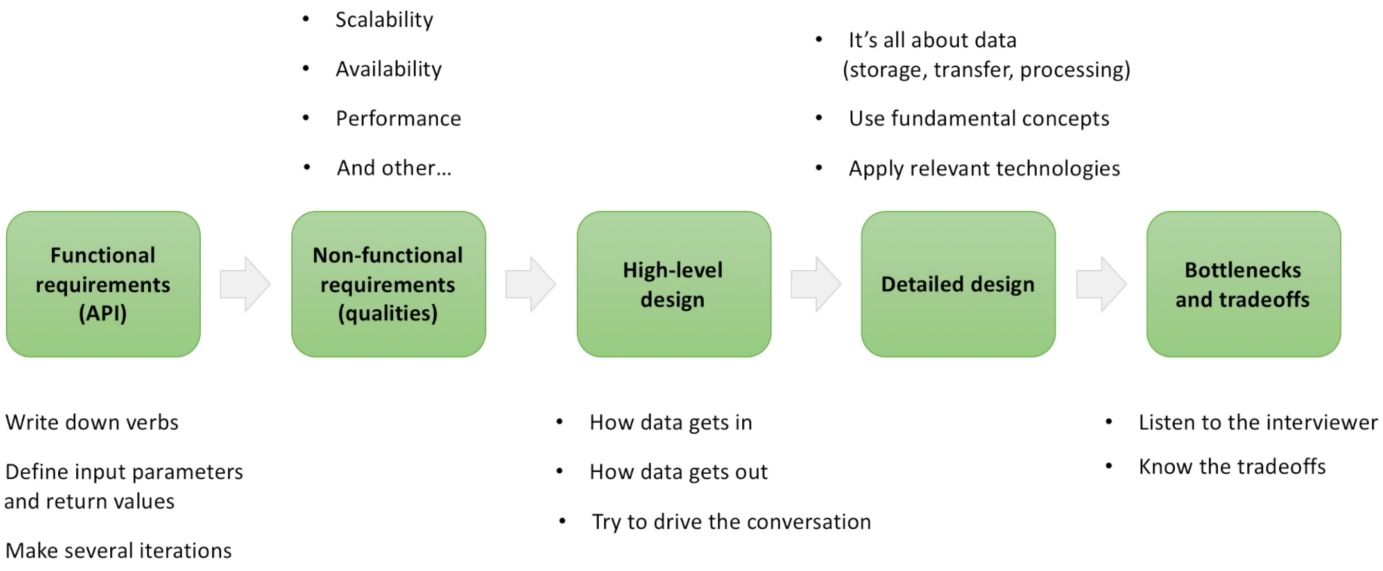
How to - Video Count View

Reference

https://www.youtube.com/watch?v=bUHFg8CZFws&ab_channel=SystemDesignInterview



Requirement clarification

- User/Customers
  - Who will use the system
  - How the system will be used
- Scale (read/write)
  - how many read read queries per second
  - how much data is queried per request
  - how many video views are processed per second
  - can there be a spike in traffic
- Performance
  - what os expected write-to-read data delay
  - what is expected p99 latency for read queries
- Cost
  - should the design minimize the cost of development
  - should the design minimize the cost of maintenance

Functional requirements - API

- system has to count video view events
  - countViewEvent(videoId)
  - countEvent(video, eventType) -> eventType={view, likes,share}
  - processEvent(videoId, eventType, function) --> function={ count, sum,...)

- system has to return video views count for a time period
  - getViewCount(videoId, startTime, endTime)
  - getCount(videoId, eventType, startTime, endTime)
  - getStats((videoId, eventType, function))

Not-functional requirements

- scalable (tens of thousands of video views per second)
- highly performant (few tens of milliseconds to return total view count for a video)
- highly available (survive hardware/failure, no single point of failure)

Talk about CAP theorem

- choosing between availability and consistency
  - go for availability

What do we store

- Individual events (every click)
  - pros
    - fast writes
    - can slice and dice data however we need
    - can recalculate numbers if needed
  - cons
    - slow reads
    - costly for large scale
- aggregate data (per minute) in real-time
  - cons
    - fast reads
    - data is ready for decision making
  - pros
    - can only query data how it was aggregated
    - requires data aggregation pipeline
    - hard to fix errors

