# System Design Cheat Sheet

Reference: Tech Dummies , System Design, Netflix, GeeksForGeeks

For you to go through just before the interview :).

***Disclaimer: This article is still under work. Still have to add notes and more architecture designs. Feel free to add comments on what else to add.***

## Load Balancers (LB)

*Selects Servers/Databases/Caches following some algorithm*

1. Round Robin: Select servers one after another
2. Weighted Round Robin: Admin assigns weight, i.e. probability of getting selected. Server 1 with twice weight of Server 2 will be allotted twice as many requests.
3. Least Connection/Response Time/ Resource Based: Dynamic Load balancing. Server with Least Connection/Response Time/ Resource Based is allotted next request. The values are calculated using client installed at servers.
4. Similarly we have Weighted flavor of Least Connection/Response Time/ Resource Based.

Types: (Ex AWS)

- L4 : Makes balancing decision only on IP address , tcp port. Cannot see request header, client, type etc.
- L7 : Has info about url, message, request type, header, client everything. Can route request based on type of request

*Use L4 when you need to make simple reliable and fast balancing decision on server load, **which has reliable TCP connection.** Use L7 when you need to route request to appropriate resource server, such as image request will go to image server etc.*

***Types of Load Balancers in AWS : ELB, ALB, NLB***

See Table : https://iamondemand.com/blog/elb-vs-alb-vs-nlb-choosing-the-best-aws-load-balancer-for-your-needs/

## Caches

*High Speed Key Value storage. Two popular Implementation Memcached (used by Netflix & Facebook) & Redis (used at Pinterest )*

*Use Memcached when* : **Simple Key Value storage needed**. Need to only store string. No need to perform any operational query on cache. Scaling vertically by using more cores and threads is easier. When keys are maximum 250 B and Values maximum 1MB, when you are ok with only LRU eviction policy. **Its Volatile.**

*Use Redis when:* You need to store objects (**don't want to serialize deserialize : access or change parts of a data object without having to load the entire object**). Scaling horizontally is easier. You need to store Set, Hash, List, Sorted Set (A non-repeating list of string values ordered by a score value). When you want to chose from multiple eviction policies**. When you would want to save data (its non volatile)**

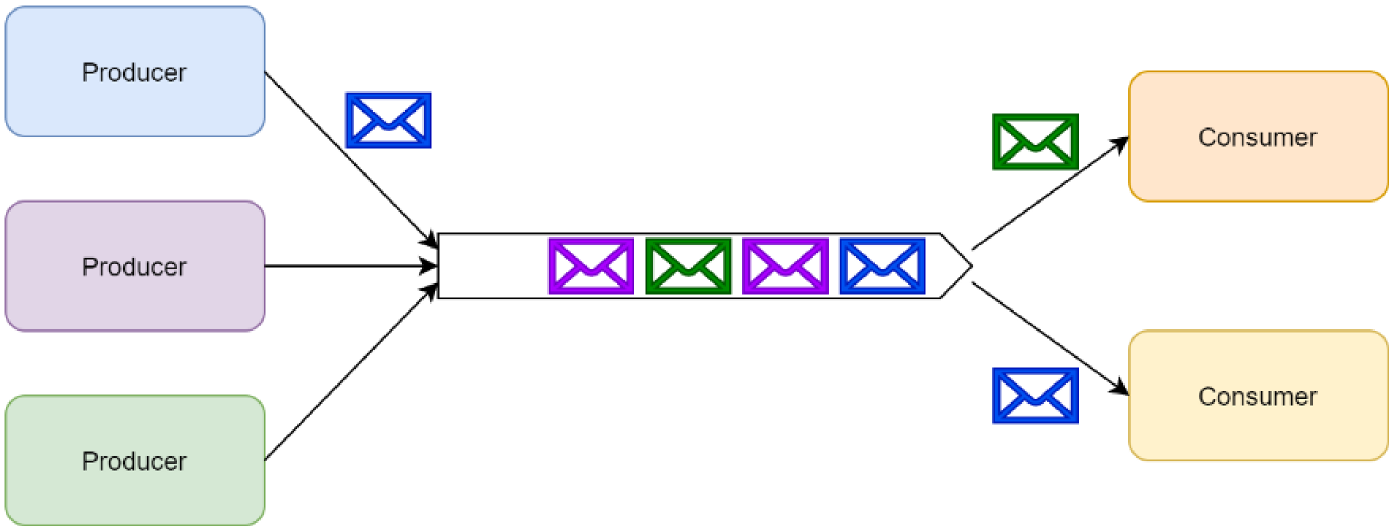
Eviction policies

- **No eviction** returning an error the memory limit is reached.
- **All keys LRU** removing keys by the least recently used first
- **Volatile LRU** removing keys, that have an expiration time set, by the least recently used first.
- **All keys random** removing keys randomly
- **Volatile random** removing keys, that have an expiration time set, randomly
- **Volatile TTL** removing keys, that have an expiration time set, by the shortest time to live first.

## Queues

Two popular async message queuing services: RabbitMQ and Kafka. Supports *message queuing* and *publish/subscribe.*

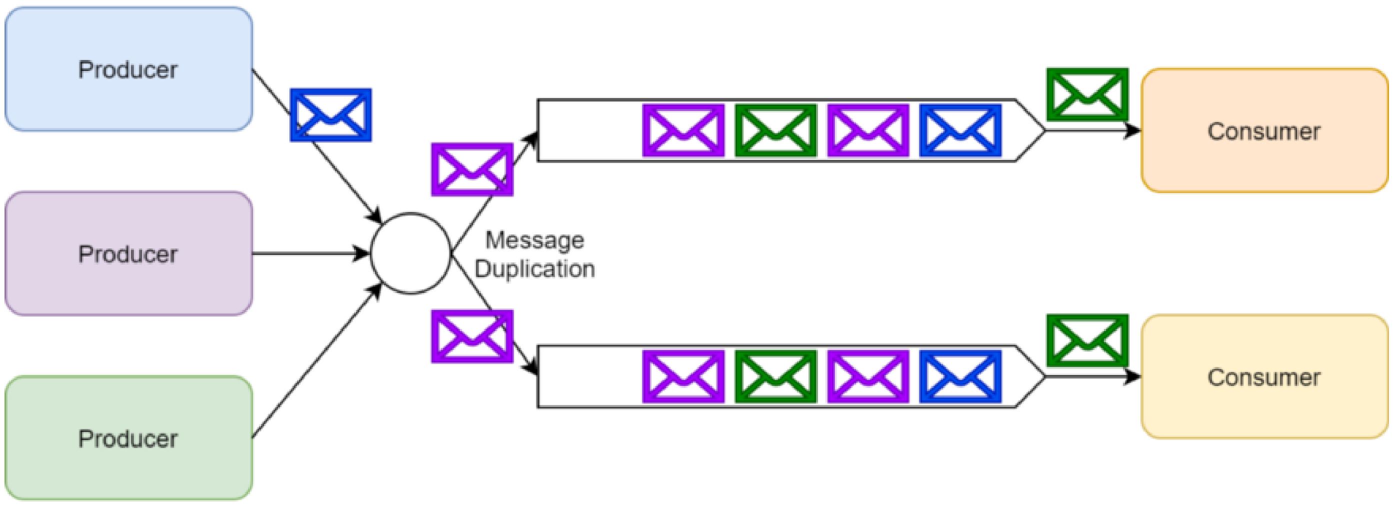
*Message Queue: Used to decouple producer from consumer. Name a queue as X, publish to X, consume from X. Can be used for events based intra service communication. Consumer listen for events in queue.*



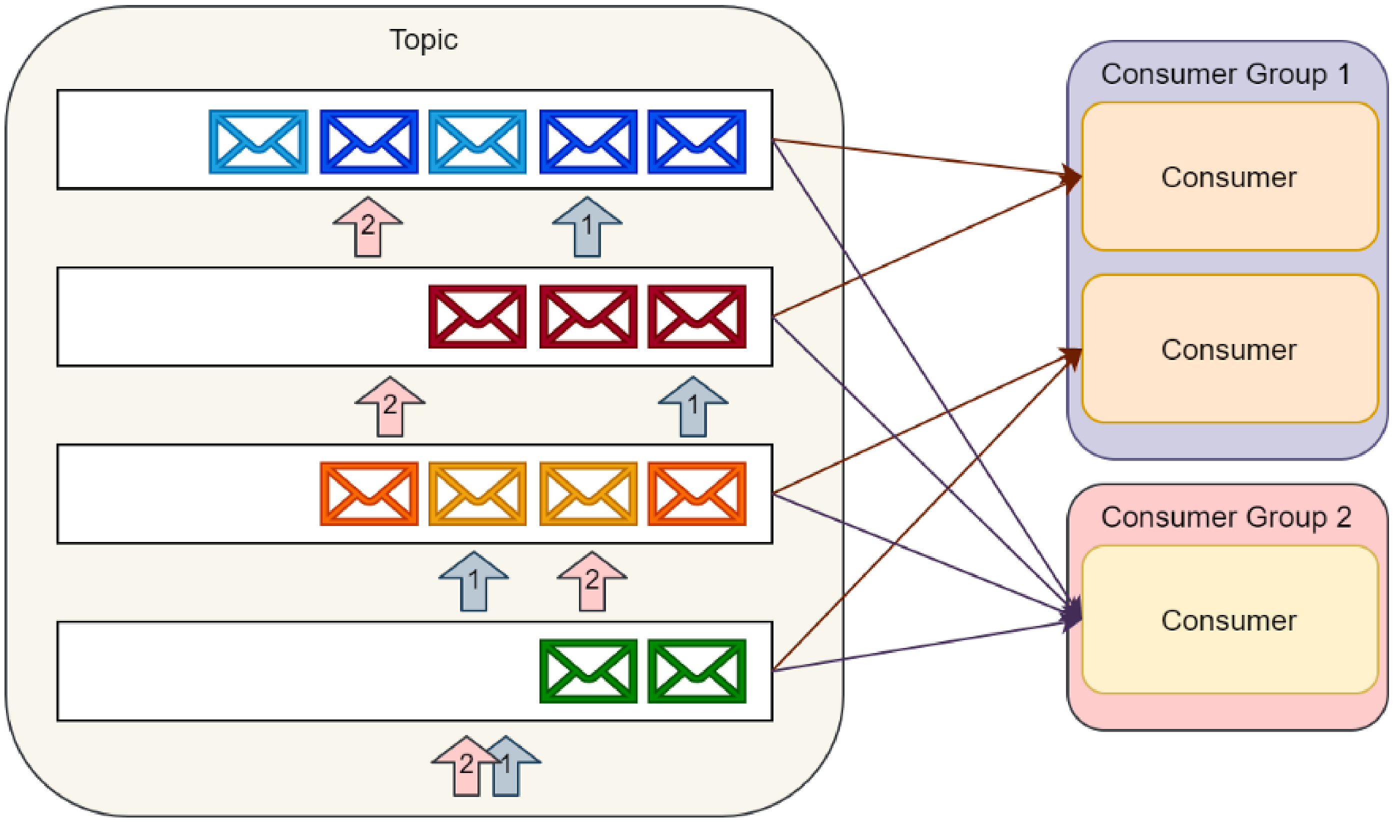
Message Queuing

*Publish/subscribe: Used in Notification system, backend job that takes lot of time and one action that triggers multiple services. Provides : lose coupling between message production and consumption, fault tolerance, retry messages failed to be consumed and independently scalable.*

*Can be done in following ways*



*Message Exchange ( as in Rabbit MQ, SQS):* Producers submit messages to exchange. Consumers create a queue, subscribes to exchange. Exchange sends message to appropriate queues. It also takes care of failure, retry, expiry, routing. **Ordering of consumption not guaranteed.**



*Streaming Platform ( as in Kafka, Kinesis):* Producers just write to their specific logs(Commit log/Write ahead logs). Consumers read data from logs using offset. Failures, retries, deletion, filtering all handled by consumers. **Order is guaranteed per consumer per partition per topic.** Replaying past message is possible(move the offset). Uses Zookeeper ( store information about brokers, topics, partitions, partition leader/followers, consumer offsets, etc.).

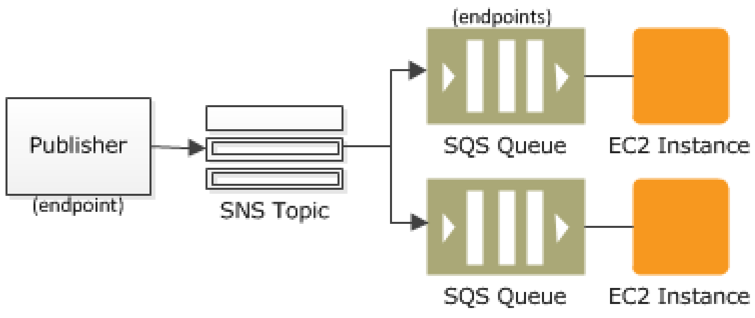
Topics in Kafka is like table in database. Thus, each message is associated with topic. Consumers subscribe to topic ( like update event on table). Multiple subscribers read from a topic( eg, one can send mail, other send phone message on order ship message in shipping topic). Messages not deleted on reading ( can have TTL), and written on disk. Since its sequential write its faster.

Central system managing queue is called Broker.

*See Kafka Use cases* : https://kafka.apache.org/uses

*See RabbitMQ usecases* : https://www.rabbitmq.com/getstarted.html

1. Work Queues : Distributing tasks among workers (the competing consumers pattern) : Cron job.
2. Pub Sub: Sending messages to many consumers at once : Notification service, intra service communiation
3. Routing: Receiving messages selectively : Queuing, retries
4. Topics: Receiving messages based on a pattern (topics) : Selective Notification
5. RPC: Request/reply pattern example.



