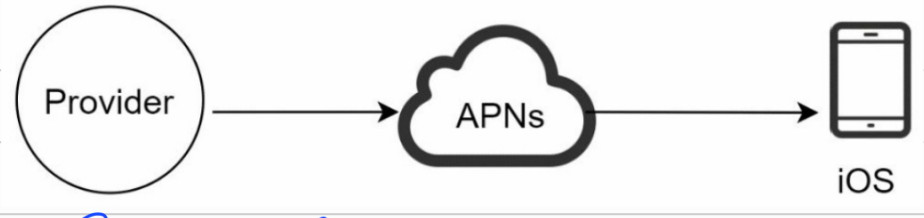
Design a Notification System

- Step 1 - Understand the problem and establish design scope.
  - Should support push notification, sms message and email.
  - It is a soft real-time system. User should receive notifications as soon as possible.
  - Support iOS | android | laptop | desktop.
  - Notification can be triggered by client applications and also can be scheduled on server side.
  - Users can choose to opt out of notifications.
  - 10 million mobile push notification, 1 million SMS, 5 million emails.

- Step 2 - Propose high-level design and get buy-in
  - High Level Design is structured as follows:
    - Different types of notifications.
    - Contact info gathering flow.
    - Notification sending / receiving flow.

- Different Types of Notifications
  - iOS push notification



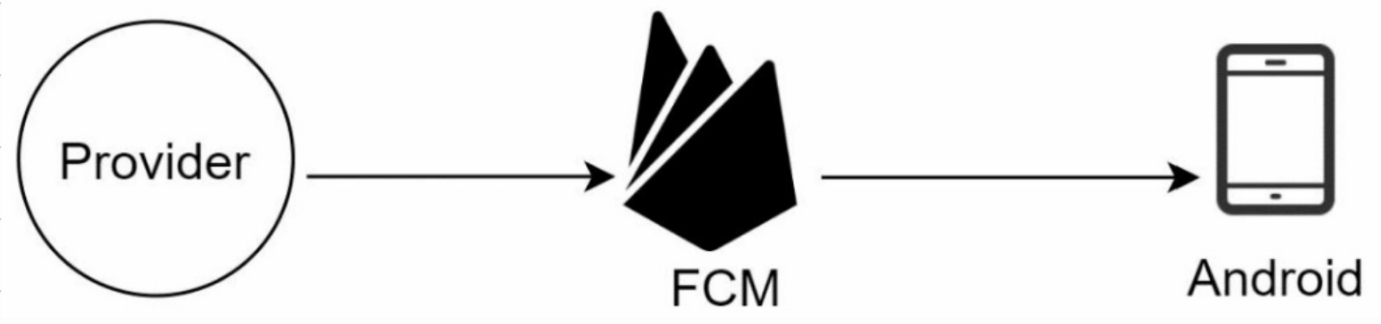
  - Provider: A provider builds and sends notification requests to Apple Push Notification Service (APNS). To construct a push notification, the provider provides the following data:

- Device Token : This is a unique identifier used for sending push notifications.
- Payload : This is a JSON dictionary that contains a notification's payload.

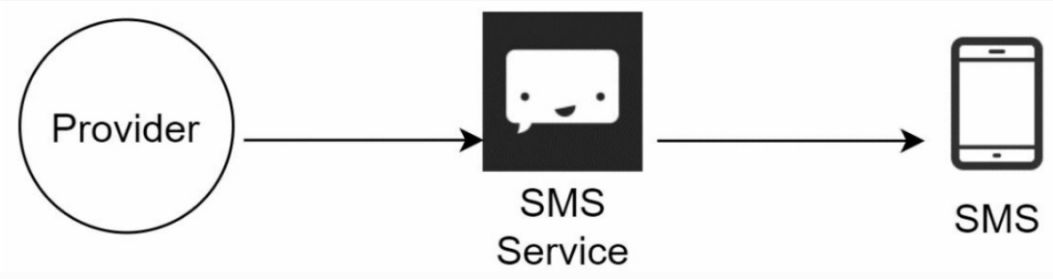
```
{
  "aps" : {
            "alert" : {
               "title" : "Game Request",
               "body" : "Bob wants to play chess",
               "action-loc-key" : "PLAY"
            },
            "badge" : 5
         }
}
```

- APNS : This is a remote service provided by Apple to propagate push notifications to iOS devices.
- iOS Device : It is the end client, which receives push notification.

