Tabla de Contenido

Tabla de gráficos

Diagrama de Contexto

- Modelo general de la solución planteada, en el cual se observa la iteración del usuario o Customer con los sistemas Core del banco e Internet Banking System que permiten obtener la información del usuario, cuentas, balances y transferencias entre sus cuentas. Es importante comentar que, al realizar una transferencia, se debe notificar al usuario por lo cual se plantea dos mecanismos que son:
    - Envío de notificación por correo electrónico, el cual puede ser un servicio interno o externo.
    - Envío de mensajes de textos a dispositivos móvil, conocido como sms, se recomiends utilizar el servicio de AWS Amazon SNS que permite implementar fácilmente este tipo de mensajería.

      Estos dos mecanismos son parte del External notification System.

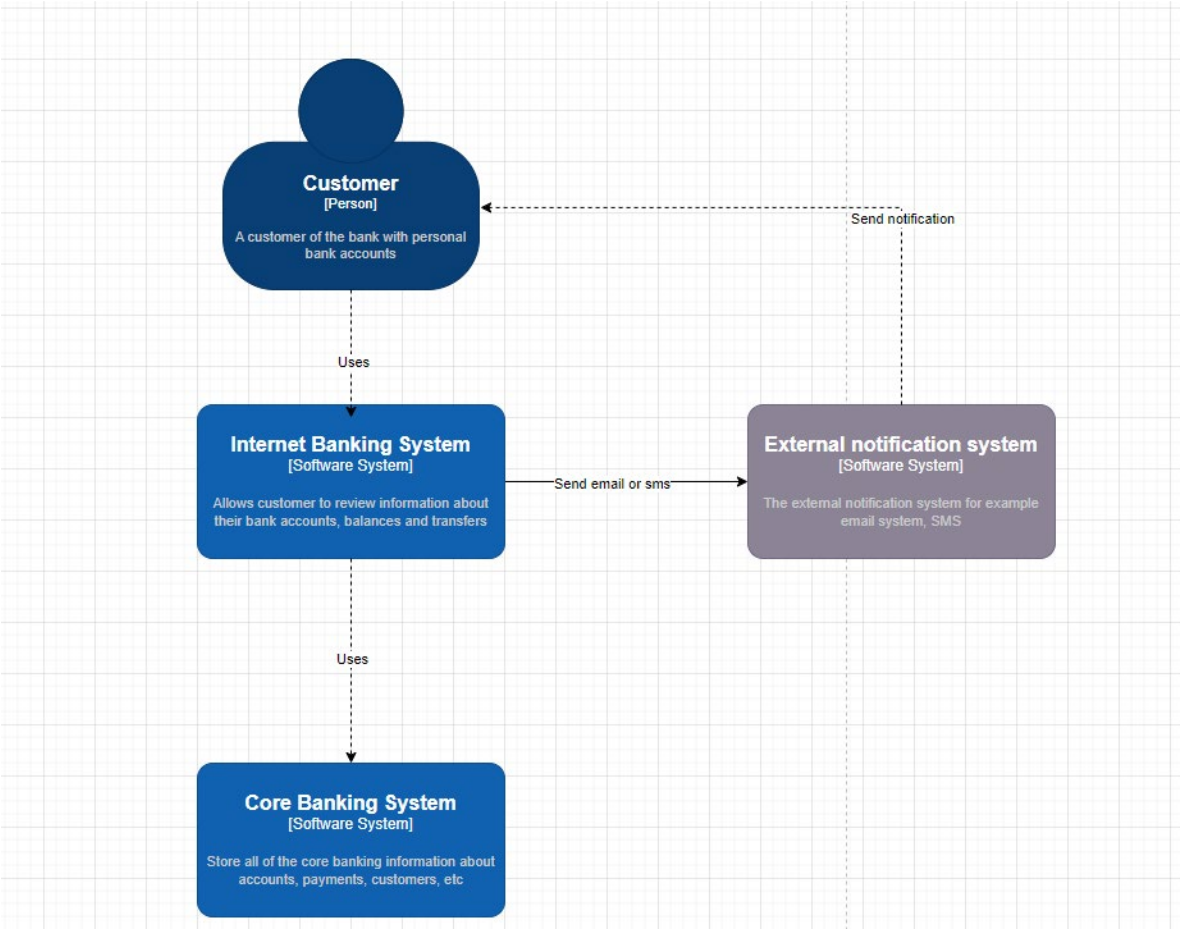

Figura 1 Diagrama de Contexto

Diagrama de Contenedores

En este diagrama se puede observar la tecnología que se plantea para la implementación de este proyecto como se detalla a continuación.

- SPA: se plantea utilizar JavaScript con React o Angular como tecnología que nos permite desarrollar un sitio web con características óptimas para que el usuario pueda utilizar la página de manera sencilla.
- Mobile App: Se recomienda utilizar React Native o Angular que son las tecnologías más utilizadas y que nos permite tener aplicaciones multiplataforma sin tener una afectación considerable en el rendimiento de la misma. Flutter una tecnología de Google también es recomendada por su sencilles en el aprendizaje y actualmente va incrementando su adopción para aplicaciones móviles.

