Notification Systems

System Design
- DSA optimization
- Code / DB schema / E-R
  - Low level Design
- microservices / data flow / fault tolerance / scaling challenges
  - High level Design

No Code today

5 step approach to any design problem

1. Problem Statement
2. Functional Requirements
3. Non Functional Requirements
4. Scale Estimation
5. Design

} Prerequisites

Problem

Notification System – generic notifications ← SMS, whatsapp, email, app ...

Scaler

Education Company

Toy Store

E-commerce Company

notification is NOT their core business

core business – something else

notifications – just a feature

Notification.com

SAAS

core offering → Notification SAAS

tons of clients → clients have users

v. high user count

v. high volume

lots of features

Functional Requirements

MVP features

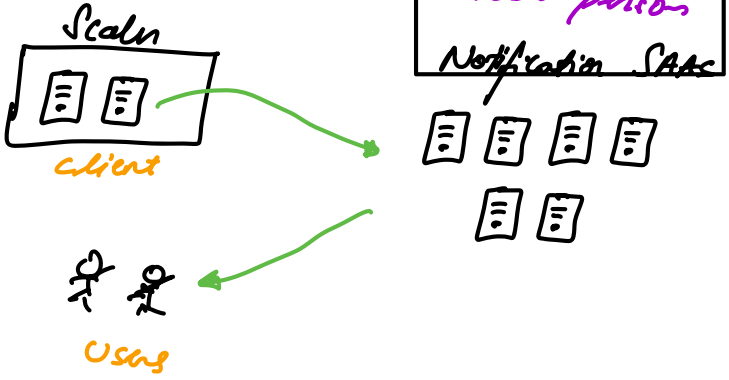
Minimal Viable Product

Notification : any small (<1Kb) content that we wish to send to a user via some means

text / audio / video / images — URL

sms / whatsapp / appl.. — flexible

1. Clients (company's backend) can send notification to users (actual person)



Post notify

(api-key (client id), user-id, content, json, time)

→ Success 2xx / failure 4xx 5xx

{ msg: ..., attach: ..., icon: ... }

→ Scheduling

2 System should support multiple channels

whatsapp / sms / email / in-app ...

must be pluggable / extensible

+ slack + call

seamlessly add more channels without changing the architecture

3 clients should be able to BROADCAST notifications to multiple users

Scaler → Broadcast masterclass reminder → 1000 users

Twitter → Elon Musk Tweets → 100 M users

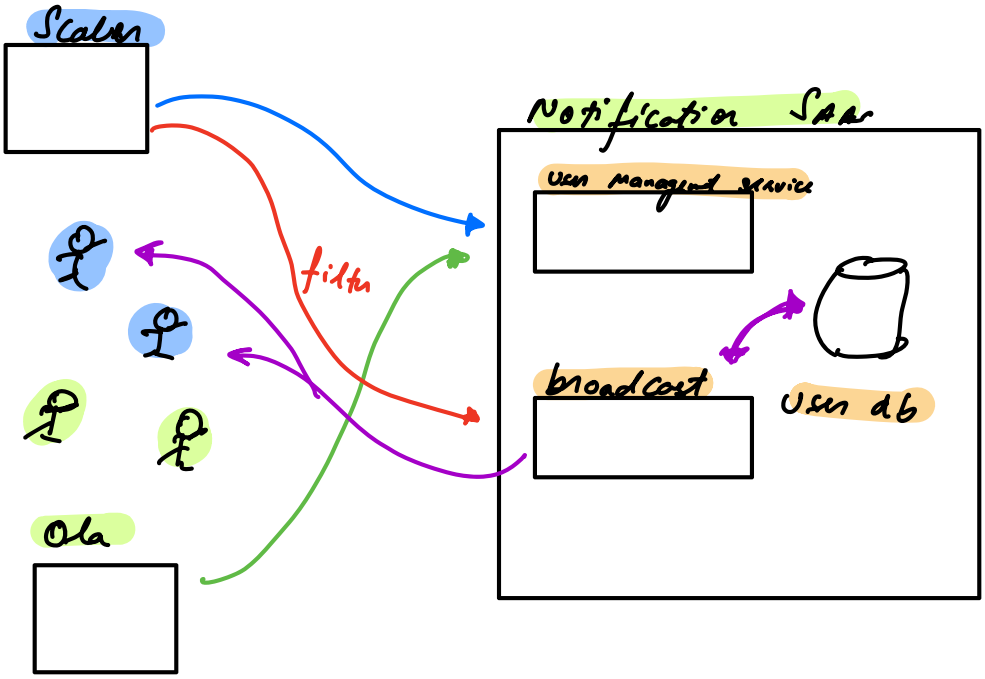
POST notify (api-key (client id) filter content json) → success 2xx / failure 4xx 5xx

