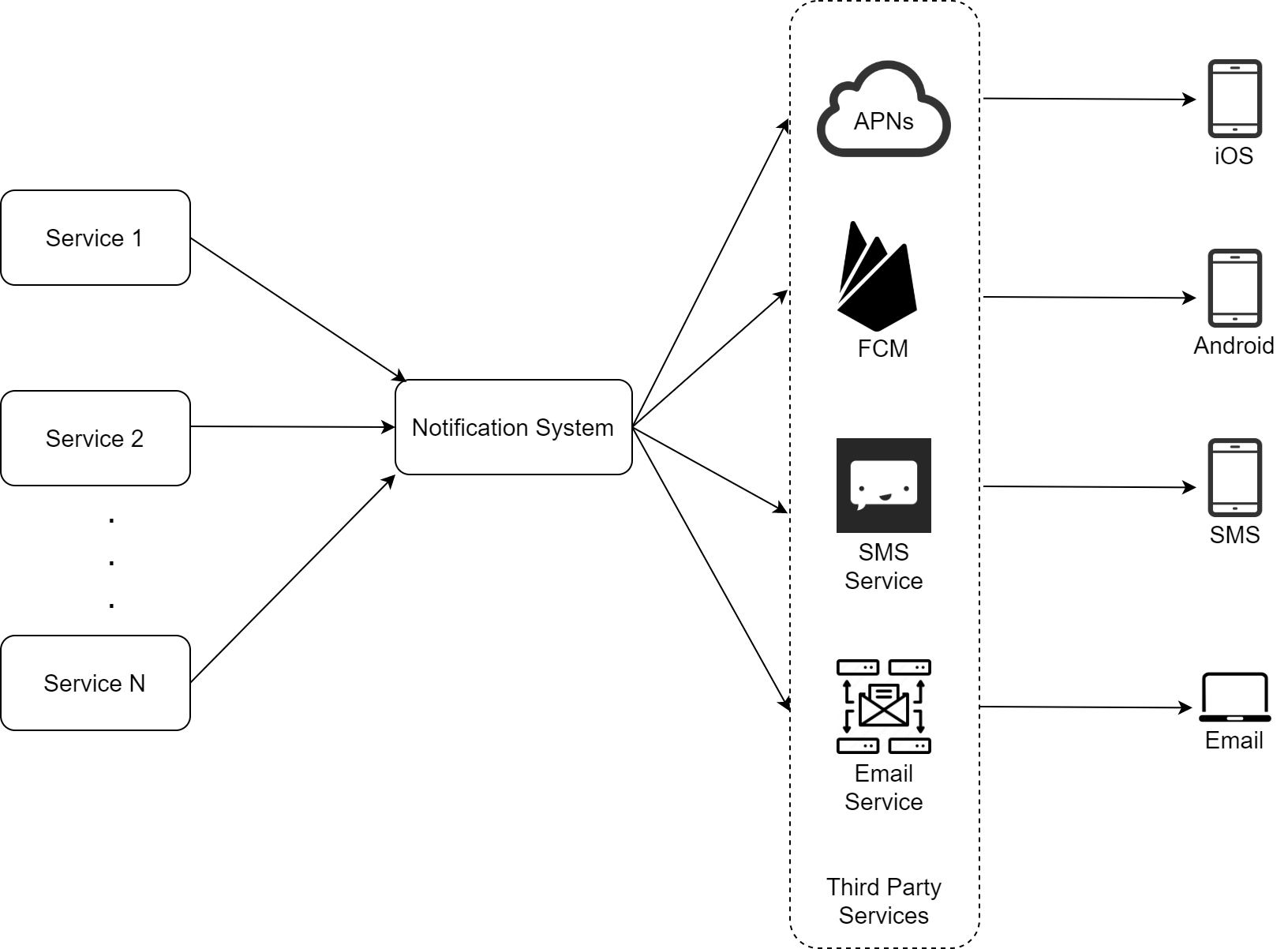


**Table needed basic data here:**

A screenshot of a computer

Description automatically generated

**Initial Design here:**



**What are different services?**

1. Biling Service: microservice:

 For example, a billing service sends emails to remind customers of their due payment or a shopping website tells customers that their packages will be delivered tomorrow via SMS messages.

