Notification System Design

Practice On: 18 Jan 2025

Time Taken: 85 Mins (1 Hour 25 mins)

Qs for the Interviewer:

1. What kind of notification needs to be send? SMS, email, in-app notification, automated call?
2. What type of notification will be send? Promotional, Transactional, Status Update(Order)
3. Bulk notification supported? like for flash sale?
4. User can block / set custom setting for the notification?
5. Notification would be sent to user / system?
6. Rate-limiting is needed or not doe sending notification?
7. Future notification can be scheduled?
8. Can we have a template for an notification?
9. DO we need to stored the sent notification somewhere for audit?
10. Analytics is needed?
11. All notification will be at same level or will have some kind of priorities? tractional (p0) order status (p1) promotional (p2)

Requirements:

Functional Requirements:

1. User should be getting SMS, email, in-app notification.
2. System should able to sent SMS, email, in-app notification
3. Bulk notification supported

Non fictional requirements:

1. Notification system should be highly scalable
2. Notification system should have very low latency for sending the notification
3. Notification system should be reliable > client using system should get a guarantee that notification is send/received

Capacity Estimation

Assumptions
1. Avg size of an notification: 1 kb
2. Avg number of notification a user get's in a day(includes all notification): 10
3. Active users: 10 million

Notification sent in a day > 10 million * 10 = 100 million notification/day
Notification throughput => 100 million / 60 * 60 * 24 ~ 1200 notification/second

Storage = 100 million * 1 kb = (100*10^6) * (1*10^3) = 100*10^9 = 100 GB of storage needed to store notification of a day

Core System Entity

1. Notification message
2. User

High level system design

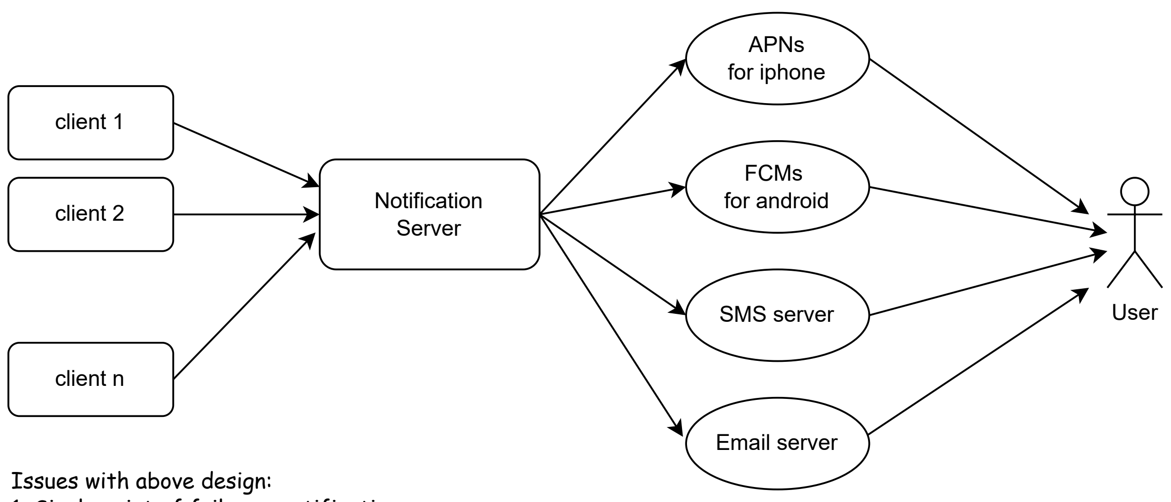
Store user's info as they sign-in for the very first time - User_info_database_table

As user sign in with mobile number / email we store that info so that when we need to send the notification to this user
we know all the info - User database. In same database we can also store user preference over the notifications.

