Design a Notification System

A notification is more than just mobile push notification. Three types of notification formats
are: mobile push notification, SMS message, and Email.

Functional Requirement

1. Type of Notifications: Push notification, SMS Message, Email
2. System should be soft real time (Slight delay can be accepted)
3. Supported Devices: iOS, Android, laptop/Desktop
4. User can opt-out & no longer can receive the notification

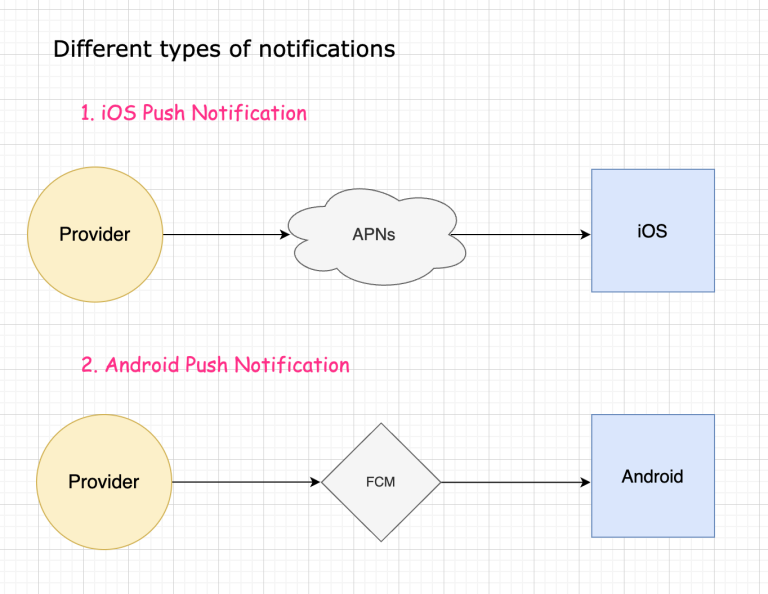
Non-Functional Requirements

1. 10 Million mobile push notifications
2. 1 Million SMS Messages
3. 5 Million Emails

High Level Design

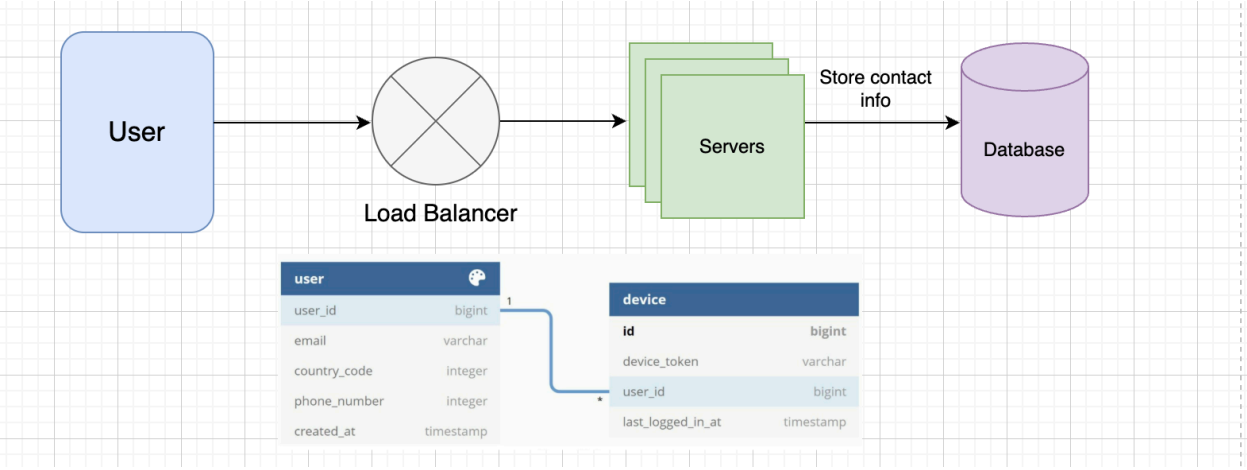
Different types of notifications

1. Android push notification
2. iOS push notification
3. SMS push notification
4. Message push notification



Contact info gathering flow

To send notifications, we need to gather mobile device tokens, phone numbers, or email
addresses. When a user installs our app or signs up for the first time,
API servers collect user contact info and store it in the database.