1. Third party service here: also important

**What are the biggest problems in this design?**

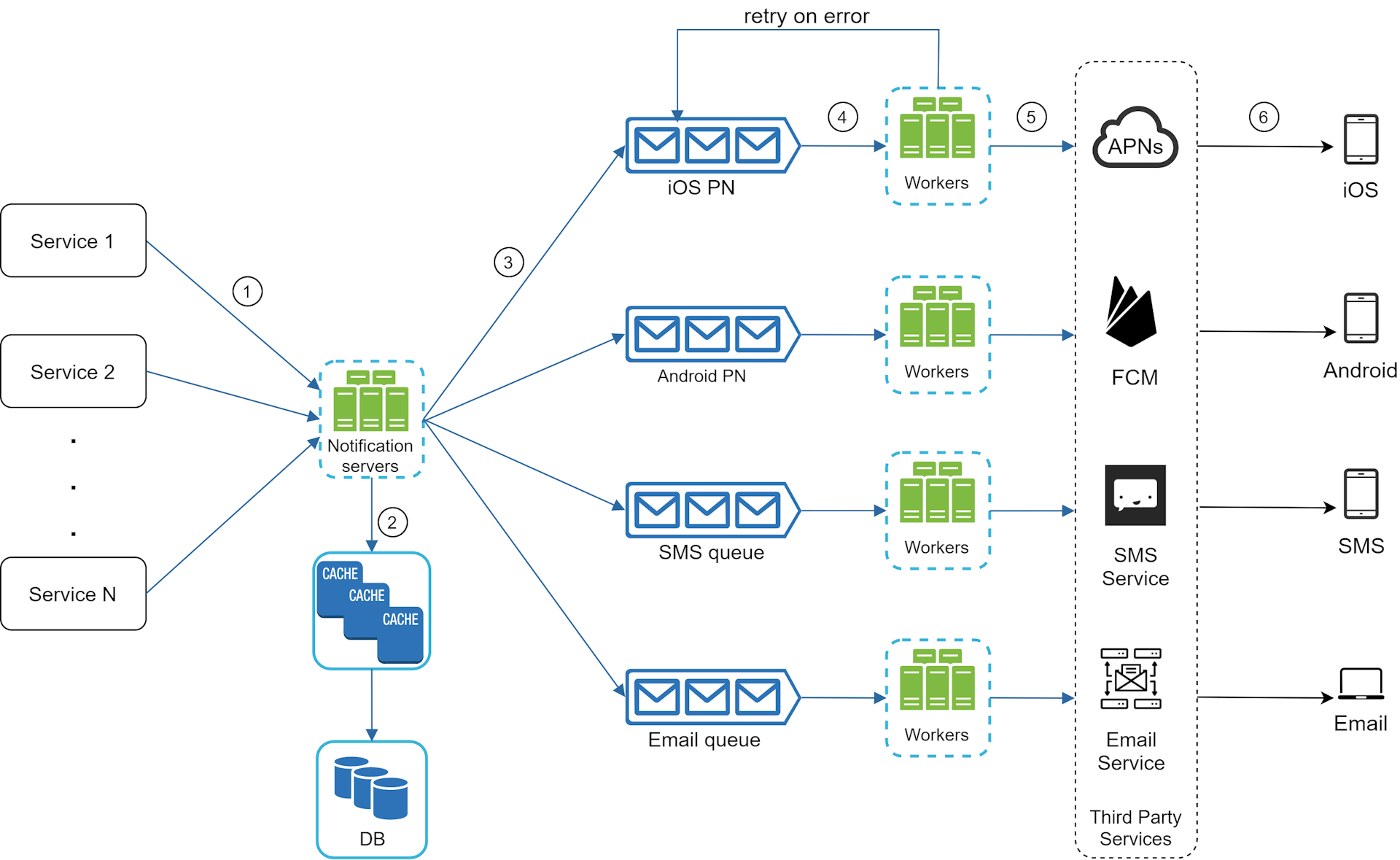
**Three problems are identified in this design:**

* Single point of failure (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 components independently.
* 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.

**How do we improve it?**

After enumerating challenges in the initial design, we improve the design as listed below:

* Move the database and cache out of the notification server.
* Add more notification servers and set up automatic horizontal scaling.
* Introduce message queues to decouple the system components.

**Improved design** 

**What’s job of the notification server?**

1. Expose api for sending notifications
2. Carry out basic notification to verify emails,
3. Query the database or cache to fetch data needed to render a notification.
4. Put notification data to message queues for parallel processing. ( this is very important)

What does typical api post request look like?

POST <https://api.example.com/v/sms/send>

**Request body**

{

"to":[

{

"user\_id":123456

}

],

"from":{

"email":"from\_address@example.com"

},

"subject":"Hello World!",

"content":[

{

"type":"text/plain",

"value":"Hello, World!"

}

]

}

**What’s responsbility of the queue?**

Used as buffer with high volume of notificatoins here.

Each notification type is assigned with a distinct message queue so an outage in one third-party service will not affect other notification types.

**What are workers?**They are the consumers nodes listening for events from the rabbit mq here

Workers are a list of servers that pull notification events from message queues and send them to the corresponding third-party services.

**Can you please walk me through the flow here?**

Next, let us examine how every component works together to send a notification:

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. For instance, an iOS push notification event is sent to the iOS PN queue.

4. Workers pull notification events from message queues.

5. Workers send notifications to third party services.

6. Third-party services send notifications to user devices.

**Part 3: Deep dive**

* Reliability.
* Additional component and considerations: notification template, notification settings, rate limiting, retry mechanism, security in push notifications, monitor queued notifications and event tracking.
* Updated design**.**

And then here

**How do we prevent data loss here?**

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. The notification log database is included for data persistence, as shown in Figure 11.

**How to prevent data from being seen twice here?**

When a notification event first arrives, we check if it is seen before by checking the event ID. If it is seen before, it is discarded. Otherwise, we will send out the notification. For interested readers to explore why we cannot have exactly once delivery, refer to the reference material [5].