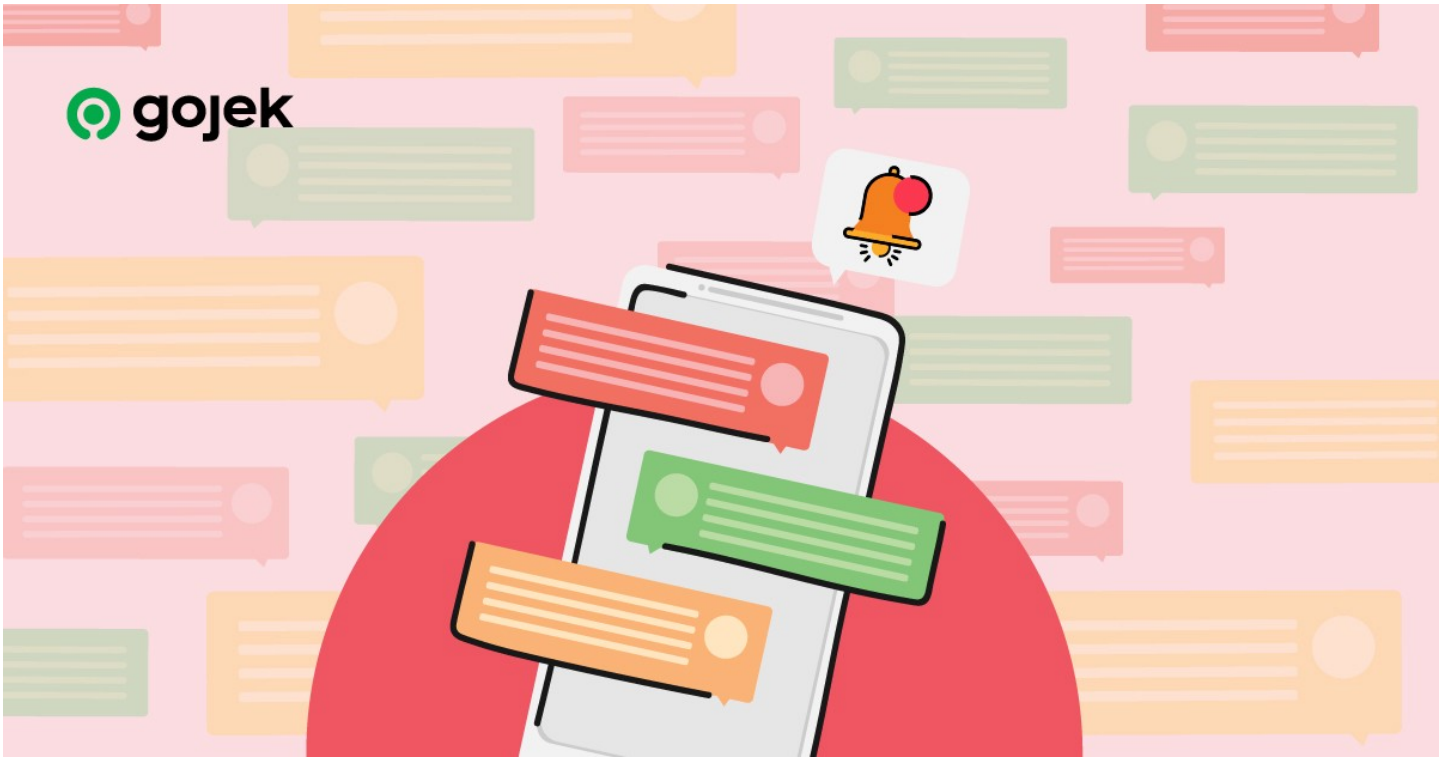
How We Manage a Million Push Notifications an Hour

3 million+ orders a day across 20+ products on multiple devices, operating systems, and services. That's a lot of notifications. 😅