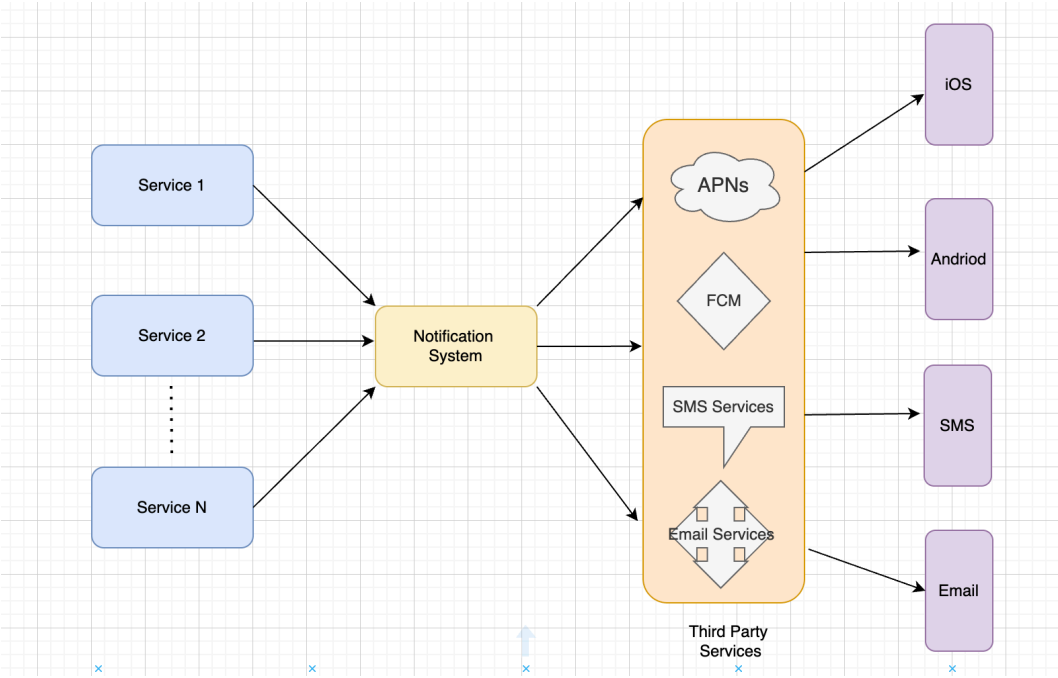


Notification sending/receiving flow

- **Service**: A service (micro-service, cron job, or distributed system) triggers notifications. For example, billing services send payment reminders via email,

while shopping sites send delivery updates through SMS.

- **Notification system:** The notification system is the centerpiece of sending/receiving
  notifications.
- **Third-party services:** Third party services are responsible for delivering notifications to
  users.



Three problems are identified in this design:

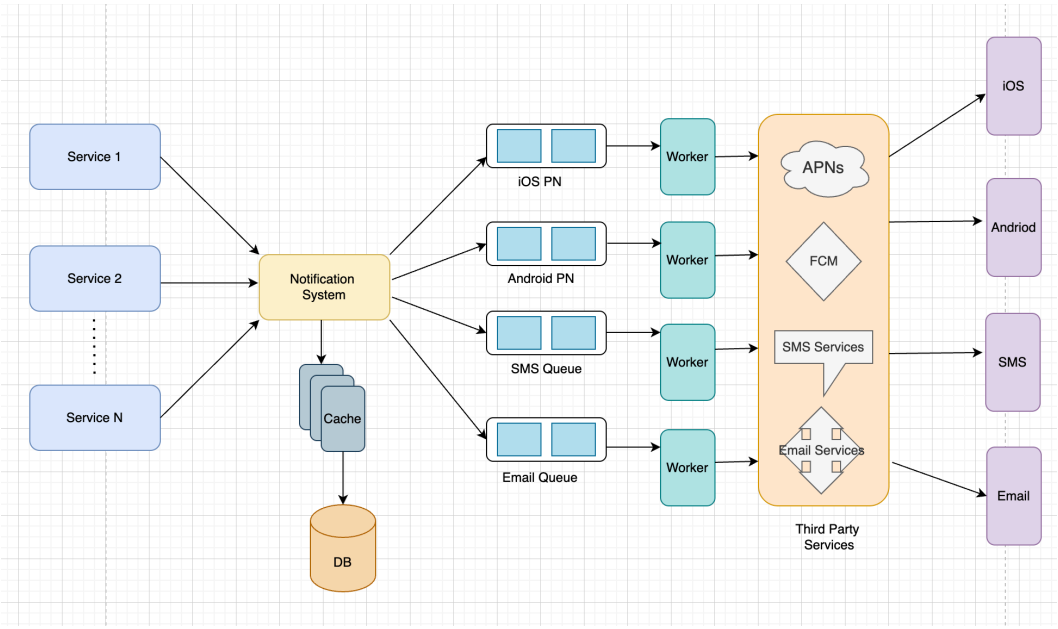
1. SPOF : single point of failure. I.e if Notification system goes down, whole system will colapse.
2. Hard to Scale: Since notification system contains all the things inside it, like Cache & Database, so it becomes difficult to scale independently.
3. Performance bottleneck: Processing and sending notifications can be resource intensive.

For example, constructing HTML pages and waiting for responses from third party
services could take time. Handling everything in one system can result in the system
overload, especially during peak hours.

Overcoming above problem

In order to scale this system, we need to move out individual componenets.

1. Take cache & DB out of notification system.
2. This will help us to scale each component individually and use automatic horizontal scaling.

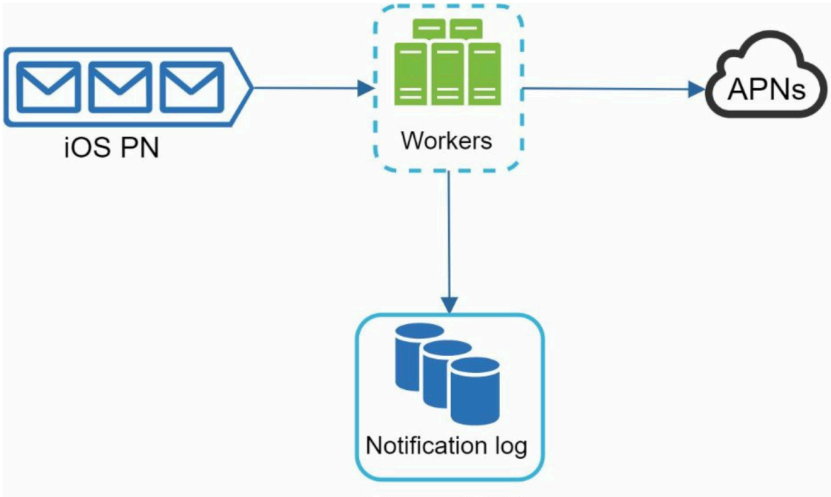


1. Service is the responsible for triggering the notification. Once it get triggered, it is transfered to Notification system server.
2. Notification system server takes all the info. from the cache/DB regarding the user phone no/email etc. & carry out some validation and put the notification into message queue.

3. Message queues: They remove dependencies between components. Message queues serve as buffers when high volumes of notifications are to be sent out.
4. Workers are a list of servers that pull notification events from message queues and
send them to the corresponding third-party services.
5. Third-party services: Third party services are responsible for delivering notifications to
users.

How to prevent data loss?