How notification works in iphone, android, SMS and email

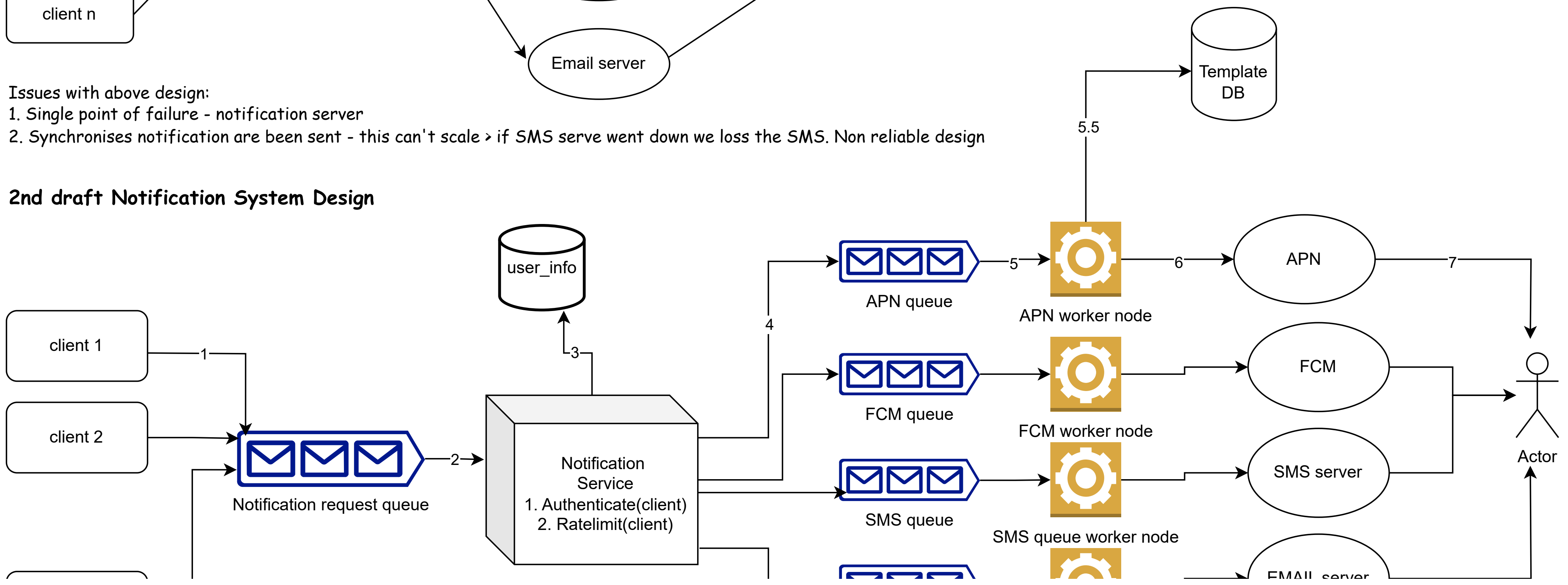
1. Iphone uses APNs
2. Android uses FCMs
3. SMS uses 3rd party message service provider like twilio
4. Email uses 3rd party email service provider like twilio/ own email server

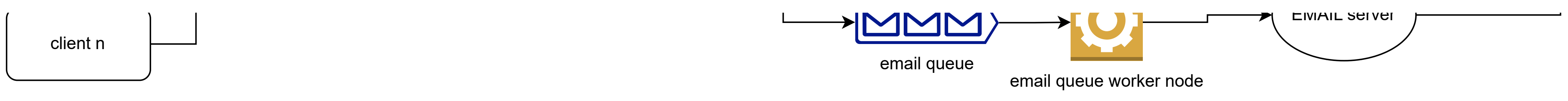
1st draft for notification system



Issues with above design:
1. Single point of failure - notification server
2. Synchronises notification are been sent - this can't scale > if SMS serve went down we loss the SMS. Non reliable design

2nd draft Notification System Design





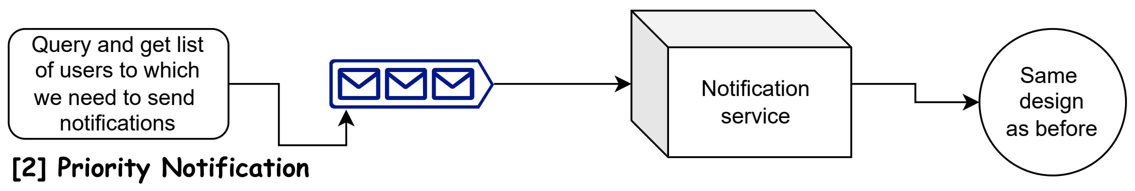
Steps
1. lets is client is payment service & customer makes and payment for an order
2. payment service is one of the client
3. payment service have the user's mobile number, email, and app info
3. payment service sends a message to notification service to send an notification to 8756XXXXXX and karanshergill@gamil.com that payment of 999.00 was successfully received.
4. We are using "Notification request queue" as to omit notification service as a SPF. Also in peak hours notification service won't get overwhelmed
5. Notification service listens to the queue and picks new message from the queue
6. It authenticates that Payment Service can sent SMS and email to user, checks the client is sending messages in limit
7. It checks the user DB to know user preference for the app notification and other notifications
8. Notification push in app notification is their respective queue , SMS notification their respective queue and email in their queue
9. From each queue respective worker picks the message and use the template to generate the final notification message.
10. User get separate notification from separate providers