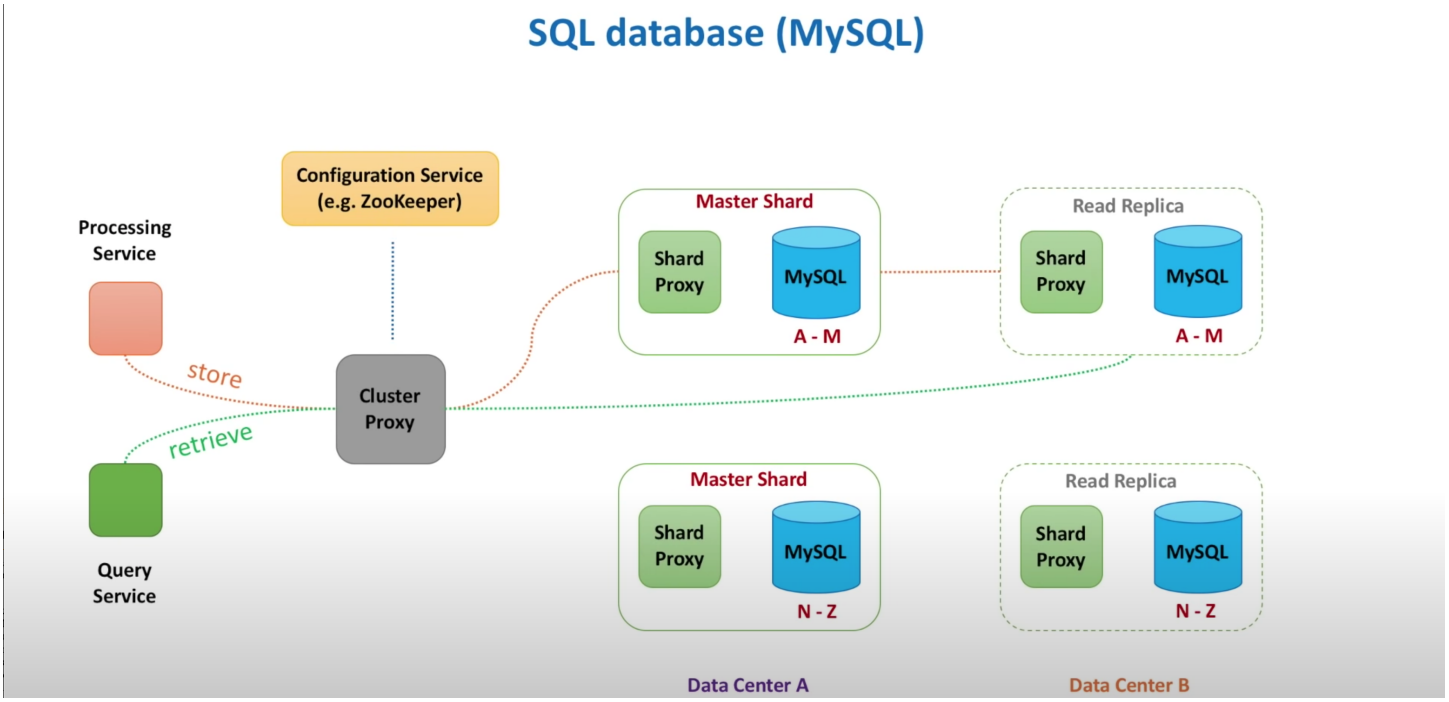
When do we process the data?

- in real time --> stream processing
- later --> batch processing

Requirements for data storage

- How to scale writes
- How to scale reads
- How to make both writes and reads fast
- How not to lose data in case of hardware faults and network partitions

- how to achieve strong consistency? hat are tradeoffs
- how to recover data in case of an outage
- how to make it extensible for are model changes in the future



- Zookeeper - configuration service to monitor each shard
  - gossip protocol - shards talk to each other

shard Proxy - https://docs.oracle.com/en/database/oracle/oracle-database/21/shard/how-database-requests-are-routed-shards.html#GUID-AD6433EC-91E3-4BC6-908F-B284A3F9CA45

Processing Service - min 32:00

Questions to ask

- how to scale
- how to achieve high throughput
- how not to lose data when node is crashing
- what to do when DB is slow or not available

