Informe de Diseño: Agenda Distribuida

Gabriel Andrés Pla Lasa
Grupo C411

Carlos Daniel Largacha Leal
Grupo C412

Universidad de La Habana
Facultad de Matemática y Computación

Abstract

This report details the design and architecture of a distributed calendar system, conceived to meet high availability and fault tolerance requirements. The architecture is based on custom-implemented failover mechanisms for its entry points and messaging layer, along with the Raft consensus algorithm for consistent data persistence. The challenges of distributed systems are addressed, including communication, coordination, service discovery, consistency, and security, with the objective of achieving a robust solution that is transparent to the end-user when compared to a centralized implementation.

Resumen

El presente informe detalla el diseño y la arquitectura de un sistema de agenda distribuida, concebido para cumplir con los requisitos de alta disponibilidad y tolerancia a fallos. La arquitectura se fundamenta en mecanismos de conmutación por error (failover) implementados a medida para sus puntos de entrada y su capa de mensajería, junto con un algoritmo de consenso Raft para la persistencia consistente de los datos. Se abordan los desafíos de los sistemas distribuidos, incluyendo comunicación, coordinación, descubrimiento de servicios, consistencia y seguridad, con el objetivo de lograr una solución robusta que sea indiferente para el usuario final frente a una implementación centralizada.

1 Introducción

La gestión eficiente del tiempo y la coordinación de actividades en grupo son problemas fundamentales en la interacción humana y profesional. El proyecto de Agenda Distribuida se presenta como una solución tecnológica a este desafío, implementando una arquitectura de microservicios distribuidos en Go. El objetivo de este informe es detallar el diseño de un sistema plenamente funcional y tolerante a fallos, capaz de operar sobre una infraestructura de red virtualizada en múltiples máquinas. Para ello, se resuelven los problemas canónicos de la computación distribuida mediante la implementación de soluciones a medida: un mecanismo de conmutación por error (failover) a nivel de red para el punto de entrada, un servicio supervisor para la capa de mensajería asíncrona, y el algoritmo de consenso Raft para garantizar la consistencia de los datos.

2 Diseño del Sistema Distribuido

El diseño del sistema se ha concebido para garantizar la continuidad del servicio ante fallos de componentes individuales. A continuación, se detallan las distintas facetas de la arquitectura distribuida, desde su organización estructural hasta los mecanismos de seguridad implementados, argumentando las decisiones de diseño tomadas para cada aspecto.

2.1 Arquitectura

La arquitectura del sistema se estructura bajo un modelo de microservicios, diseñado para evitar puntos únicos de fallo y asegurar que cada componente pueda operar de forma independiente. Esta organización facilita la distribución de responsabilidades y permite que cada servicio cumpla un rol claramente delimitado en la interacción general del sistema.

En primer lugar, el punto de entrada se implementa como un par de instancias del API Gateway configuradas en modo Activo–Pasivo. Su función es recibir todas las peticiones externas y exponer una interfaz estable al usuario mediante una Dirección IP Virtual, lo que garantiza continuidad incluso ante la caída del nodo activo. Tras esta capa se ubican los servicios de lógica de negocio, responsables de las operaciones relacionadas con usuarios y grupos. Estos servicios son completamente independientes del estado, lo que habilita su replicación y reemplazo sin afectar la coherencia del sistema.

El tercer rol lo ocupa la capa de mensajería, implementada mediante un par Redis Primario–Réplica cuya función es canalizar la comunicación asincrónica entre los servicios. Su supervisión se delega a un proceso especializado que opera sobre esta capa y asegura su disponibilidad, pero desde el punto de vista arquitectónico esta capa actúa como mecanismo de desacoplamiento entre componentes.

Finalmente, la persistencia se delega al DB Service, un clúster de cinco nodos ejecutando un algoritmo de consenso para mantener una réplica consistente de los datos. Este componente constituye la fuente de verdad autoritativa del sistema.