{ msg: ... attach: ... icon: .. }

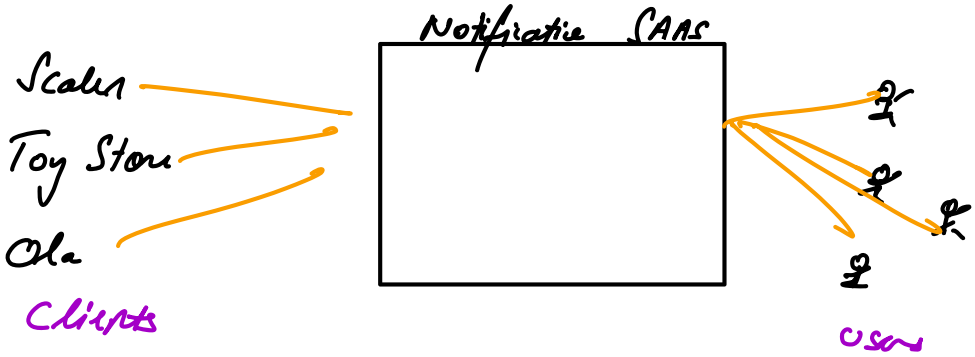
Note: providing list of user-ids in API call is bad design

4 Clients should be able to manage their own list

Hidden feature



5 Rate Limiting



① Client limits (Quota limits)

based on the purchase type

free → 500 notifications/day

basic → 50,000

team → 50,000 + additional team

Enterprise → unlimited

② User side limits

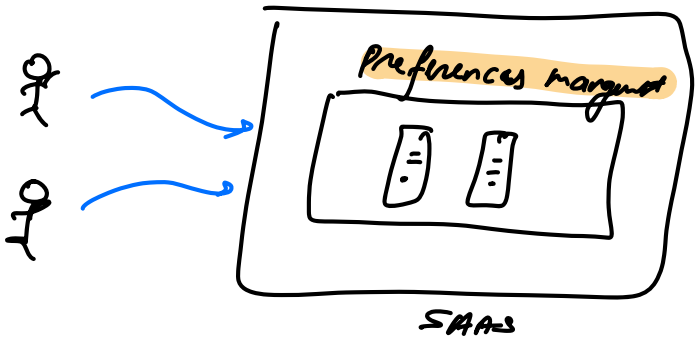
- no user should recieve too many notifications
- users should be able to customize their notification subscriptions

Sensitive / critical — OTP / emergency alerts

marketing

channel — email, sms, in-app

Push - notifications



⑥ Observability

- detect something is wrong
- diagnose the issue
  fix
- verify the fix is working

} Observability

2020 ~ 10M servers @google

- # notifications sent / client / day / type / channel
- # exceptions / hour

- logs
- Search & analytics

Non-functional Requirements – Design Goals

① Compliance

TRAI

- Very strict rules on what SMS/phone calls we can send
- email providers (SMTP) → own rules

② Priority should be considered

OTPs — v. high priority

Promotions — lowest priority

notify(..)

{ type — otp / ... }

③ High Availability

fault tolerant

④ Reliable

backups / retry mechanism

⑤ Low latency

⑥ Single-delivery / Idempotency

∵ we have an auto-retry mechanism

~~⑦ Security~~ v. difficult to do correctly

leave it to a security expert

Scale Estimation

100 users only — no need of HLD

Scale nothing

Total # users (across all clients) ≤ 2B ($B = 10^9$)

avg notifications / user / day = 10

Avg # notifications /sec = 2×10^9 ~~users~~ * $\frac{10 \text{ notifications}}{\text{~~user~~} * \text{day}}$