To support scalability do

- partitioning

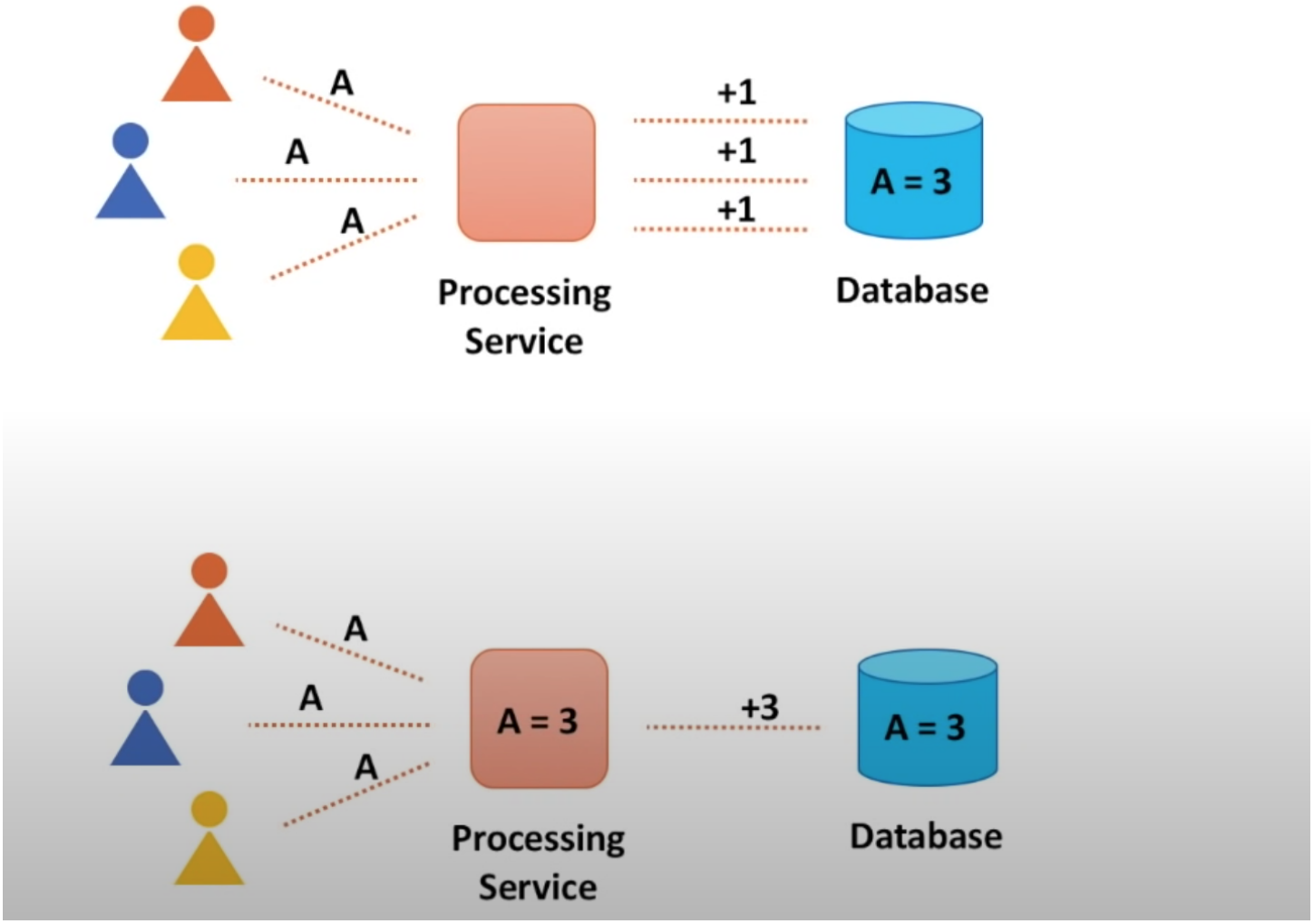
Relieable

- replication and checkpointing

Fast

- in memory

Where to aggregate data in the processing service (for example increment count)?



- increment in DB
- increment in processing service by accumulating data over a certain time window by incrementing in-memory counter. Then writes update to DB

Use push or pull?

- send some events from another service in a synchronous fashion
- pulls events from temporary storage

