Design a Notification System

1.Main Types of Notifications

1. Push Notifications

- Sent to mobile apps / browsers
- Example:
  - “Your order has been shipped”
  - “New message received”

2. Email Notifications

- Sent through email services (SMTP, SES, SendGrid)
- Example:
  - Account verification
  - Monthly reports
  - Password reset

3. SMS Notifications

- Sent to phone numbers via SMS gateways
- Example:
  - OTP verification
  - Banking alerts

4. In-App Notifications

- Shown inside the application UI
- Example:
  - Instagram like notifications
  - LinkedIn connection request

5. Webhook / System Notifications (service-to-service)

- One system notifying another system
- Example:
  - Payment success webhook
  - Order completed event

Another Important Classification

Based on timing

1. Real-time notifications
   - OTP, chat message
2. Scheduled notifications

    - Daily reminders
3. Batch notifications
    - Weekly reports sent to millions of users

When interviewer says “Design a Notification System”, you should ask these questions.

1. Product Requirement Questions

- What types of notifications should we support?
    - Email, SMS, Push, In-App?
- Is the system real-time, scheduled, or batch notifications?
- Should users be able to opt-in / opt-out of notifications?
- Do we need user preference settings (mute categories, quiet hours)?

2. Scale Questions

- How many daily active users?
- How many notifications per second expected?
- What is peak traffic?

3. Reliability Questions

- Should notifications be guaranteed delivery or best effort?
- Do we need retry mechanisms if sending fails?
- Should notifications be deduplicated (avoid sending twice)?

4. Latency Questions

- What is acceptable delivery delay?
    - milliseconds?
    - seconds?
    - minutes?

5. Storage Questions

- Do we need to store notification history?
- For how long should notifications be retained?

6. Third-party integration

- Are we using external providers (Firebase, SES, Twilio)?
- Should system support multiple providers fallback?

API Interactions

1. Retrieve User Info

Used by notification service to get delivery details (email, phone, device token).

API

GET /users/{user_id}

Response

```
{
  "user_id": "123",
  "email": "user@gmail.com",
  "phone": "+919876543210",
  "device_token": "abcxyz"
}
```

2. Send Notification

Used by other services (order service, payment service, etc.) to trigger notifications.

API

POST /notifications

Request

```
{
  "user_id": "123",
  "type": "push",
  "title": "Order Shipped",
  "message": "Your order #123 has been shipped"
}
```

Response

```
{
  "notification_id": "n789",
```

```
  "status": "queued"
}
```


3. Fetch Notification History

User fetches all past notifications.

API

GET /users/{user_id}/notifications

Optional pagination:

GET /users/{user_id}/notifications?page=1&limit=20

Response


```
[
  {
    "notification_id": "n1",
    "title": "Payment Successful",
    "message": "Payment of ₹500 completed",
    "status": "delivered",
    "timestamp": "2026-02-09T10:00:00Z"
  }
]
```


These are called third-party notification delivery providers.

They are external services that actually deliver the notification to the user's device.
Your notification system does not directly send SMS/email/push — instead it calls these providers' APIs.

Examples

1. Push Notification Providers

- **APNs (Apple Push Notification service) → sends notifications to iOS devices**
- **FCM (Firebase Cloud Messaging) → sends notifications to Android / Web**

2. SMS Providers

- Twilio
- Nexmo (Vonage)

Used to send:

- OTP messages
- alerts
- transactional SMS

3. Email Providers

- Mailchimp
- SendGrid
- Amazon SES

Used for:

