

# *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 organización del sistema se basa en un paradigma de microservicios con el objetivo fundamental de eliminar puntos únicos de fallo. Para lograrlo, se ha diseñado una arquitectura en capas, donde cada capa implementa una estrategia de alta disponibilidad específica a su rol. El rol de punto de entrada es asumido por un par de instancias de API Gateway en modo Activo-Pasivo, cuya presencia en la red se abstrae detrás de una Dirección IP Virtual. Esta decisión, si bien limita la escalabilidad del punto de entrada a la capacidad de una sola máquina, se adapta al requisito de operar sobre dos nodos físicos y permite implementar un mecanismo de failover robusto. Tras esta capa operan los servicios de lógica de negocio, User y Group Service, diseñados como servicios sin estado para permitir una replicación simple que garantice la redundancia. La comunicación asíncrona, clave para el desacoplamiento, se apoya en una capa de mensajería Redis en un par Primario-Réplica, cuya conmutación por error es orquestada por un servicio dedicado, el Redis Supervisor. Finalmente, la capa de persistencia se materializa en el DB Service, un clúster de cinco nodos que utiliza el algoritmo Raft para ofrecer una base de datos consistente y tolerante a fallos.

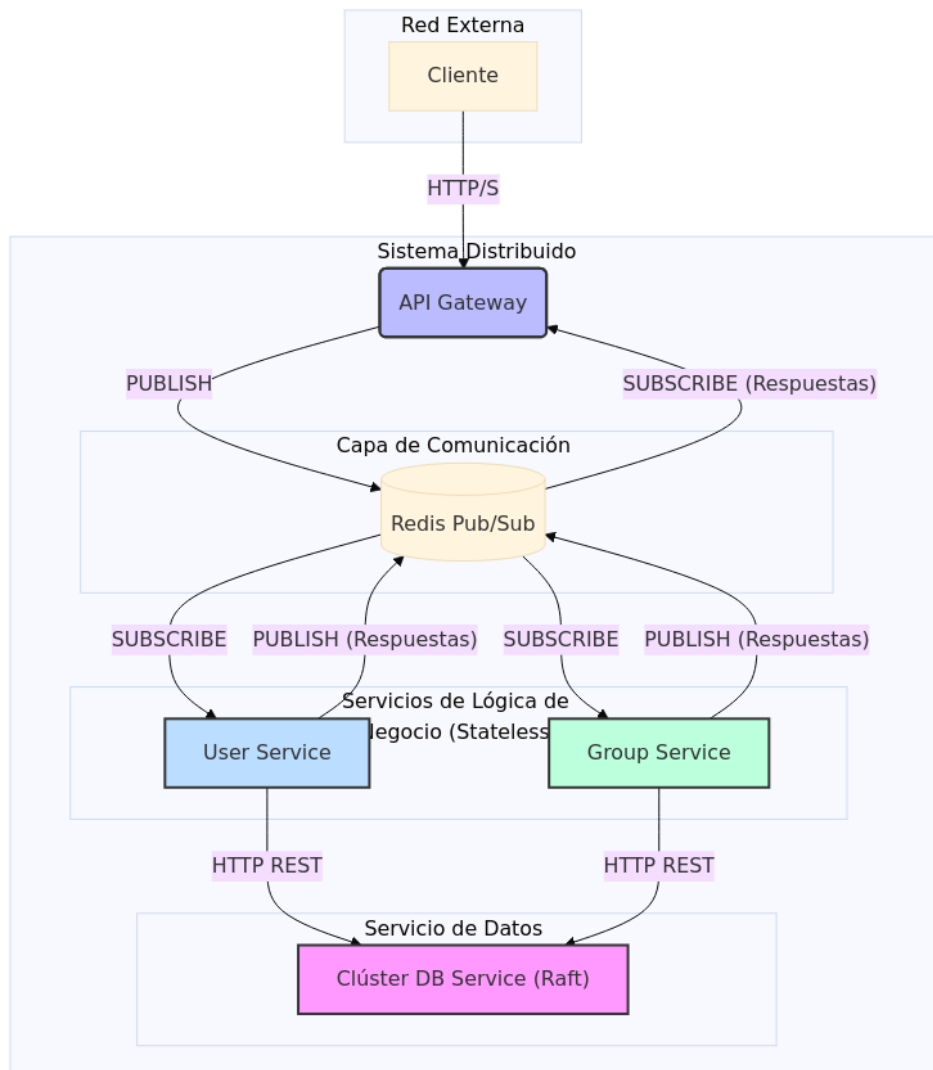


Figure 1: Diagrama General de la Arquitectura de Alta Disponibilidad

## 2.2 Procesos

El sistema se compone de varios tipos de procesos, cada uno aislado en un contenedor Docker para facilitar su despliegue y gestión. Se distinguen los procesos sin estado, como el API Gateway y los servicios de lógica, de aquellos con estado, como Redis y el DB Service. Adicionalmente, se define un proceso supervisor, el Redis Supervisor, cuya única responsabilidad es una tarea de orquestación. La organización de estos