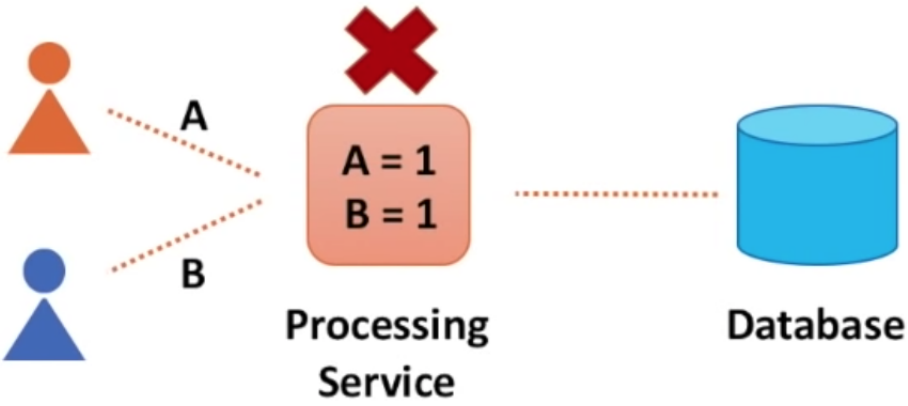
Pull with more advantages

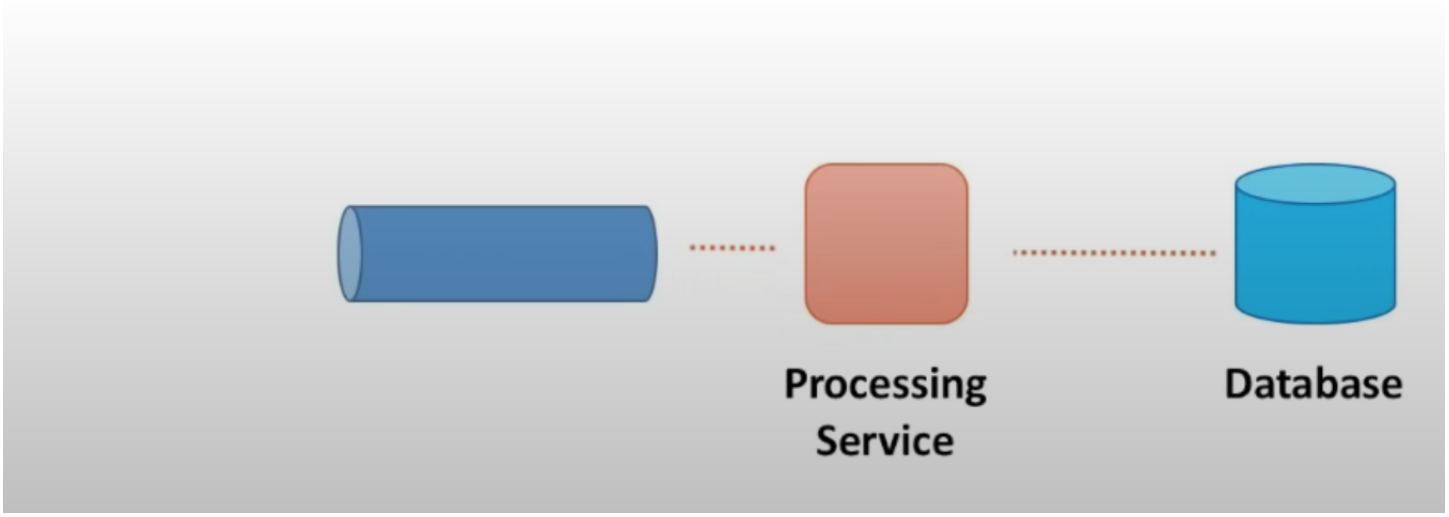
- better fault-tolerance
- easier to scale

Explanation

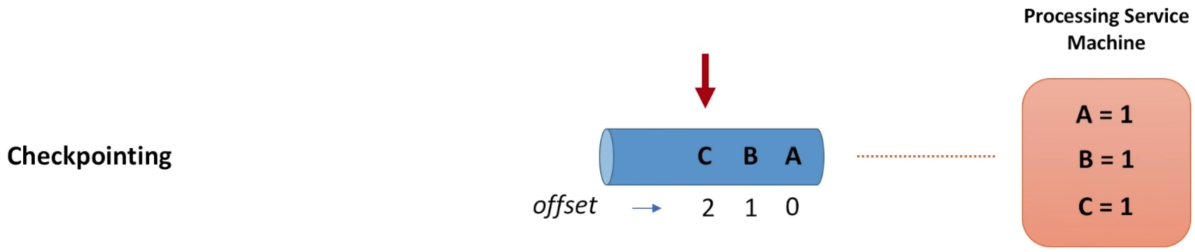
Explanation

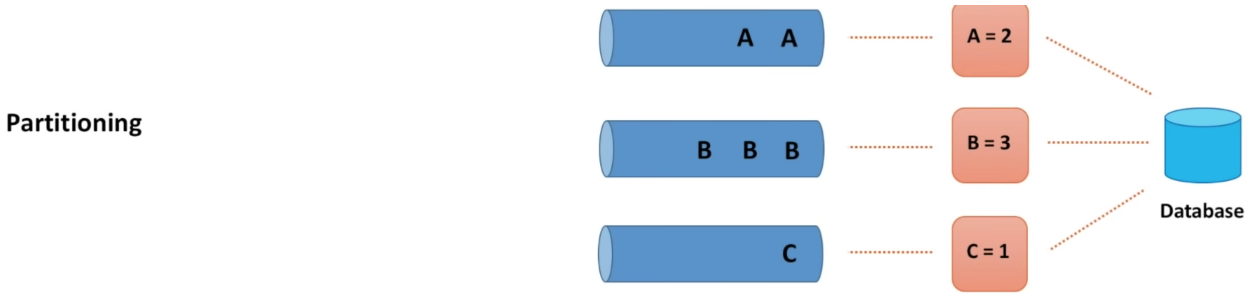
- In the diagram below the first picture shows the push approach where the processing service crashes. Data is lost
- In the second the data is pulled from a temporary storage in this case a queue