200,000 /s

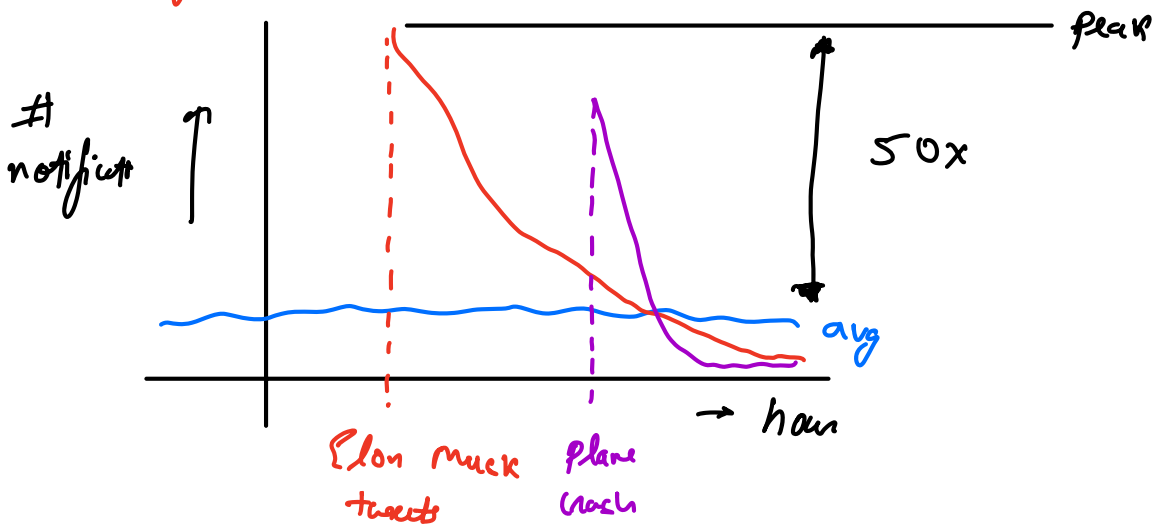
$= 2 \times 10^9 \times 10 \frac{\text{notifications}}{\text{day}}$

day = 24 x 60 x 60 seconds
= 86,400 seconds
$\approx 10^5$ seconds

$= \frac{2 \times 10^9 \times 10 \text{ notifications}}{10^5 \text{ seconds}}$

$= 2 \times 10^5$ notifications /sec

Peak = 10 M notifications /sec



2×10^5 avg notifications/sec * 50

= 10 M notifications /sec → our system should be able to handle this!

System Design

multiple concepts
- LB
- Auth
- API gateway
- Brokers
- ...

Scaler HLD Curriculum

- **How backend systems work**
  - DNS / vertical vs horizontal scaling / load balancing / sharding
- **Load Balancing & Consistent Hashing**
  - Healthcheck & Heartbeat / Roll outs / DNS as load balancer / Untrusted Layer / Consistent hashing - deep dive
- **Caching**
  - Browser / CDN / Backend
  - Local vs Global / Single vs Distributed
  - Redis
  - Cache Invalidation
    - Write through
    - TTL
    - Write back
    - Write ...

Cache Eviction — FIFO
LRU
LFU
MRU

- Caching Case Studies

  Scaler Code Judge / Scaler Leaderboard / Facebook News Feed

- CAP theorem & Replication

  Availability / Consistency / Partition Tolerance

  Immediate Eventual

  CAP theorem / PACELC /

  Master-Slave — Eventual
  With Synchronous
  Tunable Consistency — Quorum

- SQL vs NoSQL

  Pros & Cons

  Sharding Key

  NoSQL Types — Key-Value
  Document
  Wide Column
  Search
  File store

  Graph
  Vector
  In-memory
  timeseries

- Database Internals

Orchestrator / Config Management / Reserved Servers / Shard Creation / Multi-master