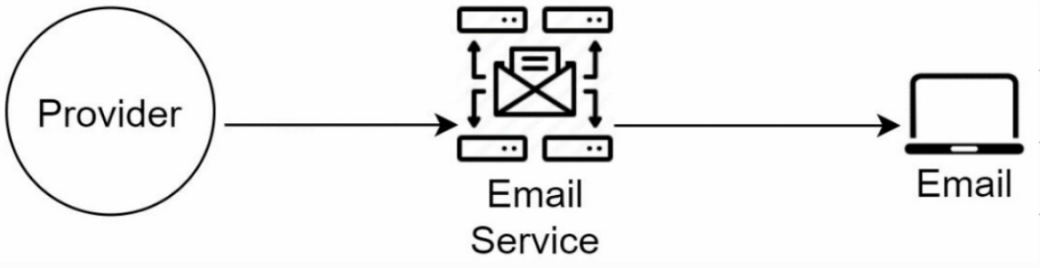
- Android Push Notification
- Android has the similar flow. Instead of using APNs, Firebase Cloud Messaging (FCM) is commonly used.

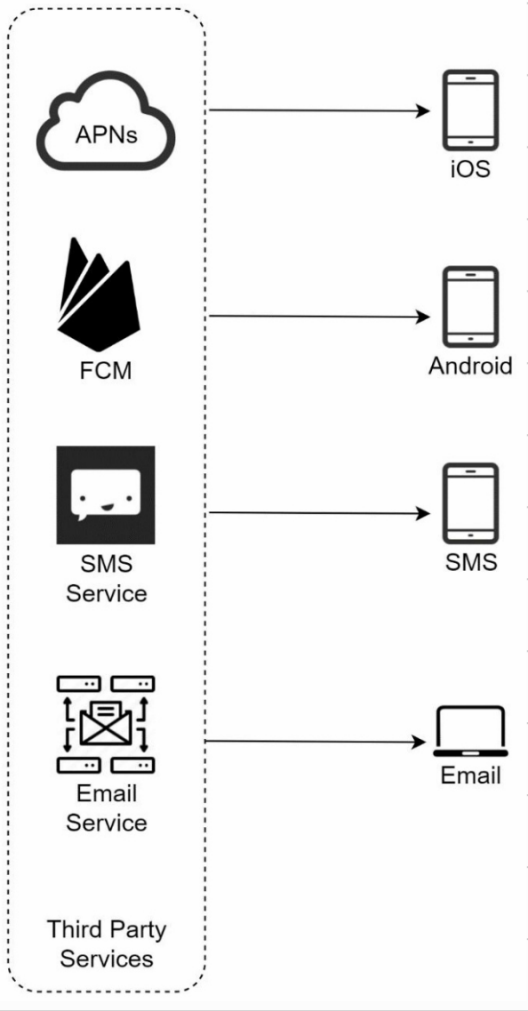


- SMS Message
  - For SMS messages, third party SMS services like Twilio, Nexmo and many others.