procesos sobre una infraestructura de dos máquinas físicas está diseñada para maximizar la resiliencia; el par Activo-Pasivo del Gateway y el par Primario-Réplica de Redis se distribuyen con una instancia en cada máquina, garantizando que la falla de una máquina no elimine ambos componentes. De igual forma, los cinco nodos del clúster Raft se reparten entre las máquinas para sobrevivir a la pérdida de un nodo físico. Para el desempeño, el sistema aprovecha intensivamente el modelo de concurrencia de Go. El uso de goroutines permite a cada proceso manejar un gran número de operaciones de red concurrentes con un bajo costo computacional en comparación con los modelos tradicionales basados en hilos, lo cual es ideal para la naturaleza de los microservicios, que son fundamentalmente I/O-bound.

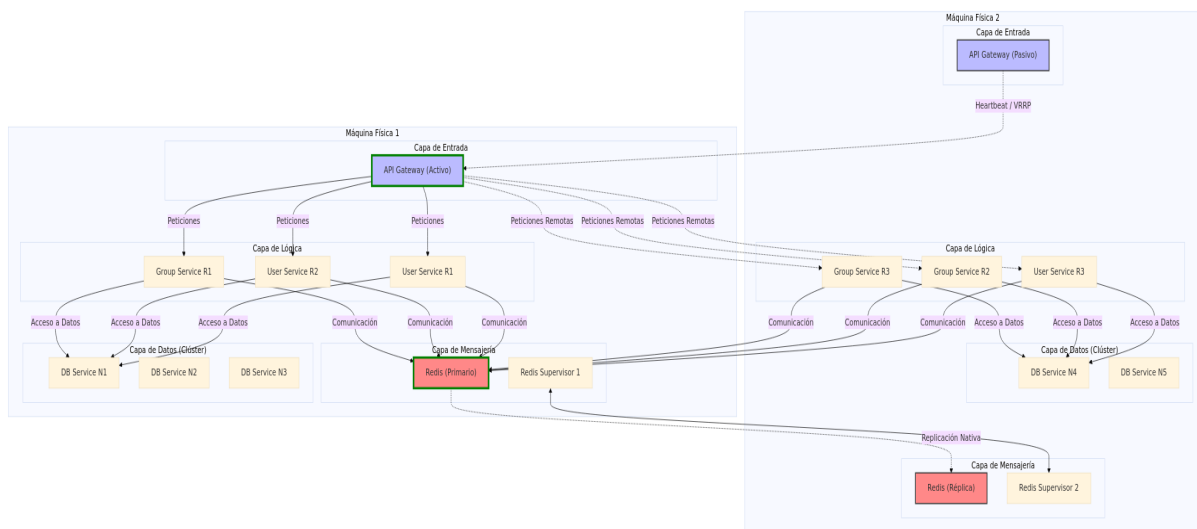


Figure 2: Distribución Física de Procesos en las Máquinas

## 2.3 Comunicación

El sistema emplea un enfoque de comunicación híbrido, seleccionando el protocolo más adecuado para cada tipo de interacción. La comunicación cliente-servidor se realiza exclusivamente a través de HTTP REST sobre TLS (HTTPS), utilizando la IP Virtual como punto de contacto estable. La comunicación interna, sin embargo, es más diversa. Se optó por un patrón de mensajes de Publicación/Suscripción sobre Redis para el flujo asíncrono entre el Gateway y los servicios de lógica. Esta elección fomenta el desacoplamiento, permitiendo que los servicios evolucionen de forma independiente. Para la comunicación síncrona, como las consultas de los servicios de lógica al líder del clúster de datos, se utiliza HTTP REST por su simplicidad. Finalmente, para la comunicación de alta frecuencia y baja latencia requerida por el algoritmo Raft, se emplea RPC sobre sockets TCP directos, minimizando el overhead que introducirían protocolos como HTTP.