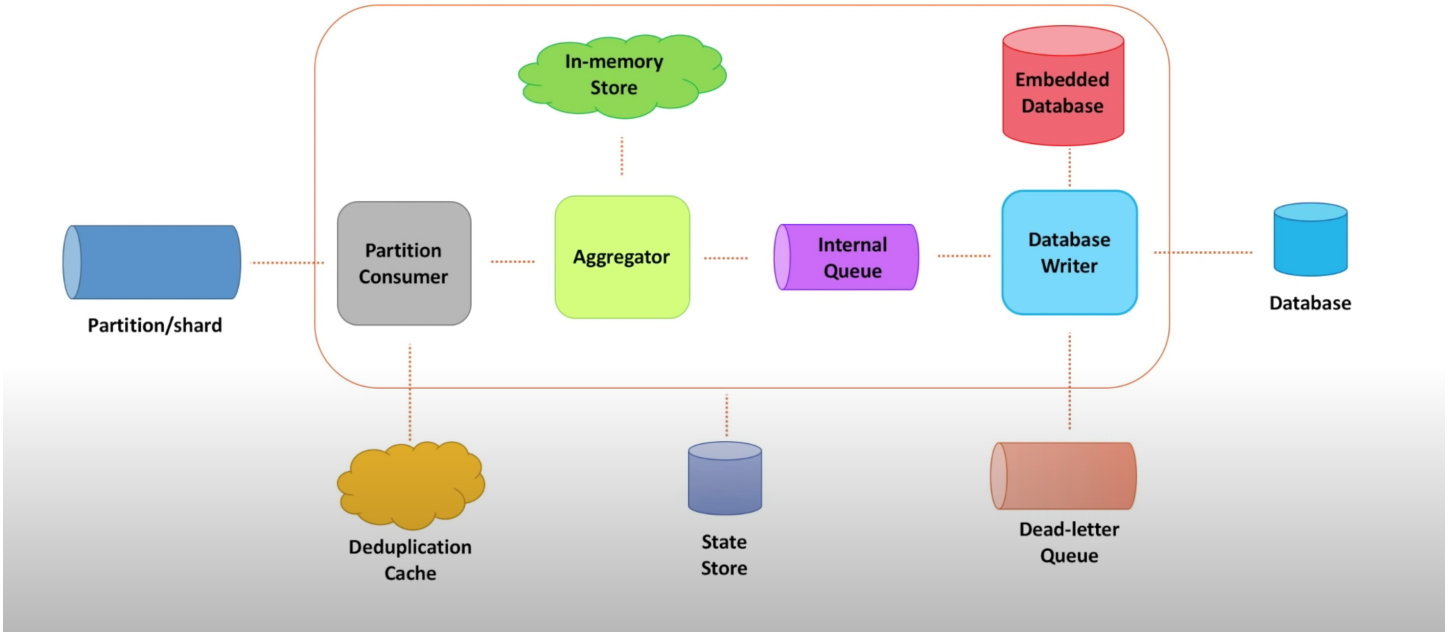
Two concepts





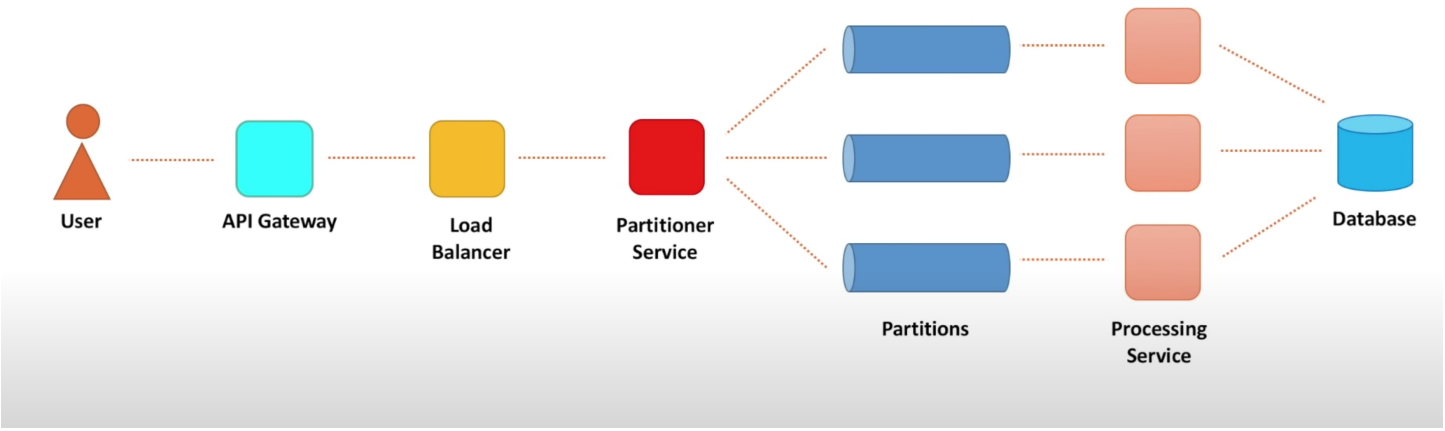
- checkpointing
  - https://en.wikipedia.org/wiki/Application_checkpointing
  - **Checkpointing** is a technique that provides fault tolerance for computing systems. It basically consists of saving a snapshot of the application's state, so that applications can restart from that point in case of failure.
  - in case of counting example we record the offset where we are. When processing service crashes we know from where we can continue
- partitioning
  - enables to support processing in parallel
  - queues are on different machines