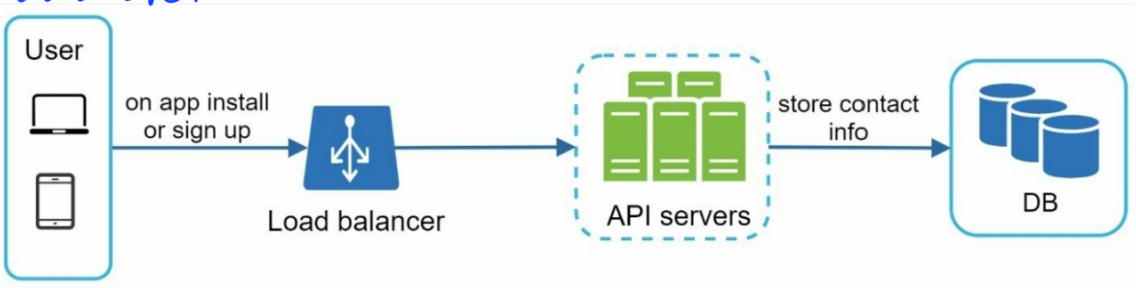


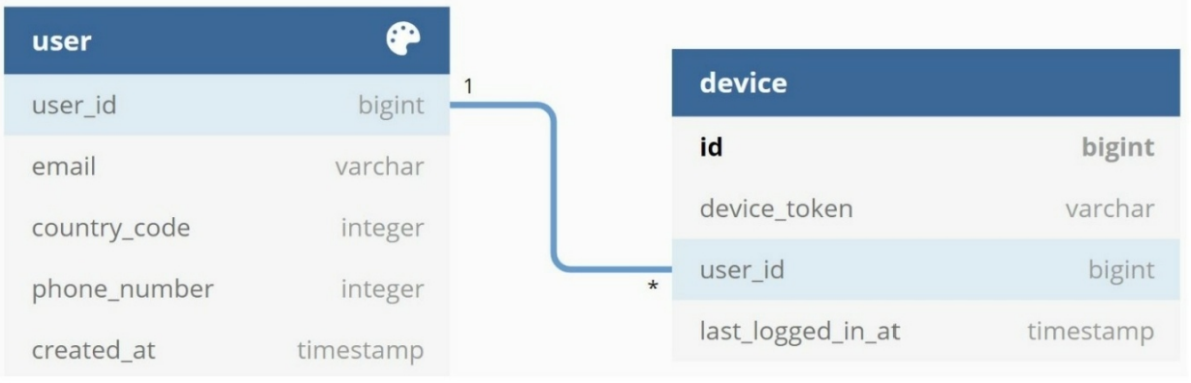
- Email
  - Although companies can set up their own email servers, many of them opt for commercial email services.



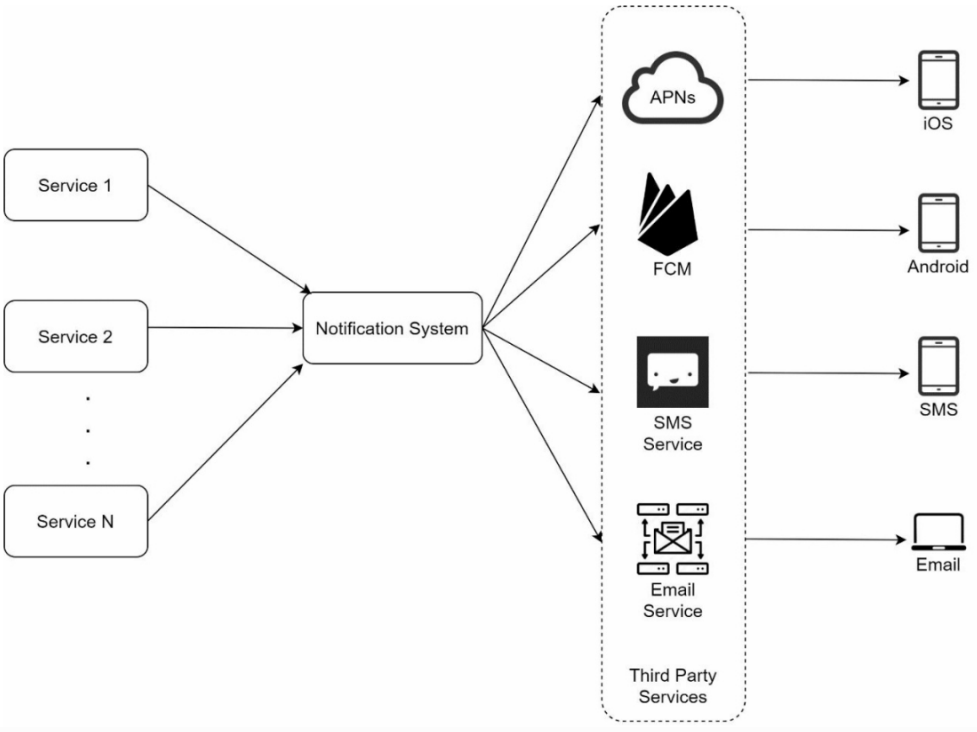


- Contact Info Gathering Flow
  - To send notifications, we need to gather mobile device tokens, phone numbers, or email addresses.
  - When a user installs our app or signs up for the first time, API servers collect user contact info and store it in the database.