**SNS SQS**

In SNS SQS, SQS is simply a queue, SNS on the other hand behaves like a broker. SQS behaves as subscribers EC2 behaves as consumers of SQS messages So SNS+SQS = Kafka. SQS alone is not. SQS is more like like partition in Kafka.

## Configuration Service

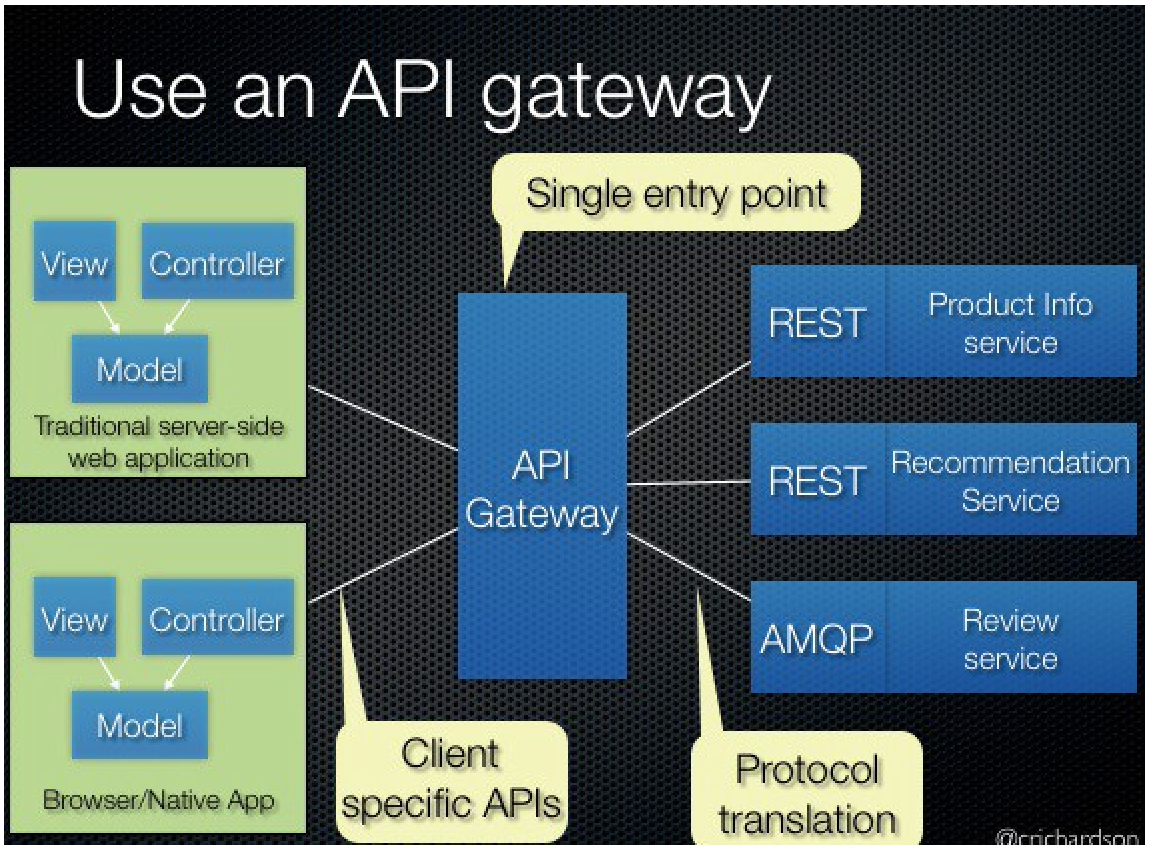
Zookeeper : Provides configuration information, naming, synchronization and group services over large clusters in distributed systems.

*Used to manage cluster of servers, databases or caches. External services can interact with clusters via zookeeper. Zookeeper names services, knows their ips, elects leader, provides failure recovery.*

## API Gateway vs Service Mesh

Popular apps: Zuul, Amazon API Gateway. Read : My experiences with API gateway

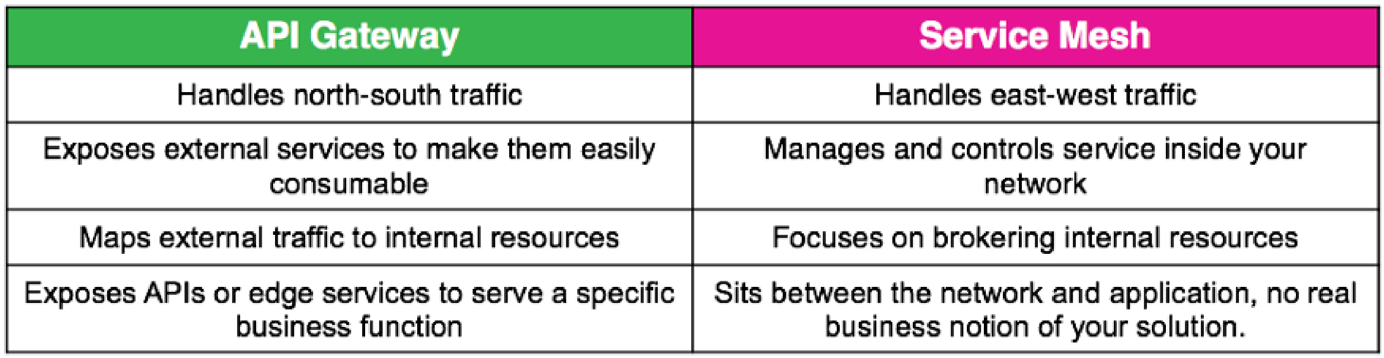
A service mesh's primary purpose is to manage internal service-to-service communication, while an API Gateway is primarily meant for external client-to-service communication. Read more.



API gateway pattern (microservices.io)

API Gateway provides a single entry point for a client for a number of different underlying APIs (system interfaces/web services/Rest APIs, Lambdas, etc.). Performs traffic management, authorization and access control, monitoring, and API version management.

API Gateway operates at Layer 7. Just like layer 7 load balancers it knows about content. Thus induces an overhead in extra hop at API gateway.



| API Gateway | Service Mesh |
|---|---|
| Handles north-south traffic | Handles east-west traffic |
| Exposes external services to make them easily consumable | Manages and controls service inside your network |
| Maps external traffic to internal resources | Focuses on brokering internal resources |
| Exposes APIs or edge services to serve a specific business function | Sits between the network and application, no real business notion of your solution. |

API Gateway vs. Service Mesh — DZone Microservices

Service Mesh (Popular example Istio) operates at layer 4. Provides inter service communication. Works like a sidecar to services. Handles service discovery, circuit breaking, timeouts, retries, encryption, tracebility, authentication and authorization among services.

### CDN

Content Delivery Networks (Akamai, Cloudfare etc) : Mostly store static data which can be pushed by servers or CDN can pull from servers on a miss or at speculated times. Globally distributed so stays near clients and is fast.

> *Use when you have some static data to serve near client, like images of restaurants/food on yelp, telephone directory*

### How to Scale database

Reference: Understanding database scaling patterns | by Kousik Nath | Medium

Brief: Query optimization -> vertical scaling -> Master slave -> multi master -> partitioning -> sharding -> multi-datacenter replication

### Cassandra

1. Wide Column NoSQL
2. Tunable consistency -> read by qourum. type of quorum defines consistency level
3. Fast wright -> Writes in log sequentially. Tombstoning and Compation in backend.
4. A KKV value storage [Like HashMap(partition) of HashMap(Rows or clusters)]: First K is the partition key and is used to determine which node the data lives on and where it is found on disk. The partition contains *multiple rows* within it . Second K (clustering key) finds the row within a partition.
   The clustering key (Second K) acts as both a primary key within the partition and how the rows are sorted. You can think of a partition as an ordered dictionary.

See how Discord used it to scale their storage. They made KK in KKV as `((channel_id, bucket), message_id)` . Message id was a snowflake id.

### Snowflake at Twitter

Generates tens of thousands of ids per second in a highly available manner and is 64 bits (UUID is 128)

These ids need to be *roughly sortable*, meaning that if tweets A and B are posted around the same time, they should have ids in close proximity to one another since this is how most Twitter clients sort tweets.

To generate the roughly-sorted 64 bit ids in an uncoordinated manner, ids are generated as composition of: **timestamp, worker number and sequence number.**

Sequence numbers are per-thread and worker numbers are chosen at startup via zookeeper.

### Some numbers

**Availability** : 99.99% ~ 50 min downtime/year | 99.999 % ~ 5min downtime /year | 99.9999% ~= 30seconds downtime/year

**Bandwidth** : Average EC2 instance is 5Gb/s

**Requests/second :** On average 1 server can process 1000 requests/second

**Max Websocket Connection**: 65k (65,536) socket connections i.e max number of TCP ports available. *Each server can handle 65,536 sockets per single IP address. So the quantity can be easily extended by adding additional network interfaces to a server. (HaProxy, ELB does TCP balancing as well)*

Low end dedicated MySQL server (2 cores, 4 GB RAM) can serve 100 requests/sec on an average ~10 million/day with CPU idle rate of 90%.

### Some More Numbers

***Approximation : 1 day ~ $10^4$ seconds → 1 billion daily active users => $10^8/10^4$ =$10^5$ = 10k requests/sec***

**Daily active users :** Twitter: ~200M/day (200/sec), Facebook: ~2Bn (2000/sec), Whatsapp: ~200M, Netflix: 200M/day

**Photos uploaded:** 200M/day (200/sec)(Instagram)

**Videos uploaded:** 500 hours/minute (Youtube)

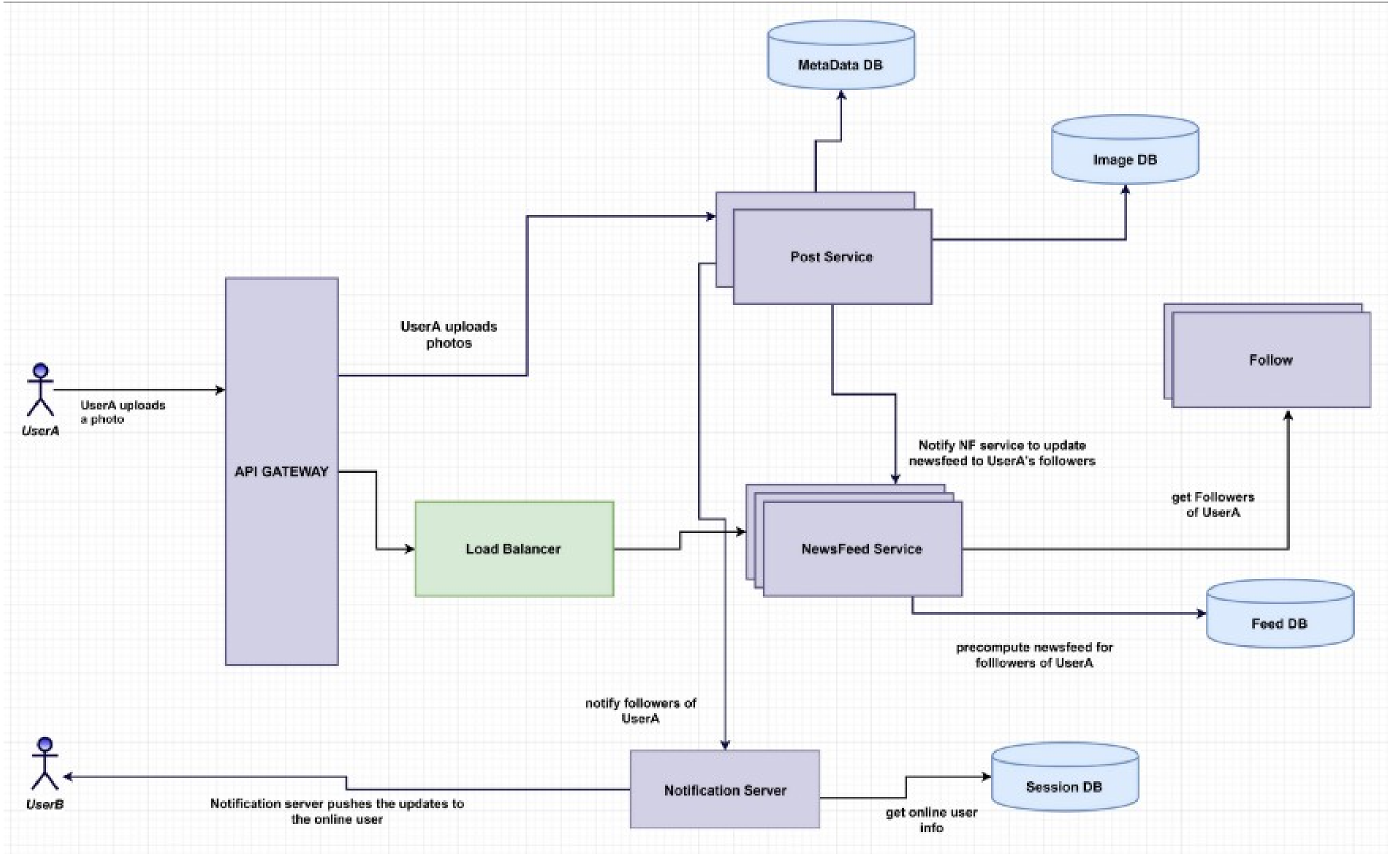
**Uber:** 20 million trips/day (20*$10^6$ / day => 2000/sec)

