Task explanation file

Problem 1:

Imagine you are given a dataset containing transactional records of different investors in various syndicates over several years. Each record contains an investor ID, syndicate ID, transaction amount, and transaction date. Your task is to identify and list the top 5 investors who have invested in the highest number of unique syndicates, along with the total amount they have invested.

Solution: I'll use mySQL to get the data with just one select query.

```
SELECT

    investor_id,

    COUNT(DISTINCT syndicate_id) AS unique_syndicates,

    SUM(transaction_amount) AS total_investment

FROM transactions

GROUP BY investor_id

ORDER BY

    unique_syndicates DESC,

    total_investment DESC

LIMIT 5;
```

Explanation: Using GROUP BY to get all the transactions of a single investor. Then calculate the total amount and count unique syndicate_id. Lastly, Order by unique_syndicates count to get the top investor first and limit the result to 5 rows only.

Problem 2:

Auptimate wants to build a real-time alerting system that notifies fund managers of unusual transaction activities within their syndicates. The system should monitor transactions in real-time and trigger alerts for the following scenarios:

A single transaction amount exceeding a pre-defined threshold.

A sudden spike in the number of transactions within a short period (e.g., 10x the average rate in the last hour).

Design and implement a prototype of this system, focusing on the algorithm and data structures that will enable real-time processing and analysis.

Solution:

A single transaction: To monitor a single transaction amount threshold, The simple solution is to use an observer design pattern. When a transaction is created, we will trigger a validation check to verify if it's below threshold or not. If it's exceeding the threshold, we can trigger a notification.

Now the scalability, data integrity, and fault tolerance part. To trigger the check we can use a MySQL trigger or a code trigger and send it to a queue channel like Apache Kafka or SQS.

To send the notification reliably we can use any third party email or SMS service like SES, twilio

A sudden spike: To get a sudden spike in an hour, we can count last 24 hours transactions' and keep that in cache. And also we can keep the last one hour's transaction in another cache. And we can check periodically if the count of last hour's transactions are more than 24 hour's average. If yes, we can check how much is the percentage and check with a predefined threshold. If the percentage crosses the predefined transaction, we can send the alert.

Now the scalability, data integrity, and fault tolerance part. We can use reliable cache storage like Kafka, Redis, Amazon ElastiCache etc.

To send the notification reliably we can use any third party email or SMS service like SES, twilio

Coding part: Here I've used Laravel's default system to demonstrate the principle.

Repo Link: https://github.com/sohag-pro/auptimate

What I've done: I've added an Observer to observe transaction creation. Then it'll trigger a job to queue. The job will check the single transaction threshold and will add the current transaction to the cache and remove if there was any transaction that is older than 1 hour.

I’ve created a scheduler to check periodically if there are any sudden spikes in average transactions. Currently it’s set to check every minute. We can configure it as per our need.

The files location:

Observer: app/Observers/TransactionObserver.php

Job: app/Jobs/TransactionAlert.php

Schedule Command: app/Console/Commands/TransactionMonitor.php

Comments and Trade-offs:

