****Member Management System****

**Version: 1.0**

**Date: 2025-09-11**

# 1. System Overview

**The Member Management System (MMS) is a scalable, secure, cloud-native backend service suite designed to handle user registration, authentication, profile management, and membership organization for a massive user base (up to 1 billion members). The architecture is based on microservices principles, leveraging the Spring Cloud ecosystem for resilience, discoverability, and centralized API management.**

# 2. Architectural Goals & Principles

**Scalability: Horizontal scaling of stateless services to meet the demand of 1B+ users.**

**Security: Centralized authentication/authorization, secure credential storage, and data protection.**

**Performance: Low-latency responses (<2s for auth) achieved through caching, database optimization, and efficient load balancing.**

**Resilience: Fault tolerance to prevent single points of failure and ensure system availability.**

**Maintainability: Loosely coupled, independently deployable services.**

# 3. Proposed Technology Stack

**Component Technology Choice Justification**

**API Gateway Spring Cloud Gateway Lightweight, performant, and provides essential routing, security, and cross-cutting concern logic.**

**Service Discovery Spring Cloud Netflix Eureka Allows microservices to locate and communicate with each other dynamically in a cloud environment.**

**Configuration Server Spring Cloud Config Server Externalized and centralized configuration management for all microservices.**

**Primary Database MariaDB A robust, open-source, relational database. Meets requirements for data integrity, transactions, and complex relationships.**

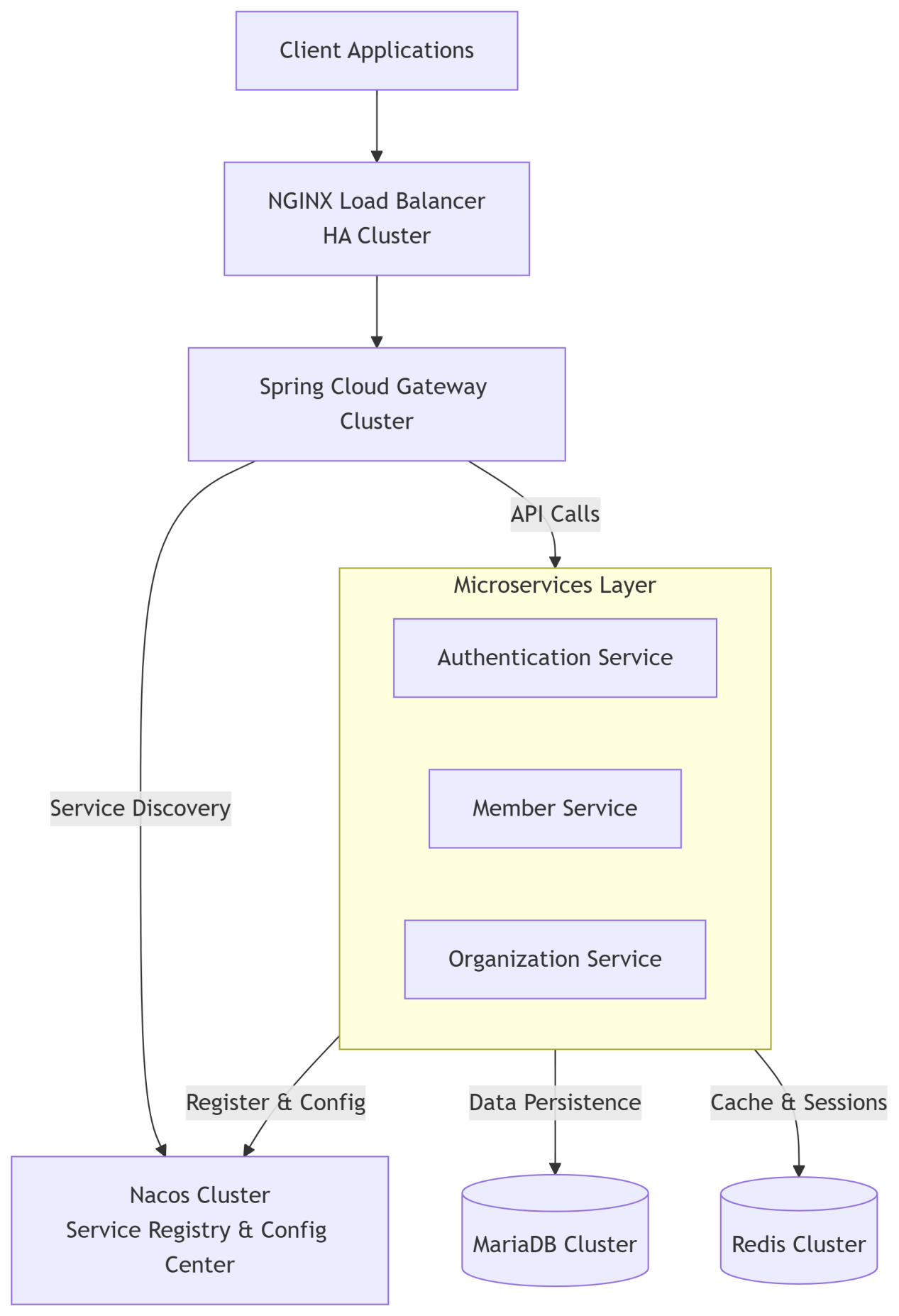
**Caching Layer Redis In-memory data store for session management, caching frequently accessed data (profiles, permissions), and rate limiting. Extremely fast, meeting PERF-1 and PERF-3.**