Soham Kamani
Aug 29, 2019 · 6 min read



gojek.jobs

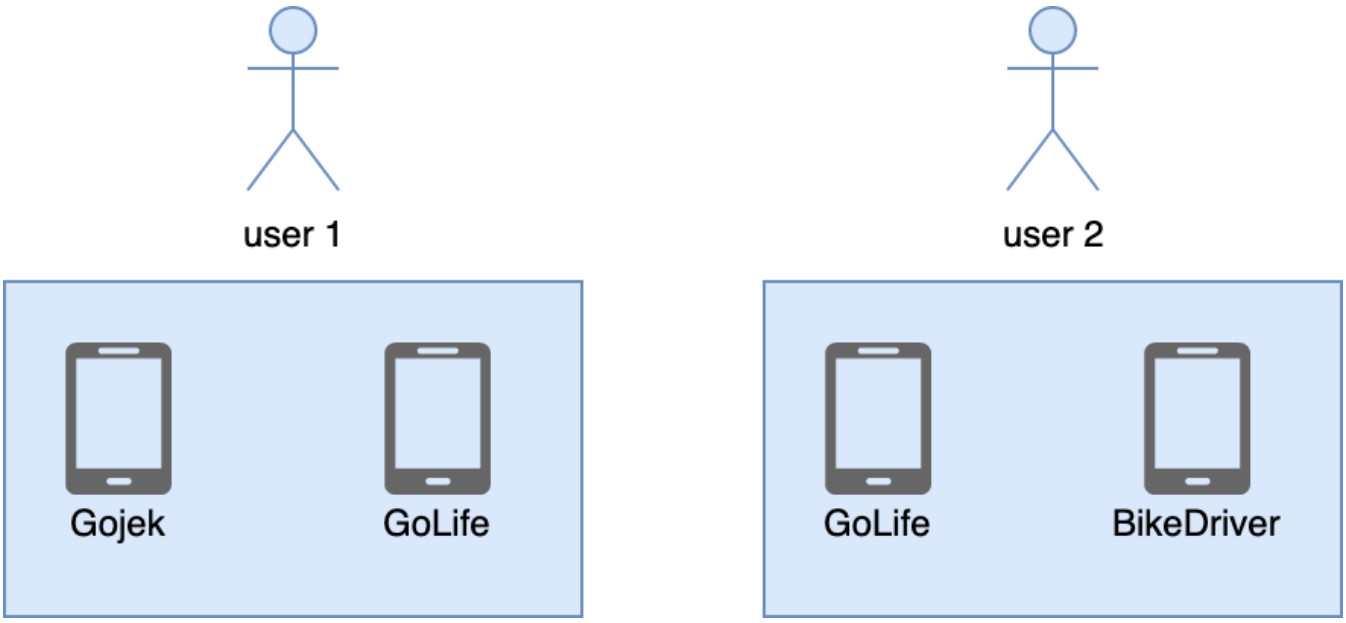
Push notifications are an essential tool to inform users about events that require their immediate attention. At Gojek, we handle more than 3 million orders a day across our 20+ products.

As you can imagine, the number of push notifications we send are proportionally large — about a million every hour to be exact.This post describes the challenges we faced while working with such high notification volumes, and the solutions we developed to solve them.

Volume is only one part of the problem, many additional challenges were somewhat unique to Gojek:

1. Multiple applications

Gojek isn't comprised of just a single app. In addition to the Gojek app for customers, we also have GoLife, as well as our driver apps, merchant apps, and service provider apps for GoLife.