La distribución física de estos servicios se realiza sobre dos redes Docker ejecutadas en dos máquinas diferentes. La disposición asegura que cada par crítico (como el Gateway o Redis) mantenga una instancia en cada máquina. Los nodos del clúster Raft se reparten equilibradamente entre las dos instancias, lo que permite preservar el quórum incluso ante fallas parciales. De esta manera, un fallo físico no afecta simultáneamente a ambos elementos de una misma función, preservando la disponibilidad del sistema.

2.2 Procesos

El sistema está compuesto por un conjunto de procesos independientes, cada uno desplegado en su propio contenedor Docker para asegurar aislamiento y portabilidad. Estos procesos pueden agruparse en dos categorías fundamentales: aquellos que mantienen estado y aquellos que operan sin él. Los procesos sin estado incluyen el API Gateway y los servicios de lógica de negocio, lo que permite replicarlos fácilmente o reiniciarlos sin afectar la continuidad de la aplicación. Por el contrario, Redis y el DB Service pertenecen a la categoría de procesos con estado, pues requieren mantener información persistente o volátil que debe estar siempre disponible y correctamente replicada.

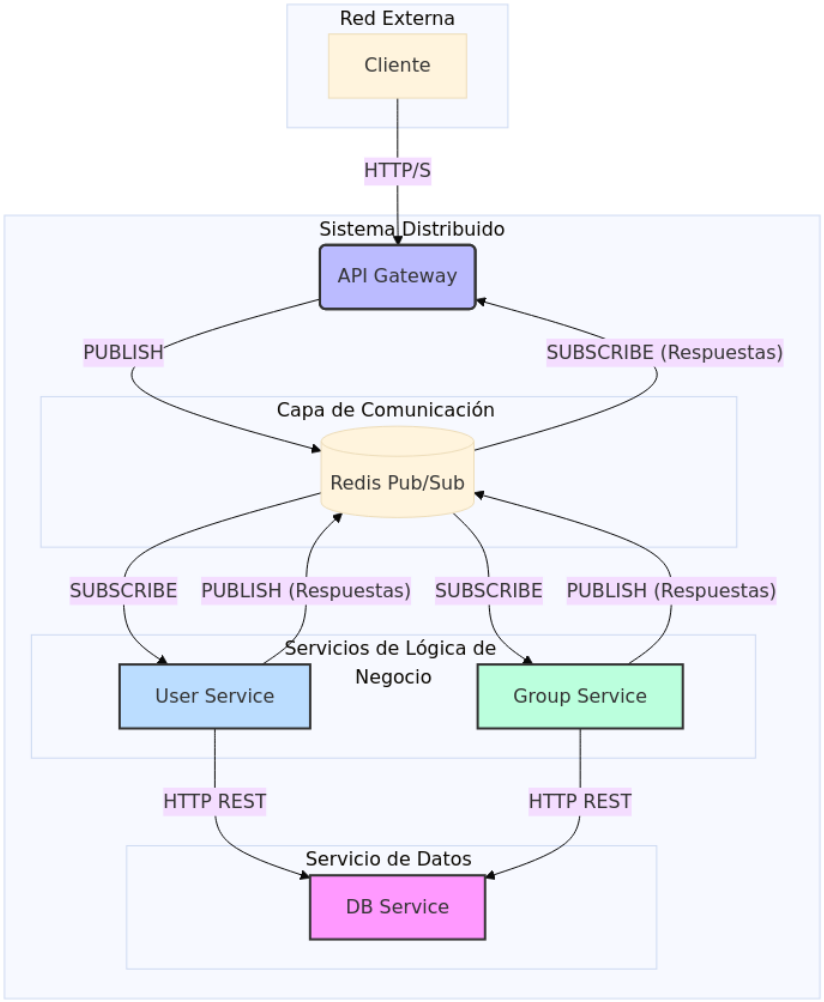

Figure 1: Diagrama General de la Arquitectura

La organización de estos procesos en las dos máquinas físicas responde a la necesidad de balancear la carga y garantizar tolerancia a fallos. Cada par crítico (como Gateway o Redis) se ubica dividido entre ambas máquinas, mientras que los nodos del clúster de datos se distribuyen de manera que ninguna de las máquinas contenga una mayoría por sí sola. Esto evita que la caída de una máquina comprometa el funcionamiento global del sistema.

