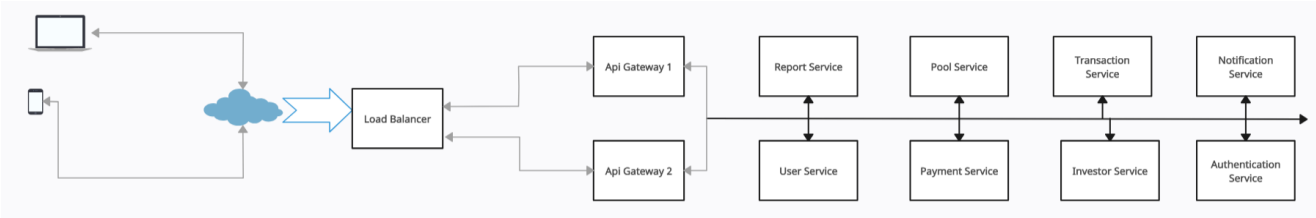
Problem 3



Diagram

1. Frontend Application:

- Develop a user-friendly web application for fund managers and investors to interact with the system.

2. Load Balancer:

- Place a load balancer in front of the application servers to distribute incoming traffic and ensure high availability.

3. Api Gateway:

- These servers host the application logic and communicate with various microservices. They should be stateless to enable horizontal scaling.

4. Microservices:

a. User Service: - Manages user information and stores user info securely.

b. Pool Service: - Handles the creation, management, and deletion of investment pools. - Manages pool-specific information and settings.

c. Investor Service: - Manages investor details and their participation in various investment pools. - Tracks investment amounts for each investor in each pool.

d. Transaction Service: - Handles real-time updates of investment amounts, distributions, and related transactions within investment pools. - Utilizes a distributed ledger or database to maintain transaction history.

e. Notification Service: - Sends notifications to fund managers and investors about updates, changes, and critical events related to their investment pools. - Ensures message delivery reliability.

f. Payment Service: - Manages financial transactions, such as contributions, distributions, and fees. - Integrates with external payment gateways for processing payments securely.

g. Report Service: - Generates reports and analytics on the performance of investment pools. - Provides fund managers and investors with insights. - Ensures data is up-to-date and accurate.

h. Authentication Service: - Manages user authentication and provides tokens for secure communication between services. - Enforces strong security practices to protect sensitive data.

5. Caching Layer:

- Implement a caching layer to improve the performance of frequently accessed data, such as user profiles and recent transaction information.

6. gRPC:

- Use gRPC to communicate between microservices. This enhances scalability and fault tolerance.

7. Monitoring and Logging:

- Use tools like Prometheus and Grafana for monitoring and centralized logging to identify and address performance or security issues in real-time.

8. Backup and Disaster Recovery:

- Regularly backup data and implement a disaster recovery plan to ensure data availability in case of unexpected outages.

Technologies and Tools:

- **Web Applications:** Node.js or Python (Django or Flask for the web applications).
- **API Gateway:** NGINX or Kong for routing and load balancing.
- **Authentication Service:** JWT for user authentication and authorization.
- **Database:** MySQL for storing Data.
- **Load Balancer:** Nginx, HAProxy
- **Messaging/Event Bus:** RabbitMQ for handling real-time updates and notifications (e.g, Email, SMS)
- **Microservices Communication:** gRPC for communication between microservices.
- **Container Orchestration:** Kubernetes for container management.

Potential Bottlenecks and Strategies:

- **Database Scalability:** As the number of users and transactions grow, consider database sharding, caching, and read replicas to handle increased load.

- **Real-time Updates:** Use a message queue (e.g., RabbitMQ) to handle real-time updates efficiently, ensuring data consistency.

- **High Availability:** Implement redundancy and failover mechanisms for critical components, and use auto scaling for web applications.

- **Security:** Implement robust security measures, including data encryption, secure API endpoints, and regular security audits to protect sensitive financial information and ensure compliance with financial industry PCI DSS rules.

Implementing and Deploying in a Remote-First Environment:

- **Development:** Developers can collaborate using version control systems Git (For security reasons you can use your own git server) and use project management tools like Jira or Trello

- **Continuous Integration/Continuous Deployment (CI/CD):** Use tools like Jenkins, Gocd or GitHub Actions to automate testing and deployment.

- **Documentation:** Create thorough documentation for development, deployment, and maintenance procedures to ensure consistency and knowledge sharing among remote teams and create a proper documentation of how system business logic works.