  Se recomienda utilizar microfrontends que nos permite desagregar las funcionalidades y adicionalmente permite que la aplicación permita tener una arquitectura desacoplada lo que nos da flexibilidad en el caso de actualizar backend o front end.

- Como orquestador se puede utilizar MODYO que nos permite utilizar Widgets los cuales son componentes desarrollados en Angular o React (Reac Native para funciones específicas del dispositivo) y que se puede utilizar tanto en aplicaciones móviles, como web, lo que facilita su mantenimiento y desarrollo ágil con equipos grandes.
- A nivel de servicios se recomienda utilizar Spring Webflux sobre Spring Framework ya que al ser una programación rectiva basada en Reactor permite ejecutar peticiones de forma asíncrona y no bloqueante como sucede con Spring Framework que no es óptimo para alto tráfico recurrente. Adicionalmente utilizamos se recomienda utilizar Spring Boots para crear microservicios lo cual nos ofrece una mayor flexibilidad y escalabilidad.
- Se plantea utilizar los servicios de AWS como CodeBuild, CodePipeline, Route53, Amazon Api Gateway, VPC, ALB, NLB, Fargate, EKS, RDS. Se va a utilizar una arquitectura lo más serverless posible, no solo porque permite reducir costos de hardware y licenciamiento por ejemplo si se utilizara un servidor de Oracle en un ambiente onpremise en el cual se requiere hardware y un equipo que administre, actualice el servidor, por eso la idea es utilizar los servicios que están disponibles en la nube de AWS.
- Finalmente, si se pretende migrar a la nube se recomienda implementar OAuth 2.0 con el flujo de autorización para Amazon Cognito usando AWS lambda y Amazon DynamoDB.
- Seguridad: Para evitar ser vulnerados todos los datos de entrada sensibles deben ser validados, encriptados, la información como tarjetas de crédito, números de cuentas deben enviarse de manera encriptada al API para que en el servicio se des- encripte y se envíe a buscar la cuenta o tarjeta requerida para una transferencia.

  La comunicación entre las APIs y AWS se lo puede realizar mediante un protocolo seguro (HTTPS) para que permita que la comunicación vaya cifrada mediante certificados TLS.