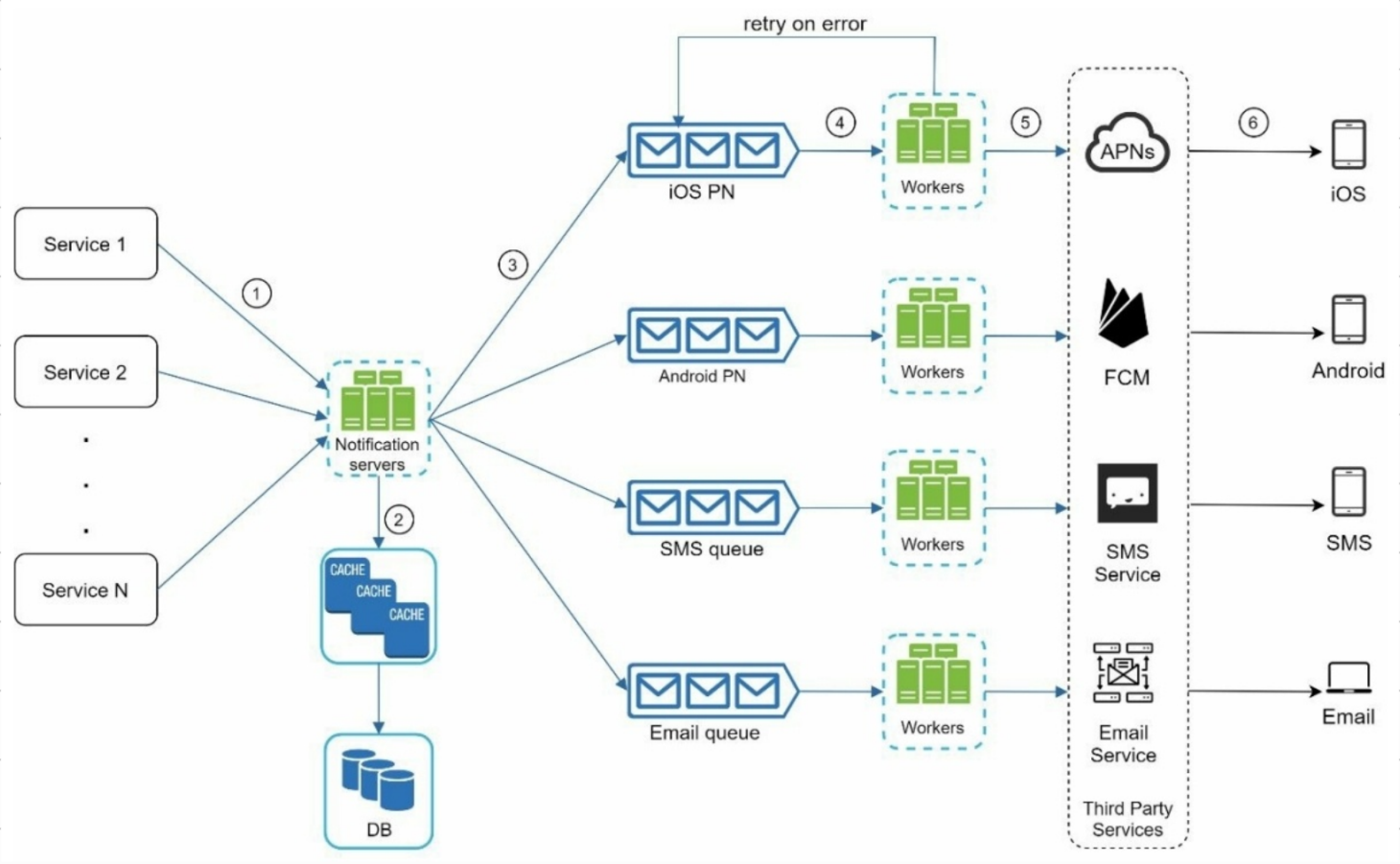


- Notification sending / receiving flow
  - High Level Design



- Service 1 to N: A service can be a micro-service, a cron job, or a distributed system that triggers notification sending events.
- Notification System: The notification system is the centerpiece of sending / receiving notifications. Starting with something simple, only 1 notification server is used. It provides APIs for services 1 to N and builds notification payloads for 3rd party services.
- Third Party Services: Third party services are responsible for delivering notification to users. While integrating with third party services, we need to pay extra attention to extensibility. Good extensibility means a flexible system that can easily plugging or unplugging of a third party service might be unavaivalable in new markets or in the future.
- iOS, Android, SMS, Email: Users receive notifications on their device.
- Three problems with the design:
  - SPOF: A single notification server means SPOF.
  - Hard to Scale: The notification system handles everything related to push notifications in one server. It is challenging to scale databases, caches and different notification processing component independently.
  - Performance Bottleneck: Processing and sending notifications can be resource intensive.