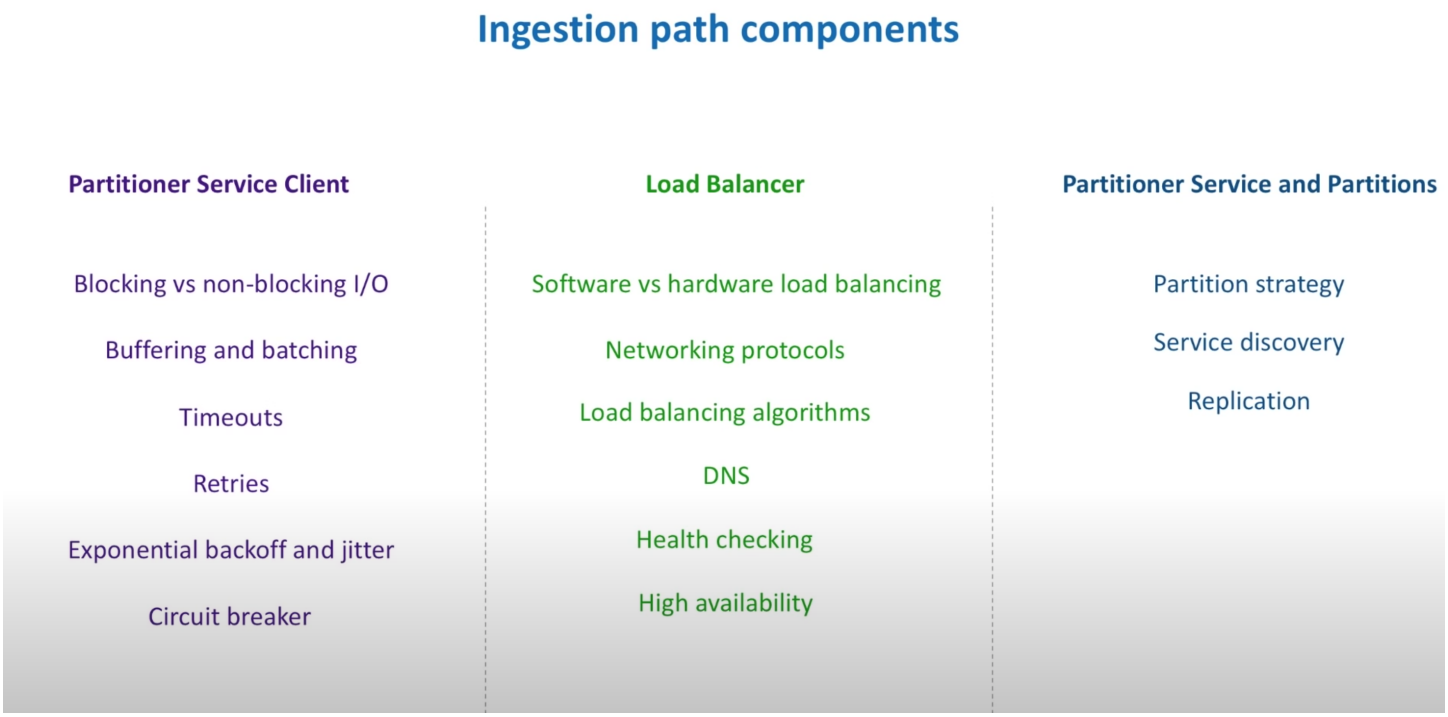
35:00 min



Q: Why Embedded Database? What is embedded DB?
Q: State Store
Q: Why Internal queue
Q: Deadletter queue

Data ingestion path





Partitioner Service Client

Blocking vs non-blocking

- blocking - create one thread per connection. Maybe can handle 100 of concurrent connections per machine. But when a slow down happens and number of threads increase. Could cause death spiral
  - easier to debug by looking into the thread stack
- non-blocking - higher through-put with multiple job queue. IO is processed later in the system. Piling up jobs in queues is less expensive than piling up threads
  - not as easy to debug

- not as easy to debug

Buffering and batching

- instead of sending individual events, place events into a buffer and send them as a single request (batch request). Wait several seconds or wait till buffer fills up
  - pros
    - increases through put
    - helps to save on costs
    - compression is more effective
  - cons
    - increases complexity on client and server side
    - in failure scenarios sending complete batch and partial

Timeouts

- connection and request timeouts
- connection timeout - how long a client is willing to wait till connection request is considered be not successful (in ms)
- request time -
  - latency timeout - measure request
    - what to do with request - retry

Retries

- smart retries like exponential backoff with jitter (jitter - add randomness)

Circuit breaker

- stop sending requests when threshold limit is exceeded
  - system gets more difficult to test
  - need to know about thresholds and error timeout

Load Balancer

- function - distribute data traffic between multiple servers
  - hardware/software loadbalancers
- AWS
  - ELB
  - network load balancer
    - tcp
    - http - Application load balancer
- algorithm to distribute load
  - round robin - distribute in order across list of servers

    - least connection - to server with lowest number of existing connections
    - least response time - to server with fastest response time
    - hashed - based on identifier

Questions

- how does client know about load balancer
- how does load balancer know about service partitioner
- how does load balancer guaranty high availability