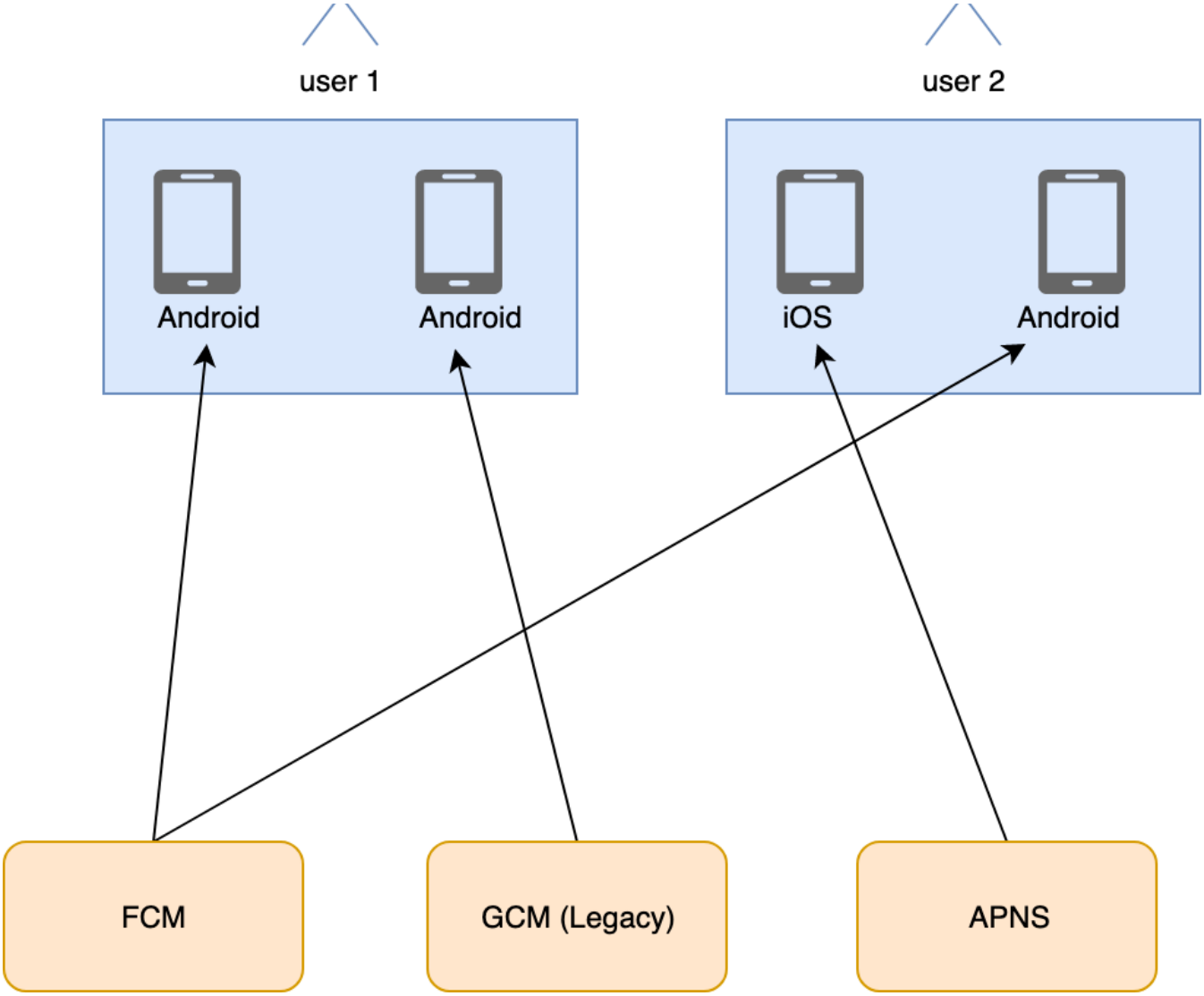
When one of our systems wants to send a notification, it could be for a specific application of a user (for example, we don't want to send GoLife booking notifications to the Gojek app), or for all applications (for example, a promotional notification).

Our system needed to be flexible enough to choose between an option to broadcast the notification to all applications, or to a specific application.

2. Multiple notification providers

Since we support both iOS and Android for our customer applications, we needed to support multiple notification systems.

For Android devices, we have FCM (Firebase Cloud Messaging) and the deprecated GCM (Google Cloud Messaging) API. For iOS, we have APNS (Apple Push Notification service)



Each notification provider has different API keys and tokens for different application IDs. For example, the FCM API key would be different for GoLife as opposed to Gojek.