- Database Internals

  LSM Trees / Bloom Filters / Sparse Index / Compaction / MemTable / WAL

- Case Study — Google Typeahead

  Tries / Redis / Recency / Sampling / Batching / Precomputation

- Case Study — Messaging apps

  Slack groups / Whatsapp / Facebook Messenger

- Case Study — Kafka + Zookeeper

  event driven architecture / Pub-Sub / topics & partitions / msg order

- Case Study — Elastic Search

  Inverted Index / TF-IDF / Sharding / Tunable Latency

- Case Study — File Storage

  upload / download / chunking

- Case Study — Uber

  Quad Trees / traffic

Traffic Patterns / Request driven

- Case Study — Rate Limiting

  DDoS / Token Bucket / Leaky Bucket / Sliding Window

- Case Study — Id generation

  idempotency / Unique / incremental

- Case Study - Video Streaming

  Netflix / Youtube / Hotstar

  Concurrency

  ABS / chunking / CDNs

- Microservices

  Monolith vs Microservices vs Service Oriented

  Breaking a Monolith

  Observability / Distributed Tracing / Logging

  Thundering herd / Cascading failure / Circuit Breaker

  Distributed Transactions

  Two Phase Commit / Saga / CQRs /

  Orchestration & Choreography

  Event Sourcing

SDE-3 @ google

60 LPA ... 1.2 Cr base
+ Stocks + Performance bonus
+ Health benefits ...
India - Pune / Hyd / Bangalore

Senior Architect / Principle Engineer @ google

1 Cr — 3 Cr base

8:57 ⟶ 9:15 Break

System Architecture

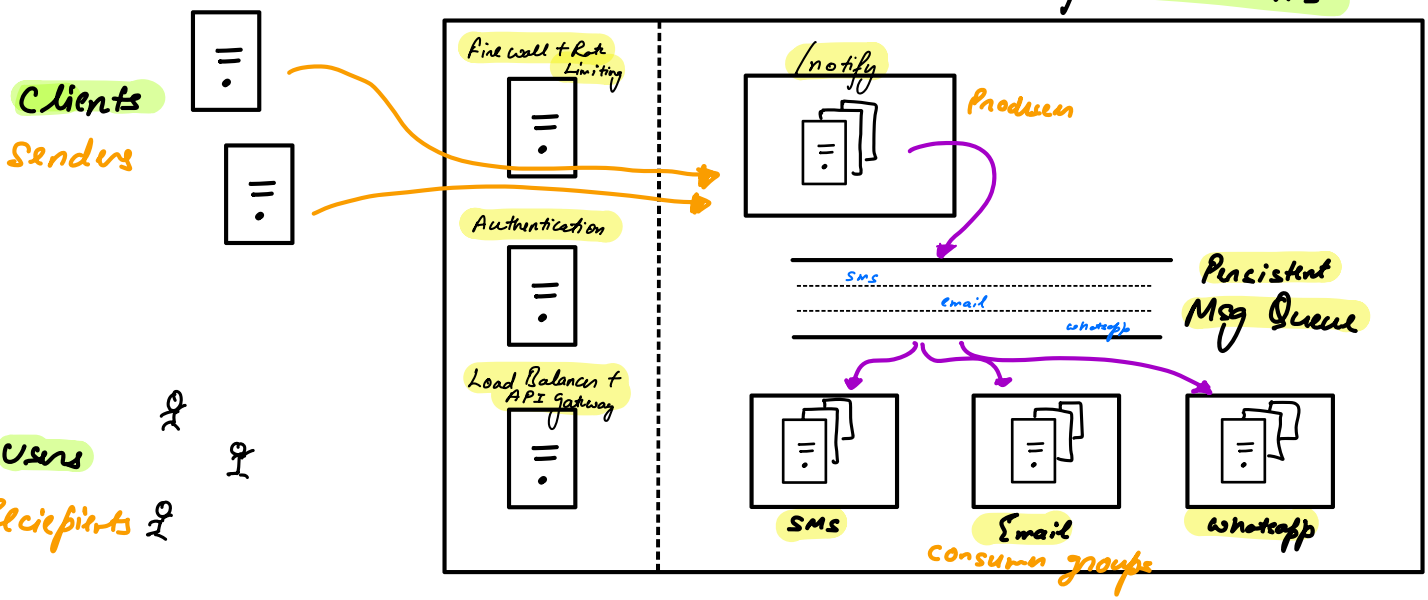
1. Clients should be able to send notifications

notify (api-key / user_id / content)
json

2. System should support multiple channels

type : OTP / promotional / general / new index / ..
channels : {
sms : { msg : Hello , icon : info , attachment : URL }
email { subject : Hi , body : 'Hello' , ... }
} apn { msg : Hello , action : ' open (record) '

Notification SAAS



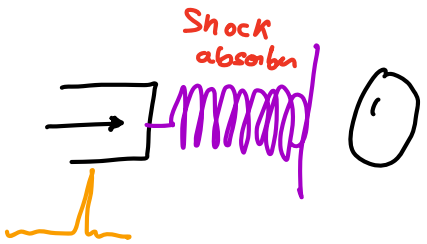
Users
Recipients

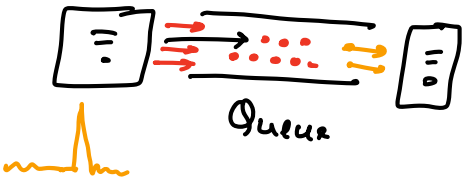
Untrusted layer

Trusted VPC (Virtual Private Cloud)

Auth — Authentication — Untrusted login only
Auth — Authorization — at every service