En cuanto al desempeño, todos los procesos escritos en Go aprovechan el modelo de concurrencia nativo del lenguaje. La utilización de goroutines y canales permite gestionar numerosas operaciones de red de forma eficiente, reduciendo el costo asociado a la creación de hilos del sistema operativo. Esto resulta especialmente útil en los servicios de lógica y en los componentes que realizan monitoreo o comunicación recurrente, para los cuales la naturaleza asincrónica del modelo contribuye a mantener latencias bajas y un uso moderado de recursos.

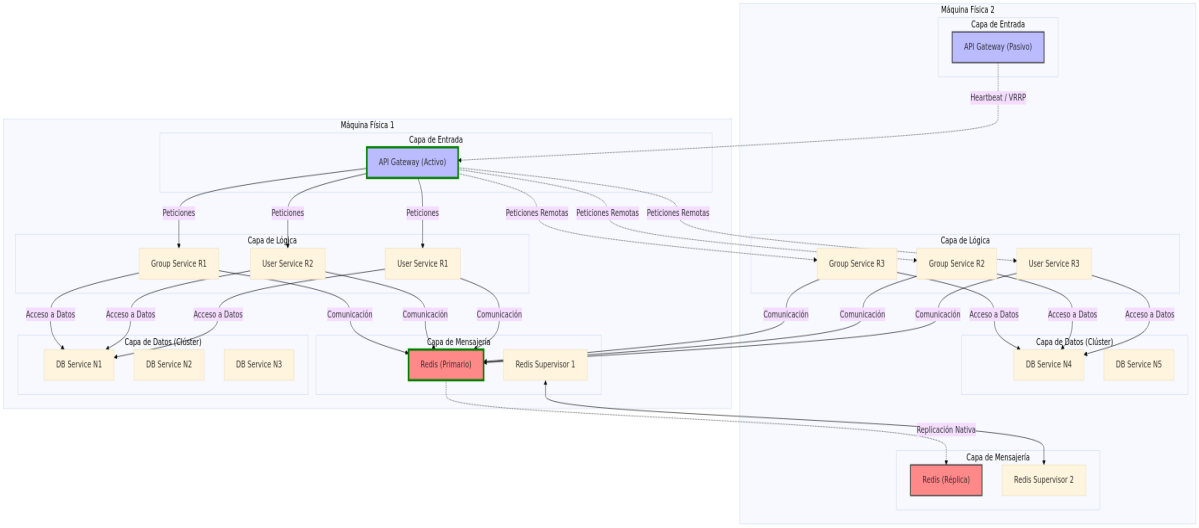

Figure 2: Distribución Física de Procesos en las Máquinas

2.3 Comunicación

La comunicación entre los componentes del sistema se ha diseñado combinando distintos mecanismos según el tipo de interacción requerida. Esta selección responde a la necesidad de equilibrar simplicidad, eficiencia y desacoplamiento, tres propiedades esenciales en un entorno distribuido.

En la comunicación entre el cliente y el sistema se utiliza exclusivamente HTTP REST sobre TLS. Este enfoque proporciona una interfaz clara, fácilmente integrable y segura, y permite que el usuario interactúe de forma transparente con el punto de entrada sin necesidad de conocer la estructura interna del sistema. Todas las peticiones se realizan a través de la Dirección IP Virtual del Gateway, lo cual oculta la dinámica interna de disponibilidad de esta capa.

La comunicación interna adopta una estructura más variada. El intercambio asincrónico entre el API Gateway y los servicios de lógica de negocio se implementa mediante un patrón de publicación y suscripción sobre la capa de mensajería Redis. Esta decisión permite desacoplar temporalmente a los productores y consumidores de eventos, de modo que los servicios puedan procesar solicitudes sin depender del ritmo o disponibilidad inmediata de los demás componentes.

Para las interacciones que requieren una respuesta inmediata, como las consultas de los servicios de lógica hacia el nodo líder del clúster de datos, se emplea nuevamente HTTP REST. Su sencillez y bajo costo de implementación lo convierten en una opción adecuada para solicitudes puntuales y de baja frecuencia.

