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

November 29, 2025

Abstract

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.

2.1 Arquitectura

La organización del sistema se basa en un paradigma de microservicios, donde cada componente posee un rol definido y se implementan estrategias de alta disponibilidad para eliminar puntos únicos de fallo. La arquitectura se compone de un punto de entrada de alta disponibilidad que utiliza un par de instancias de API Gateway en modo Activo-Pasivo, cuya visibilidad externa se gestiona a través de una Dirección IP Virtual. Tras esta capa, operan los servicios de lógica de negocio, User y Group Service, los cuales son sin estado (stateless) y pueden ser replicados para redundancia. La comunicación asíncrona entre servicios se apoya en una capa de mensajería Redis, también configurada para alta disponibilidad en un par Primario-Réplica. La gestión de este par es orquestada por un servicio dedicado, el Redis Supervisor. Finalmente, la persistencia de los datos se delega en el DB Service, un clúster de cinco nodos que utiliza el algoritmo de consenso Raft para garantizar la consistencia y durabilidad del estado del sistema.

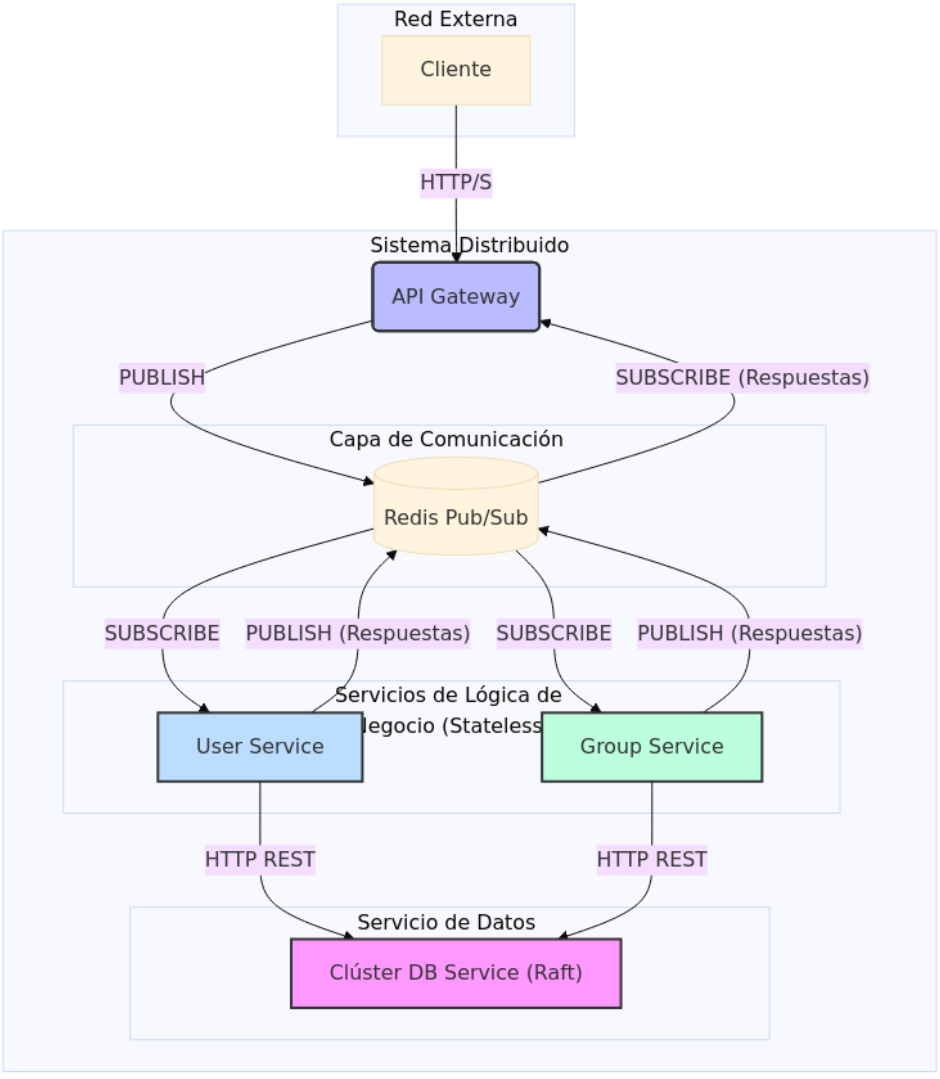

Figure 1: Diagrama General de la Arquitectura de Alta Disponibilidad

2.2 Procesos

El sistema se compone de varios tipos de procesos, cada uno aislado en un contenedor Docker. Además de los procesos correspondientes a los servicios de Gateway, lógica de negocio y base de datos, se define un proceso supervisor, el Redis Supervisor, cuya única responsabilidad es orquestar la alta disponibilidad de la capa de mensajería. La organización de estos procesos sobre una infraestructura de dos máquinas físicas sigue un modelo de failover. El API Gateway se despliega como un par Activo-Pasivo, con cada instancia en una máquina distinta, y lo mismo ocurre con el par