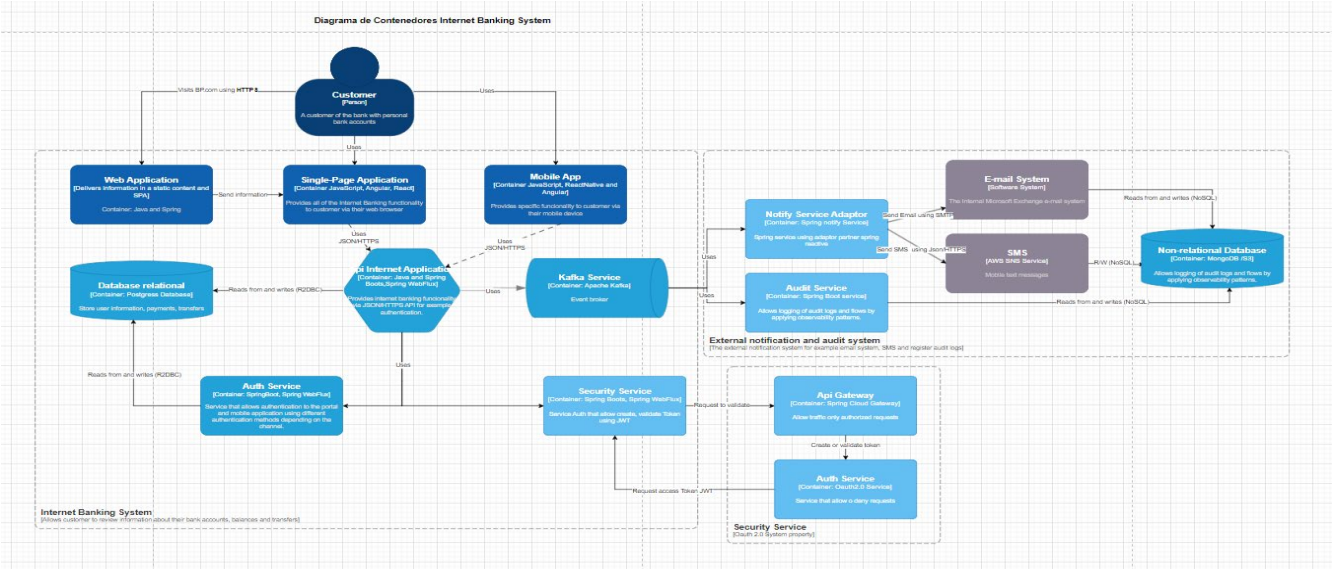

Figura 2 Diagrama de contenedores Internet Banking System

Para el sistema Core se considera el uso de Kafka aplicando el patrón de observabilidad con el cual podemos integrar varias herramientas, se recomienda utilizar servicios como AWS CloudWatch (monitorea el performance de las respuesta de una aplicación), CloudTrail (monitorea y registra la actividad de cuentas a través de la infraestructura de AWS), AWS X-Ray (herramienta utilizada para ayudar a los desarrolladores a analizar y debuguear aplicaciones), AWS MQ (permite reenvío de eventos a otros sistemas), tiene una menor curva de aprendizaje. Este esquema nos permite tener toda la trazabilidad end to end ya que tenemos un registro de eventos y auditoría lo que nos facilita las tareas de operación y mantenimiento.

Existen otras herramientas como Istio Console, Jaeger, Elasticsearch (ELK) y Prometheus, las cuales también trabajan en conjunto con Apache Kafka, hay que considerar que la curva de aprendizaje es alta, difícil de implementar sino se tiene experiencia en Servicios Mesh, podría generar sobrecarga en entornos con tráfico alto.

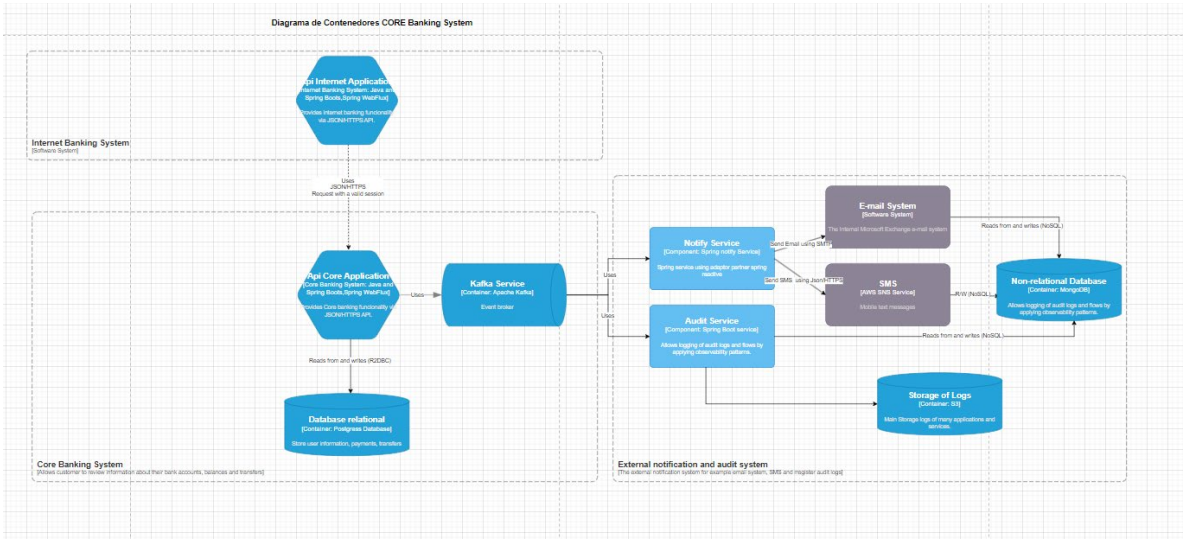

Figura 3 Diagrama de contenedores Core Banking System

Diagrama de Componentes

Diagrama de componentes Autenticación

En este diagrama se puede observar cómo se realizaría el proceso de autenticación (SSO) y como se puede implementar en web y app (considerando el enrolamiento de un usuario y su mecanismo de autenticación que puede ser reconocimiento facial, biometría, etc). Importante indicar que se utiliza el patrón de diseño creacional como es Factory para que dependiendo del canal (web o app) se cree la instancia del componente que va a permitir continuar el proceso de autenticación, adicionalmente se utiliza el patrón de diseño strategy ya que nos permite agregar nuevos métodos como para reconocimiento facial, huella dactilar o un mecanismo básico de autenticación sin cambiar el código base, nos permite elegir la estrategia en tiempo de ejecución por ejemplo si el dispositivo no soporta FaceId, utiliza la huella, de esta forma el contexto se vuelve independiente y genera un código más limpio.

Se recomienda utilizar servicios de AWS como AWS Rekognition (detección facial), FingerPrintManager (Android), LocalAuthentication (FaceId/TouchId) que son las herramientas de alto rendimiento, escalabilidad automática, bajo de tiempo de respuesta y están listas para usar, se integra fácilmente con S3 (si se requiere almacenar las imágenes o una base de datos non relacional), disponible SDK's para múltiples lenguajes como Phyton, Java, JavaScript, etc.