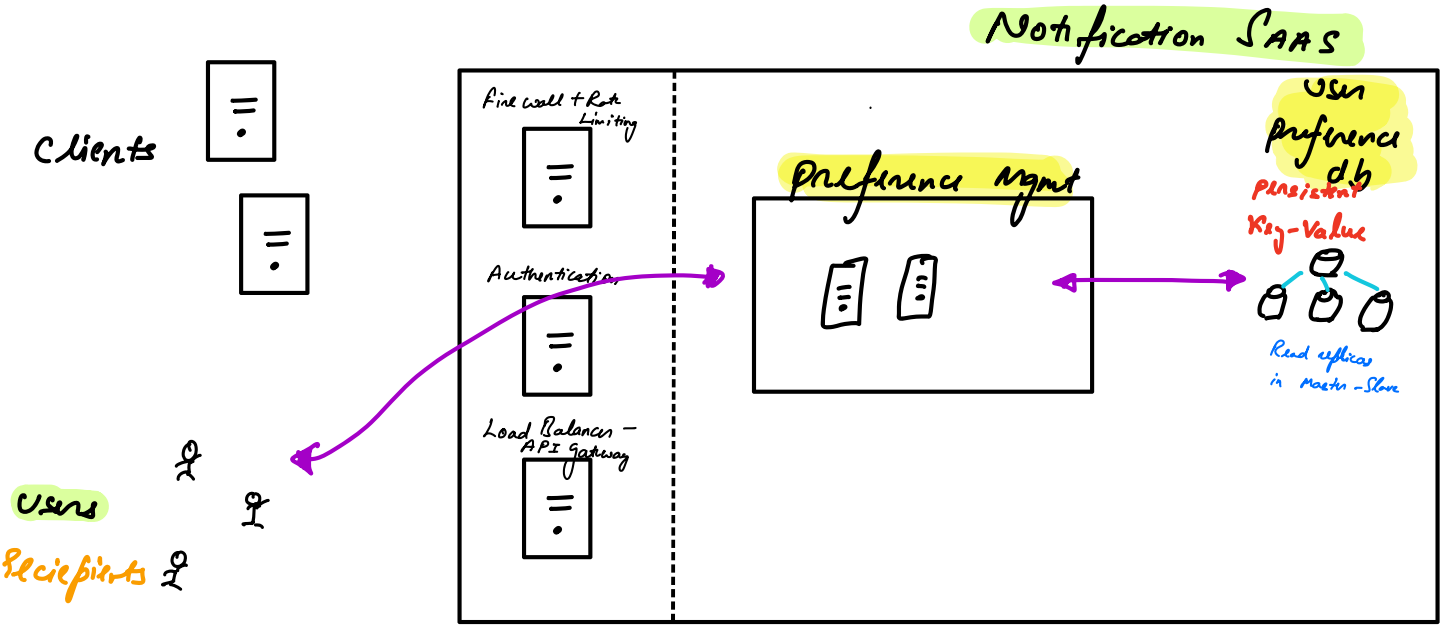
Msg Queue — Pub Sub
Msg Queue — Service Discovery
Msg Queue — Event driven
Msg Queue — Shock absorber





(3) Users should be able to customize their notification preferences

System must respect the user settings



Ideal DB for user-preferences

Strengths | Weaknesses

SQL

Postgres / mySQL / DB2 / RDS / ..

ACID, Relations, Joins, Normalization, Schema, Complex data model

Slow reads & writes
don't scale well
manual sharding

Key-Value

Redis / Memcache / Dynamo DB

Simple, fast, read optimized ✓

No relations
Complex data modeling is not possible
No search
Simple queries
No transaction

document

Mongo / firebase / Elastic search / Couch / Cockroach / ...

