**📷 Instagram Clone – Scalable Microservices Architecture**

A feature-rich Instagram backend clone built with a focus on **microservices**, **asynchronous communication**, **scalability**, and **resilience**. This system includes essential functionalities like posts, comments, likes, profile management, and secure authentication using modern cloud-native technologies.

**🧭 Overview**

This project is a scalable clone of Instagram’s backend, implemented as a distributed system composed of multiple microservices. Each service is independently deployable, stateless, and communicates with others via **Kafka**, **REST**, or **Feign Clients**.

The backend supports:

* Uploading and displaying media posts with metadata
* User profiles and authentication
* Comments and likes on posts
* Subscription/follow functionality
* Real-time profile updates via event streaming
* Asynchronous processing and fault tolerance

**✅ Functional Features**

**📄 Post Service**

* Create, update, and delete posts
* Upload multiple media files (images/videos) to **AWS S3**
* Indexing in **Elasticsearch** for efficient search
* Event-based communication with Profile and Feed services via **Kafka**

**👤 Profile Service**

* Stores user profile data: username, avatar, bio, status
* Reacts to post creation events via Kafka
* Cached profile lookups using **Redis**

**💬 Comment Service**

* Post and retrieve comments per post
* Asynchronous persistence to optimize performance
* Designed to handle high-volume interactions

**❤️ Like Service**

* Add and remove likes on posts
* Separate microservice to isolate write pressure
* Publishes like events to analytics or feed system

**🔐 Authentication & User Management**

* Registration, login, and secure JWT-based token authentication
* Password encryption and validation
* Role-based authorization

**📦 Media Storage**

* Uses **AWS S3** for storing images and videos
* Pre-signed URLs for secure, client-side uploads
* Links are stored in the Post Service

**⚙️ Architecture Highlights**

* **Microservice Architecture**: Each domain (posts, users, likes, comments, calls) runs as an independent Spring Boot service
* **Service Discovery**: Integrated with **Spring Cloud Eureka** for dynamic routing
* **API Gateway**: Routes all external traffic and aggregates documentation (Swagger UI)
* **Async Communication**: **Kafka** is used to propagate events (e.g., PostCreated, UserUpdated) across services
* **Caching Layer**: **Redis** is used for hot-profile caching and token verification
* **Object Storage**: **AWS S3** is used to persist and serve large media content
* **Internal REST**: Lightweight **Feign Clients** connect internal services
* **Secure Auth**: JWT tokens protect endpoints and enforce authorization

**📈 Non-Functional Goals**

**⚡ Asynchronous & Non-Blocking Design**

* Kafka-based messaging decouples services
* Each service is reactive-ready and supports high-concurrency scenarios

**💥 Fault Tolerance & Resilience**

* Kafka ensures message delivery even if target services are temporarily down
* Redis caches user/profile data to reduce DB load
* Retry mechanisms are implemented for media uploads and inter-service calls