- account emails
- marketing emails
- transactional emails

How it works in architecture

Application Service

↓

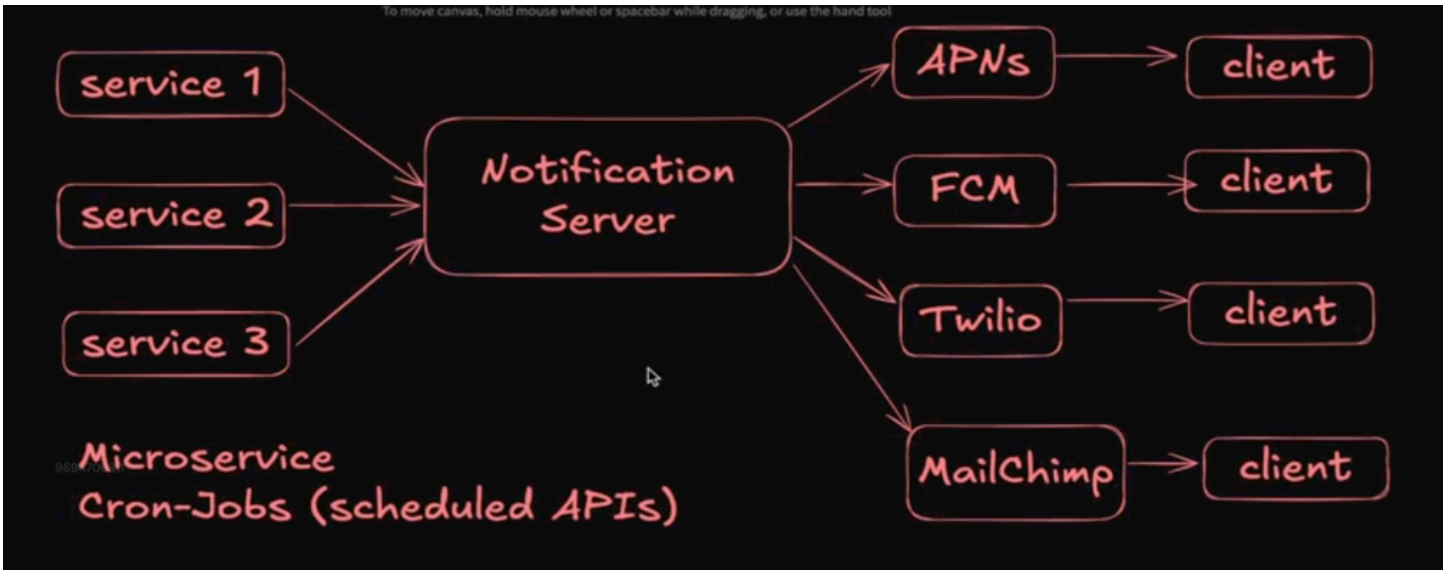
Notification Service

↓

Third-party provider (FCM / APNs / Twilio / SES)

↓

User device



It represents a centralized Notification Server that receives notification requests from multiple internal services and sends them to users through different delivery providers.

Flow Explanation

1. Producer Services

- Service 1, Service 2, Service 3
- These are application microservices (Order service, Payment service, Social service, etc.)
- They call the Notification Server API to request sending a notification.

2. Notification Server

- Central system responsible for:
  - validating requests
  - selecting notification channel (push, SMS, email)
  - formatting message
  - forwarding to correct provider

3. Third-Party Providers

- APNs → iOS push notifications
- FCM → Android / Web push notifications
- Twilio → SMS
- Mailchimp → Email

These providers actually deliver the notification to the client device.

4. Cron Jobs / Scheduled APIs

- Used for scheduled notifications
  - reminders
  - daily summaries
  - marketing campaigns

Problems in this Design

1. No Queue (Biggest issue)

Requests go directly from services → Notification Server → providers.

Problems:

- Traffic spikes can overload notification server
- Requests may fail if provider is slow
- No buffering mechanism

2. Single Notification Server = Single Point of Failure

If the notification server crashes:

- **All notifications stop**
- **Entire system becomes unavailable**

3. No Retry Mechanism

If:

- **Twilio fails**
- **FCM temporarily down**

Notifications will be lost because no retry logic exists.

4. Tight Coupling with Providers

Notification server directly calling:

- **APNs**
- **FCM**
- **Twilio**
- **Mailchimp**

Any provider latency will block processing.