Finalmente, la comunicación entre los nodos del DB Service se establece a través de llamadas RPC sobre sockets TCP. Esta vía de comunicación es ligera y eficiente, y resulta especialmente adecuada para el intercambio frecuente de mensajes de control y actualización que exige el algoritmo de consenso ejecutado en esta capa. El uso de sockets directos evita la sobrecarga de protocolos de nivel superior y contribuye a mantener la latencia dentro de los márgenes necesarios para preservar la estabilidad del clúster.

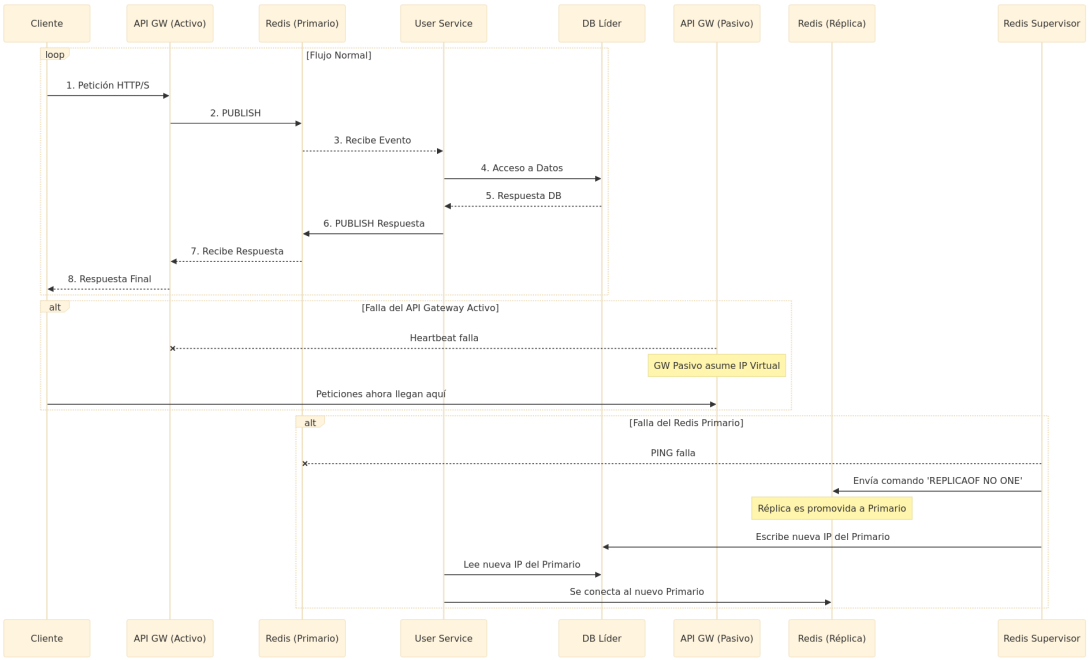

Figure 3: Diagrama de Secuencia de Comunicación y Escenarios de Failover

2.4 Coordinación

La coordinación de acciones entre los distintos componentes del sistema es un elemento esencial para garantizar un comportamiento global coherente y predecible. En el diseño de la Agenda Distribuida, este problema se manifiesta en tres niveles claramente diferenciados, cada uno con un mecanismo propio adaptado a sus necesidades.

El nivel de coordinación más estructurado corresponde al clúster de datos del DB Service, donde se ejecuta un algoritmo de consenso de Raft. Su función principal es establecer un orden global sobre las operaciones que modifican el estado persistente. Este ordenamiento actúa como un mecanismo de exclusión lógica, asegurando que las actualizaciones se procesen secuencialmente mediante un líder. De esta manera, el sistema evita condiciones de carrera y garantiza que todas las réplicas converjan hacia el mismo estado. Este mecanismo constituye la forma de coordinación más estricta dentro del sistema, ya que determina cuándo una operación es válida y cuál nodo tiene autoridad para gestionarla.

