Notification System Design

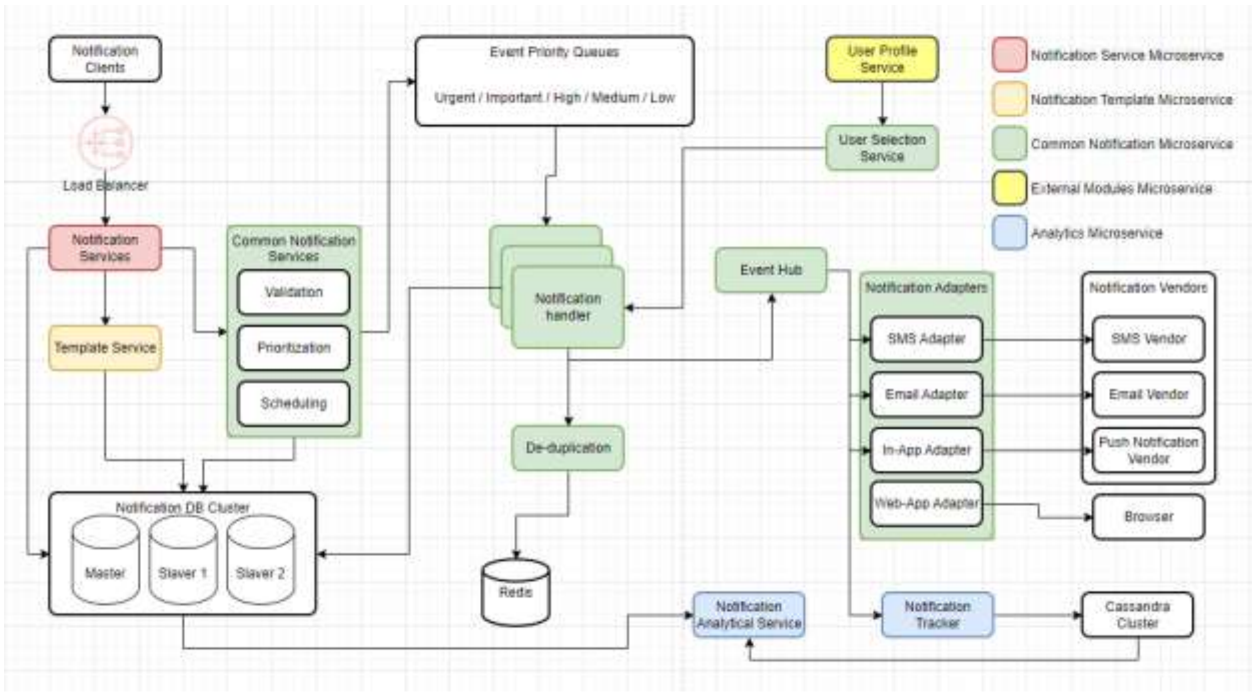
Functional Requirements

- The system should be able to send notifications to subscribed consumers.
- The system should be able to prioritize notifications. OTPs are high-priority messages while news feed updates may be a lower priority.
- The system should be able to support Email, SMS, and push notifications on mobile and web browsers.
- Single/simple and bulk notification messages
- No same notification twice
- Tracking every notification, delivered, opened, seen, unsuccessful, canceled

Non Functional Requirements

- The system should be highly available.
- Latency should be low. OTP messages are time-critical.
- It should be scalable, to handle a growing number of subscriptions.
- Extendable/Pluggable design to add more clients, adapters and vendors.

System Design Architecture:



These are the solution design considerations and components:

1. **Notification clients:**

These clients will request using API calls. These clients will send notification messages to notification services:

2. **Notification Services:**

These services are entry services which will expose REST APIs to clients and interact with the clients. They are responsible to build notification messages by consuming Template Service.

3. **Template Service:**

This service manages all ready-to use templates for OTP, SMS, Email and other push notification messages. It also provides REST APIs to create, update, delete and manage templates.

4. **User Selection Service:**

This service will provide services to choose target users and various application modules. There could be use cases to send bulk messages to specific group of users or different application modules.

5. **User Profile Service:**

This service will provide various features including managing users profile and their preferences. It will also provide feature to unsubscribe for notifications and also notification receiving frequency etc. Notification Service will be dependent on this service.

6. **Common Notification Service**

- **Scheduling**: provide APIs to schedule notifications like immediate or any given time or auto-triggered messages based on the scheduled times.

- **Validation**: validating notification messages against business rules and expected format.
- **Prioritization**: It will also prioritize notification based on urgent, important, high, medium and low priorities. OTP notification messages have higher priority with a time-bound expiry time, they will always be sent in higher priority.

7. **Notification Handler:**

This service will consume notification messages from Event Hub by polling event priority queues based on their priority. High precedence will be given to "High" queue and so on so forth. Finally, It will send notification messages to message specific adapter thru Event Hub.

This service will also fetch target user/applications from User Selection Service.

8. Notification DB

- Store all notification messages with their delivery time and status
- Metadata contains the destination and sources
- Using a noSQL database for Schema Flexibility, Write-heavy System

9. Notification Adapters:

These are adapters which will transform incoming messages from event hub (Kafka) and send to external vendors according to their supported format.

10. Notification Tracker

This service will continuously read Event hub queues and track all sent notifications. It captures metadata of the notifications like transmission time delivery status, communication channel, message type etc.

11. Cassandra Database Cluster

This database cluster will persist all notifications for analytics and reporting purpose. It's based on write more and read less concept.

12. De-duplication

Identify if the payload is duplicate or not is to hash the entire payload and compare it in the database. If the hash is found then the notification should be discarded.

- Hash the incoming notification payload
- Query cached database to check and see if it exists or not
- If the hash doesn't exist then store the hash and process the event (send the notification)
- If hash exists then skip sending the notification

API Design

Method	Endpoint	Description
Get	/notification/template	List template
Get	/notification/template/:id	Template detail
Post	/notification/template	Create template
Patch	/notification/template/:id	Update template
Delete	/notification/template/:id	Delete template
Get	/notification	List notification
Get	/notification/:id	Notification detail
Post	/notification	Push notification
Post	/notification/scheduling	Schedule a specified time to send notifications
…	…	…

Scalability

Notification service is horizontally scalable (by adding more servers) and it is decoupled from the other components by using a message queue.