- High-Level Design (Improved)
- Design can be improved as follows:
  - Move the database and cache out of the notification server.
  - Add more notification servers and setup automatic horizontal scaling.
  - Introduce message queues to decouple the system components.



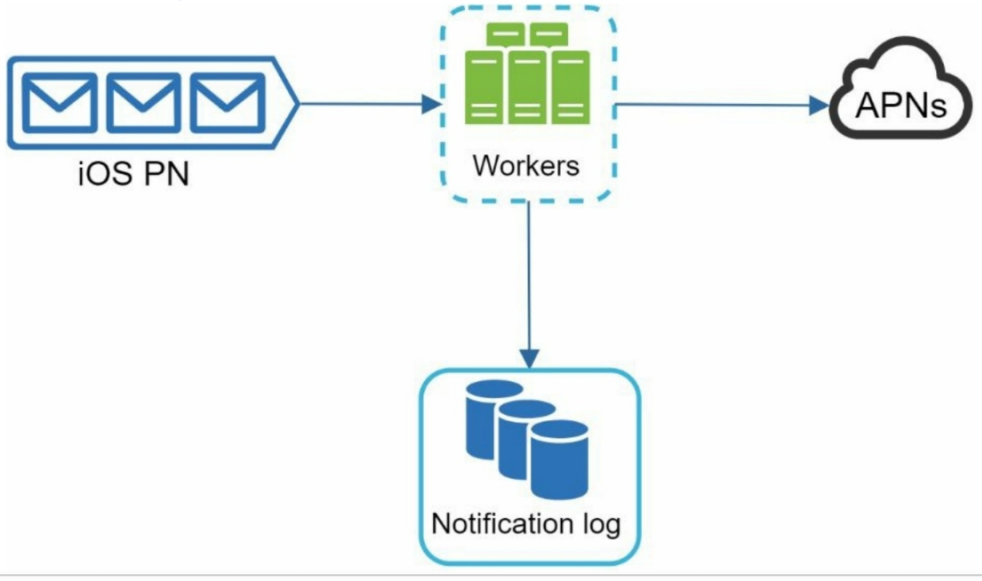
- Service 1 to N: They represent different services that send notifications via APIs provided by notification servers.
- Notification Servers: They provide following functionalities:
  - Provide APIs for services to send notifications. Those APIs are only accessible internally or by verified clients to prevent spams.
  - Carry out basic validations to verify emails, numbers etc.
  - Query the database or cache to fetch data needed to render a notification.
  - Put notification data to message queues for parallel processing.

- Cache: User info, device info, notification templates are cached.
- DB: It stores data about user, notification, setting etc.
- Message Queues: They remove dependencies between components. Message queues serve as buffers when high volumes of notifications are to be sent out. Each notification type is assigned with a distinct message queue so an outage in one third-party service will not affect other notification types.
- Workers: Workers are a list of servers that pull notification events from message queues and send them to the corresponding third-party services.

- Components coorks as follows:
  1. A service calls APIs provided by notification servers to send notifications.
  2. Notification servers fetch metadata such as user info, device token and notification setting from the cache or database.
  3. A notification event is sent to the corresponding queue for processing.
  4. Workers pull notification events from message queue.
  5. Workers send notifications to third party services.
  6. Third party services send notifications to user devices.

- Step 3 - Design Deep Dive
  - Following will be explored in deep dive:
    - Reliability
    - Additional components and considerations
    - Updated design

- Reliability
  - 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.



    - Will recipients receive a notification exactly once?
      - No, due to the distributed nature of the system, a notification could be sent more than once.
      - To reduce duplication occurance, a dedupe mechanism can handle failure cases.
      - When a notification event arrives, we check if it is seen before by checking event ID. If it is seen before, it is discarded otherwise notification is sent.