Redis Primario-Réplica y sus respectivos supervisores. Los servicios de lógica de negocio se replican en ambas máquinas para redundancia, mientras que los nodos del clúster de base de datos se distribuyen entre ambas para maximizar la resiliencia. En cuanto al desempeño, el sistema aprovecha el modelo de concurrencia de Go para un manejo eficiente de las operaciones.

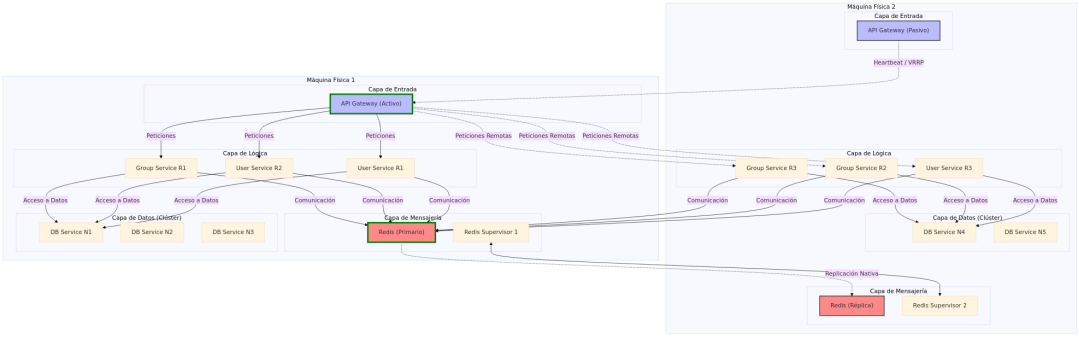

Figure 2: Distribución Física de Procesos en las Máquinas

2.3 Comunicación

El sistema emplea un enfoque de comunicación híbrido y consciente de la topología. La comunicación cliente-servidor se dirige a una IP Virtual estable, que a nivel de red siempre apunta al API Gateway activo. La comunicación interna, por su parte, utiliza un patrón de mensajes de Publicación/Suscripción sobre Redis para el flujo asíncrono. Un aspecto clave de este diseño es que los servicios clientes de Redis deben ser dinámicos; antes de establecer una conexión, consultan una fuente de verdad consistente (el propio DB Service) para obtener la dirección del nodo Redis primario actual. La comunicación con el DB Service se realiza mediante HTTP REST al líder del clúster, y la comunicación interna para el consenso Raft utiliza RPC sobre sockets TCP para máxima eficiencia.

2.4 Coordinación

La coordinación de acciones es un aspecto fundamental que se manifiesta en tres niveles distintos. El nivel más robusto es el consenso Raft dentro del DB Service, que utiliza la elección de líder y el log replicado para la toma de decisiones distribuidas sobre el estado de los datos, garantizando el acceso exclusivo y previniendo condiciones de carrera. Un segundo mecanismo de coordinación es el implementado por el Redis Supervisor, el cual debe decidir de forma autónoma cuándo iniciar un failover y orquestar la promoción de una réplica a nuevo primario. Finalmente, el mecanismo de failover del API Gateway