**Bandwidth :** Say 200 reads/second, each read needs 10kb data ($10^4$ characters ) 2000kb/sec => 2MB/sec

## ……………….Architecture Designs…………………..

### Instagram

Reference: System Design Analysis of Instagram

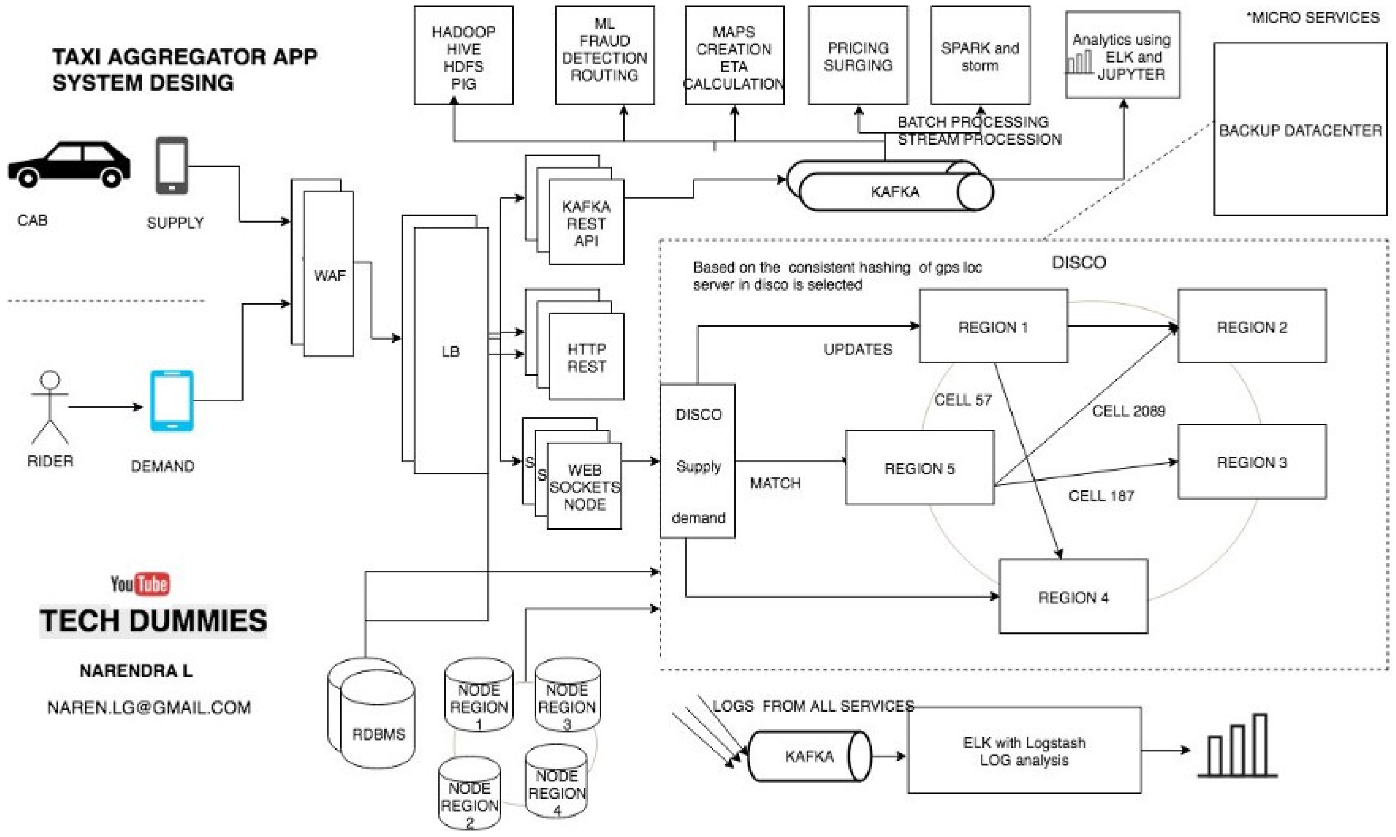


- Precompute : User A posts “x”. User B,C,D follows. Then “x” can be appended to User B,C,D news feed inside cache. This is called fanning out. In hybrid approach, if A is a celebrity, then its post is not fanned out ( 1 post need to be written to millions cache).
- Notification server needs to maintain a persistent websocket connection to push data to User B
- Place Metadata service to : provide separation of concern, access to DB via API, also can act as a caching layer, metadata can be accessed via service cache rather than db.

## Uber

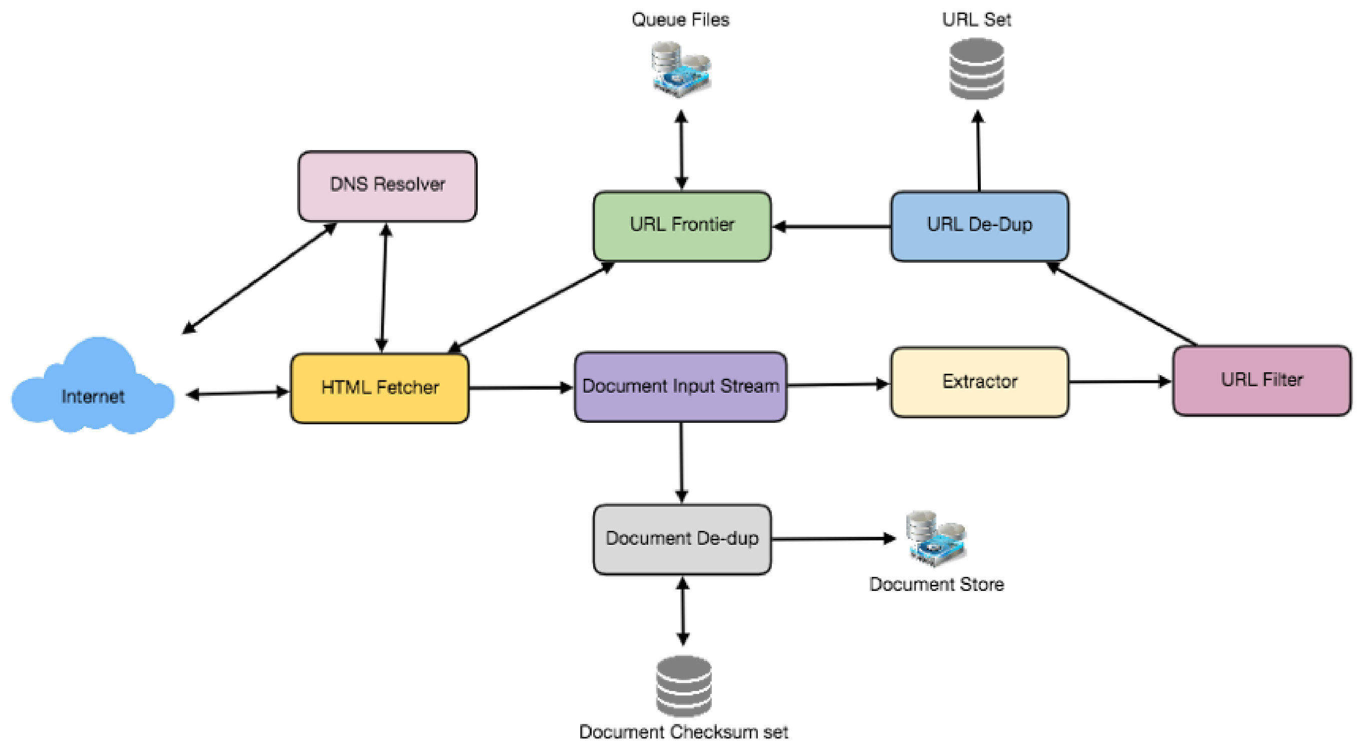
Reference: UBER System design

Best Article so far: A must read



## Web Crawler

Reference: RoadtoArchitect



https://www.educative.io/collection/page/5668639101419520/5649050225344512/5718998062727168

Note:

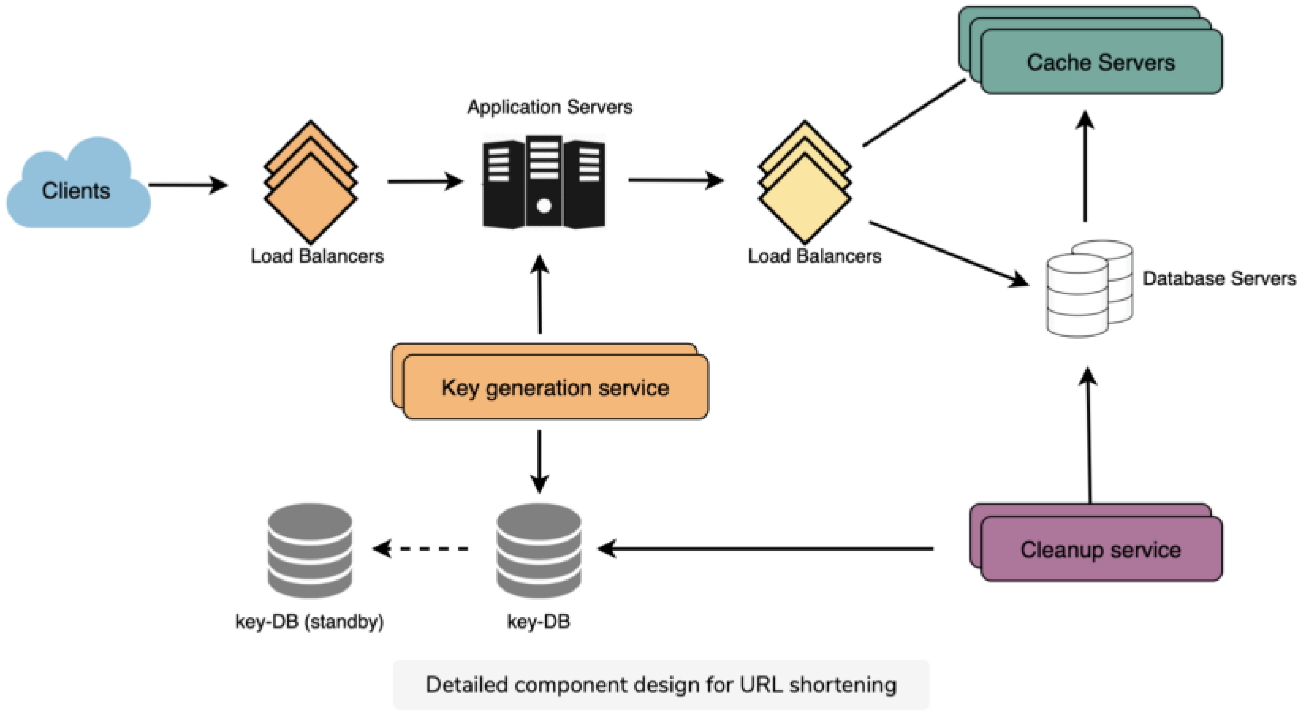
Get document, remove duplicate documents using checkum, extract URLs in page, remove duplicate urls, feed to the queue to parse.

Fetcher will map host name to robots.txt values.

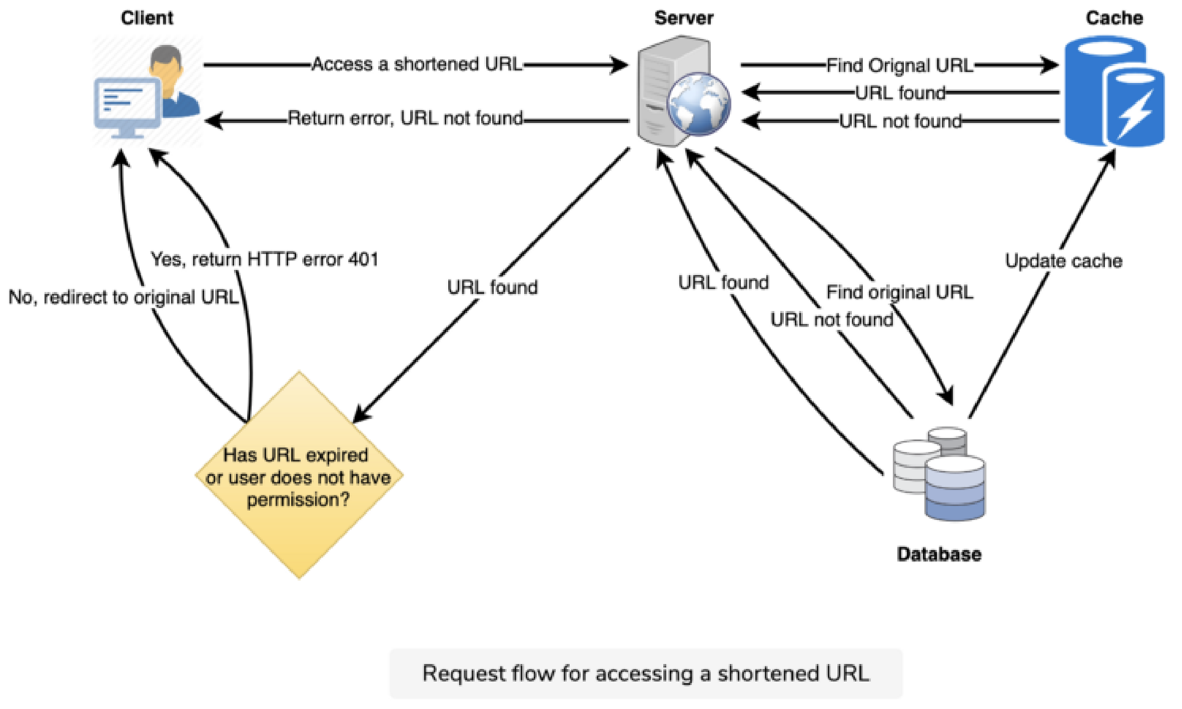
**Domain name resolution:** Before contacting a Web server, a Web crawler must use the Domain Name Service (DNS) to map the Web server's hostname into an IP address. DNS name resolution will be a big bottleneck of our crawlers given the amount of URLs we will be working with. To avoid repeated requests, we can start caching DNS results by building our local DNS server.