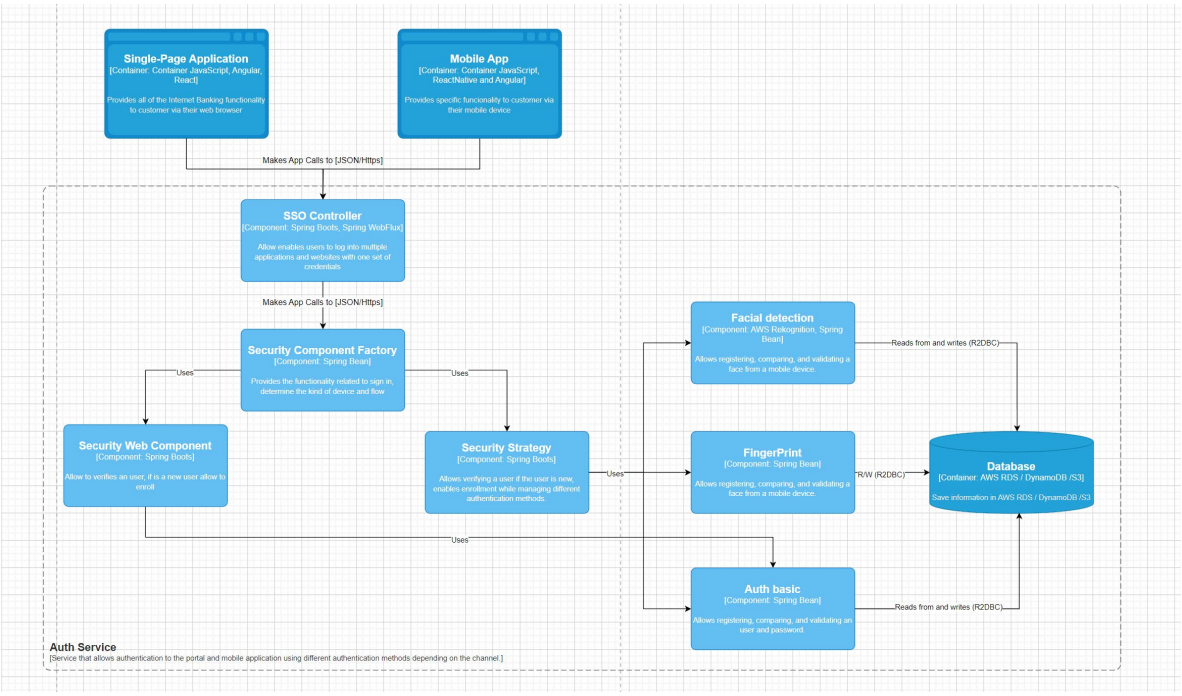

Figura 4 Diagrama de componentes Autenticación

Diagrama de componentes de Seguridad

Al contar con el mecanismo de seguridad con el estándar OAuth 2.0, se recomienda el flujo en el cual en primera instancia cualquier petición primero se valida en el api Gateway en el cual se valida el nivel de acceso de cada petición, una vez validado se puede utilizar EurekaServer para protección de recursos y Keycloack que nos permite implementar la autenticación a aplicaciones de servicios seguros con un mínimo esfuerzo, adicionalmente nos permite el manejo de la validación y generación del token Access (JWT).
En este diagrama se puede observar el flujo recomendado para implementar un mecanismo de seguridad.

