System Design

System : System is a collection of architecture or software technologies that interact with each other to cater the certain requirements of a certain set of users.

Things to consider while designing system :

Components
User
Requirements

Design : Understanding the requirements , users of the system, Selecting the components and how they will interact with each other is design.

System Design : It is the process of understanding the requirements of the user, understanding the user, selecting the components and intertwining the components such that they are capable of serving the needs of the user.

Components :

Logical :

Database
Server
Application Layer (APPs, mobile)
Presentation layer (what we see)
Networking (Protocols) Communication
Cache
Infrastructure

Every system(app) doesn't have a presentation layer like a logger application.

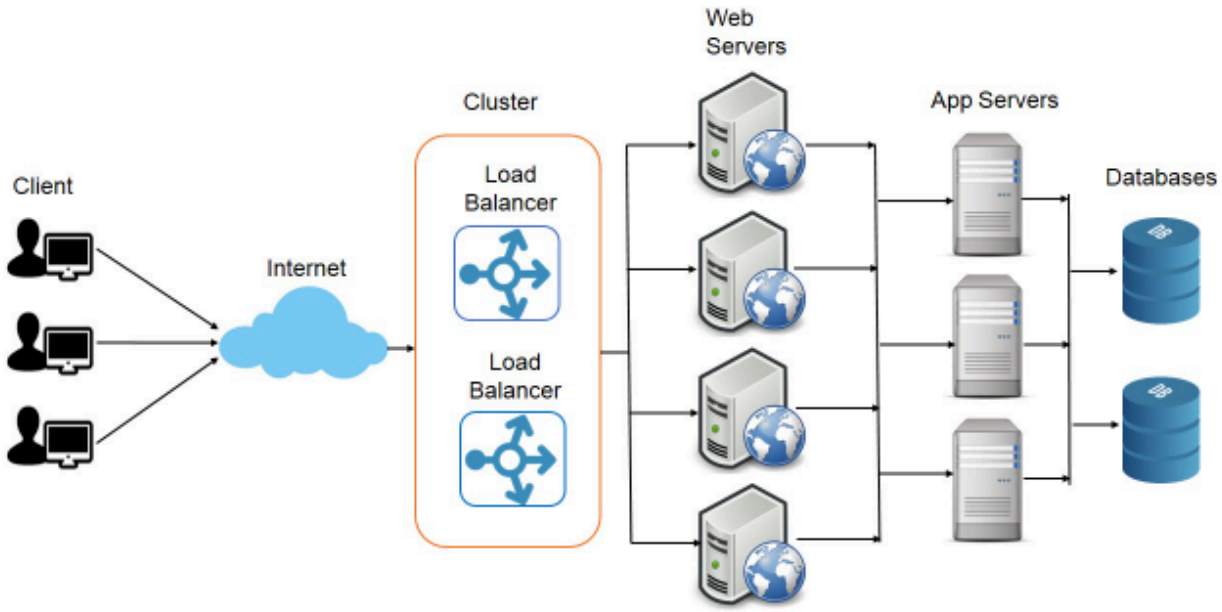
Tangible :

Text, image, videos
Mongodb, MySQL, Cassandra
Java, Golang, Python, Amber, React
Redis, memeCache
Kafka, RabbitMQ
AWS, GCP, Azure
APIs, RPCs, Messages

Here tangible are the instances of the logical components that we have represented above.

System Image :

System Design



No Bottle-Neck

Client server Architecture :

Different Tiers of Client Server Architecture :
Frontend = (Presentation + Logic)
Backend = (Logic + data)

Thick Client : (Presentation + Logic) (video games, Streaming applications**)**
Thin Client : Presentation only Logic sits on backend (ecommerce, streaming applications)

0) Single tier : Client(Presentation), Logic(backend) and data(database) are in the same machine i.e Developer's machine.