## URL Shortner

Reference: Educative



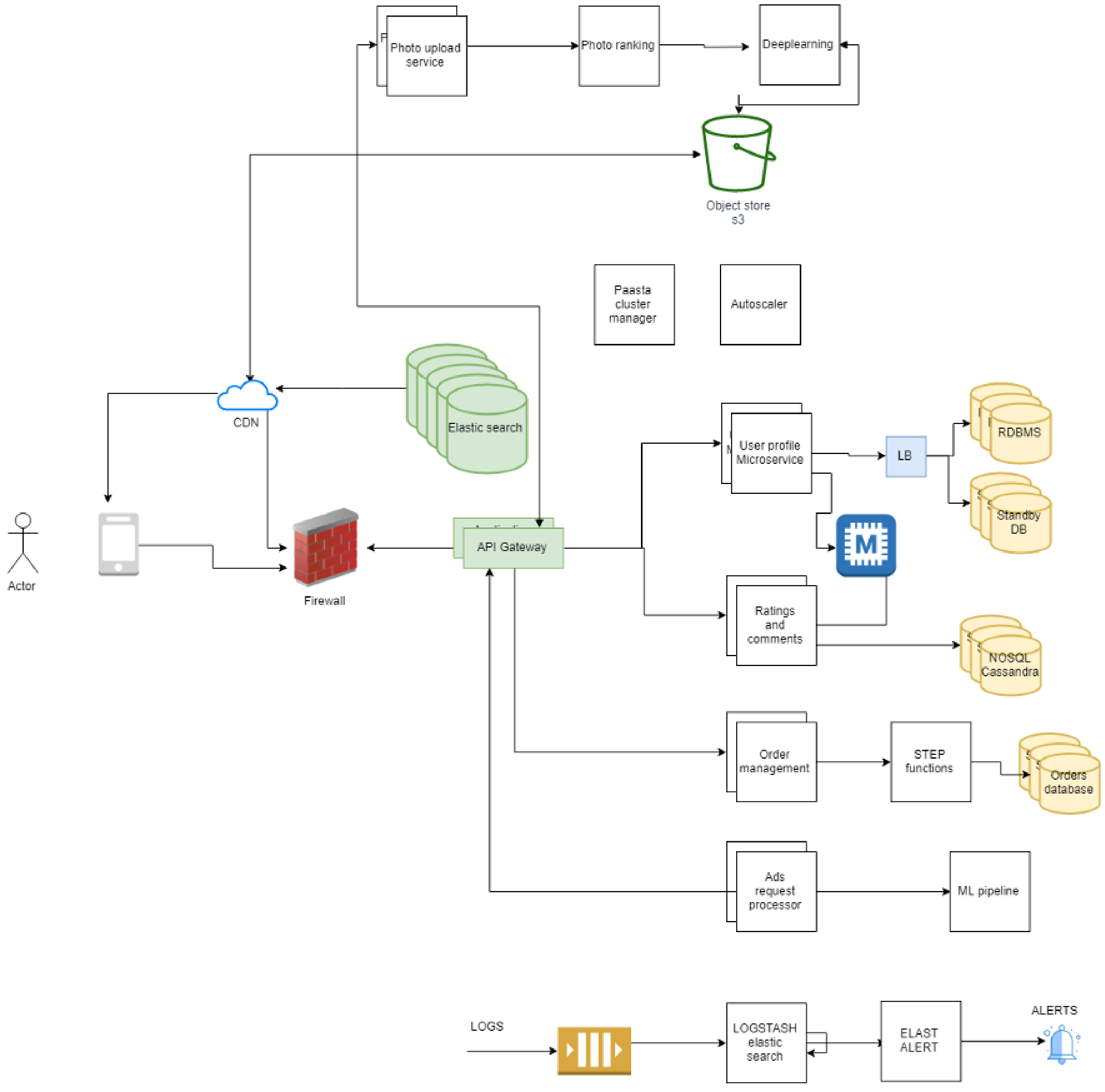
Detailed components


Client
Server
Cache
Access a shortened URL
Return error, URL not found
Find Orignal URL
URL found
URL not found
Yes, return HTTP error 401
No, redirect to original URL
URL found
URL found
Find original URL
URL not found
Update cache
Has URL expired or user does not have permission?
Database
Request flow for accessing a shortened URL