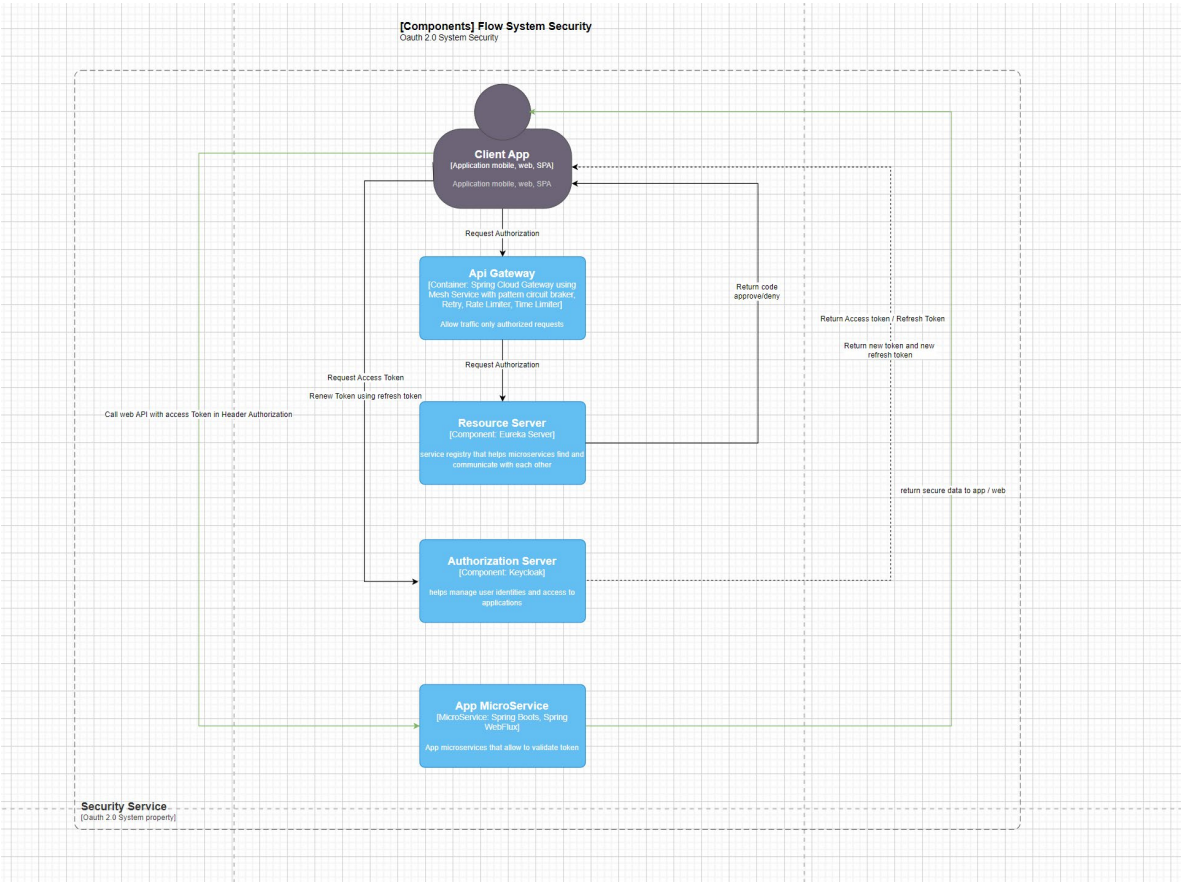

Figura 5 Diagrama de componentes de Seguridad (Flujo)

Diagrama de componentes de Auditoría y notificaciones

Se puede observar los componentes que deben ser auditados por lo que se puede visualizar el AmazonApigateway que nos sirve para securizar los requests, adicionalmente se puede implementar un MeshService en el cual podemos configurar los patrones de observabilidad como circuit braker, Retry para lo cual podemos utilizar Resilience4j para la comunicación entre servicios y que integrado con un Mesh (gestiona la comunicación segura entre microservicios) nos permite balanceo y seguridad.

Posteriormente se puede utilizar un ALB, NGiNX, EKS clúster para utilizar como un orquestador a Kubernetes, un ECR para el registro de las imágenes Docker de los microservicios y Kakfa en conjunto con servicios como AWS CloudWatch , CloudTrail, AWS X-Ray, AWS MQ, que nos permite aplicar de manera eficiente el patrón de observabilidad.

Finalmente, al utilizar programación reactiva podemos implementar por medio del patrón de observabilidad los dos mecanismos de notificación a través de eventos, que se va a implementar como un sistema de correos o envío de sms utilizando el servicio AWS SNS que nos permite hacerlo de manera sencilla.

