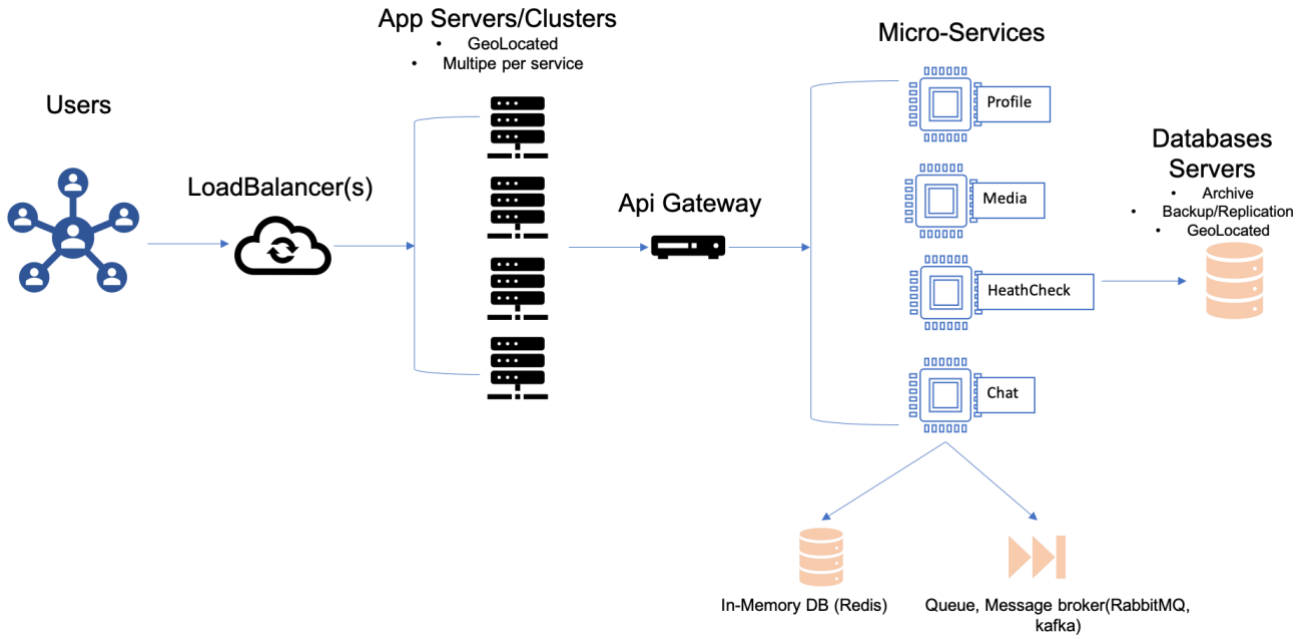
Messaging backend for apps like WhatsApp or Skype

A messaging backend like WhatsApp or Skype would require a distributed architecture that can handle millions of users simultaneously. Here is a high-level overview of the architecture schematic for such a system:



Load Balancer: To ensure that application servers can handle the load from millions of users, a load balancer should be used to distribute traffic across multiple servers/clusters. The load balancer should be intelligent enough to route traffic to the least busy server, ensuring that no single server is overwhelmed with traffic.

Application Servers: to handle user's requests and pass it along to the API gateway to be handled by respective micro-services accordingly.

Api Gateway: enable a centralized and holistic management of all APIS of various micro-services.

Micro-services:

- **Profile**: to manage each user's KYC data (storing, updating, retrieval)
- **Media**: manage images, videos, documents exchanged through users chat conversations
- **Health Check**: to monitor, benchmark usage for traceability and proactive improvements
- **Chat**: Process, store and exchange messages either user to user or groups of users
    - **Push Notification Sub-Service**: To enable real-time messaging, a push notification service should be used to notify users instantly when they receive a new message. The push notification service should be designed to

be scalable and fault-tolerant, ensuring that users receive their notifications in real-time.

Databases: to persist data, a geolocated setup would decreases latency, improve user experience and bring on board the real time factor.

Scalability issues:

- Handling a large number of concurrent connections: As the number of users in the messaging app grows, the system needs to be able to handle a large number of concurrent connections efficiently. This can be addressed by using a load balancer to distribute traffic across multiple messaging servers and using a non-blocking I/O model to ensure that the server can handle a large number of connections efficiently.

- Scaling the database: As the amount of data stored in the database grows, the database may become a bottleneck. This can be addressed by using sharding to distribute the database workload across multiple servers, caching to reduce the load on the database server, and using a highly available and fault-tolerant database.

- Real-time communication: Real-time communication between clients requires low latency and high throughput. This can be addressed by using web sockets for real-time communication and optimizing the network stack to minimize latency.

Performance issues:

- Message processing: Processing a large number of messages can be CPU-intensive and can cause performance issues. This can be addressed by using distributed systems and parallel processing to distribute the workload across multiple servers. Furthermore**,** Caching can help reduce the load on the database server by storing frequently accessed data in memory.

- Message history: Storing and retrieving message histories can be slow and can cause performance issues. This can be addressed by using caching to store frequently accessed messages, using a highly available and fault-tolerant database (in-memory databases where applicable) and optimizing database queries.

- Push notifications: Sending push notifications can be slow and can cause performance issues. This can be addressed by using a push notification service that is optimized for performance and can handle a large number of requests.

The following approaches can help address these issues:

- **Distributed Architecture**: Using a distributed architecture that can handle multiple servers can ensure that the system is scalable and can handle a large number of users.

- **Load Balancing**: Load balancing can help distribute traffic across multiple servers, ensuring that no single server is overwhelmed with traffic.

- **Caching**: Caching can help reduce the load on the database server by storing frequently accessed data in memory

- **Sharding**: Sharding can help distribute the database workload across multiple servers, ensuring that the system can handle a large number of users.

- **Compressio**n: Compressing messages and media files can help reduce the amount of data that needs to be transferred, improving performance and reducing server load.

- **Real-time Messaging**: Real-time messaging can help reduce the load on the server by minimizing the amount of time that clients need to stay connected to the server.

Overall, designing a messaging backend like WhatsApp or Skype requires careful consideration of scalability and performance. By using a distributed architecture, load balancing, caching, sharding, compression, and real-time messaging, it is possible to design a system that can handle millions of users simultaneously.