Key point here is a separate key generation service and cleanup service

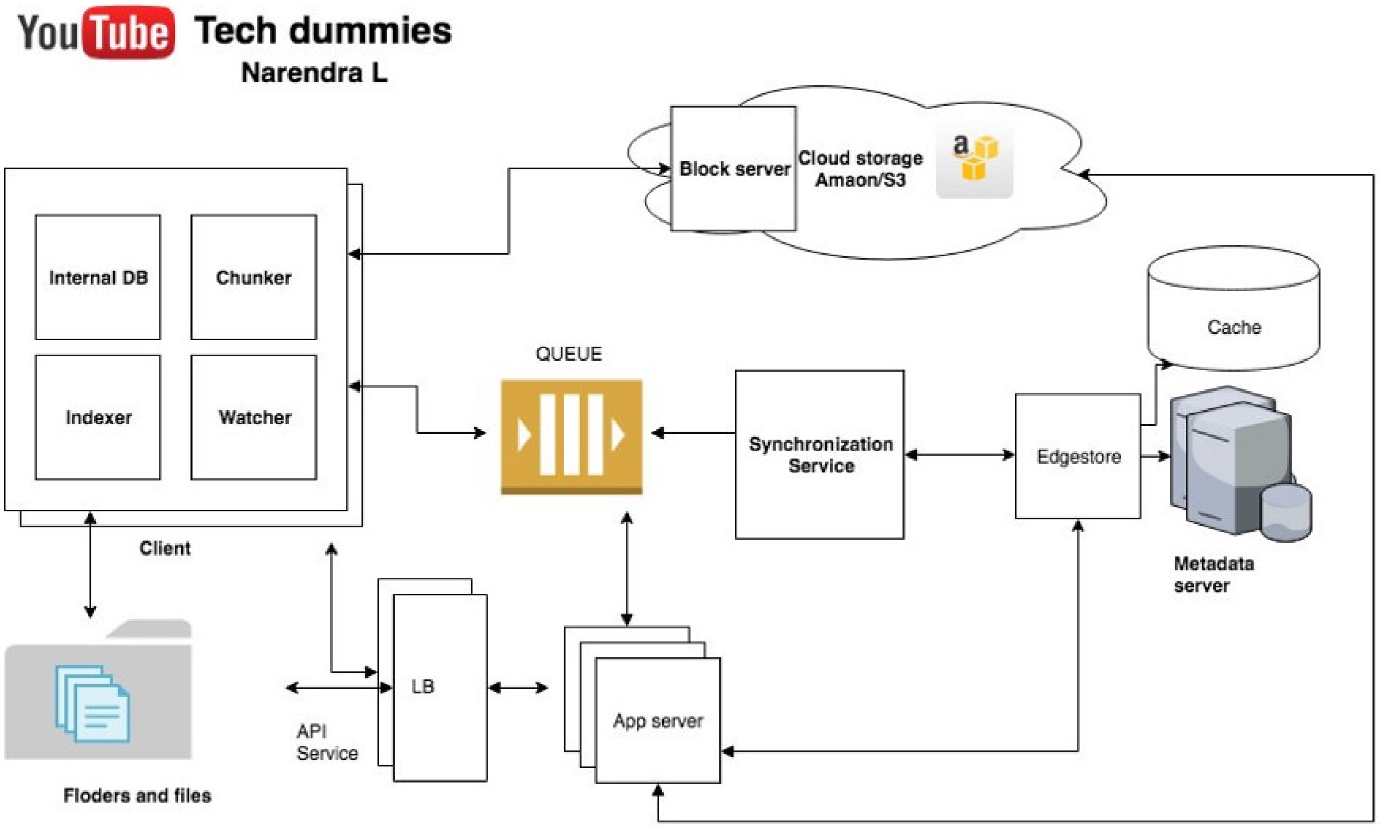
## Yelp

Reference: Tech Dummies



## Dropbox

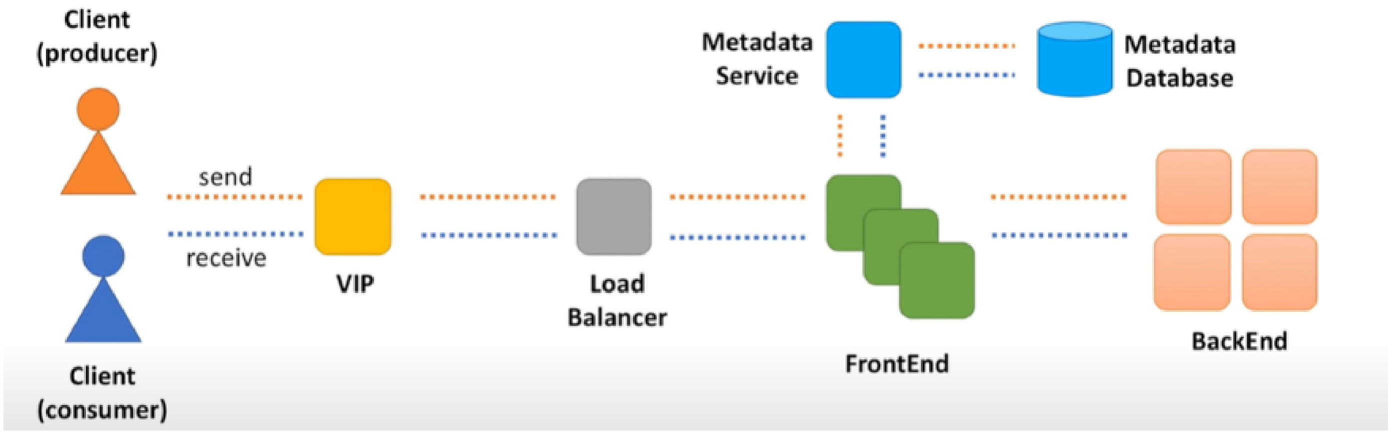
Resource: Dropbox system design



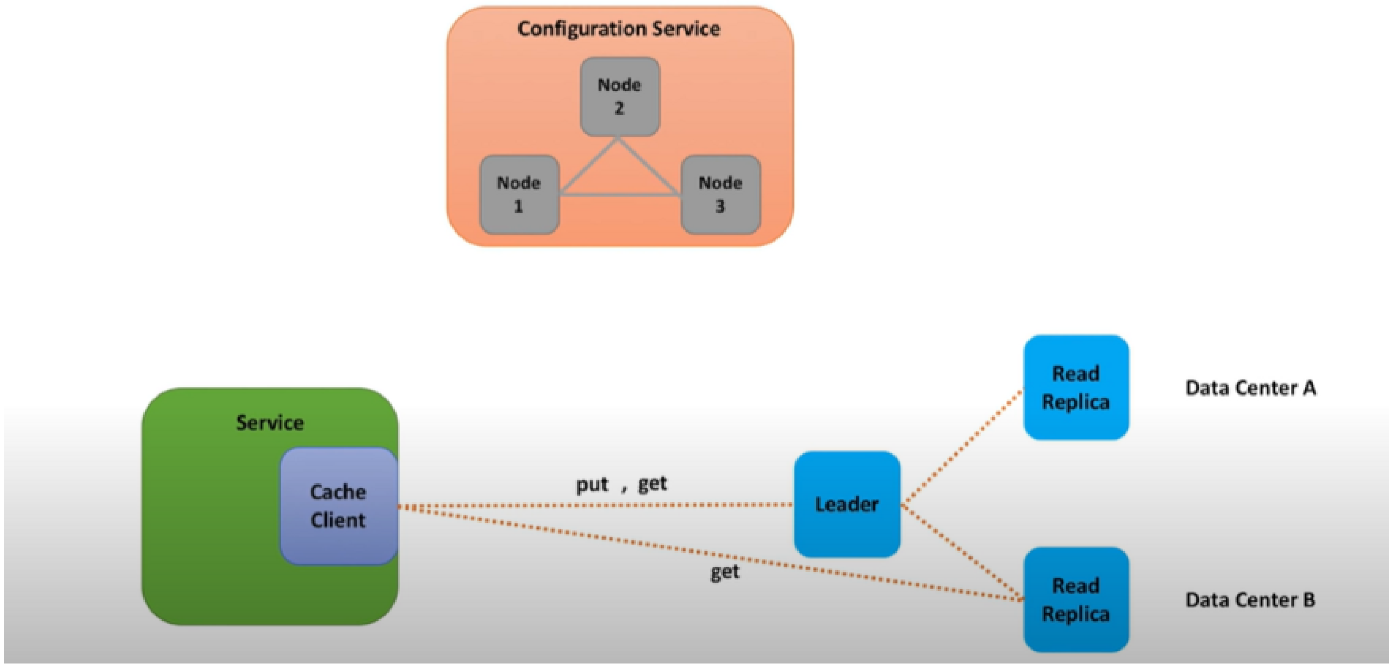
File share/storage System design

## Distributed Message Queue

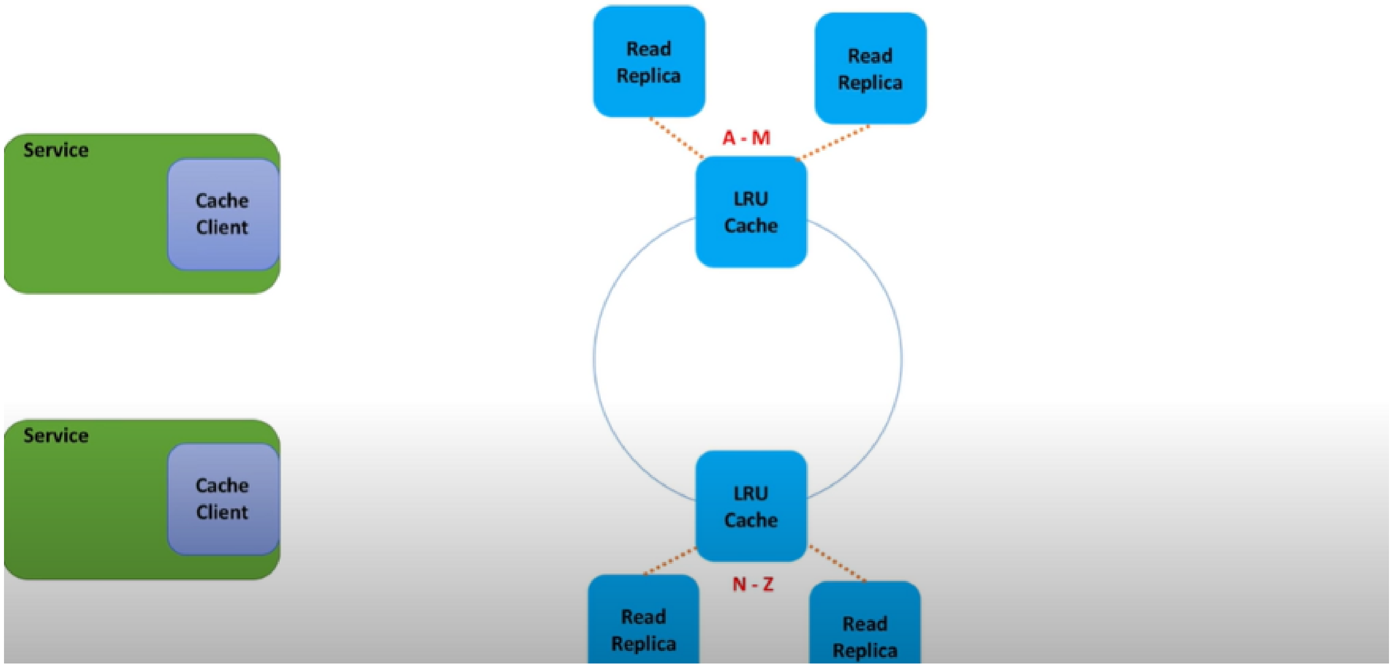
Resource : System Design Interview



## Distributed Cache



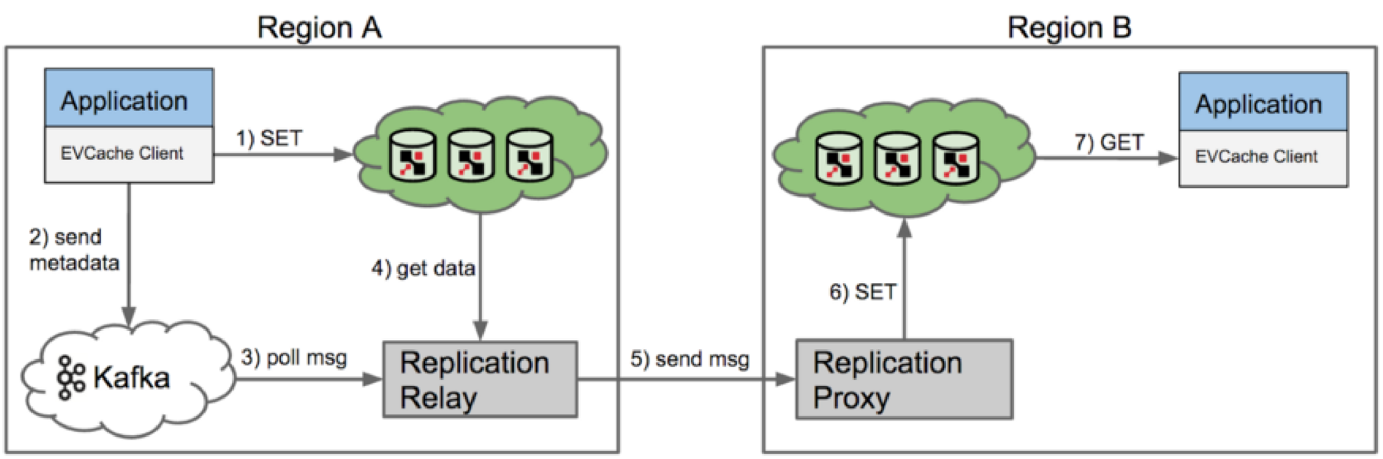
Unsharded



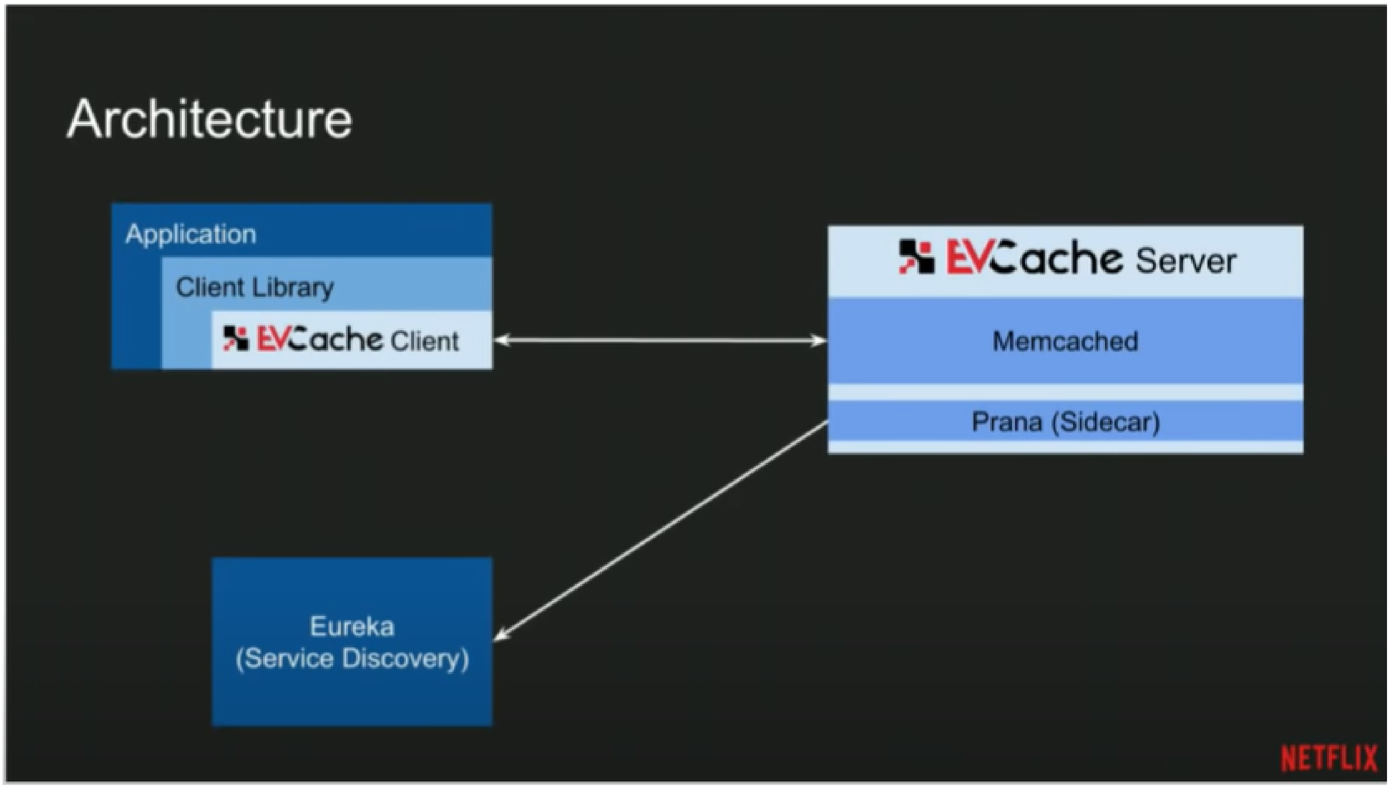
Sharded

## Distributed Cache at Netflix

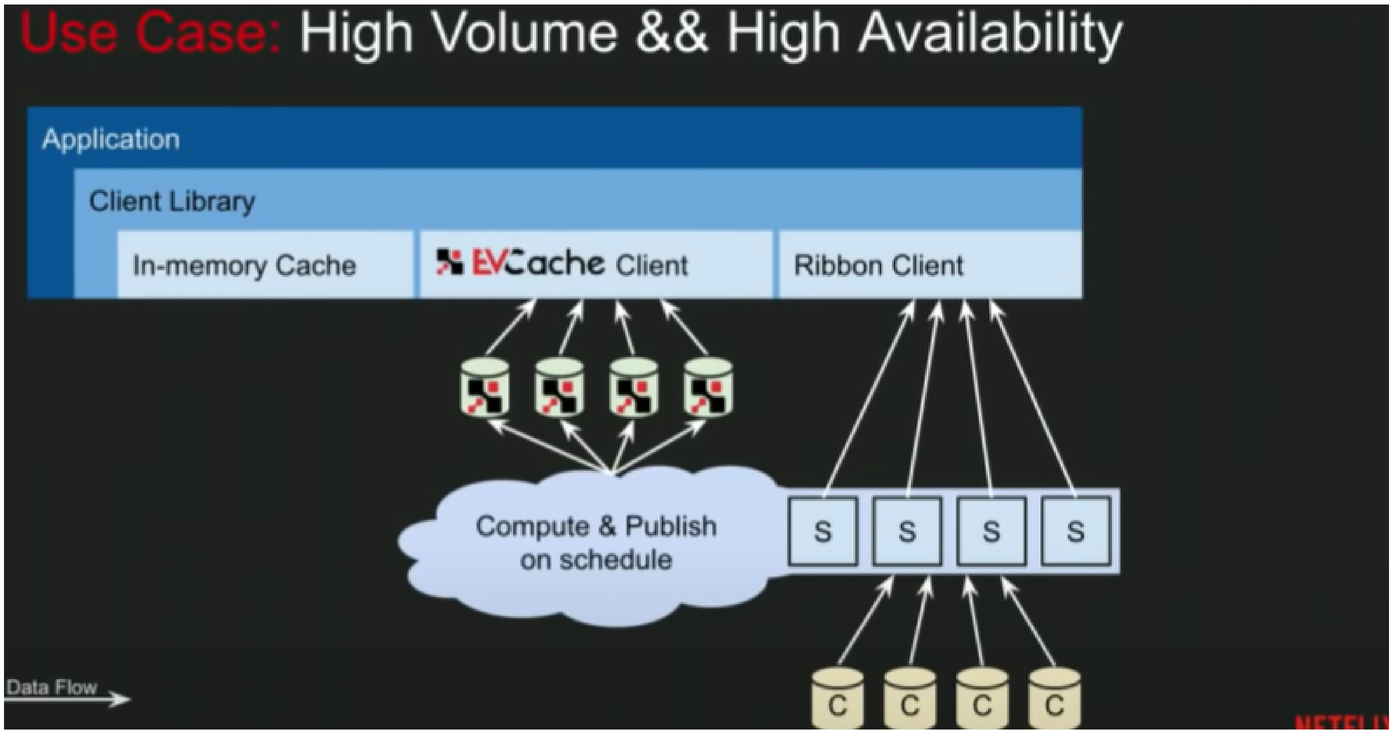