3. Multiple devices per user

We allow our users to stay signed in to multiple devices at a time. For us, this also means that any push notification sent to the user must be sent to all of the devices that the user is currently logged into.

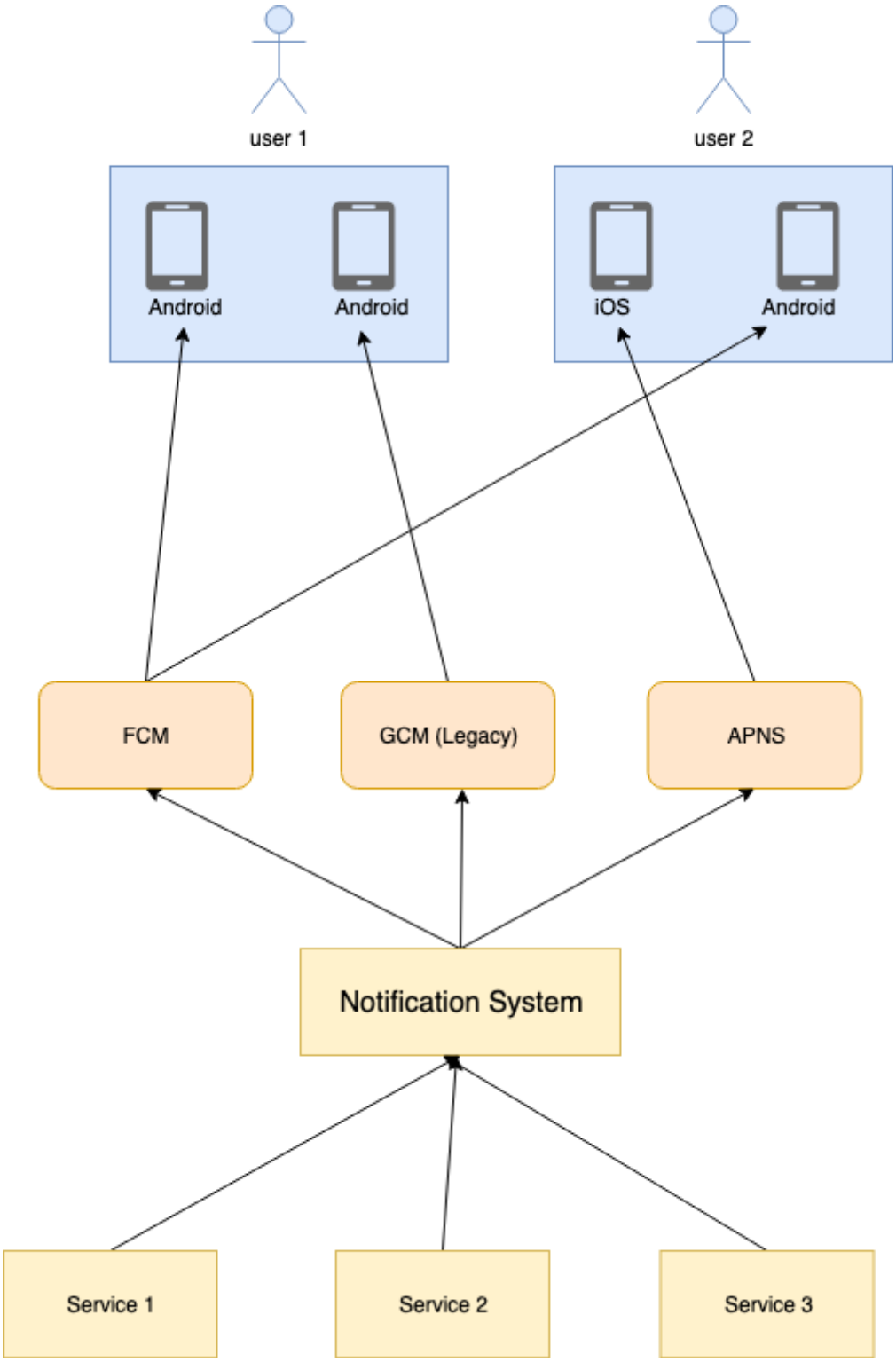
This is also a combination of the previous two problems:

1. The user can be logged into multiple applications on a single device (Gojek + GoLife)
2. The user may be logged into different devices, where each device requires a different push notification provider. For example, a user can be logged into Gojek on their Android, as well as iOS device.

