de auditoría, ya que los datos registrados suelen ser sensibles, cifrado AES-256, hashing SHA-256.
- **Capa de Monitoreo y Detección de Anomalías**. - Es esencial implementar un sistema de monitoreo para analizar los eventos registrados y detectar comportamientos sospechosos o patrones inusuales, con Azure Machine Learning, Azure Monitor.
- **Capa de Reportes y Consultas**. - La solución de auditoría debe permitir a los usuarios consultar y generar informes detallados sobre los eventos registrados, Elasticsearch, Power BI.

10. Conocimientos de regulaciones bancarias y estándares de seguridad

- LOPD - Ley orgánica de protección de datos personales emitida el 26-mayo-2021, tratamiento de los datos sensibles y custodia de los mismos, como la difusión.
- Normas generales para las instituciones del sistema financiero (https://www.superbancos.gob.ec/bancos/codificacion-de-normas-de-la-sb-libro-uno-sistema-financiero/), las principales:
    - Se requiere doble método de autenticación un MFA, el primero usuario y clave, el segundo puede ser biométrico, reconocimiento facial, OTP, etc.

  - La información sensible no debe salir del territorio ecuatoriano, esta información son las identificaciones, nombres, etc.
- La norma ISO/IEC 27001, normativa reconocida internacionalmente para la gestión de la seguridad de la información. Permite establecer, implementar, mantener y mejorar continuamente un sistema de gestión de la seguridad de la información (SGSI)., lo importante de la normativa:
  - Protección de datos
  - Cumplimiento del regulatorio legal y requerimientos como GDPR
  - Gestión de riesgo enfocado a reducir vulnerabilidades

11. Implementación de alta disponibilidad y Tolerancias a fallos

- Se recomienda utilizar en on-prem DCA y DCP
- En la nube se requiere utiliza Azure Front Door que es un CDN que ofrece alto rendimiento, escalabilidad y experiencias de usuario seguras para su contenido y aplicaciones, ya que es un servicio global que permite distribuir requerimientos entre regiones, como se detalla en el diagrama de componentes.

A nivel de DR:

Se requiere definir un HA con:

- **Redundancia de Componentes Críticos** como servidores implementando load balancing, discos y almacenamientos implementando RAID o almacenamiento distribuidos mediante un NAS
- **Balanceo de carga** Implementar balanceadores de carga de nivel de red (DNS) y de nivel de aplicación (como NGINX o HAProxy).
- **Replicación de base de datos** SQL server con replicación nativa y oracle con data guard
- **Monitoreo continuo** con Nagios, Prometheus, o Zabbix que supervisen constantemente el estado de los sistemas y alerten sobre posibles fallos

Recuperación ante Desastres (Disaster Recovery - DR)

- Tener definido los objetivos
  - **RPO (Recovery Point Objective):** El máximo tiempo de datos que la empresa está dispuesta a perder. A menor RPO, más frecuente debe ser la replicación de datos.
  - **RTO (Recovery Time Objective):** El tiempo máximo en que los sistemas deben estar operativos tras un desastre.
- Copia de Seguridad (Backups)
- Sitios de Recuperación (DR Sites)

12. Implementación de Monitoreo

El monitoreo de aplicaciones permite supervisar el rendimiento y la integridad del software en ejecución. Aquí es donde se miden tiempos de respuesta, tasas de error y uso de recursos por parte de las aplicaciones.

Métricas clave:

- **Tiempos de respuesta**: Medición de la latencia en las respuestas del servidor.
- **Tasas de error**: Cantidad de errores HTTP (500, 404, etc.).
- **Tiempos de ejecución de consultas**: Supervisión de consultas lentas en la base de datos.
- **Tiempos de respuesta de API**: Monitorización de endpoints y sus tiempos de respuesta.
- **Transacciones y operaciones**: Seguimiento del ciclo completo de una transacción o solicitud dentro del sistema.

Herramientas comunes:

- **Elastic APM**: Proporciona monitoreo de rendimiento de aplicaciones, siendo parte del Elastic Stack.
- **New Relic**: Herramienta de APM que ofrece una visión detallada del rendimiento de las aplicaciones, con alertas y diagnósticos.
- **Azure Application Insights**: Integrado en el ecosistema de Azure para monitoreo y análisis de aplicaciones .NET.
- **Dynatrace**: Plataforma que permite rastrear automáticamente el rendimiento de aplicaciones distribuidas.

Monitoreo de Logs

El monitoreo de logs es crucial para tener visibilidad sobre los errores, advertencias y eventos importantes dentro del sistema. Permite realizar análisis forense en caso de problemas o ataques.

Métricas clave:

- **Registros de errores**: Captura de excepciones no manejadas y errores críticos.
- **Logs de auditoría**: Monitoreo de accesos a datos sensibles y cambios en la configuración.
- **Logs de seguridad**: Detección de intentos de acceso no autorizados o ataques como el brute force.

Herramientas comunes:

- **Elastic Stack (ELK):** Conjunto de herramientas compuesto por Elasticsearch (almacenamiento y búsqueda), Logstash (ingesta de logs) y Kibana (visualización).
- **Graylog**: Plataforma que permite la recopilación y análisis de logs en tiempo real.
- **Splunk**: Solución comercial avanzada para el monitoreo y análisis de logs a gran escala.