Reference: Netflix



Netflix Implementation



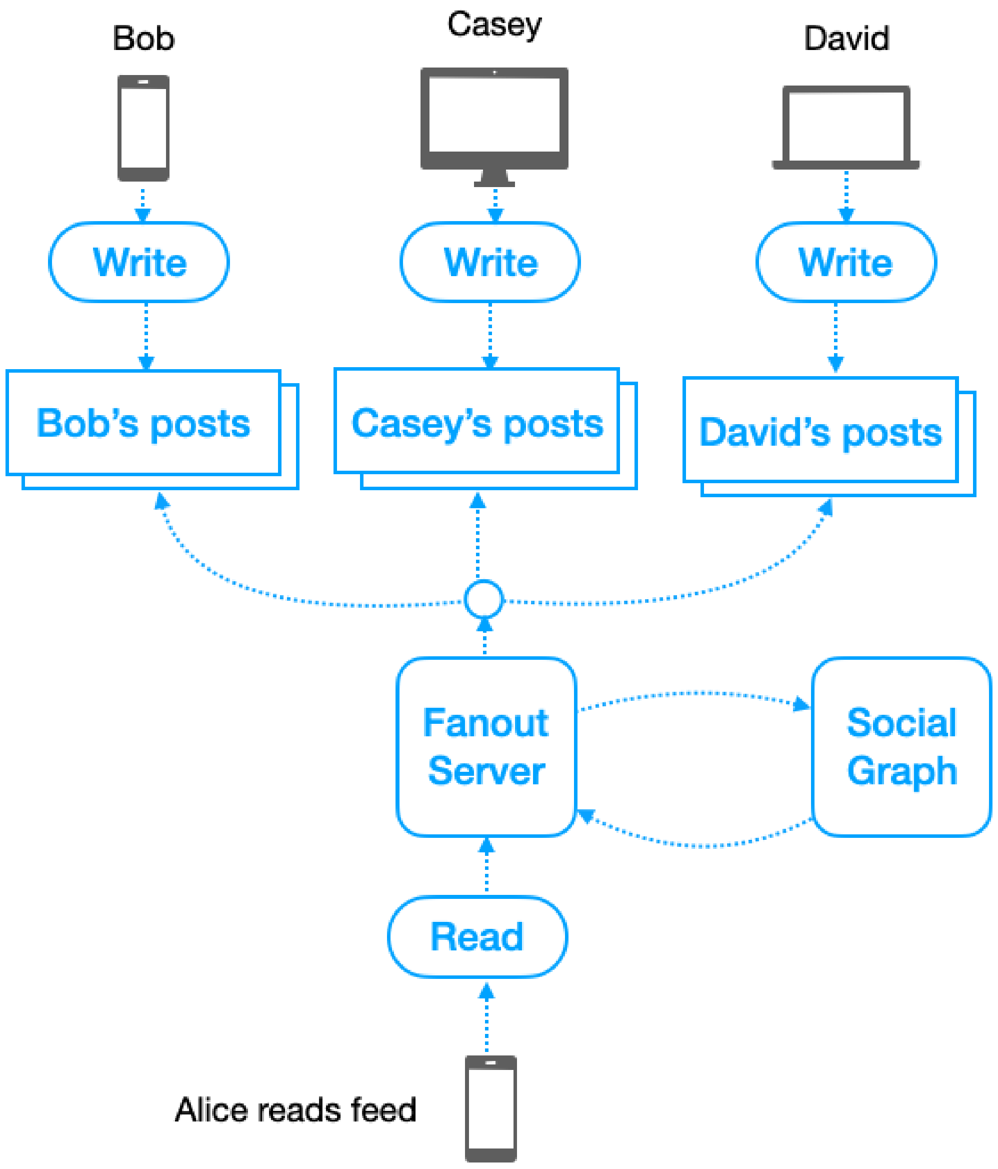
Level 1

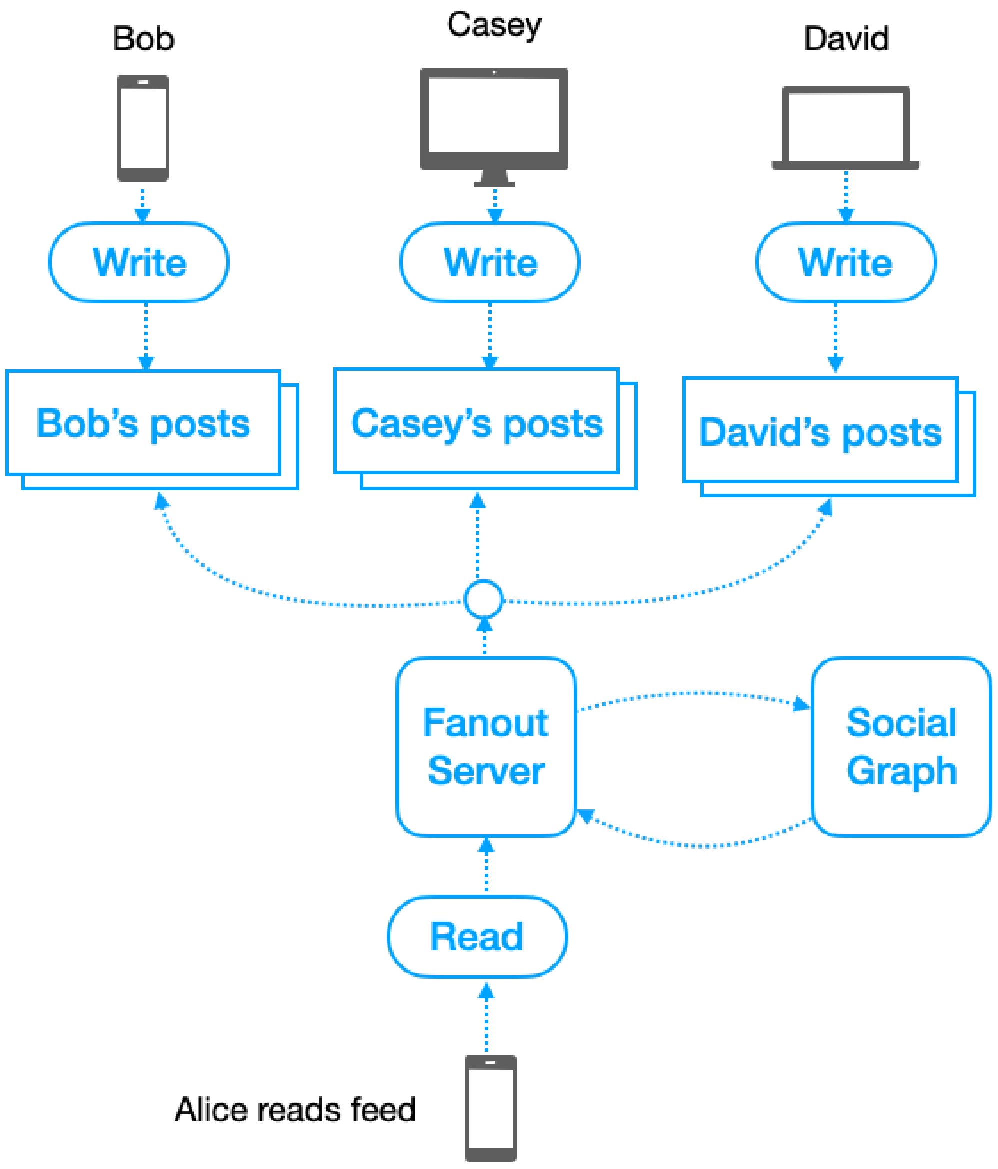


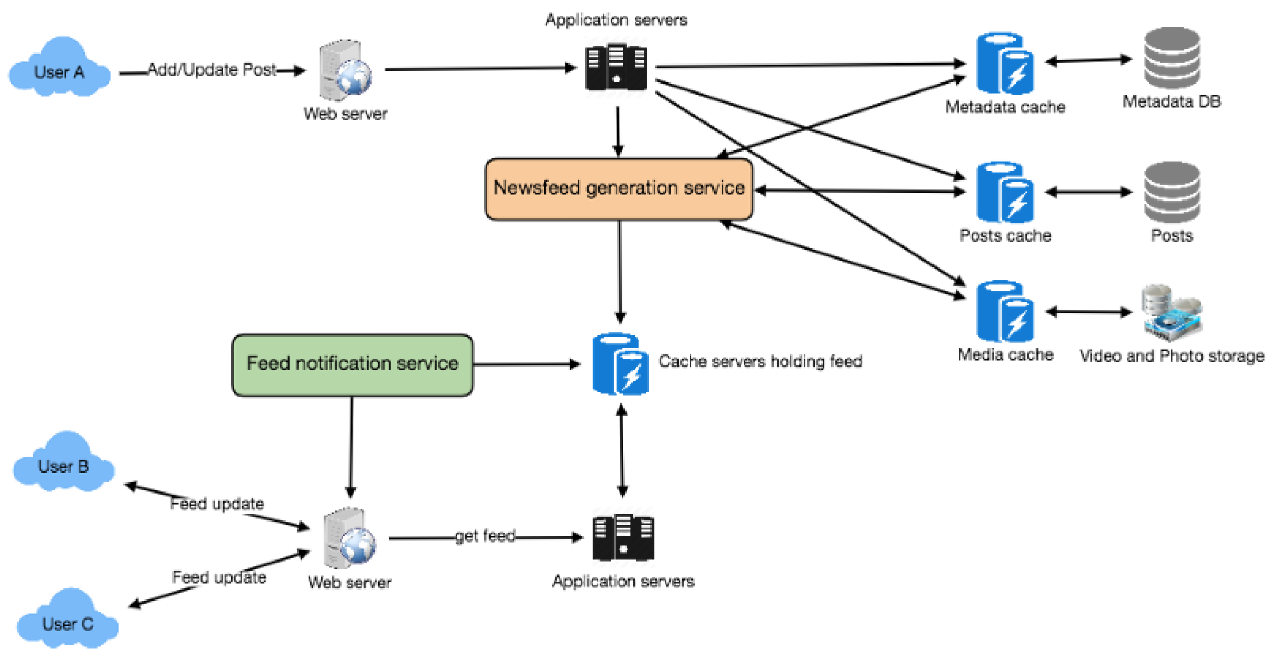
Level 2

## News Feed Generation Service

Source: Design a News Feed System — System Design (gitbook.io)


Bob
Casey
David
Write
Write
Write
Bob's posts
Casey's posts
David's posts
Fanout Server
Social Graph
Read
Alice reads feed


Bob
Casey
David
Write
Write
Write
Bob's posts
Casey's posts
David's posts
Fanout Server
Social Graph
Read
Alice reads feed



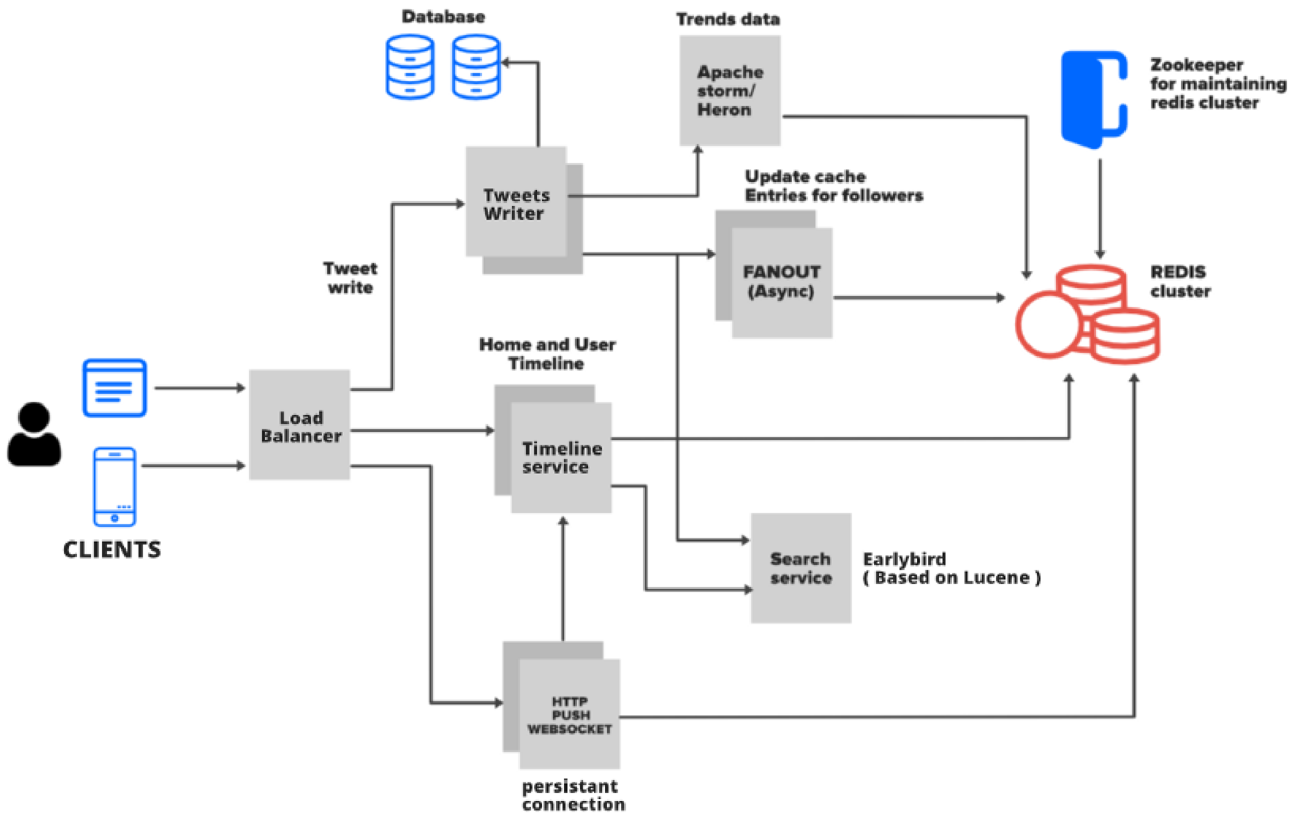
Notes:

getFeed(UserId, pageNumber, maxPerPage)