4. Multiple services that want to send notifications

Gojek uses a micro-service architecture, where the services of each of our products need to send notifications to our users.

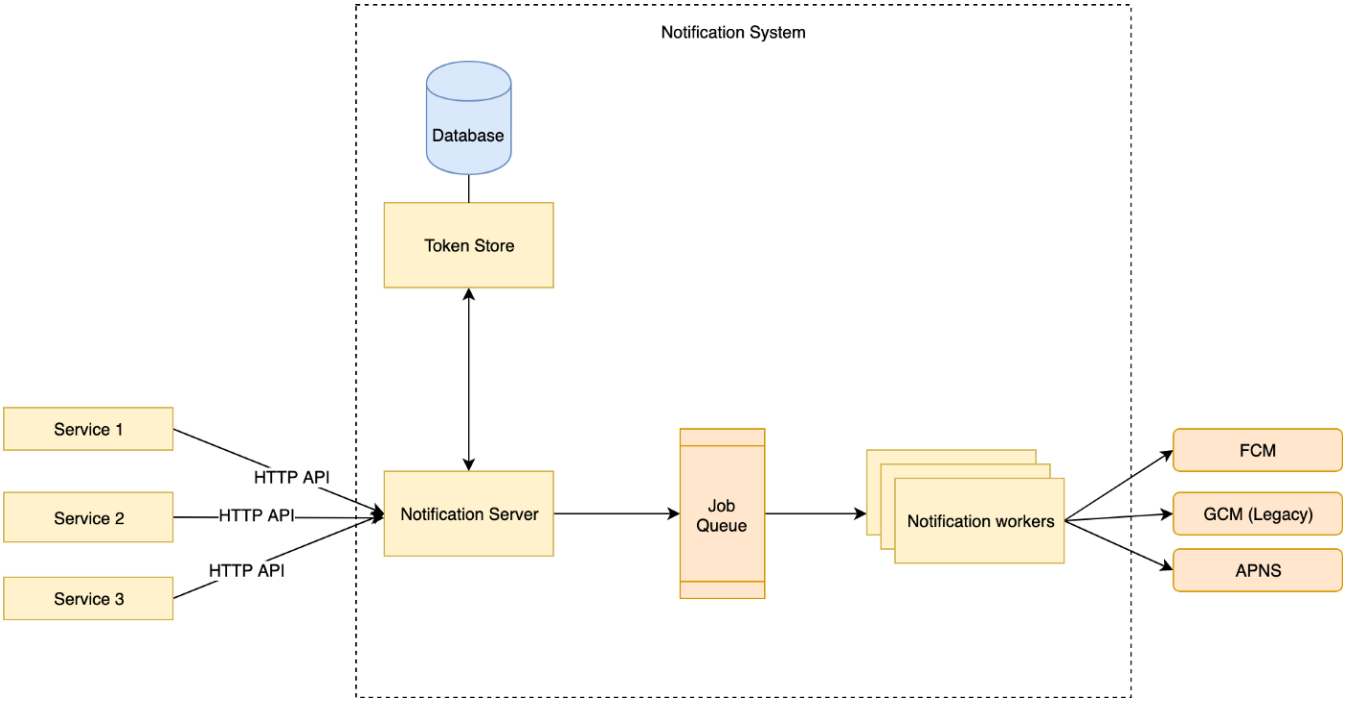
We want to make it as easy as possible for any service to be able to send a notification, without worrying about the multiple devices and providers that we discussed.