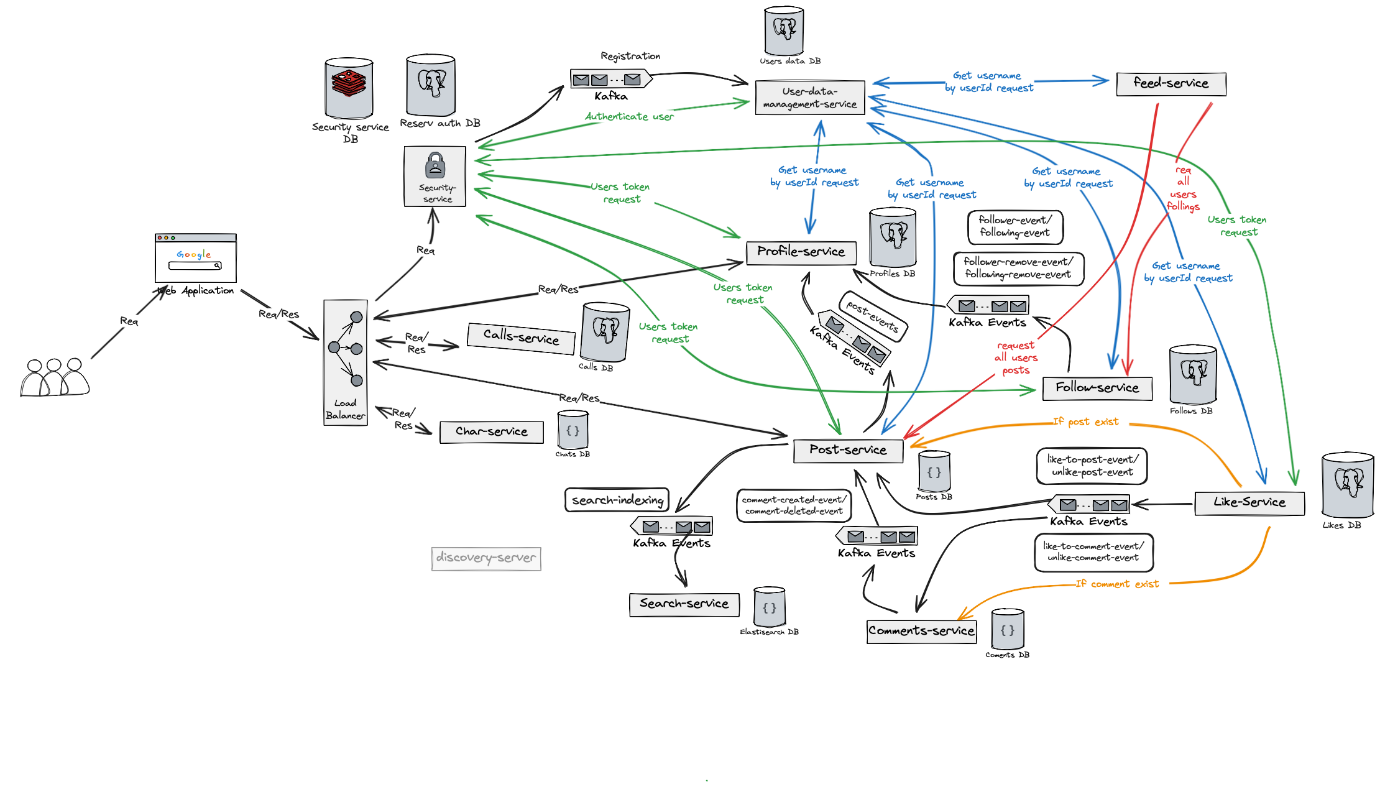
**🚀 Scalability**

* Horizontal scaling possible per service
* Stateless architecture allows container-based deployment (e.g., Docker Compose)

**🧰 Tech Stack**

* **Java 21**, **Spring Boot 3.x**
* **Spring Cloud Gateway**, **Eureka**, **Feign**
* **PostgreSQL**, **MongoDB**
* **Apache Kafka** (KRaft mode)
* **Redis**
* **AWS S3** for media
* **Elasticsearch** for search
* **Docker / Docker Compose**

**Architecture diagram**

[](https://private-user-images.githubusercontent.com/128328679/443692468-86b0e27f-63c2-4a04-b9e3-8e61f2814f7f.png?jwt=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJnaXRodWIuY29tIiwiYXVkIjoicmF3LmdpdGh1YnVzZXJjb250ZW50LmNvbSIsImtleSI6ImtleTUiLCJleHAiOjE3NDc3NDg1MDAsIm5iZiI6MTc0Nzc0ODIwMCwicGF0aCI6Ii8xMjgzMjg2NzkvNDQzNjkyNDY4LTg2YjBlMjdmLTYzYzItNGEwNC1iOWUzLThlNjFmMjgxNGY3Zi5wbmc_WC1BbXotQWxnb3JpdGhtPUFXUzQtSE1BQy1TSEEyNTYmWC1BbXotQ3JlZGVudGlhbD1BS0lBVkNPRFlMU0E1M1BRSzRaQSUyRjIwMjUwNTIwJTJGdXMtZWFzdC0xJTJGczMlMkZhd3M0X3JlcXVlc3QmWC1BbXotRGF0ZT0yMDI1MDUyMFQxMzM2NDBaJlgtQW16LUV4cGlyZXM9MzAwJlgtQW16LVNpZ25hdHVyZT05MGZhYTY2YjQzYzAxOWVjZDc2ZTBiNzg4MDI4ZDgyN2YxMWUxYTA5ZGU4M2E5M2YzMWI1ZDI5OGNmYzVhZWU3JlgtQW16LVNpZ25lZEhlYWRlcnM9aG9zdCJ9.7wy8HV2oNutHtbkZuP9BZuo7US2jY3Rm6cZwROjOT7Y)

**📚 API Documentation**

Explore the available REST API endpoints using the centralized Swagger UI via the API Gateway:

🔗 <http://localhost:8013/swagger-ui.html>

Each microservice is documented individually and accessible through the gateway:

* 🛡️ **Authentication Service**  
  <http://localhost:8013/authentication-service/swagger-ui/index.html>
* 👤 **User Data Management Service**  
  <http://localhost:8013/user-data-management-service/swagger-ui/index.html>
* 🧑‍💼 **Profile Service**  
  <http://localhost:8013/profile-service/swagger-ui/index.html>
* 📝 **Post Service**  
  <http://localhost:8013/post-service/swagger-ui/index.html>
* 💬 **Comment Service**  
  <http://localhost:8013/comments-service/swagger-ui/index.html>
* ❤️ **Like Service**  
  <http://localhost:8013/like-service/swagger-ui/index.html>
* 📞 **Calls Service**  
  <http://localhost:8013/calls-service/swagger-ui/index.html>
* 👥 **Follow (Subscription) Service**  
  <http://localhost:8013/follow-service/swagger-ui/index.html>

All documentation is automatically aggregated and exposed via **Springdoc + Spring Cloud Gateway**.

**🗺️ Future Roadmap**

* WebSocket notifications for real-time updates
* Feed aggregation service
* Story support with auto-expiry
* Admin moderation panel
* Rate limiting and analytics

**🧪 Development & Deployment**

All services are containerized and can be launched via:

docker-compose up --build