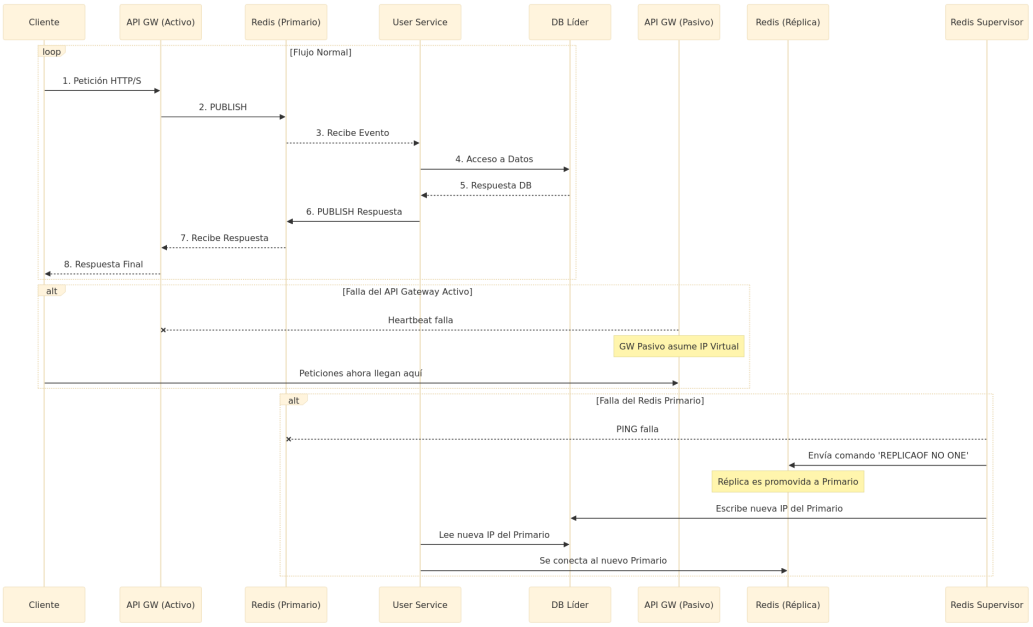

Figure 3: Diagrama de Secuencia de Comunicación y Escenarios de Failover

requiere una coordinación a nivel de red, donde el nodo pasivo debe detectar la ausencia del activo para reclamar la posesión de la IP Virtual.

2.5 Nombrado y Localización

La localización de recursos en el sistema se resuelve con estrategias adaptadas a la criticidad de cada servicio. Para el descubrimiento de los servicios de lógica de negocio, que son sin estado y replicados, se utiliza un protocolo de Gossip descentralizado, permitiendo que cada nodo mantenga una visión actualizada de sus pares. Sin embargo, la localización de los servicios con estado, como el Redis primario y el líder del clúster Raft, es más crítica y no puede depender de un mecanismo eventualmente consistente. Por ello, sus direcciones se registran en una ubicación centralizada y altamente consistente, el propio DB Service, que actúa como fuente de verdad para que los demás servicios del sistema puedan localizar estos puntos de conexión vitales.

2.6 Consistencia y Replicación

La estrategia de replicación varía según el rol del servicio. La replicación de los datos principales se gestiona en el DB Service con cinco réplicas, donde el algoritmo Raft garantiza consistencia fuerte. Para la capa de mensajería, se utiliza el modelo de replicación Primario-Réplica nativo de Redis, que prioriza el rendimiento. La consistencia de este servicio se ve reforzada por el Redis Supervisor, que, aunque no puede prevenir la pérdida de mensajes en tránsito durante un fallo, garantiza la recuperación automática del servicio. Los servicios de lógica de negocio, al ser sin estado, se replican para disponibilidad, pero no requieren replicación de datos entre sus instancias.