Push notification service architecture

In order to address the above challenges, and make sure that the API remains as simple as possible, we built our notification system as three components:

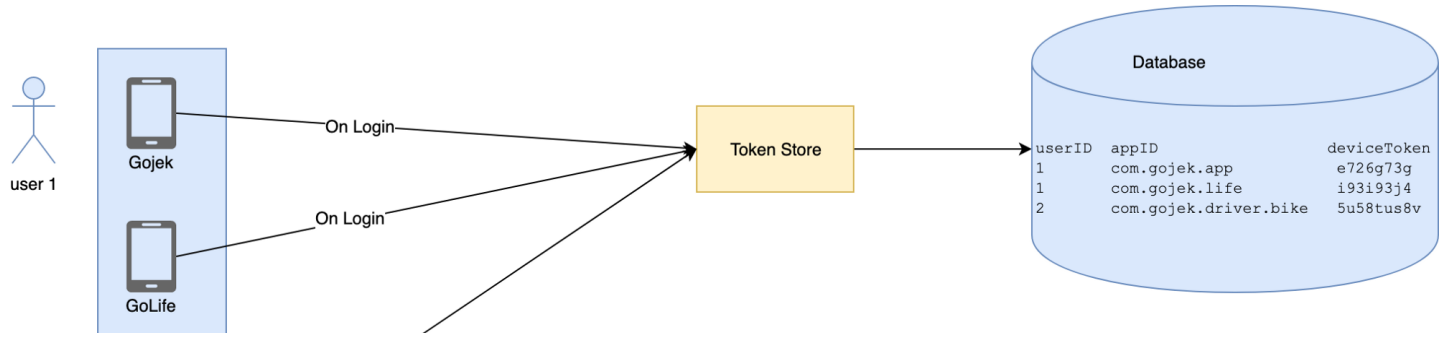
1. **Notification server** — exposes the API to send notifications, and pushes it as a job on our job queue
2. **Token store** — stores the devices and devices tokens of all the currently logged-in users
3. **Notification worker** — consumes jobs on the job queue and sends notifications via the notification providers

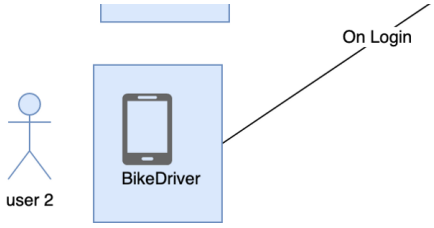


Each component abstracts away some part of the problems we discussed above. Let's take a look at each part in more detail:

1. Token store

Once a user is logged into our application, the application makes a call to the token store API with their device token and application ID.





This entry is then removed when the user logs out.

> *The token store abstracts the process of deciding which devices to send a notification to for each user*

2. Notification server

This is an HTTP server that exposes an API internally to send notifications.

In order to make things simple, the API accepts the user ID, and an optional application ID as HTTP headers, and notification information in the request body:

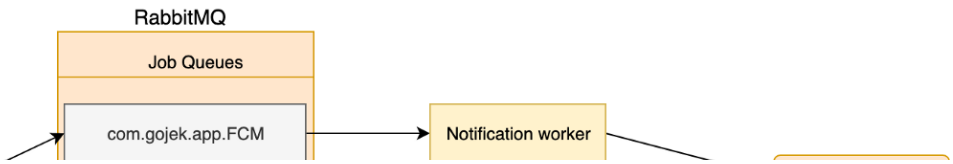
```
POST http://<base_url>/notification
user_id: <user_id>
application_id: <application_id>

{
        "payload": {},
        "title": "You driver is here",
        "message": "Please meet your driver at the pickup point"
}
```