Un segundo nivel de coordinación aparece en la capa de mensajería mediante el servicio redis-supervisor. Aunque su función se relaciona con la disponibilidad, el proceso que emplea para tomar decisiones constituye un ejemplo claro de coordinación distribuida. Cada instancia del supervisor mantiene conocimiento de las direcciones de las dos instancias de Redis y determina de forma autónoma cuál de ellas actúa como primaria mediante consultas periódicas al estado de replicación reportado por cada nodo. A partir de esta información, el supervisor realiza un monitoreo continuo mediante comandos de verificación que permiten detectar la ausencia del nodo primario. En caso de fallo, la promoción de la réplica se convierte en una decisión distribuida en la que cada instancia del supervisor está capacitada para actuar. Para evitar inconsistencias tras la elección, el supervisor que realiza la promoción registra de forma centralizada la nueva ubicación del nodo primario en el DB Service, permitiendo que el resto de los servicios adopten una visión coherente del sistema.

Finalmente, la capa de entrada incorpora un mecanismo de coordinación a nivel de red para gestionar la posesión de la Dirección IP Virtual asignada al API Gateway. La instancia pasiva supervisa la disponibilidad de la instancia activa mediante señales periódicas de presencia. Si dichas señales cesan, la instancia pasiva asume de manera exclusiva el rol activo y reclama la dirección virtual, lo que garantiza que siempre exista un único punto de entrada operativo. Aunque más simple que los mecanismos anteriores, este proceso sigue las mismas ideas fundamentales de la coordinación distribuida: detección, decisión y exclusividad.

En conjunto, estos tres mecanismos permiten que el sistema distribuido opere como una unidad coherente. Cada capa adopta el nivel de coordinación adecuado a su función, evitando sobrecargar el sistema con complejidad innecesaria y asegurando que las decisiones críticas se tomen de manera consistente y ordenada.

2.5 Nombrado y Localización

El problema del nombrado y la localización en un sistema distribuido consiste en determinar cómo identificar de manera inequívoca a los servicios y cómo permitir que los distintos procesos puedan encontrarse entre sí incluso en presencia de fallos o cambios en la configuración. En la Agenda Distribuida este desafío se aborda mediante una combinación de mecanismos descentralizados y centralizados, seleccionados en función de las necesidades de consistencia y estabilidad de cada componente.

Los servicios de lógica de negocio constituyen la capa más flexible dentro del sistema. Al tratarse de procesos sin estado que pueden replicarse libremente, su localización no requiere de un modelo de consistencia estricta. Para ellos se emplea un protocolo de descubrimiento basado en la difusión periódica de información entre nodos, lo que permite que cada instancia mantenga una visión actualizada de sus pares sin depender de un único punto de referencia. Este enfoque, cercano a las técnicas de propagación de información propias de los protocolos de tipo Gossip, resulta adecuado para servicios replicados donde las fluctuaciones en la disponibilidad no comprometen la semántica del sistema.

En contraste, los servicios con estado requieren un mecanismo de localización preciso y altamente confiable, ya que un error al determinar su ubicación podría conducir a inconsistencias o comportamientos incorrectos. Este es el caso de la capa de mensajería y, en particular, del nodo primario de Redis, cuya identidad puede variar tras un proceso de conmutación. Para garantizar que todos los componentes accedan a la ubicación correcta, la información sobre el primario se almacena en el DB Service, que actúa como repositorio central de configuración. De esta forma, el sistema dispone de una fuente de verdad única que refleja en todo momento la topología vigente.

Una estrategia similar se aplica al clúster de datos, donde la identificación del nodo líder es igualmente crítica. La dirección del líder se obtiene directamente del propio clúster mediante los mecanismos internos del algoritmo de consenso, evitando depender de información desactualizada. De esta manera, el sistema conserva una visión coherente de los procesos que controlan el acceso al estado persistente.

En conjunto, la arquitectura adopta un modelo híbrido de localización. Los servicios que pueden tolerar variabilidad se resuelven mediante técnicas descentralizadas y eventualmente consistentes, mientras que los que requieren estabilidad estricta se gestionan mediante un mecanismo centralizado construido sobre la capa de datos. Esta diferenciación permite equilibrar la eficiencia con la fiabilidad, garantizando que cada componente encuentre la información que necesita.