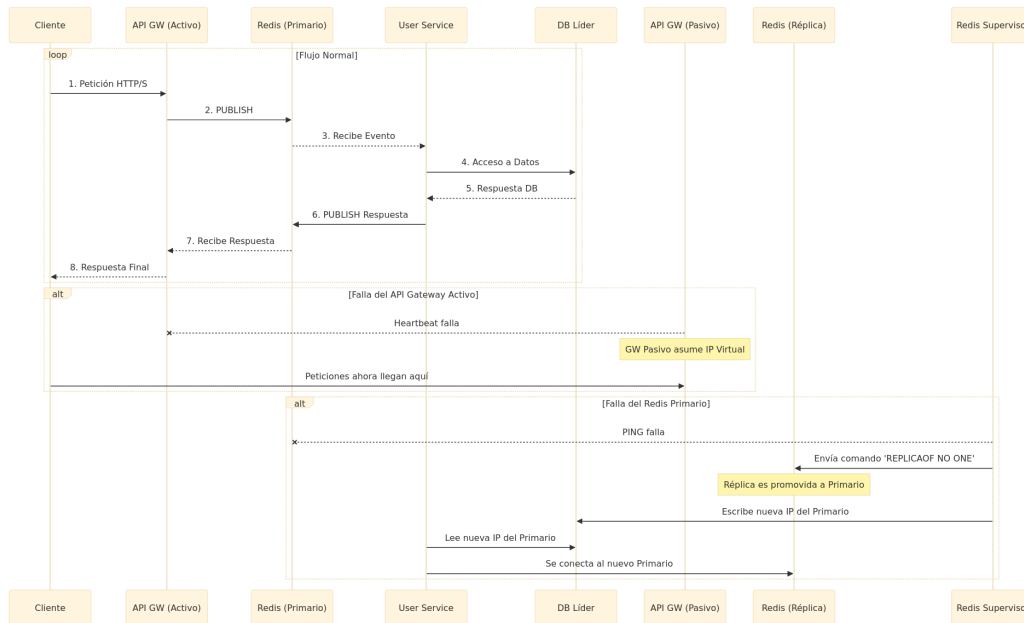


Figure 3: Diagrama de Secuencia de Comunicación y Escenarios de Failover

## 2.4 Coordinación

La coordinación de acciones es un aspecto fundamental que se manifiesta en tres niveles distintos, cada uno con un mecanismo diferente. El nivel más robusto es el consenso Raft dentro del DB Service. Este se utiliza para la toma de decisiones distribuidas sobre el estado de los datos, garantizando el acceso exclusivo a los recursos y previniendo condiciones de carrera mediante la secuenciación de todas las operaciones a través de un líder estable. Un segundo mecanismo de coordinación, más simple pero igualmente crucial, es el implementado por el Redis Supervisor. Este proceso realiza una forma de elección de líder al 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 requiere una coordinación a nivel de red, donde el nodo pasivo debe detectar la ausencia del activo para reclamar de manera exclusiva la posesión de la IP Virtual.

## 2.5 Nombrado y Localización

La localización de recursos en el sistema se resuelve con estrategias duales, adaptadas a la criticidad y consistencia requerida por cada servicio. Para el descubrimiento de los servicios de lógica de negocio, que son sin estado y replicados, un modelo de consistencia eventual es aceptable. Por ello, se utiliza un protocolo de Gossip descentralizado, permitiendo que cada nodo mantenga una visión local y actualizada de sus pares sin depender de un punto central. Por el contrario, la localización de los servicios con estado, como el Redis primario y el líder del clúster Raft, es crítica para la operación correcta del sistema. Conectarse a un nodo incorrecto podría llevar a errores o a la lectura de datos obsoletos. Por esta razón, sus direcciones se registran en una ubicación centralizada y altamente consistente, el propio DB Service, que actúa como una fuente de verdad autoritativa.

## 2.6 Consistencia y Replicación

La estrategia de replicación y el modelo de consistencia ofrecido varían 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, específicamente linealizabilidad, asegurando que todas las operaciones parezcan ocurrir instantáneamente en un único punto en el tiempo. Para la capa de mensajería, se utiliza el modelo de replicación Primario-Réplica asíncrono nativo de Redis. Este enfoque prioriza el alto rendimiento sobre la consistencia estricta, lo que implica un riesgo teórico de pérdida de mensajes si el primario falla antes de que un mensaje sea replicado. La confiabilidad del \*servicio\* de mensajería, sin embargo, se garantiza mediante nuestro Redis Supervisor, que automatiza la recuperación ante fallos. Los servicios de lógica de negocio, al ser sin estado, se replican únicamente para disponibilidad.