Nested, Semi-structured or schemaless data, search, Powerful index & queries

no global index
slow writes
no joins & relations
no ACID transaction

wide column

Cassandra / scylla / Hbase / Big Table

aggregate & analytics, v. fast writes, Pagination by time

no relations
no ACID
Reading wide rows is slow

File Storage / Vector db / graph db - -

2B ~~users~~ * 1 KB ~~preference data~~ / ~~user~~ = 2B * 1KB data

= 2TB

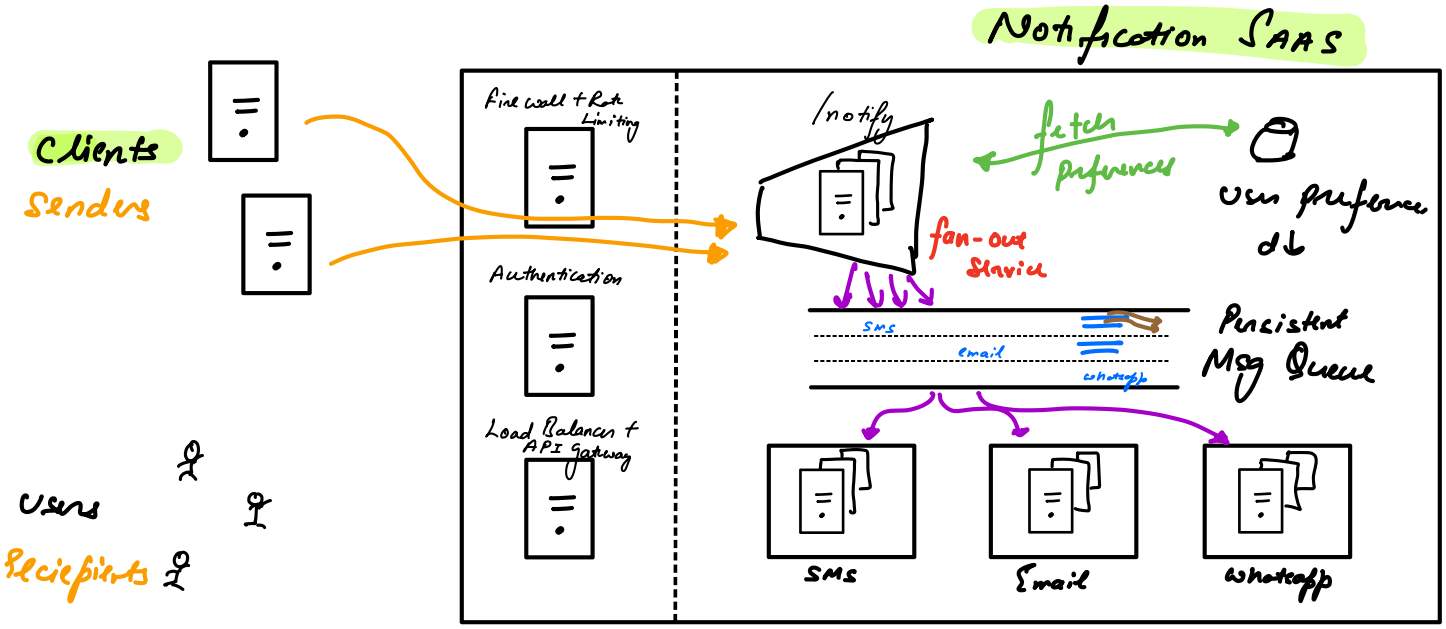
- Can be stored on a single server
- read? v. high (10M r/s)
- write? low

Too much data — Shard

Too many reads — Cache / Read Replicas → improves reads but worsens writes ∵ you have write more than once

Too many writes — Shard / Sample / Batch

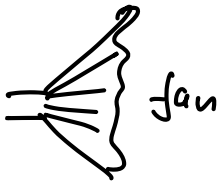
Prevent data loss — Replicate

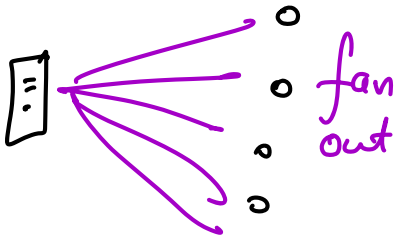


Fan-out notification service

for each notification req.

- Read the channels from request payload
- get user_id → fetch preferences from user-pref db
- filter out channels based on user pref
- broadcast messages (one for each channel) to Msg Q





To separate user-db for pref management service & notification service use CQRS

Message Queue

① Persistent — store notifications on disk (temporarily)

to ensure notification delivery.