2.6 Consistencia y Replicación

El tratamiento de la consistencia y la replicación en el sistema se articula mediante una diferenciación clara entre los distintos tipos de datos y servicios presentes en la arquitectura. Esta estratificación permite aplicar a cada componente el modelo más adecuado, evitando tanto la sobrecarga innecesaria como los riesgos asociados a un nivel de consistencia insuficiente.

La capa de persistencia constituye el núcleo más estricto del modelo. En ella se emplea un clúster de cinco nodos que ejecuta el algoritmo de consenso Raft capaz de garantizar consistencia fuerte. Esta propiedad, expresada como linealizabilidad, asegura que todas las operaciones de escritura se procesen en un único orden global y que las réplicas mantengan en todo momento una visión coherente del estado. Como consecuencia directa, cualquier lectura realizada sobre el nodo líder refleja el resultado de todas las operaciones confirmadas, incluso en escenarios donde uno o varios nodos se encuentran temporalmente fuera de servicio. Esta estrategia convierte al clúster en una fuente de verdad autoritativa para el resto del sistema.

La capa de mensajería adopta un enfoque diferente. Redis emplea un esquema de replicación Primario–Réplica de tipo asincrónico, orientado a maximizar el rendimiento y minimizar la latencia de publicación de mensajes. Aunque este modelo introduce la posibilidad de pérdida de información si el nodo primario falla antes de completar la propagación, el diseño del sistema mitiga este riesgo mediante supervisión constante. Cuando ocurre una conmutación en esta capa, el proceso supervisor actualiza de manera centralizada la ubicación del nuevo primario, garantizando que los componentes del sistema retomen la operación sobre una réplica válida en el menor tiempo posible. En este contexto, la confiabilidad del servicio de mensajería se alcanza por la vía de la recuperación rápida y coordinada, más que mediante consistencia estricta.

Los servicios de lógica de negocio constituyen el nivel más flexible en términos de replicación. Al ser procesos completamente independientes del estado, su duplicación tiene como objetivo exclusivo la disponibilidad. Cada servicio se replica en tres instancias, lo que proporciona un nivel de tolerancia a fallos de $k = 2$ dentro de esta capa, permitiendo que continúe operando aun cuando hasta dos de sus réplicas se encuentren fuera de servicio. La ausencia de estado evita la necesidad de coordinar o sincronizar su información interna, de modo que cada réplica puede atender peticiones sin depender del historial o del contexto operacional de las otras.

Un esquema particular de replicación se aplica a la capa de entrada del sistema, donde el API Gateway se despliega en dos instancias configuradas en modo Activo–Pasivo. En este caso la replicación no persigue la consistencia entre réplicas, dado que el Gateway es un servicio sin estado, sino que se orienta exclusivamente a garantizar disponibilidad continua. La presencia de una Dirección IP Virtual asegura que, independientemente de cuál instancia se encuentre activa en un momento dado, el sistema exponga siempre un único punto de acceso estable hacia el exterior. La conmutación entre ambas instancias no implica transferencia ni sincronización de información, lo que permite un cambio inmediato entre nodos sin impacto sobre el resto de los componentes.

El conjunto de estas decisiones configura un modelo híbrido de consistencia, el cual permite que el sistema cumpla simultáneamente con los requisitos de robustez, eficiencia y escalabilidad que exige un entorno distribuido.