Notification System Deep Dive

[1] Bulk Notification

Usually bulk notifications are not sent to single user but to a group / section of user like
Send reminder for flash sale, send promotional notification to group of people in one geo location

We first query people to whom we need to send the notification and have a bulk notification service


Query and get list of users to which we need to send notifications
Notification service
Same design as before

[2] Priority Notification

Transactional(p0) > Order track(p1) > Reminder(p1) > promotional notification(p2)
In Message Queue do can we set priority for messages ???? - YES, can be done in all the queues that are in our system

[3] Future notification schedule

A queue can keep messages for a week, we can schedule a message for future.
In this case then the before sending the notification the worker node need to check for the NOTIFICATION TIMING

[4] Audit & Analytics

Once notification are sent we can store that message for audit and analytics in our database

[5] Increase Scale

To increase SCALE we have keep multiple instances of the Notification Service
Keep template for similar notifications

[6] low latency

Keep template for similar notifications
Horizontal scaling of workers node

[7] Reliability

Queue guarantee to deliver the notification to user at least one time.

Design Review by ChatGPT

Strengths:

1. **Clear Requirements Understanding**:
   - Your questions to the interviewer demonstrate a strong grasp of clarifying functional and non-functional requirements before diving into the design.
   - Good coverage of aspects like bulk notifications, rate limiting, user preferences, and scheduling.
2. **Scalability Considerations**:
   - Decoupling the notification service with message queues ensures scalability and avoids overwhelming the service during peak traffic.
   - Horizontal scaling of worker nodes is mentioned, which is essential for handling increased loads.
3. **Reliability and Fault Tolerance**:
   - Use of queues ensures messages are not lost even if worker nodes or services fail.
   - Highlighting queue delivery guarantees adds robustness.
4. **Prioritization and Flexibility**:
   - Support for message priority and scheduling provides flexibility for different notification use cases.
   - Good separation of delivery mechanisms (APNs, FCM, SMS, email) via queues and worker nodes.
5. **Future-Proof Design**:
   - Support for templates, audit logging, and analytics showcases foresight for future features and system observability.

Suggestions for Improvement:

1. **High Availability of Notification Service**:
   - Although queues remove the single point of failure for notifications, the "Notification Service" itself can still become a bottleneck. Consider adding multiple instances of this service behind a load balancer to improve availability. ⭐
2. **Database Design**:
   - Expand on the "user_info" database. For instance:
     - Include schemas for user preferences (notification types and delivery channels).
     - Optimize querying for bulk notifications (e.g., geolocation indexing for regional campaigns).
3. **Handling Failures in Downstream Services**:

- Add retry mechanisms and dead-letter queues to handle failures when sending notifications via APNs, FCM, SMS, or email servers. ⭐
- Monitor downstream services and have fallbacks, e.g., switching between multiple SMS/email providers. ⭐

4. **Latency Considerations**:
    - While horizontal scaling is mentioned, highlight specific low-latency strategies like using in-memory data stores (e.g., Redis) for frequently accessed data (e.g., user preferences). ⭐
5. **Rate Limiting Granularity**:
    - Clarify how rate limiting works (e.g., per client, per user, or per notification type).
    - Consider token bucket or leaky bucket algorithms to ensure fairness across clients.
6. **Detailed Bulk Notification Flow**:
    - For bulk notifications, describe how to segment users efficiently. For example, pre-compute user segments and store them in a separate table for quick lookups.
7. **System Monitoring and Metrics**:
    - Add a monitoring layer to track metrics like notification delivery success rates, queue sizes, worker latencies, and system errors. ⭐
    - Use tools like Prometheus, Grafana, or AWS CloudWatch for this purpose.
8. **Compliance and Security**:
    - Ensure secure storage of sensitive user data (e.g., encrypt email and phone numbers).
    - Mention compliance with data protection regulations (e.g., GDPR, CCPA) for audit and analytics features.