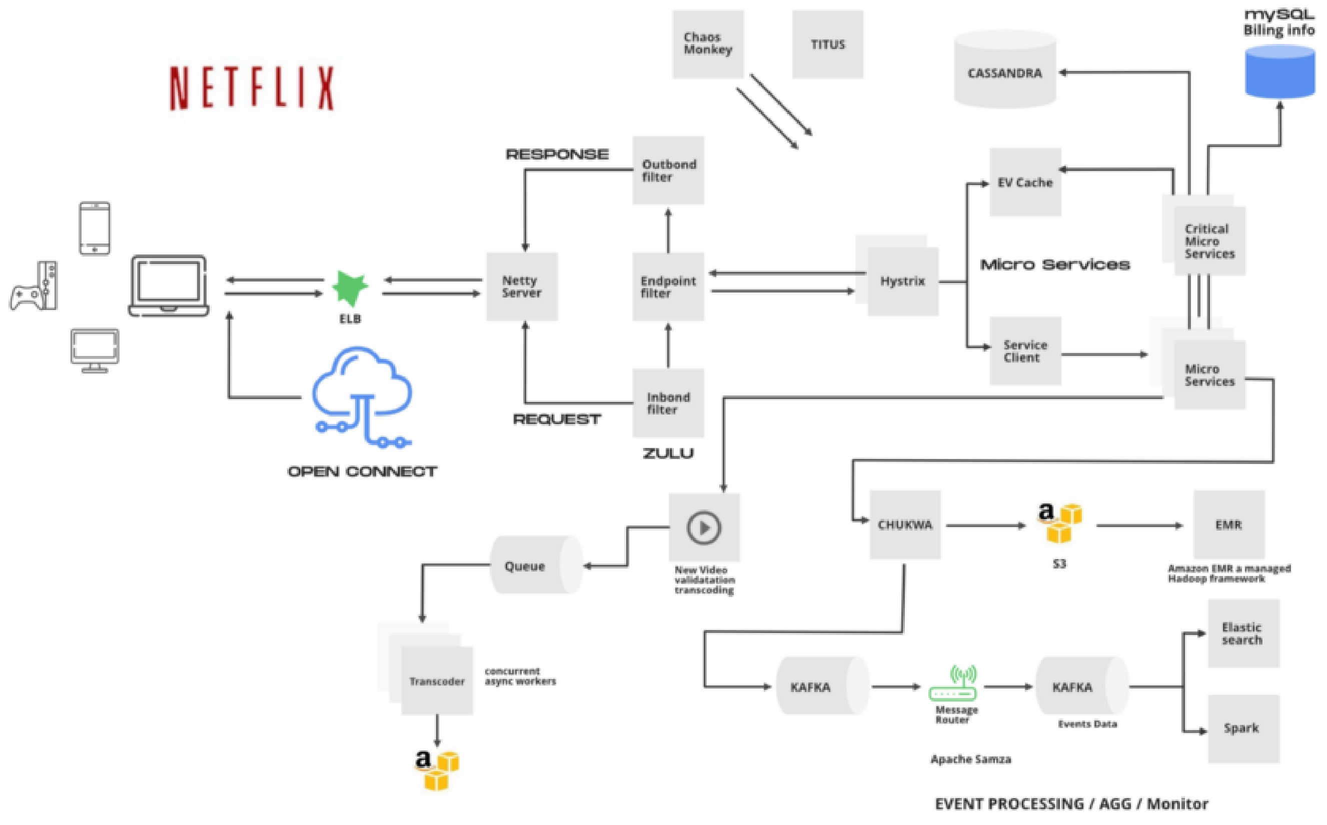
DB design : User table, Entity Table ( has entity id and post ), Post table (has media), Media table, User Followers( followers are entities or users),

## Twitter

Reference: Tech Dummies


Database
Trends data
Apache storm/ Heron
Zookeeper for maintaining redis cluster
Tweets Writer
Update cache Entries for followers
FANOUT (Async)
REDIS cluster
Tweet write
Home and User Timeline
Load Balancer
Timeline service
CLIENTS
Search service
Earlybird ( Based on Lucene )
HTTP PUSH WEBSOCKET
persistant connection

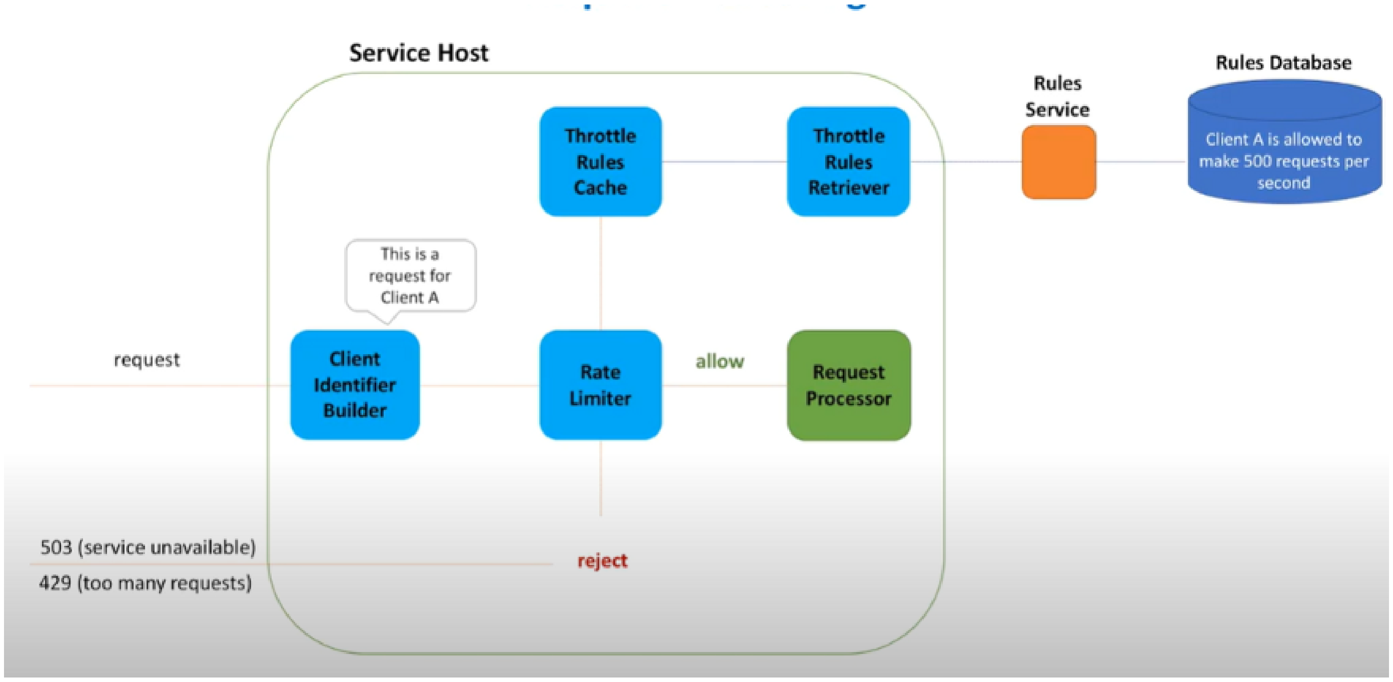
## Netflix

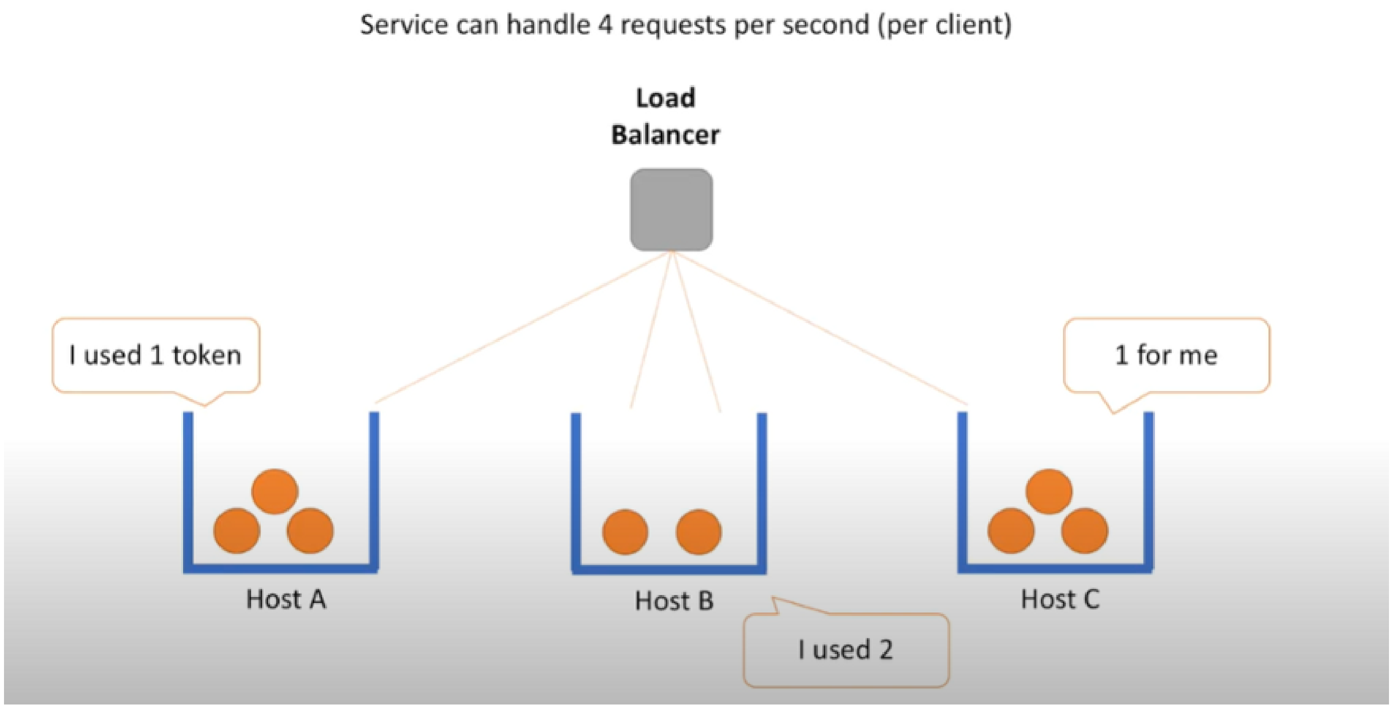


## Distributed Rate Limiter

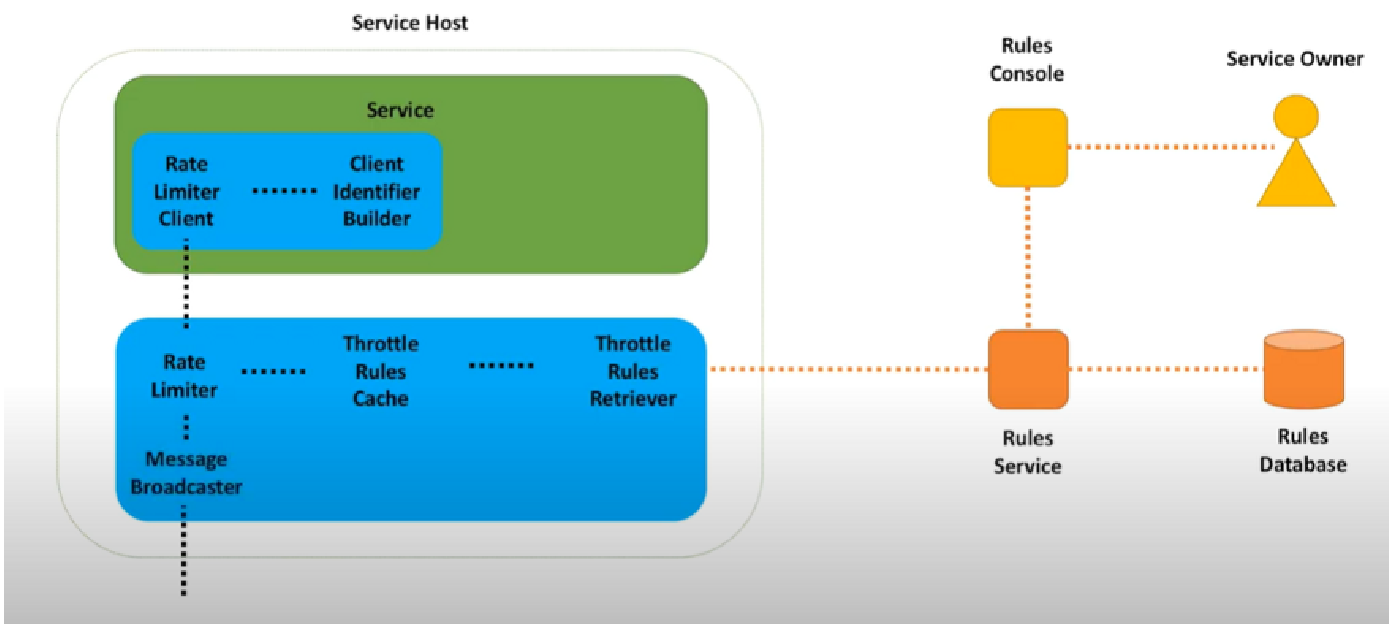
Resources: System design Interview

- Client Identifier Builder assigns unique id/client to get originator.
- Rate limiter coordinates with throttler and processor to pass or reject.
- Throttling Service implements token bucket, sliding window using requests stored in rules db.