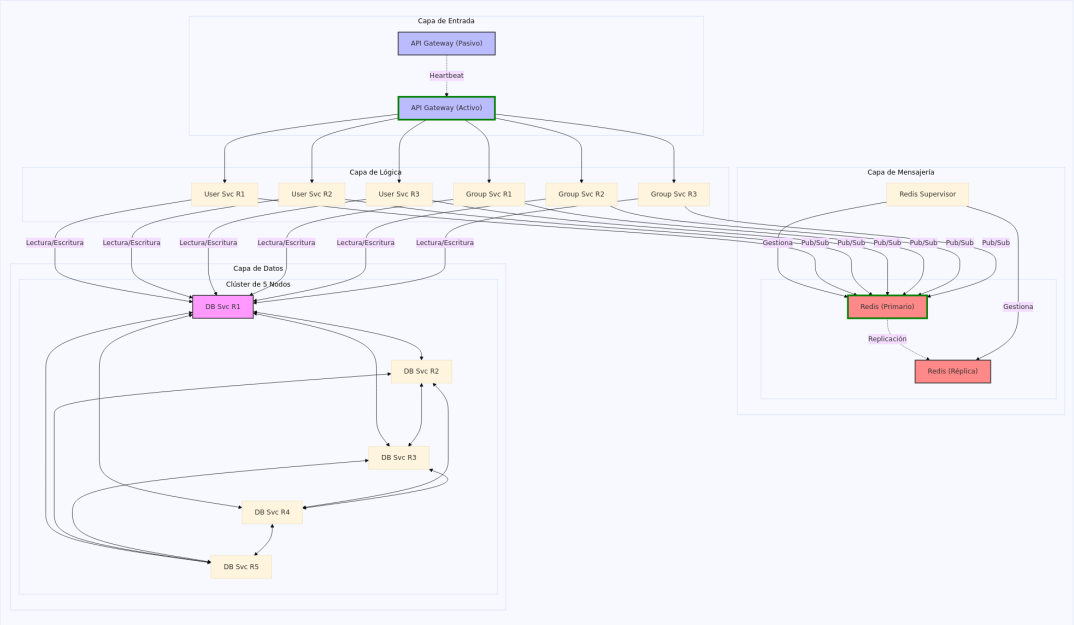

Figure 4: Modelo de Replicación de Componentes por Rol

2.7 Tolerancia a Fallos

La tolerancia a fallos constituye un principio rector en el diseño de la Agenda Distribuida, cuyo objetivo es garantizar la continuidad del servicio ante fallos parciales de la infraestructura. El sistema ha sido construido para mantener su funcionalidad incluso en presencia de interrupciones en componentes individuales, aprovechando la redundancia y la capacidad de recuperación automática en cada una de sus capas.

En el punto de entrada, la tolerancia a fallos se alcanza mediante un esquema de dos instancias del API Gateway distribuidas en máquinas distintas y unidas por una Dirección IP Virtual. La instancia pasiva supervisa al nodo activo mediante señales periódicas de presencia y, en caso de detectar su ausencia, asume inmediatamente el rol activo sin requerir intercambio de estado. Este mecanismo permite mantener un único punto de acceso ininterrumpido para el usuario, incluso cuando la máquina que alojaba al Gateway original se encuentra fuera de servicio.

En la capa de mensajería, el servicio redis-supervisor desempeña un papel fundamental en la recuperación ante fallos. Cada instancia del supervisor monitoriza de manera continua al nodo primario de Redis y es capaz de detectar su inactividad tras un número determinado de fallos consecutivos. Ante esta situación, la réplica se promueve de forma automática a nodo primario y la ubicación del nuevo responsable se registra en el clúster de datos. Este procedimiento asegura que las operaciones de publicación y suscripción puedan continuar con una interrupción mínima, aun cuando la transición implique la posible pérdida de mensajes no replicados al momento del fallo.

La capa de datos implementa un modelo más robusto y exhaustivo de tolerancia a fallos. El clúster se compone de cinco nodos y es capaz de sostener la operación normal incluso ante la caída simultánea de dos de ellos, gracias a su mecanismo de quórum. La persistencia de la información se garantiza mientras una mayoría de nodos permanezca disponible, lo que permite continuar aceptando y consolidando operaciones. Los nodos que se reintegran tras un fallo se sincronizan automáticamente con el estado vigente, evitando inconsistencias y manteniendo la integridad global del sistema.

Finalmente, los servicios de lógica de negocio dependen de una estrategia basada en replicación sin estado. Al existir tres réplicas de cada servicio, la aplicación conserva un nivel de disponibilidad suficiente para continuar su funcionamiento aun en escenarios donde hasta dos instancias se encuentren inaccesibles. La ausencia de estado facilita no solo su sustitución rápida, sino también su reincorporación transparente al sistema cuando la infraestructura afectada vuelve a estar operativa.