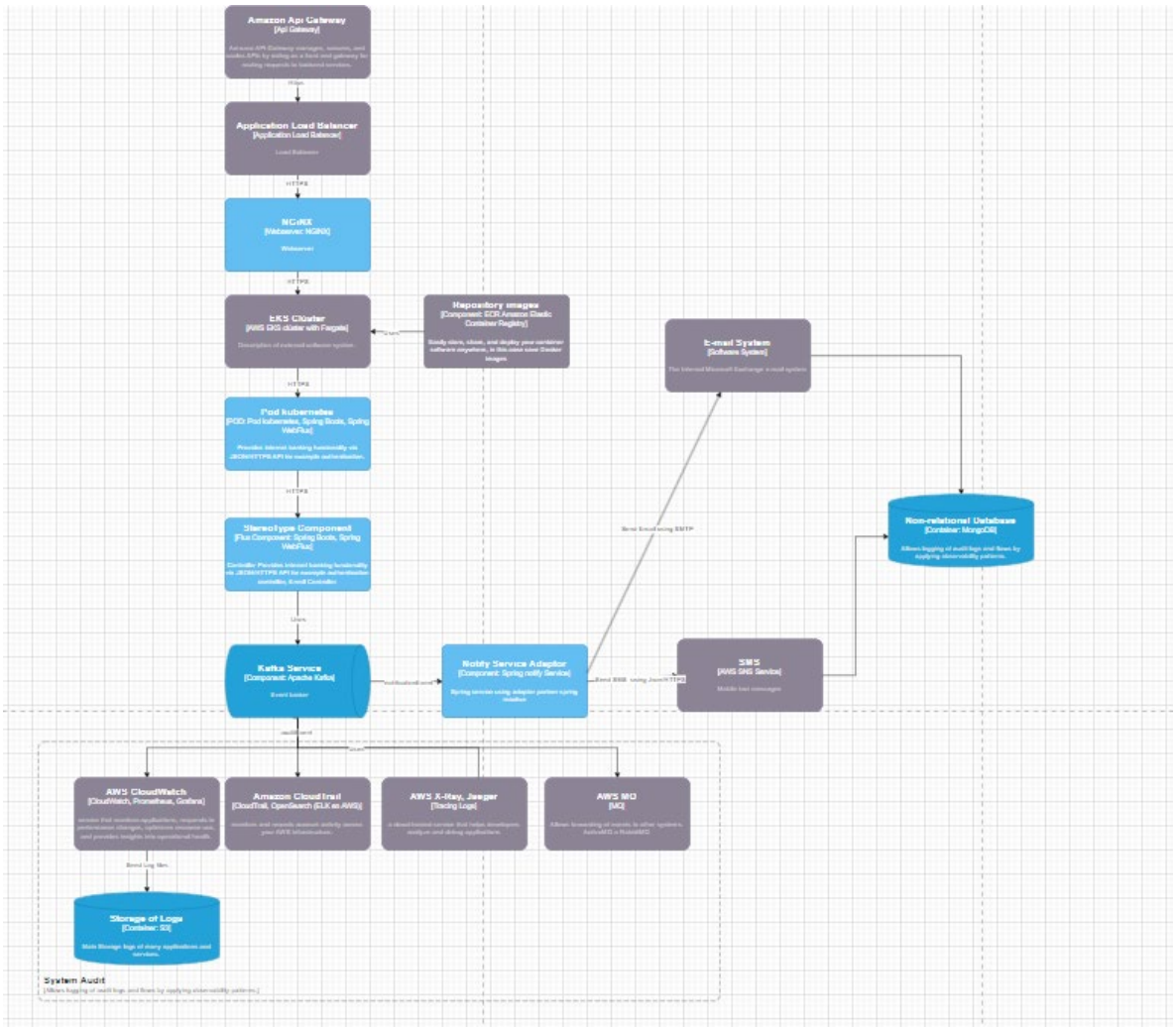

Figura 6 Diagrama de componentes de Auditoría y Notificaciones.

Diagrama de componentes de Core Banking System

Se puede observar el modelo planteado para consulta de información básica del usuario, transferencias, movimientos del cliente y como interactúa con los distintos componentes, donde se puede ver la implementación de un microservicio con su propia base de datos, todos los microservicios se comunican entre sí utilizando Resilience4j que nos permite implementar patrones de tolerancia a fallos en aplicaciones reactivas, lo que mejora la resiliencia de las aplicaciones al manejar fallos de red, tiempos de espera y sobrecarga de servicios.

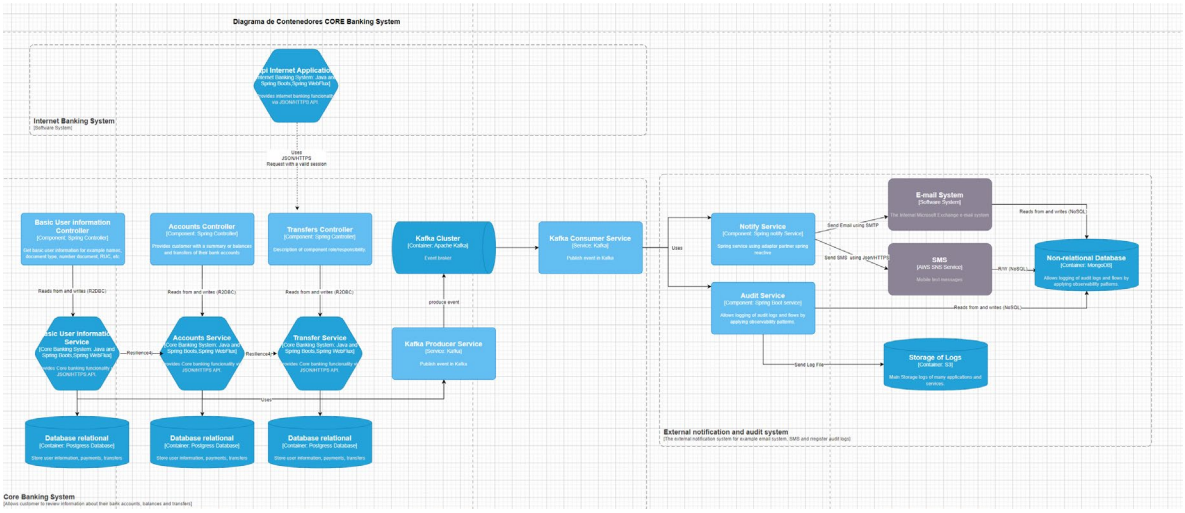

Figura 7 Diagrama de componentes de Core Bank System.