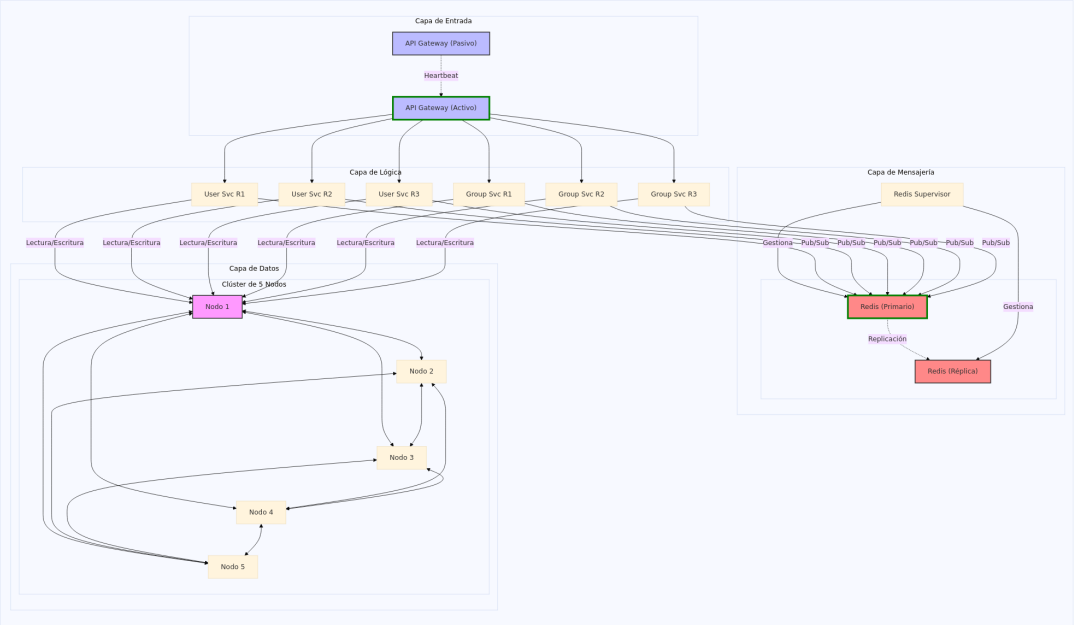

Figure 4: Modelo de Replicación de Componentes por Rol

2.7 Tolerancia a Fallas

El sistema está diseñado para ser resiliente a fallos en sus tres componentes críticos. Para el punto de entrada, se implementa un mecanismo de failover tipo VRRP, donde el nodo API Gateway pasivo monitoriza al activo mediante heartbeats a nivel de red y se autoasigna la IP Virtual si estos cesan. Para la capa de mensajería, el Redis Supervisor monitoriza la salud del Redis primario con comandos a nivel de aplicación; si este falla, promueve la réplica a nuevo primario y actualiza la nueva dirección en la configuración centralizada. Finalmente, para la capa de datos, el clúster Raft de 5 nodos, alcanza un nivel de tolerancia a fallos de k=2, pudiendo sobrevivir a la falla de hasta 2 nodos sin perder consistencia ni disponibilidad.

2.8 Seguridad

La seguridad del sistema se aborda desde tres perspectivas. Primero, la seguridad en la comunicación se garantiza cifrando todo el tráfico, tanto externo como interno, mediante TLS. Segundo, el diseño de microservicios proporciona un aislamiento natural entre componentes, limitando el impacto de una posible brecha de seguridad. Tercero, la autorización y autenticación se gestionan de forma centralizada en el API Gateway activo. Este valida los tokens JWT de las peticiones entrantes, actuando como un único punto de control de acceso cuya propia estabilidad está garantizada por el mecanismo de failover.

3 Conclusiones

La arquitectura distribuida propuesta para la Agenda Distribuida logra un alto grado de robustez y resiliencia. Mediante la aplicación de principios de sistemas distribuidos y, crucialmente, la implementación de soluciones a medida para la alta disponibilidad, el diseño aborda de manera integral los requisitos académicos. La implementación de un

mecanismo de failover tipo VRRP para el API Gateway, un supervisor de servicio para la capa de mensajería Redis, y el uso de consenso Raft para la capa de datos, constituyen un sistema completo y tolerante a fallos. El trabajo futuro se centrará en la implementación efectiva de estos algoritmos de alta disponibilidad y en la validación de su comportamiento ante fallos reales.