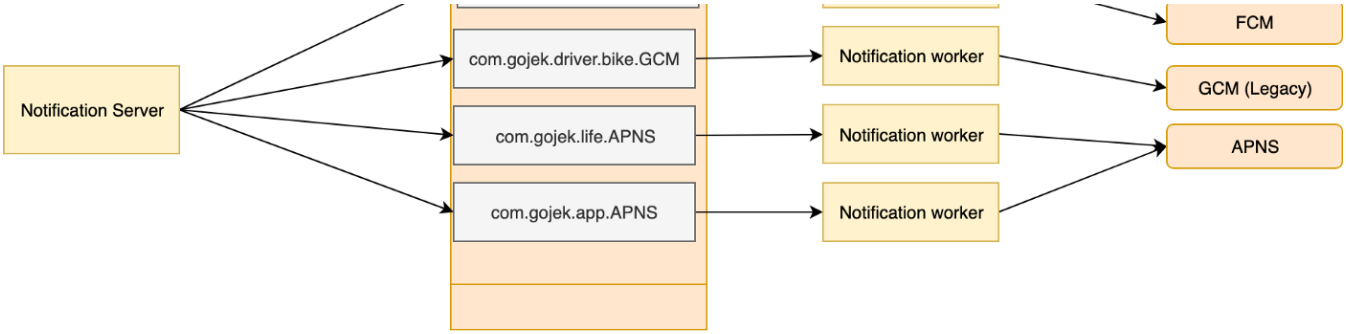
The server fetches all of the users devices from the token store, and schedules a job for each of the users devices.

> *The notification server abstracts the external interface to the system. Now, other services in Gojek that need to send out a notification just need to call our API with the user ID, and the system takes care of the rest*

3. Job Queue

We use RabbitMQ as an exchange for our job queues. We have a queue for each application ID and provider type.





This is important because we want to isolate failures for each application and provider type. For example, if our FCM token expires for `com.gojek.app`, we don't want to stall the jobs for `com.gojek.life` or `com.gojek.driver.bike`.

4. Notification workers

The worker processes consume messages from the job queues, and send out messages to the respective notification providers.

In order to make our code simpler, and to accommodate different service providers if needed, we make use of interfaces to abstract the functionality implemented by each provider:

```
type PushService interface {
        Push(ctx context.Context, m PushRequest) (PushResponse,
error)
}
```

The `Push` method takes a request object and returns a response object.

The request structure contains information related to the recipient and additional options about the notification, like expiry time, title and text:

```
type PushRequest struct {
        DeviceID   string
        Title      string
        Message    string
        Payload    map[string]interface{}
        //some other parameters omitted
}
```

The response contains information on whether the notification was sent to the provider's server successfully:

```
type PushResponse struct {
        Success         bool
        ErrorMsg        string
}
```

We then implement the interface for specific providers. For example, the code for Google FCM provider, and Apples APNS provider looks something like this:

```
type FCMProvider struct {
        // configuration for our provider, like API token and URL
endpoint
}

func (p *FCMProvider) Push(ctx context.Context, m queue.Message)
(notification.PushResponse, error) {
        // code to send a notification payload to FCMs server
}

type APNSProvider struct {
        // configuration for our provider, like API token and URL
endpoint
}

func (p *APNSProvider) Push(ctx context.Context, m queue.Message)
(notification.PushResponse, error) {
        // code to send a notification payload to FCMs server
}
```

> *The notification worker abstracts the process of selecting the correct provider to send the notification. It selects the correct provider, with the correct API key based on the application ID of the message received from the queue.*

Conclusion

By finding common patterns in the challenges we faced, and abstracting them into their own services, we turned a relatively complex problem into a suite of services that are reasonably straightforward and easy to manage on their own.

Each time we found a point which needed to handle multiple implementations of the same core logic, we put it behind a dedicated service:

1. Multiple devices for a user was put behind the token service
2. Multiple applications were given a common interface on notification server
3. Multiple providers were handled by individual job queues and notification workers

In the end, we wound up with a system that can comfortably handle **over a million notifications every hour**.

Want more updates on how we build scalable systems for Southeast Asia's #SuperApp? Sign up for our newsletter!

gojek.jobs

Tech Notifications Mobile Startup Gojek