Diagrama de componentes de Front End

Se recomienda utilizar microfrontends que nos permite desagregar las funcionalidades y adicionalmente permite que la aplicación permita tener una arquitectura desacoplada lo que nos da flexibilidad en el caso de actualizar backend o front end. Adicionalmente permite actualizar y escalar cada componente de manera independiente, sin afectar el resto del sistema, con lo cual se logra modularidad, mantenibilidad y escabilidad.

MODYO nos permite utilizar Widgets los cuales son componentes desarrollados en Angular o React (Reac Native para funciones específicas del dispositivo) y que se puede utilizar tanto en aplicaciones móviles, como web, lo que facilita su mantenimiento y desarrollo ágil con equipos grandes. A través de Modyo Connect se desacopla completamente el front end del backend, se establece un contexto delimitado para la comunicación mediante APIs.

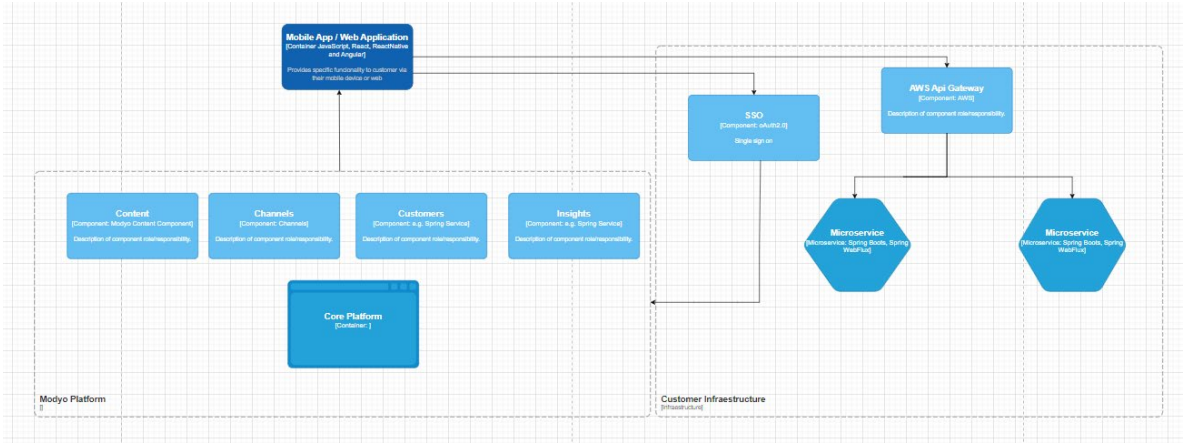

Figura 8 Diagrama de componentes Front.

Diagrama de despliegue

- Se adiciona este diagrama en el cual se puede observar un flujo que nos permitirá implementar la integración continua y entregas continuas (CI/CD) de la aplicación con lo cual utilizando servicios de AWS podemos tener un esquema de alta disponibilidad, escalamiento automático, tolerancia a fallos y podemos monitorear fácilmente.
- Se plantea utilizar servicios de AWS como se detalla a continuación.
    - AWS CodeBuild que nos permitirá compilar el código que se puede estar en un repositorio como GitHub fuente de la aplicación y generar el artefacto que posteriormente automáticamente se generará la imagen docker que será registrada en AWS ECR.
    - AWS ECR nos permite registrar los contenedores Docker de manera sencilla y que se lo utilizará en el clúster de Kubernetes.
    - AWS CodePipeline es el que permite orquestar automáticamente si determina si se realizó algún cambio en el repositorio del código fuente y se genera el artefacto, por seguridad se recomienda niveles de aprobación para su ejecución. También nos permitirá realizar el deploy continúo lo que permite actualizaciones continuas.
    - AWS EKS, nos permite crear clúster de Kubernetes en la nube de AWS.
    - AWS CloudWatch Logs, nos permite monitorear los servicios.
    - AWS Fargate nos permite ejecutar los pods de Kubernetes y que provee bajo demanda la capacidad de los contenedores, lo que nos evita tener que provisionar o escalar grupos de máquinas virtuales, sino que lo realiza de manera automática, si es necesario se crea más instancias para que la aplicación responda correctamente.
    - Se configura CronJobs a los contenedores lo que nos permitirá ahorro de recursos ya que podemos configurar en que período de tiempo van a estar online nuestra infraestructura en la nube.

Con este esquema se pretende que la aplicación tenga alta disponibilidad ya que automáticamente al utilizar un clúster de Kubernetes se puede configurar el número de PODS mínimos y máximos que son requeridos de acuerdo con la demanda, es decir que automáticamente se crean instancias y permite que la aplicación se recupere rápidamente a fallos.