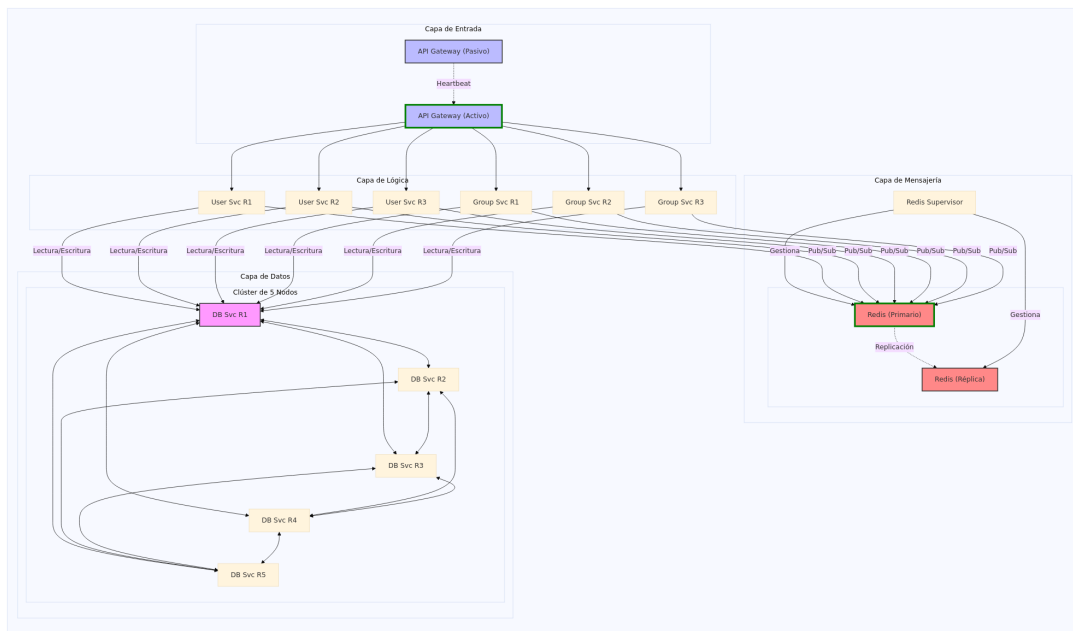


Figure 4: Modelo de Replicación de Componentes por Rol

## 2.7 Tolerancia a Fallas

El sistema está diseñado para ser resiliente a fallos en sus tres componentes críticos. La respuesta a errores se ha implementado a medida para cada caso. Para el punto de entrada, se emplea un mecanismo de failover tipo VRRP, donde el nodo API Gateway pasivo monitoriza al activo mediante heartbeats a nivel de red. Si estos cesan, el nodo pasivo ejecuta comandos a nivel de sistema operativo para autoasignarse la IP Virtual, redirigiendo el tráfico de forma transparente para el cliente. Para la capa de mensajería, el Redis Supervisor realiza un chequeo a nivel de aplicación; si el Redis primario falla, el supervisor ejecuta comandos de aplicación para promover la réplica y actualiza la nueva dirección en la configuración centralizada. Finalmente, la capa de datos, con su clúster Raft de 5 nodos, alcanza un nivel de tolerancia a fallos de  $k=2$ , pudiendo sobrevivir a la falla de hasta 2 nodos gracias a su mecanismo de quórum, sin perder consistencia ni disponibilidad.

## 2.8 Seguridad

La seguridad del sistema se aborda desde tres perspectivas para crear una defensa en profundidad. Primero, la seguridad en la comunicación se garantiza cifrando todo el tráfico, tanto el externo desde el cliente como el interno entre microservicios, mediante TLS. Segundo, el diseño de microservicios proporciona un aislamiento natural entre componentes; un compromiso en un servicio no otorga acceso directo a los datos o la lógica de los demás, 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 actúa como un punto de control de acceso (choke point), validando los tokens JWT de las peticiones entrantes. Al centralizar esta lógica, se simplifica el código de los servicios internos, que operan bajo la premisa de que toda petición recibida ha sido previamente autenticada.

## 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.