- Additional Components:
  - Notification Template
    - A large notification system sends out millions of notifications per day, and many of these notifications follow similar format.
    - Notification templates are introduced to avoid building every notification from scratch and maintaining a consistent format.

  - Notification Setting
    - Users generally receive way too many notifications daily. Thus many websites and apps give users fine-grained control over notification settings.
    - Before any notification is sent to user, we first check if a user is opted-in to receive this type of notifications.

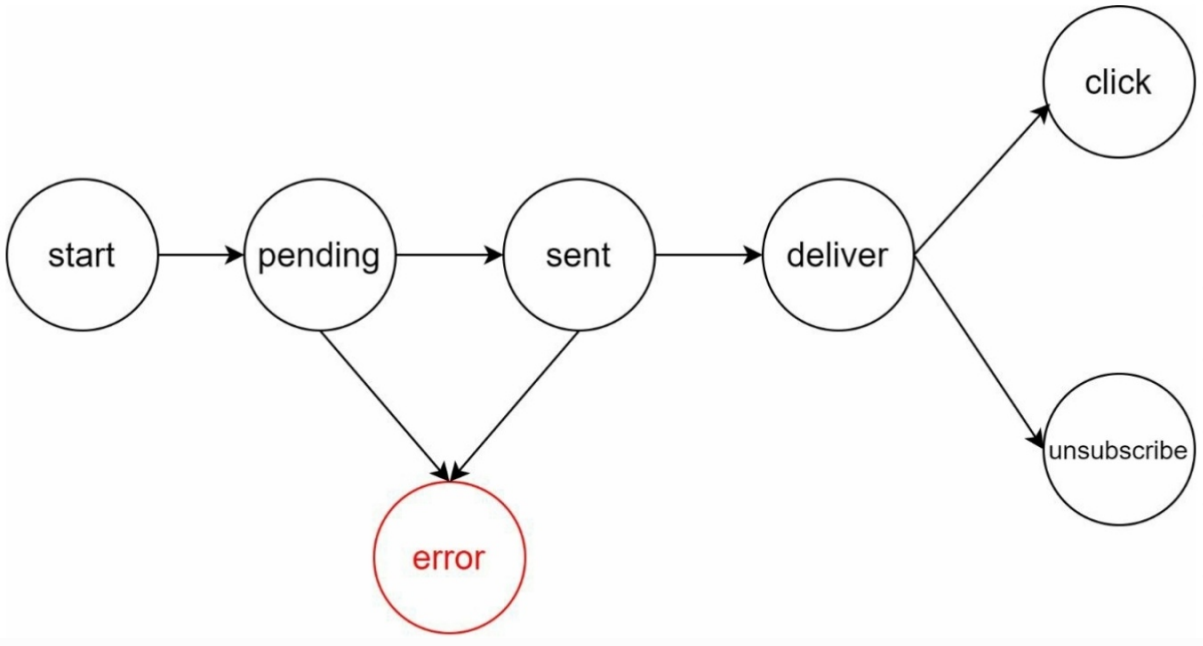
  - Rate Limiting
    - To avoid overwhelming users with too many notifications, we can limit the number of notifications a user can receive.
    - This is important because receivers could turn off notifications completely if we send too often.

  - Retry Mechanism
    - When a third-party service fails to send a notification, the notification will be added to the message queue for retrying.