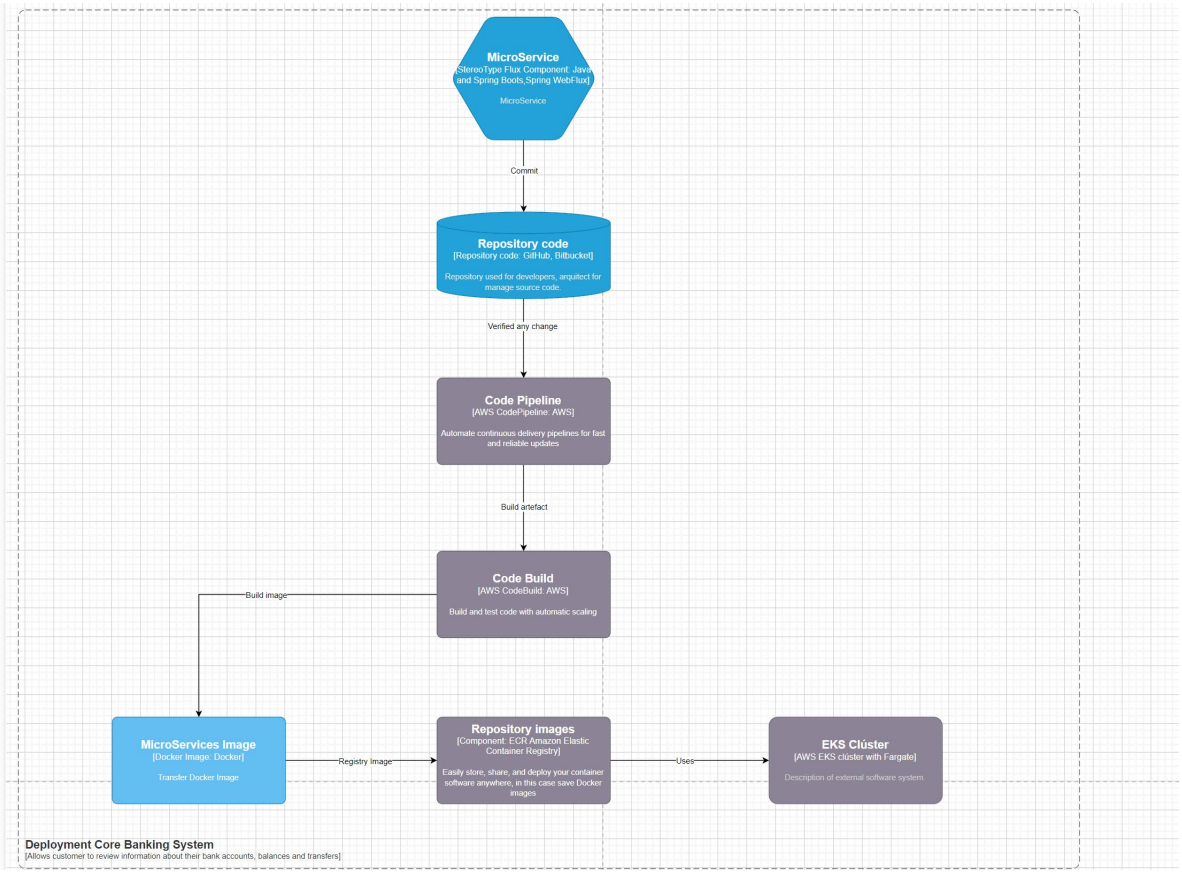

Figura 9 Diagrama de Despliegue

Manejo de costos

Al plantear la solución en la nube, para optimizar los costos se ha considerado los siguientes puntos:

- Se plantea una arquitectura serveless en lo que se pueda aplicar, con la finalidad de reducir costos tanto en hardware como licenciamiento del cliente, se plantea una solución en la nube optimizando recursos.
- Se configura CronJobs al clúster con la finalidad de optimizar recursos, por ejemplo, se configura el horario en el cual la infraestructura va a estar disponible, en un ambiente de desarrollo se puede configurar con horario de 8 – 7pm y en la noche se apagan lo que genera ahorro.
- En un ambiente productivo se puede implementar políticas para que dependiendo del horario se baje automáticamente las instancias disponibles tanto en escalamiento vertical como horizontal dependiendo del componente que se requiera optimizar, con este tipo de medidas podemos optimizar recursos.
- La arquitectura planteada recomienda utilizar software open source como Java, Spring WebFlux, Spring Boots, Eureka server, Keycloak, Docker, Kubernetes, Apache Kafka que nos permite reducir costos de licenciamiento del cliente y que a la vez nos permite integrarnos a la nube de AWS optimizando los servicios utilizados.
- Para temas de auditoría, se recomienda utilizar un S3 estándar para el almacenamiento de archivos logs más recientes que son más utilizados en diagnósticos de errores y se requiere agilidad, por ejemplo, de los últimos 3 meses, pasado este tiempo se lo puede mover a un S3 One Zone de acceso poco frecuente con lo cual podemos optimizar costos, esta política podría ser utilizada por ejemplo en almacenamiento de registros de imágenes faciales, archivo de lotes, etc. Es decir dependiendo de la frecuencia de acceso podemos elegir el S3 que sea mas conveniente para una funcionalidad, optimizando recursos.