Answer

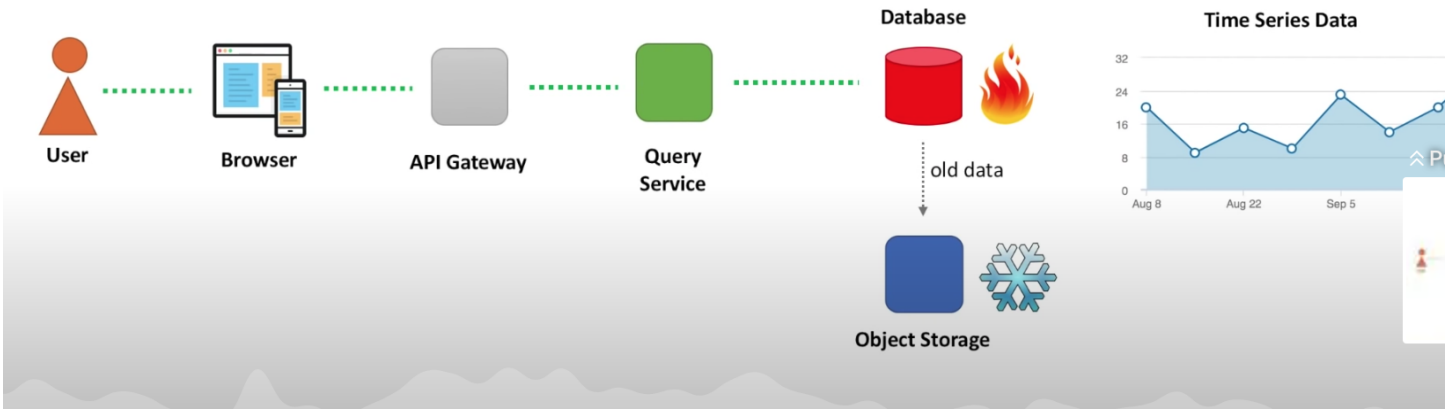
- DNS - like a phone book of internet
    - directory of domains name mapped to IP addresses
    - todo - register partitioner server in DNS
- For load balancer to know about partitioner service need to tell load balancer the IP addresses of each machine

- health check - LB need to know which service is healthy
- high availability - primary and secondary node
    - secondary monitors primary
    - primary and secondary live in different data centers

Partitioner Server and Partitions

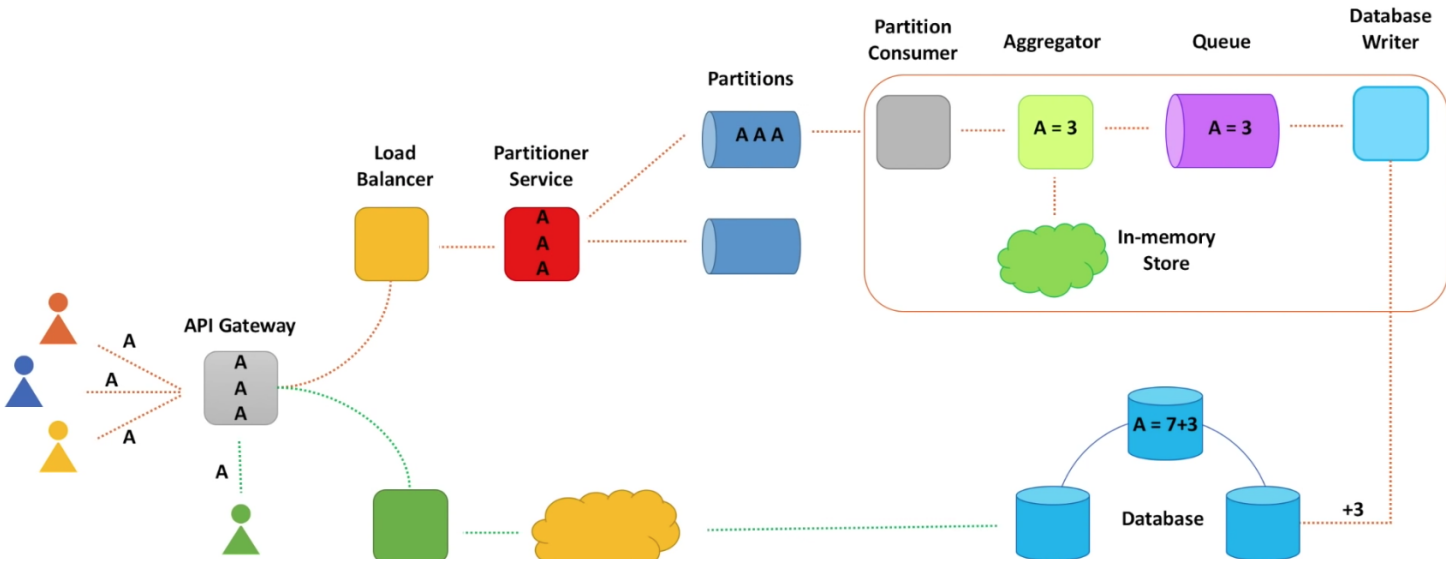
- partitioner server - what partition gets which requests
    - partition strategy - hash function based on identifier
        - problem - hot partitions
            - use event time to resolve hot partitions
            - split hot partitions into 2 new partitions - split keys
- service discovery
    - client side
    - server side
    - Zookeeper - partitions register with zookeeper
- replication - not to lose events
    - multiple leader replication
    - single leader - discussed when how to scale a Sql data base
        - leader - read and write from reader only
    - leaderless replication - when we discussed how Cassandra works
- Message format
    - textual - xml, json, csv
        - human readable
    - binary - protocol buffer, thrief
        - faster to parse
        - compact