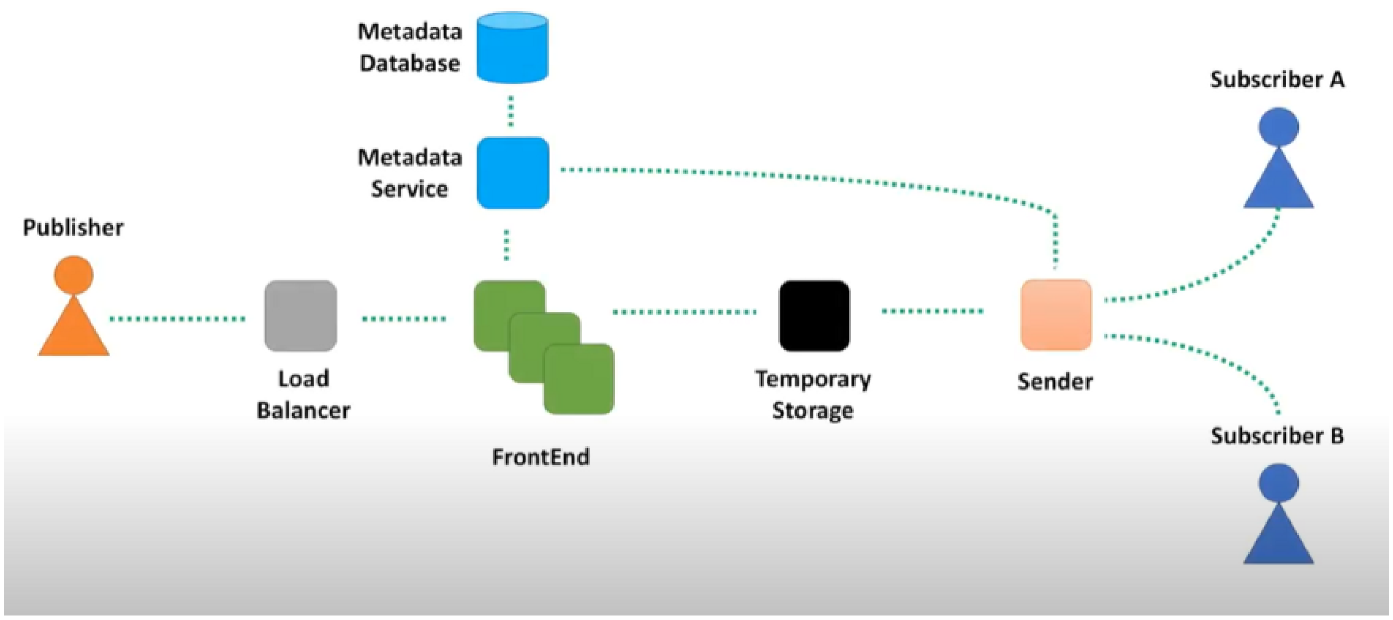


Token Bucket algorithm

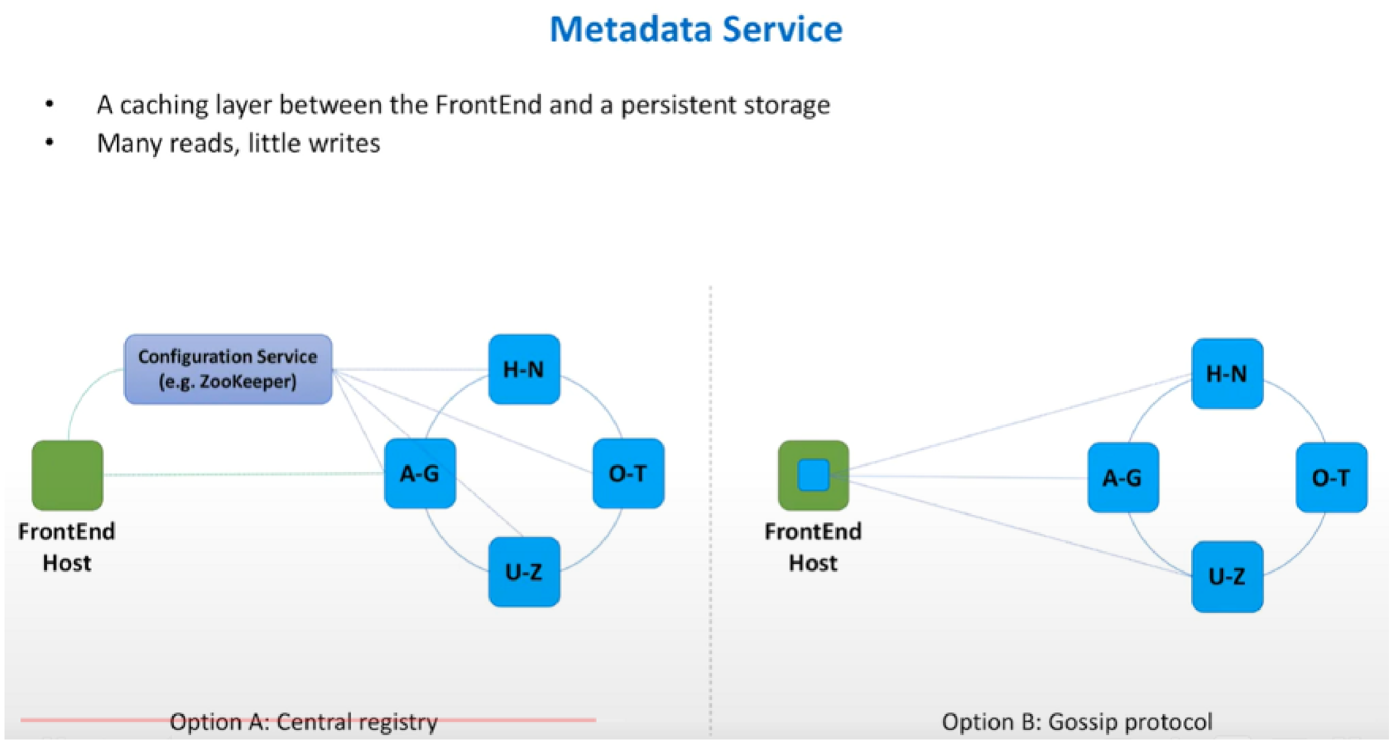


## Notification Service

Resources: System design Interview . Kind of like building RabbitMQ



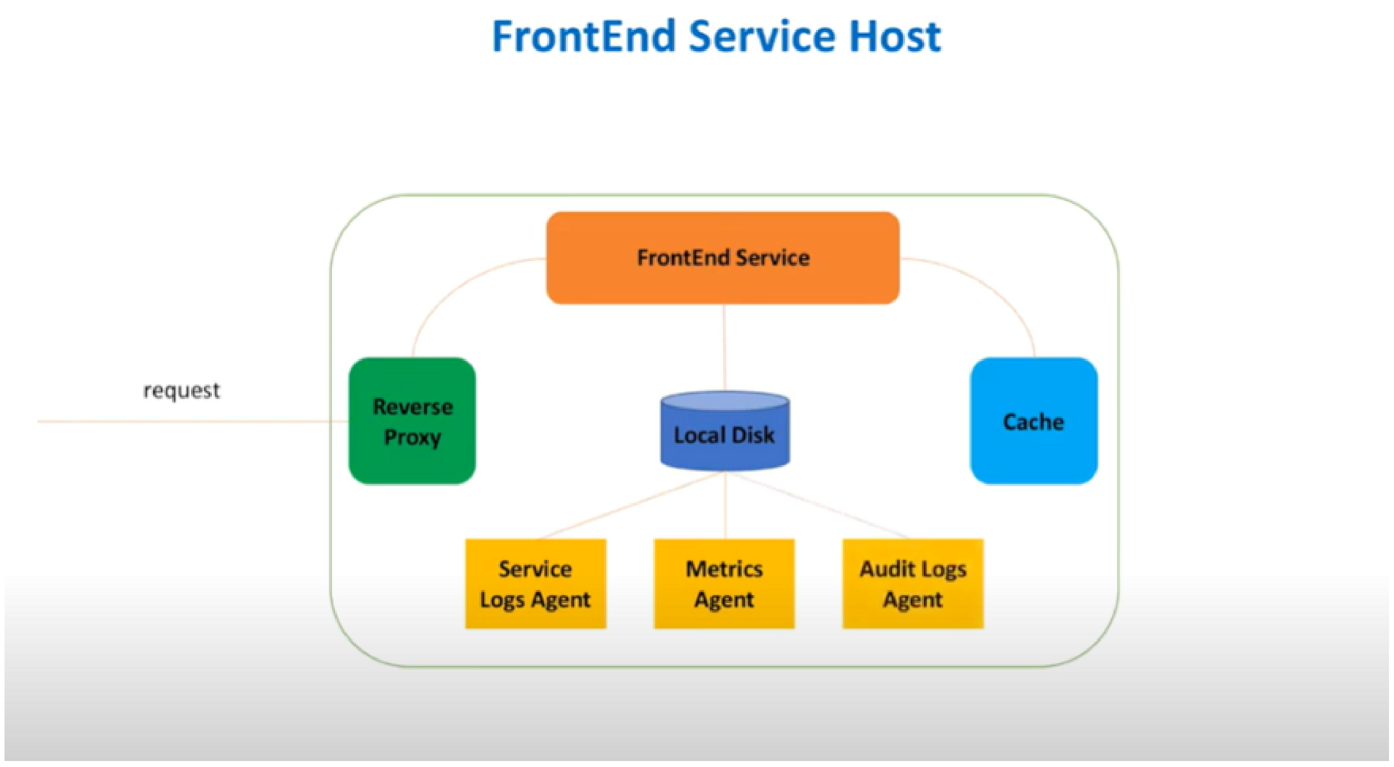
- Metadata service for caching and separation of concern.
- Temporary storage to store message for a while
- Sender reads data from Metadata service to know which message to send to whom.



## Metadata Service

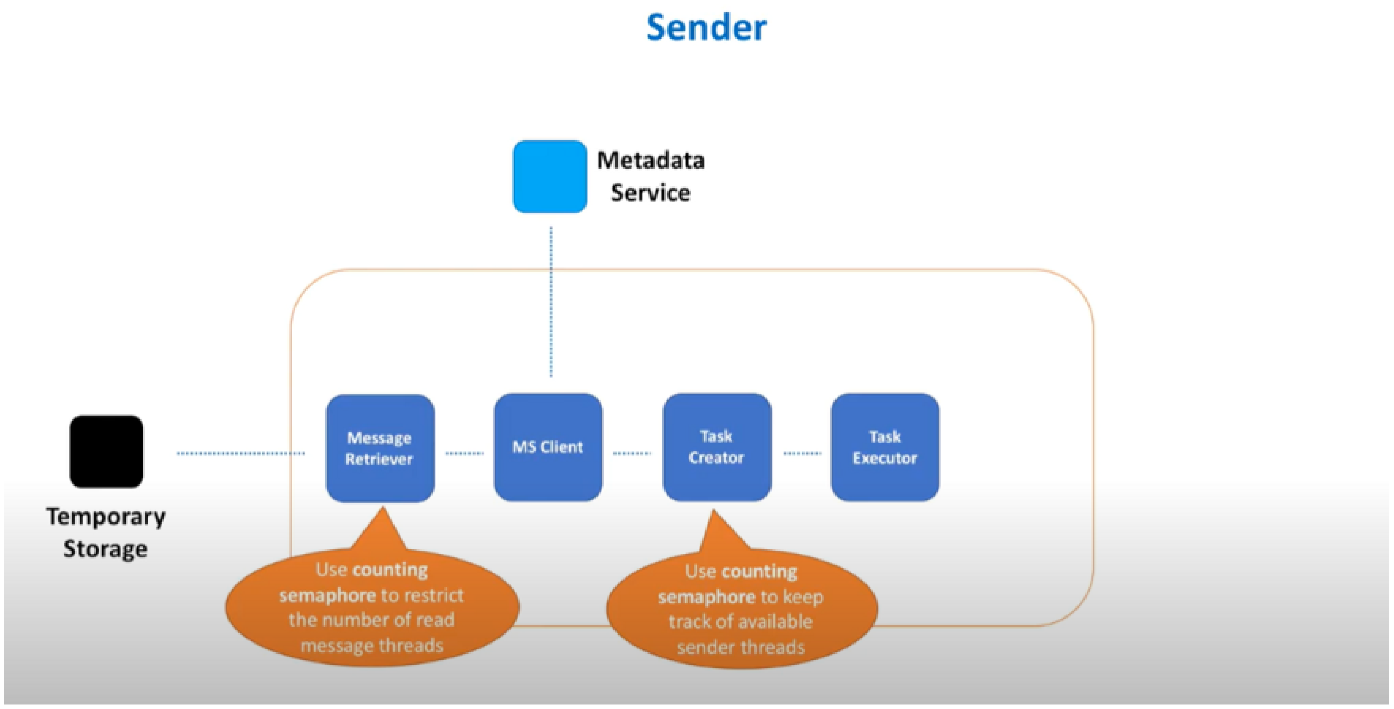
- A caching layer between the FrontEnd and a persistent storage
- Many reads, little writes

Configuration Service (e.g. ZooKeeper)
H-N
A-G
O-T
U-Z
FrontEnd Host
Option A: Central registry
H-N
A-G
O-T
U-Z
FrontEnd Host
Option B: Gossip protocol



## FrontEnd Service Host

Frontend Service Host caches the metadata

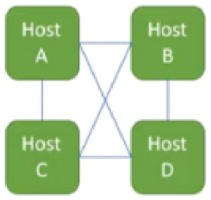

Sender
Metadata Service
Temporary Storage
Message Retriever
MS Client
Task Creator
Task Executor
Use counting semaphore to restrict the number of read message threads
Use counting semaphore to keep track of available sender threads

Temporary storage can be NoSQL, Key Value (Redis) or Column Based (Cassandra) or streaming service (kafka)

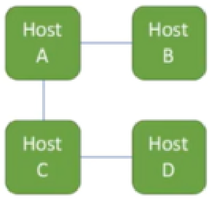
## Message sharing within cluster

Resources: System design Interview

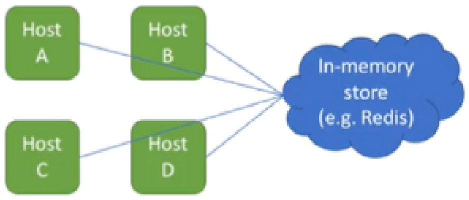
Tell everyone everything



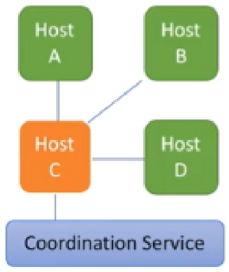
Gossip communication



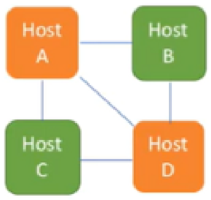
Distributed cache



Coordination service



Random leader selection


Host A
Host B
Host C
Host D