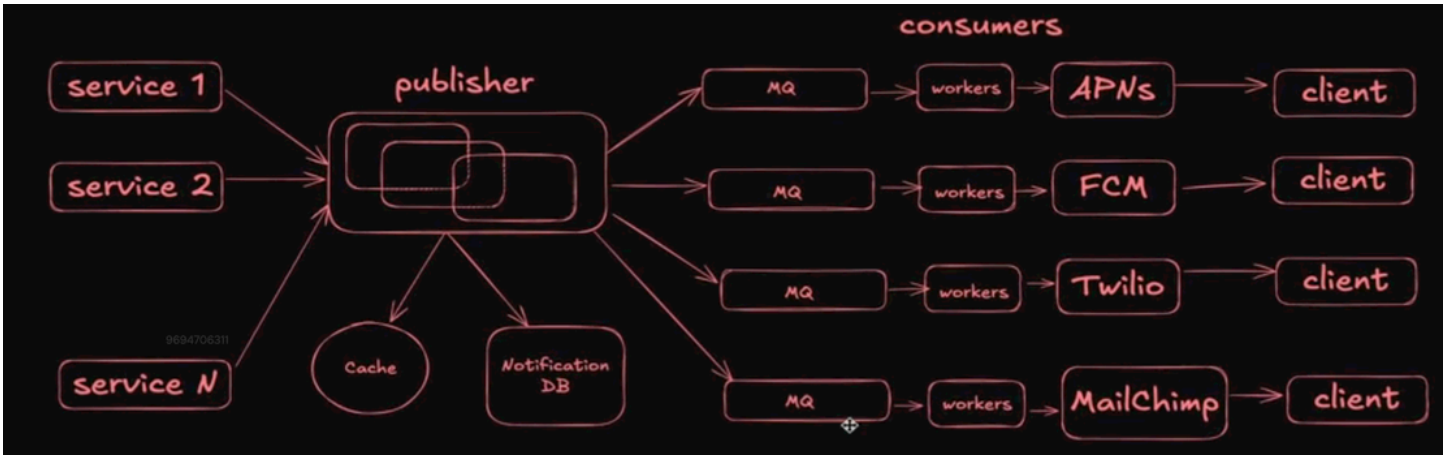
5. No Scalability

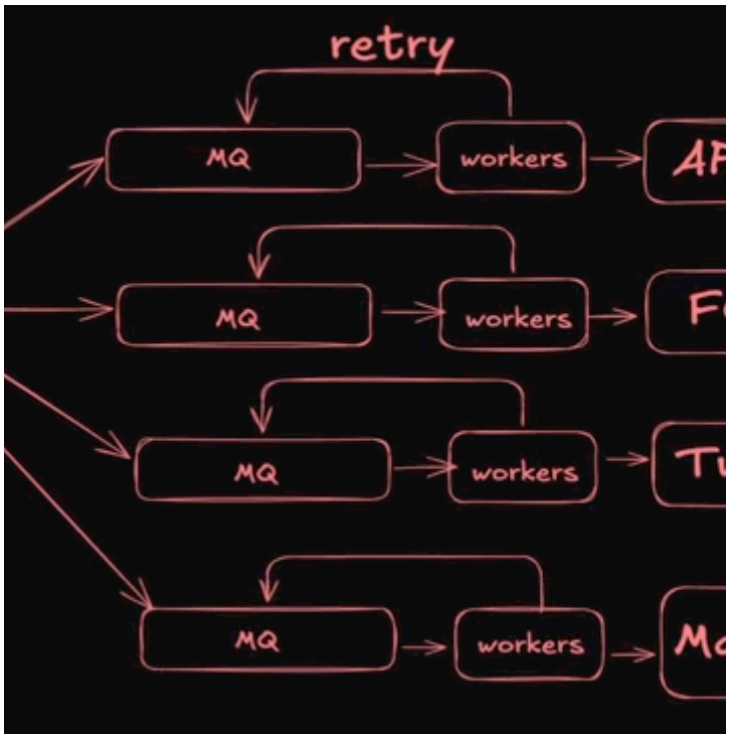
As traffic increases:

- **One server cannot handle millions of notifications/sec**
- **Horizontal scaling difficult without queues/workers**

6. No Prioritization / Scheduling Handling

- **Urgent OTP and marketing messages treated the same**
- **Scheduled notifications not clearly handled**





Step-by-step Explanation

1. Producer Services (Left side)

- Service 1, Service 2 ... Service N
- These services generate notification events (order placed, payment success, message received).
- They send requests to the Notification Publisher Service.