② support offsets to ensure single delivery

③ Scale → partition

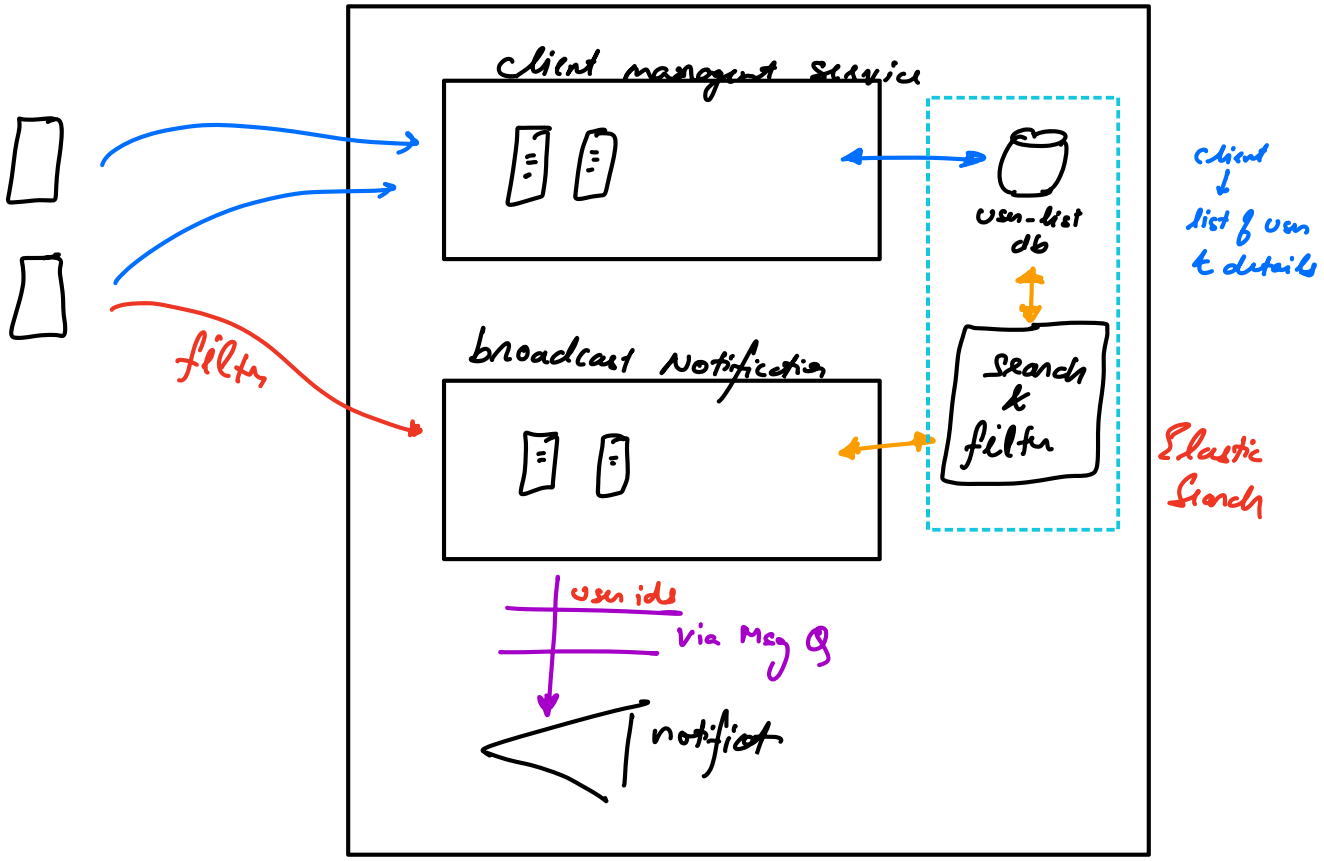
Partition Key

(Partition Key client-id, Topic id channel)