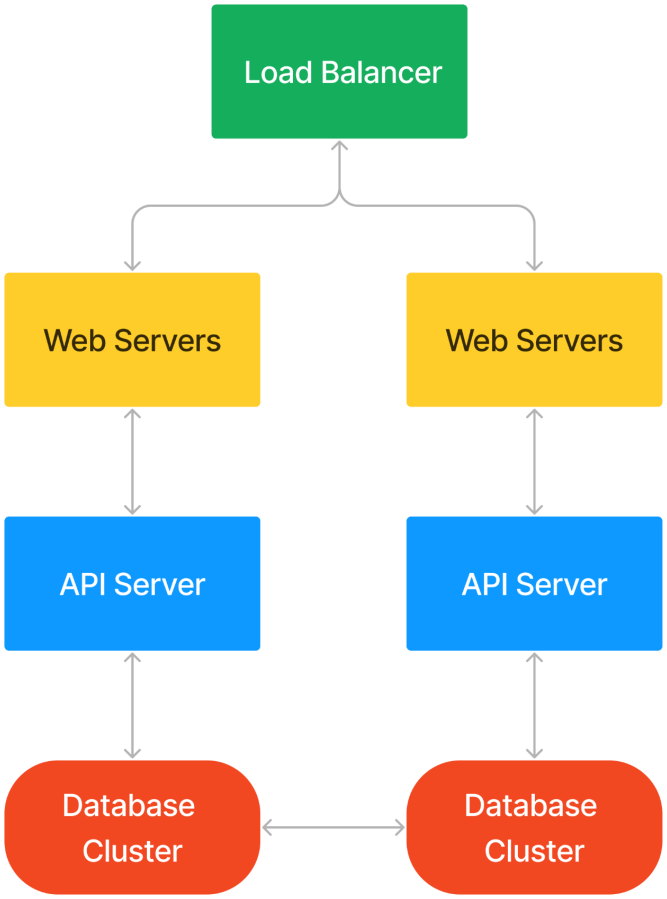
- Trade-offs might involve choosing between low-latency alerting and precision. A shorter time window for spike detection might result in more false positives.
- The choice of the technology stack for data processing and storage will affect scalability and reliability.

This design provides a robust foundation for a real-time alerting system, capable of handling large volumes of transaction data, ensuring data integrity, and responding promptly to unusual activities in the syndicate transactions. Specific implementation details will depend on the chosen technologies and architecture.

Problem 3

Auptimate is planning to launch a new feature that allows fund managers to create and manage investment pools. Each pool can have multiple investors, and each investor can participate in multiple pools. The system should handle real-time updates of investment amounts, distributions, and other related transactions.

Design the architecture for this feature ensuring scalability to accommodate a growing number of users and transactions, reliability to ensure data accuracy and availability, and security to protect sensitive financial information.



Components and Their Interactions:

Load Balancer: Distributes incoming traffic evenly (or as we want) across multiple web servers to ensure high availability and scalability.
Web Servers: Handle incoming HTTP requests from users, including fund managers and investors.

API Servers: Implement the business logic, manage real-time updates of investment amounts, distributions, and transactions, and interact with the database.
Database Cluster: Stores data related to investment pools, fund managers, investors, transactions, and other related information. It should be designed for high availability and scalability.

Technologies and Tools:
Load Balancer: Use industry-standard load balancers like NGINX, AWS Elastic Load Balancer, or Azure Load Balancer for traffic distribution.
Web Servers: Apache, NginX or any other server depending on your team's expertise.
API Servers: Utilize microservices architecture with technologies like Node.js, Python, or Go to handle real-time updates and transactions.
Database: Consider using a scalable relational database like Amazon Aurora, Google Cloud SQL, or a NoSQL database like MongoDB for flexible data storage.

Justification:
- Load balancing ensures even (or as we want) distribution of traffic, improving scalability and reliability.
- Web and API servers provide separation of concerns, making it easier to manage and scale individual components.
- A database cluster offers high availability and can scale horizontally to accommodate a growing number of users and transactions.

Potential Bottlenecks and Strategies:
Database Scalability: Implement sharding or partitioning techniques to distribute the data across multiple database servers to handle a high volume of transactions.
Real-time Updates: Use a message queue system like Apache Kafka or RabbitMQ to handle real-time updates efficiently.
Data Security: Implement encryption at rest and in transit, role-based access control, and regular security audits to protect sensitive financial data.

Implementation and Deployment Steps:
1. Develop and test the web and API servers locally or on staging environments.
2. Set up the database cluster with replication and failover mechanisms.
3. Deploy web and API servers in an auto-scaling group to ensure reliability and scalability.
4. Configure the load balancer to distribute traffic across servers.
5. Implement monitoring and alerting with tools like Prometheus and Grafana for proactive issue detection.
6. Regularly test and review your architecture for optimizations, and ensure data backup and recovery processes are in place.
7. Deploy in a remote-first environment by utilizing CI/CD pipelines and containers for easy deployment and scaling.
8. Train the team on security best practices and monitor security logs regularly.

By following these steps and considering the mentioned technologies, you can design an architecture that is scalable, reliable, and secure for Auptimate's investment pool management feature.