**Application Framework Spring Boot / Spring Cloud De-facto standard for building Java-based microservices. Provides extensive integration with the chosen stack.**

**Security Spring Security, JWT To handle authentication and authorization flows securely.**

**Build & Deployment Docker, Kubernetes (K8s) For containerization, orchestration, easy scaling, and management of microservices.**

# 4. High-Level Architecture Design Diagram

****

# 5. Service Breakdown & Responsibilities

**Service Name Responsibility Key Functions Data Store**

**API Gateway Single entry point for all client requests. Request routing, rate limiting, JWT validation, SSL termination, logging. (Stateless)**

**Authentication Service Handles user identity and security. User registration (SRS 2.1), login, JWT issuance & validation, password hashing (SEC-1). MariaDB (for credentials), Redis (for blacklisted tokens)**

**Profile Management Service Manages member profile data. Update member profile (SRS 2.2), retrieve own profile. MariaDB (profiles), Redis (cache)**

**Search Service Handles member search operations. Search for members (SRS 2.3), data filtering (SEC-4), pagination (PERF-2). MariaDB (indexed search)**

**Organization Service Manages organizational structures and roles. Assigning default org/role (SRS 2.1), managing permissions. MariaDB**

# 6. Data Design (High-Level)

**MariaDB Schema:**

**members table: Core member data (id, login\_id [UNIQUE - DATA-1], securely hashed password, status, created\_date).**

**profiles table: Extended profile information (member\_id [FK], first\_name, last\_name, email, etc.).**

**organizations table: Org structures.**

**member\_organizations table: Junction table managing the many-to-many relationship between members and orgs, including their role (supports CON-2).**

**Sharding Strategy: To scale to 1B users, the members table will be sharded horizontally using a strategy like Range Sharding or Hash Sharding on the member\_id.**

**Redis Usage:**

**Cache: Key-value store for member:{id} and profile:{id} to reduce database load (PERF-3).**

**Session Store: Store short-lived JWT refresh tokens or blacklist invalidated tokens.**

**Rate Limiting: Store request counts for API rate limiting on the Gateway.**

# 7. Key Data Flows

**1. User Registration (SRS 2.1):**

**Client -> Gateway -> Auth Service.**

**Auth Service validates verification proof (integrates with external system - CON-3).**

**Service checks for unique login\_id (DATA-1).**

**Service hashes password with bcrypt/scrypt (SEC-1) and stores user in MariaDB.**

**Auth Service calls Organization Service to assign default org/role.**

**Success response is returned to the client.**

**2. Member Search (SRS 2.3):**

**Authorized Client -> Gateway (validates JWT & permissions - SEC-3) -> Search Service.**

**Search Service queries MariaDB with filters.**

**Service applies data masking/filtering to protect sensitive data (SEC-4) before returning the paginated result set.**

# 8. Addressing Non-Functional Requirements

**SEC-1 (Password Hashing): Implemented in Auth Service using Spring Security's BCryptPasswordEncoder.**

**SEC-2, SEC-3, SEC-4 (Access Controls): Enforced at the API Gateway (JWT validation) and within each service's business logic based on roles/permissions.**

**PERF-1 (2s Auth Response): Achieved through Redis caching, efficient hashing algorithms, and horizontal scaling of the Auth Service.**

**PERF-2 (Pagination): Implemented in the Search Service database queries.**

**PERF-3 (Caching): Redis is used to cache member profiles and frequently accessed data.**

**PERF-4 (1B Members): Addressed by sharding the MariaDB database and using de-normalized data models where appropriate for read performance.**

**DATA-1 (Unique Login): Enforced by a unique constraint in the MariaDB members table.**

**CON-2 (Org Compatibility): Handled by the flexible Organization Service and its related data model.**