2. Publisher (Notification Service)

Responsibilities:

- Validate request
- Determine notification channel (Push / SMS / Email)
- Store notification in Notification DB
- Use Cache for fast user/device lookup
- Publish message into appropriate Message Queue (MQ)

Example:

- Push notification → Push MQ
- SMS → SMS MQ
- Email → Email MQ

3. Message Queues (MQ)

- Act as buffer
- Handle traffic spikes
- Ensure notifications are not lost
- Enable asynchronous processing

- **Allow horizontal scaling**

Each channel has its own queue.

4. Workers (Consumers)

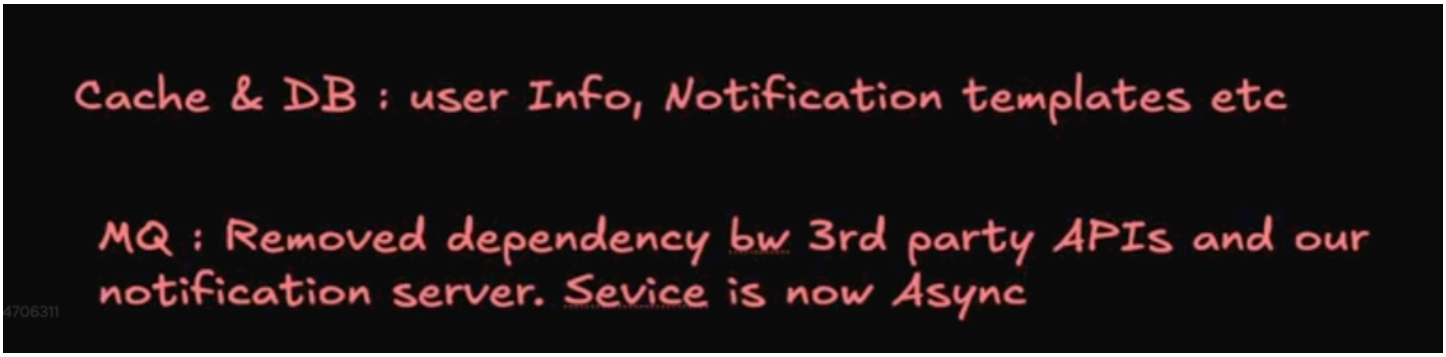
- **Workers read messages from MQ**
- **Format and send notification to the appropriate provider**
  - **APNs → iOS**
  - **FCM → Android/Web**
  - **Twilio → SMS**
  - **Mailchimp → Email**
- **Multiple workers can run in parallel for scaling.**