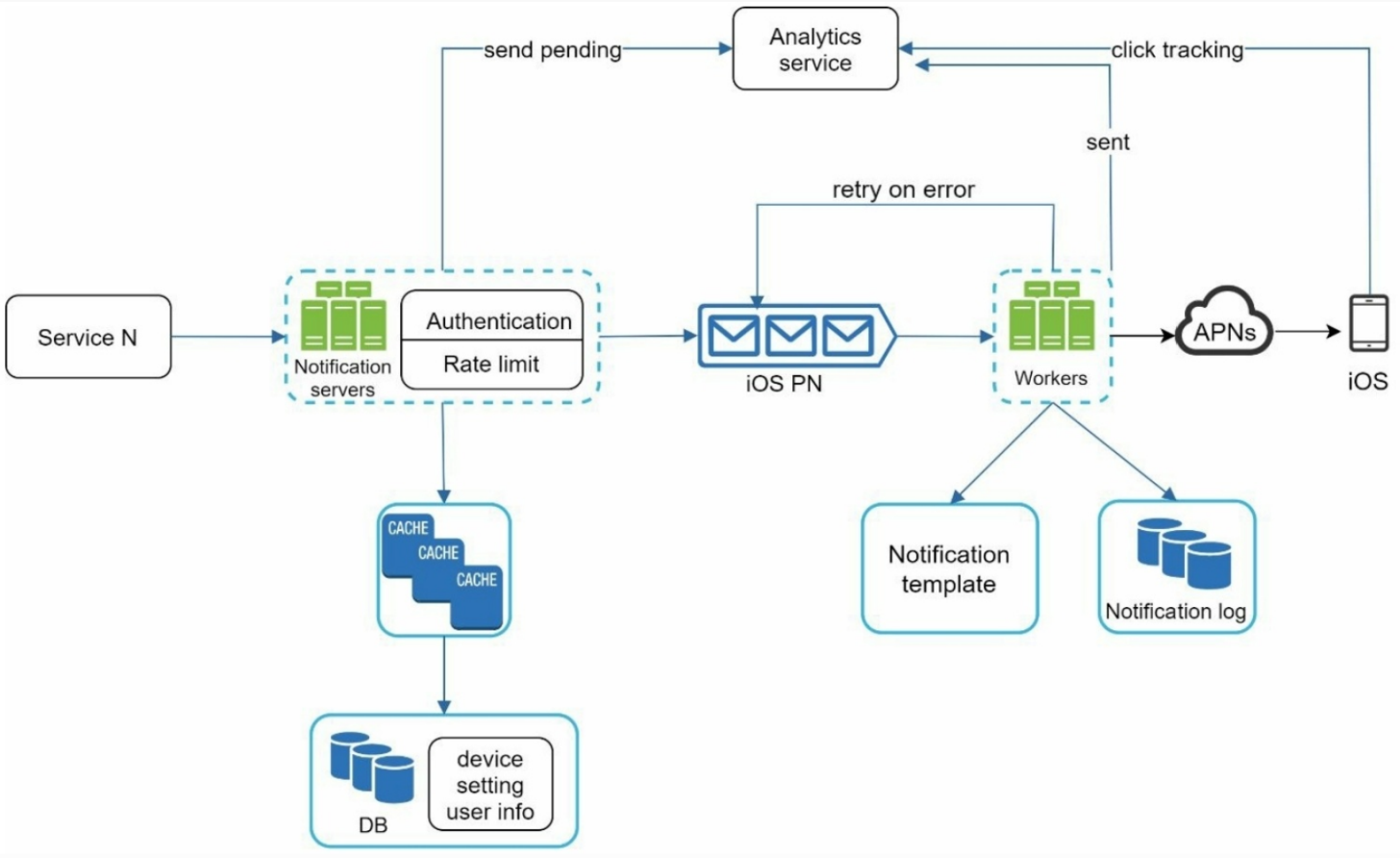
  - Security in Push Notifications
    - For iOS or Android apps, appKey and appSecret are used to secure push notification APIs.

- Monitor Queued Notifications
  - A key metric to monitor is the total number of queued notifications. If the number is large, the notification events are not processed fast enough by workers.
  - To avoid delay in the notification delivery, more workers are needed.

- Events Tracking
  - Notification metrics, such as open rate, click rate and engagement are important in understand customer behaviour.
  - Analytics service implements events tracking. Integration between the notification system and the analytics service is usually required.



- Updated Design



- New Components added:
  - The notification servers are equipped with 2 more critical features: Authentication and Rate-Limiting
  - We also add a retry mechanism to handle notification failure.
  - 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.

- Step 4 - Wrap Up
  - Reability: We proposed a robust retry mechanism to minimize the failure rate.
  - Security: AppKey / appSecret pair is used to ensure only verified clients can send notifications.
  - Tracking and Monitoring: These are implemented in any stage of a notification flow to capture important stats.
  - Respect user settings: Users may opt-out of receiving notifications. Our system checks user settings first before sending notifications.
  - Rate Limiting: Users will appreciate a frequency capping on the number of notifications they receive.