En conjunto, el sistema adopta una aproximación multicapa a la tolerancia a fallos, donde cada estrato de la arquitectura incorpora mecanismos propios de redundancia, detección y recuperación. Esta organización permite que la Agenda Distribuida mantenga su servicio activo en un amplio rango de escenarios adversos, garantizando una operación estable incluso en condiciones de fallos parciales prolongados.

2.8 Seguridad

La seguridad del sistema se estructura siguiendo un enfoque de capas complementarias que permite proteger tanto el flujo de información como la integridad de los servicios internos. Su diseño busca minimizar la superficie de ataque y garantizar que las operaciones realizadas por los usuarios y los distintos componentes se desarrollen de manera controlada y verificable.

El primero de estos niveles corresponde a la seguridad en las comunicaciones. Todo el tráfico entre el cliente y el sistema, así como el que circula internamente entre los microservicios, se encuentra cifrado mediante el uso de TLS. Esta decisión garantiza la confidencialidad e integridad de los mensajes, evitando tanto la inspección del contenido como su modificación en tránsito. El uso de certificados permite además que cada componente verifique la identidad de los demás, reduciendo el riesgo de ataques de suplantación dentro de la red distribuida.

Un segundo nivel de protección se obtiene mediante la propia organización del sistema en microservicios. La separación funcional y el aislamiento inherente que proporciona la ejecución de cada servicio en un contenedor independiente limitan el impacto potencial de un fallo o una intrusión. Un eventual compromiso de un servicio no concede acceso automático a los datos o procesos de los demás, ya que las interacciones entre ellos se encuentran restringidas a canales y protocolos estrictamente definidos. Este aislamiento reduce significativamente la posibilidad de escaladas laterales dentro del sistema.

El tercer nivel de seguridad se centra en el control de acceso. La autenticación y la autorización se gestionan de manera centralizada en la capa de entrada mediante el API Gateway. Cada petición recibida es verificada mediante un mecanismo basado en tokens firmados, lo que permite validar la identidad del usuario y determinar si posee permisos para ejecutar la operación solicitada. Esta estrategia evita duplicar lógica de autenticación en los servicios internos y garantiza que toda solicitud procesada por ellos provenga de un emisor ya validado. De esta forma, el Gateway actúa como un punto único de decisión que define claramente los límites de acceso al sistema.

La combinación de estos tres enfoques proporciona un modelo integral de seguridad que refuerza la fiabilidad y estabilidad de la Agenda Distribuida. La protección del tráfico, el aislamiento entre servicios y la verificación constante de identidades contribuyen conjuntamente a crear un entorno resistente a fallos y accesos no autorizados, acorde con los requisitos de un sistema distribuido de alta disponibilidad.

3 Conclusiones

La arquitectura desarrollada para la Agenda Distribuida constituye una solución completa y coherente a los desafíos fundamentales de los sistemas distribuidos. El diseño integra de manera estructurada distintas estrategias de disponibilidad, coordinación, consistencia y seguridad, asignando a cada capa el nivel de rigor requerido por su función. La combinación de un punto de entrada tolerante a fallos basado en direcciones virtuales, supervisión activa para la capa de mensajería y un clúster de consenso Raft para la persistencia permite alcanzar un sistema robusto capaz de operar de forma estable aun ante la presencia de fallos parciales.

Asimismo, el uso de microservicios sin estado, la replicación controlada de servicios críticos y la separación estricta de responsabilidades contribuyen a un comportamiento predecible y a una recuperación rápida ante interrupciones. Los mecanismos implementados no solo responden a los requisitos académicos planteados, sino que también reflejan buenas prácticas de ingeniería en entornos distribuidos. Como línea futura de trabajo, se prevé profundizar en la validación empírica de los escenarios de fallo y optimizar los mecanismos de supervisión y descubrimiento para adaptarlos a cargas de trabajo más exigentes.