One of the most important requirements in a notification system is that it cannot lose data.
Notifications can usually be delayed or re-ordered, but never lost. To satisfy this requirement,
the notification system persists notification data in a database and implements a retry
mechanism.



Additional components and considerations

1. Notification template

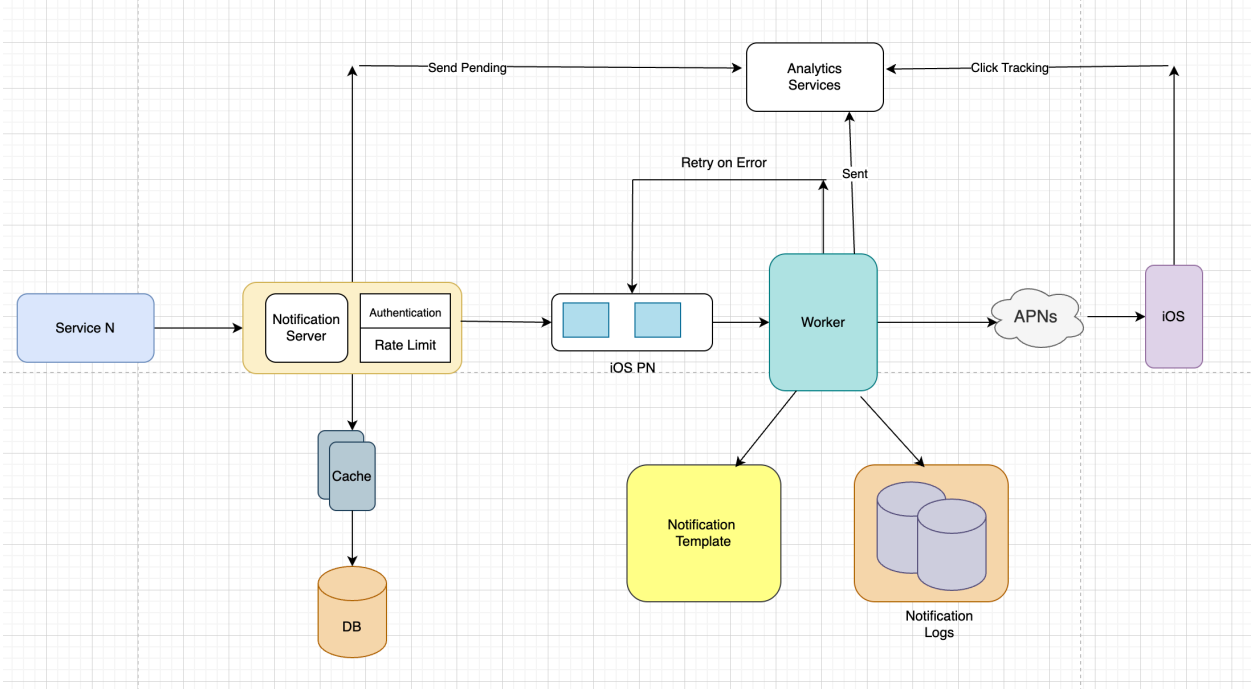
- A large notification system sends out millions of notifications per day, and many of these
  notifications follow a similar format. Notification templates are introduced to avoid building every notification from scratch. This template is mainly kept in the DB/Cache.

2. Notification setting

   - Users generally receive way too many notifications daily and they can easily feel
     overwhelmed. Thus, many websites and apps give users fine-grained control over notification settings.

3. Rate limiting

   - To avoid overwhelming users with too many notifications, we can limit the number of
     notifications a user can receive. This is important because receivers could turn off
     notifications completely if we send too often.



In this design, many new components are added in comparison with the previous design.

• The notification servers are equipped with two more critical features: authentication and
rate-limiting.
• We also add a retry mechanism to handle notification failures. If the system fails to send
notifications, they are put back in the messaging queue and the workers will retry for a
predefined number of times.
• Furthermore, notification templates provide a consistent and efficient notification creation process.
• Finally, monitoring and tracking systems are added for system health checks and future
improvements.