Data Retrieval Path

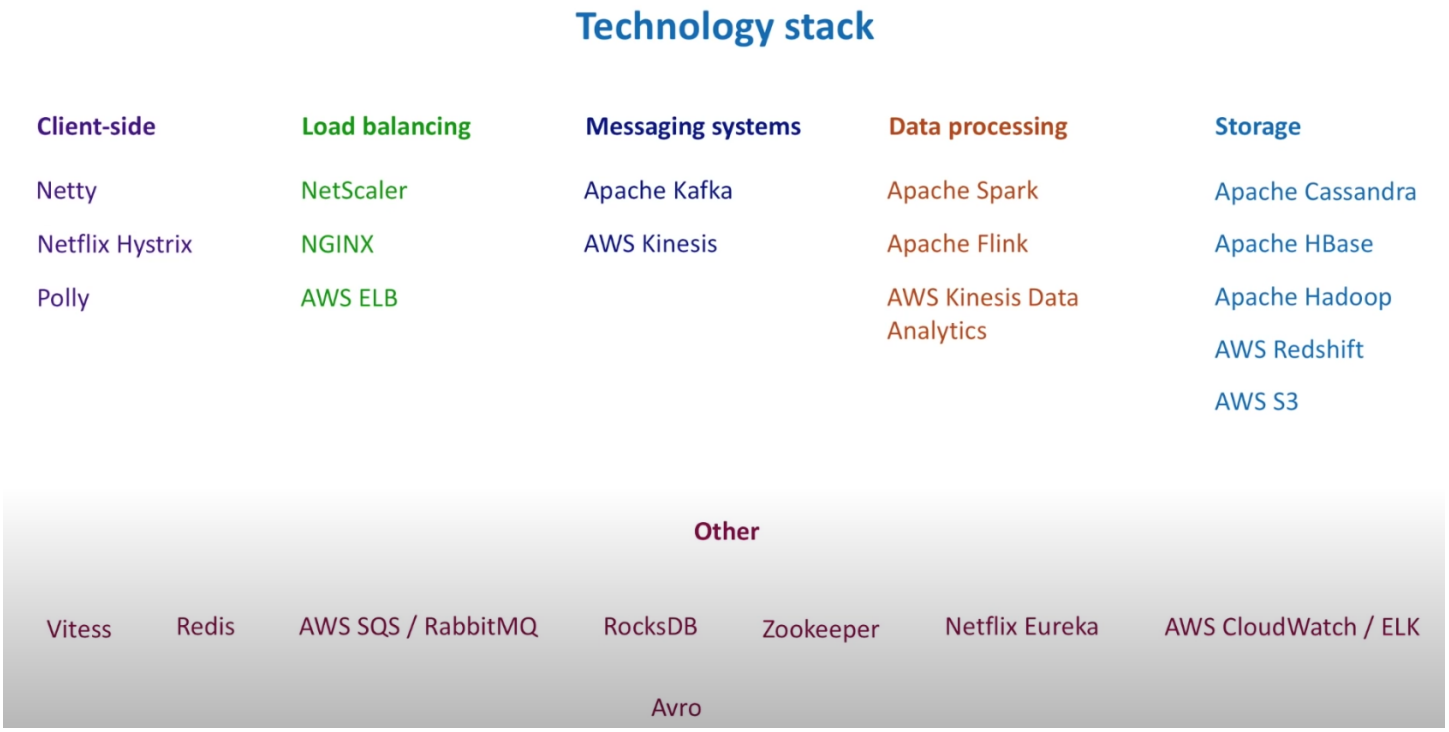


- goal to show total user count of videos
  - compute recommendations
- API Gateway routes request to backend service
- get total count directly from DB
  - single value in DB per video
  - using time series data for statistics
    - data aggregation
      - concept of rolling up data --> from days to weeks, from weeks to months
      - no need to store all data in DB
        - newer data in DB
        - older data can be stored in object store (AWS S3)
- store query results in cache

Complete Flow



Technology Stack



Bottlenecks, Tradeoffs - 1:14

Performance Test - Heavy Load
Load Testing - Testing System under specific load

- test that system can handle load
- is scalable and can handle 2 to 3 more time of current load

Stress Testing - to find breaking point of system

- which component will suffer first
- which resource will it be?

Soak Testing - **Soak testing** involves testing a system with a typical production load, over a continuous availability period, to validate system behavior under production use

- find memory leaks
    - use Jmeter

How to make sure that system works correctly --> counts correctly?

- building an audit system
    - weak - end-to-end test which runs eg every minute, we compare count with expected count
    - strong - build a lambda system which combines are stream processing and batch system in parallel

- stitch together results from both system
-