↓ map by same client maintain order

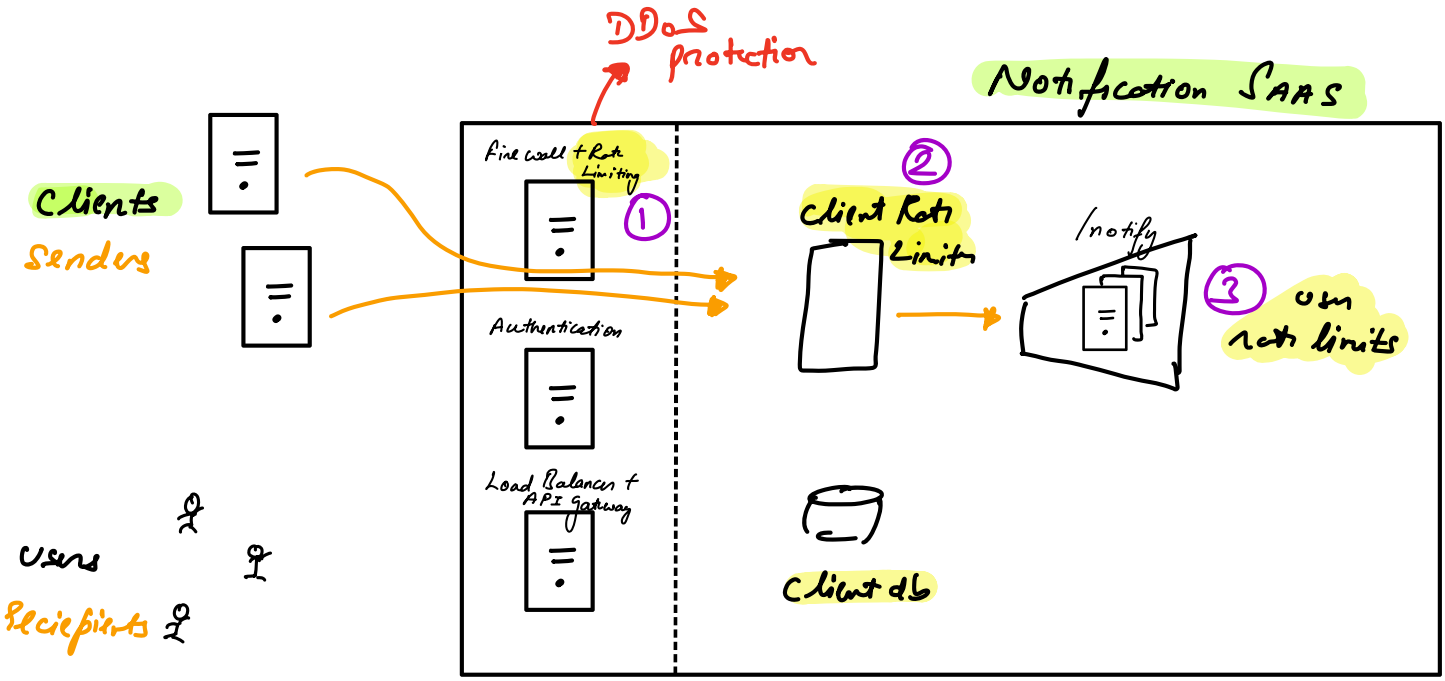
↳ ∵ consumers are channel based

(7) Clients should be able to broadcast msgs



(6) Rate limiting

- generic DDoS protection untrusted layer
- client
  don't exceed purchased Quota
- user
  don't spam user

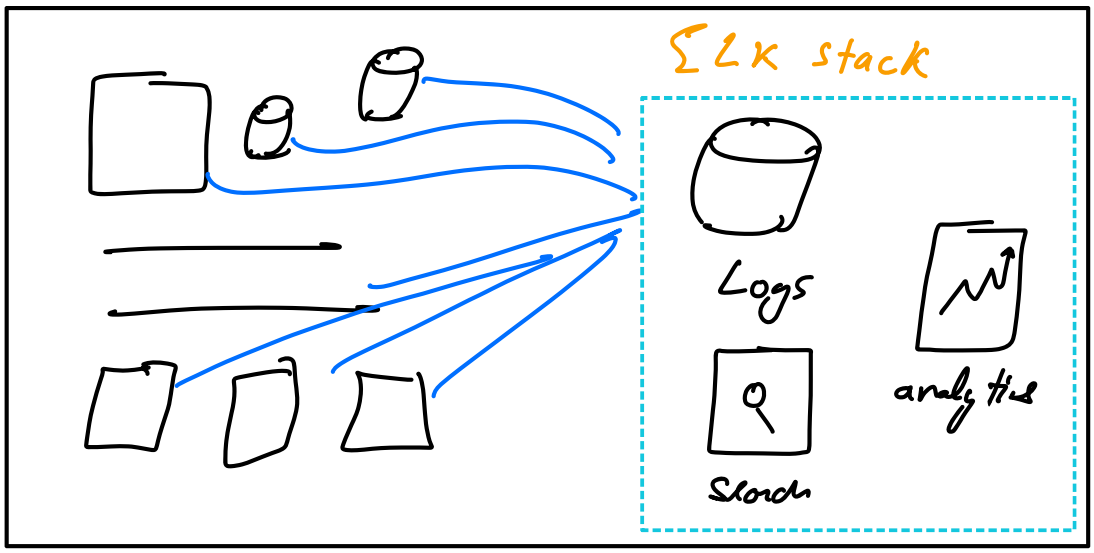


(7) Observability

(1) aggregate logs for all servers

(2) search on logs

(3) analytics (alerts / dashboards / monitoring)



Elastic Search

Log Stash

Kibana

Design Problems are open ended

1. Rate limiting algos
2. Event Driven archit
3. Celebrity Problem.

Elon Musk → 200M followers

M1 Scott → 300M subscribers.

1. Separate infra for celebrities

   Twitter.

2. Bypass the issue !!

   Subscribe → Notification ✓

Bell icon → notification ?