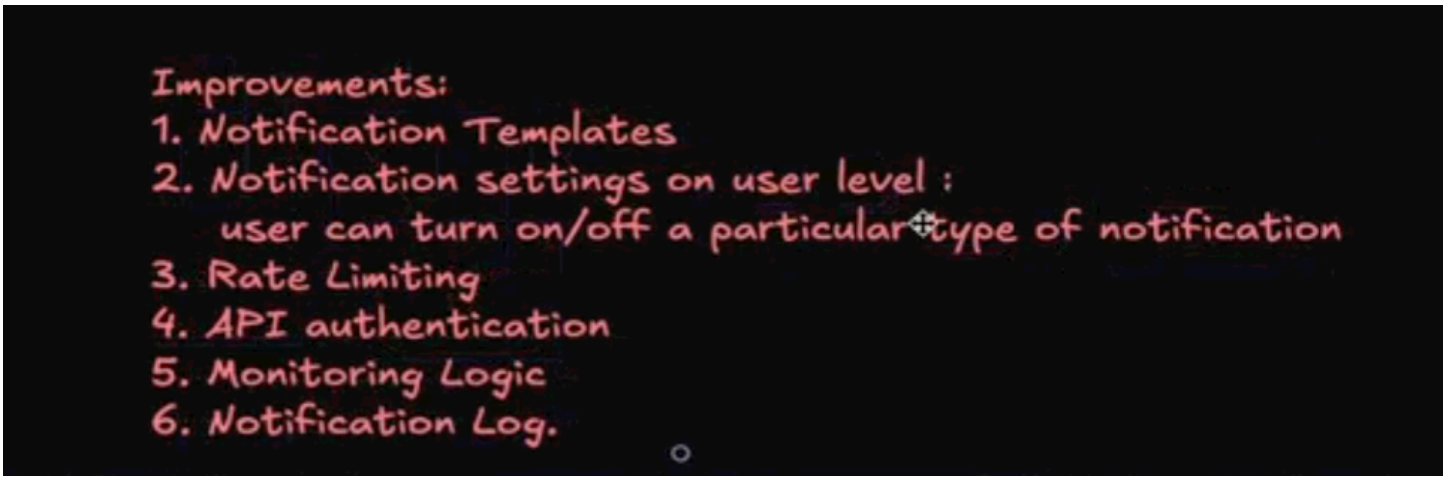
5. Delivery Providers → Client

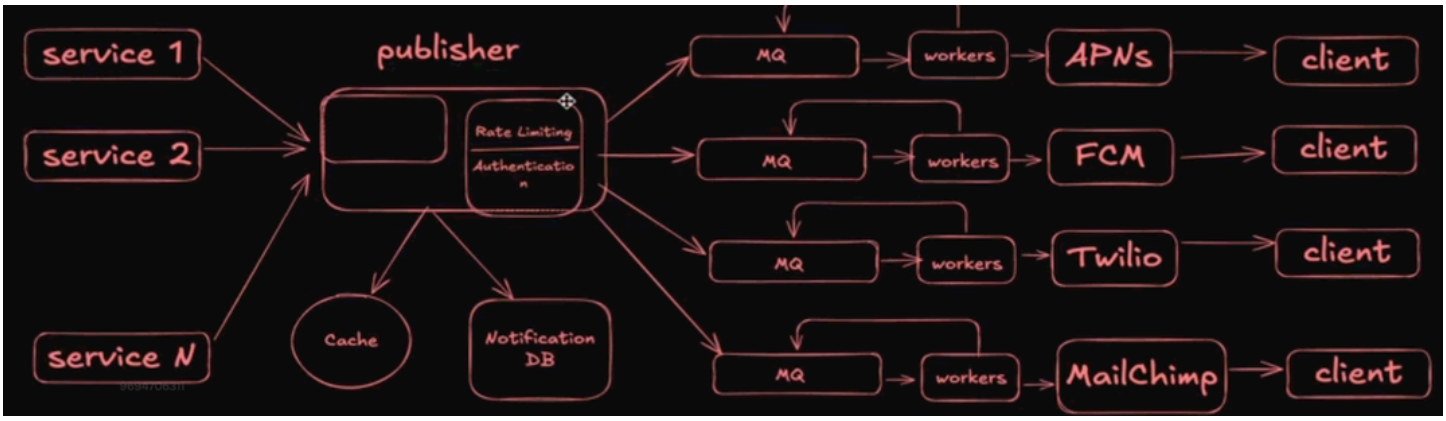
Providers deliver the notification to the user device.

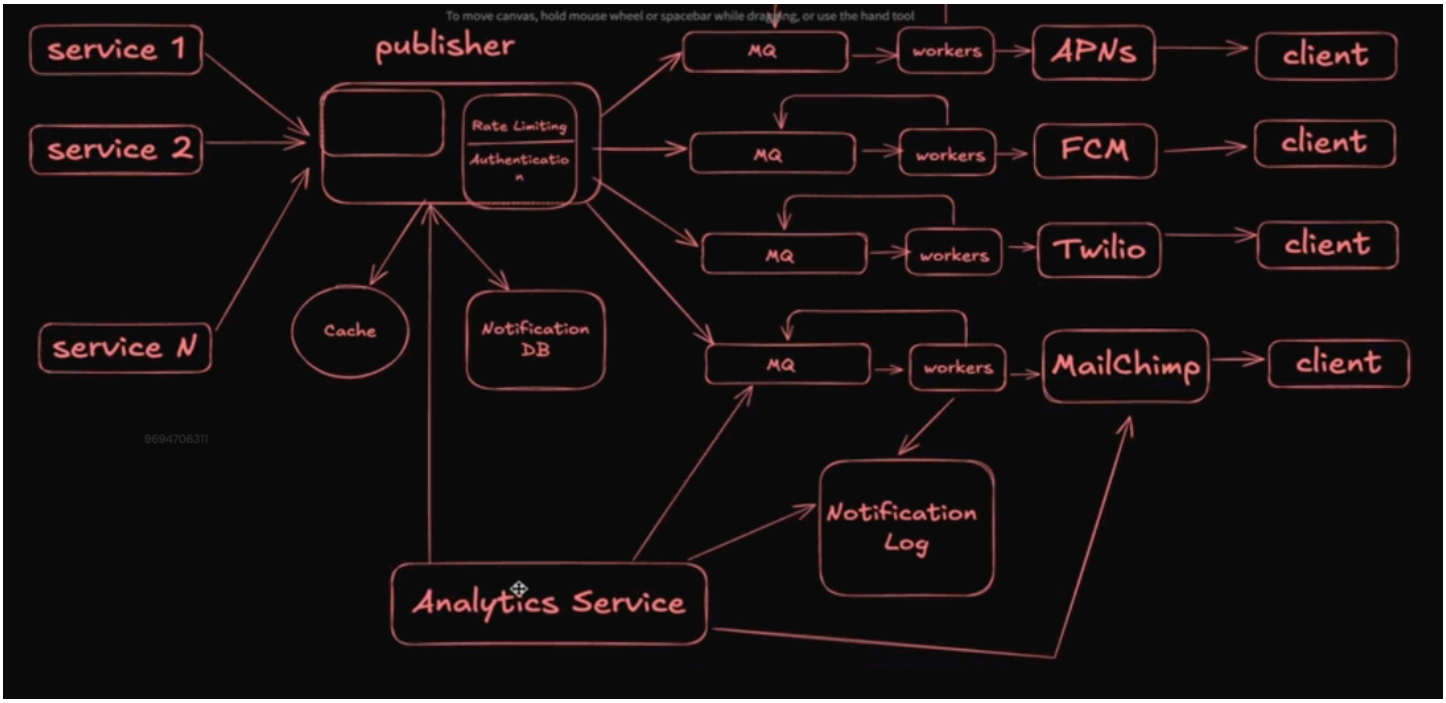
Why this design is better

- **No single bottleneck**
- **Handles traffic spikes**
- **Supports retries**
- **Horizontally scalable**
- **Reliable delivery**
- **Independent scaling per channel (email, sms, push)**









1. Producer Services

- Service 1, Service 2 ... Service N
- These services generate events:
  - order placed
  - payment success
  - new message
- They call the Notification Publisher API

2. Publisher / Notification Service

This is the entry point of the system.

Responsibilities:

- Authentication & Authorization
- Rate limiting (protect system from abuse)
- Validate notification request
- Determine channel (push / email / sms)
- Store notification in Notification DB

- **Use Cache for fast user/device lookup**
- **Push message to appropriate Message Queue (MQ)**

3. Channel-specific Message Queues (MQ)

Separate queues for:

- **Push notifications**
- **SMS**
- **Email**

Why:

- **Independent scaling**
- **Isolation of failures**
- **Prioritization possible**

Queues act as buffer during traffic spikes.

4. Workers (Consumers)

Workers read messages from MQ:

- **Format message**
- **Call third-party providers:**
  - **APNs → iOS push**
  - **FCM → Android push**
  - **Twilio → SMS**
  - **Mailchimp → Email**
- **Update Notification Log with delivery status**

Multiple workers run in parallel for scaling.

5. Notification Log

Stores:

- **delivery status**
- **retry attempts**
- **timestamps**

Used for:

- **debugging**
- **user notification history**
- **retry handling**

6. Analytics Service

Collects data:

- **notifications sent**
- **open rates**
- **click rates**
- **failures**
- **campaign metrics**

Helps business dashboards and reporting.

7. End Delivery

Third-party providers finally deliver notifications to client devices.