1) **Two tier :**
    a) **Logic and Data sits in the save server.**
    b) **IT is manipulated by the application in the client machine.)**
2) **Three tier :**
    a) **Frontend (presentation) client**
    b) **Logic(Server)**
    c) **Database(another server)**
3) **N tier :**
    a) **Client**
    b) **Load Balancer (reverse Proxies)**
    c) **Logic**

d) **Cache**
e) **Database**

Here Loadbalancer sits between client and logic and cache sits between Logic and database.

Proxies :

Forward proxy : Proxy sit's in client side.

1) Hide's client from server.
2) Bypass the blocked records.

Reverse proxy : Proxy server sit's in server side.

1) Hide's the server from the server
2) Use cases :
    a) Security
    b) Load balancing
    c) SSl encryption
    d) Case the responses

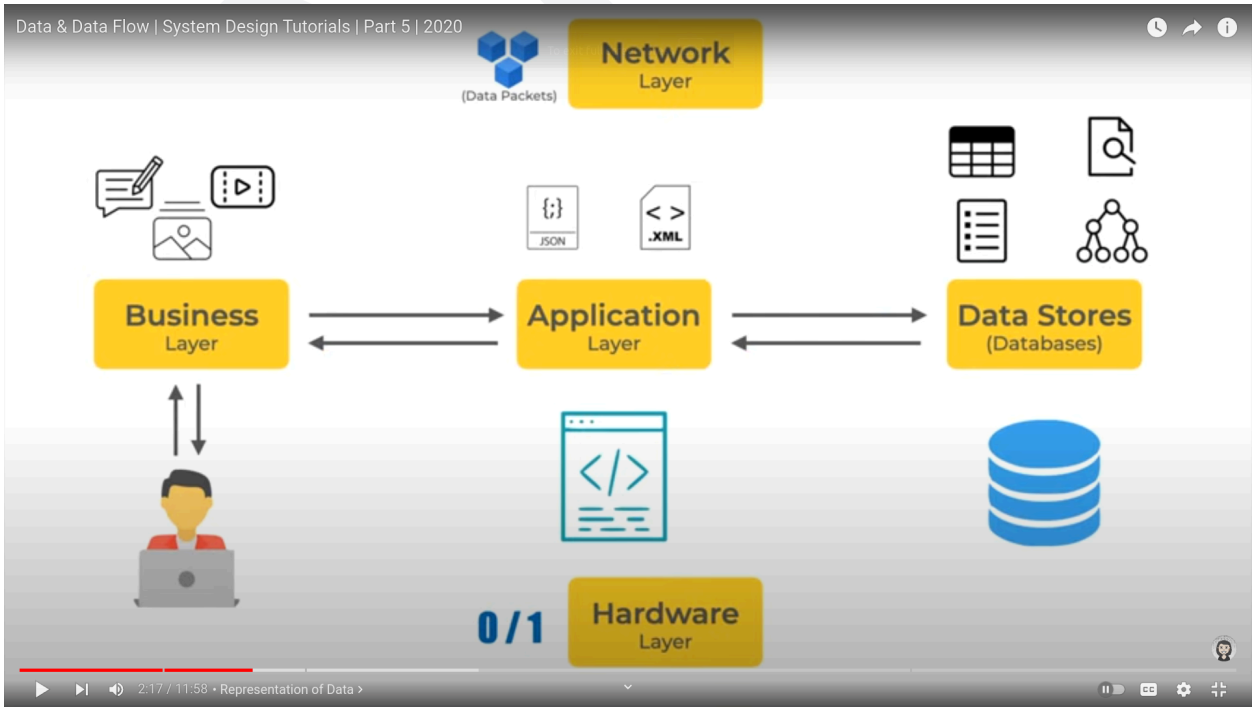
If reverse proxy fails it is the bottleneck since it is the single point of failure.

Data & Dataflow :

Data is the crux of the System or system design.

Which format is used for data transfer ?

a) JSON
b) XMl



System Design

Datastore :

1) Database
2) Queues
3) Cache
4) Indexes

Data method flow methods :

1) APIS
2) Events
3) Messages

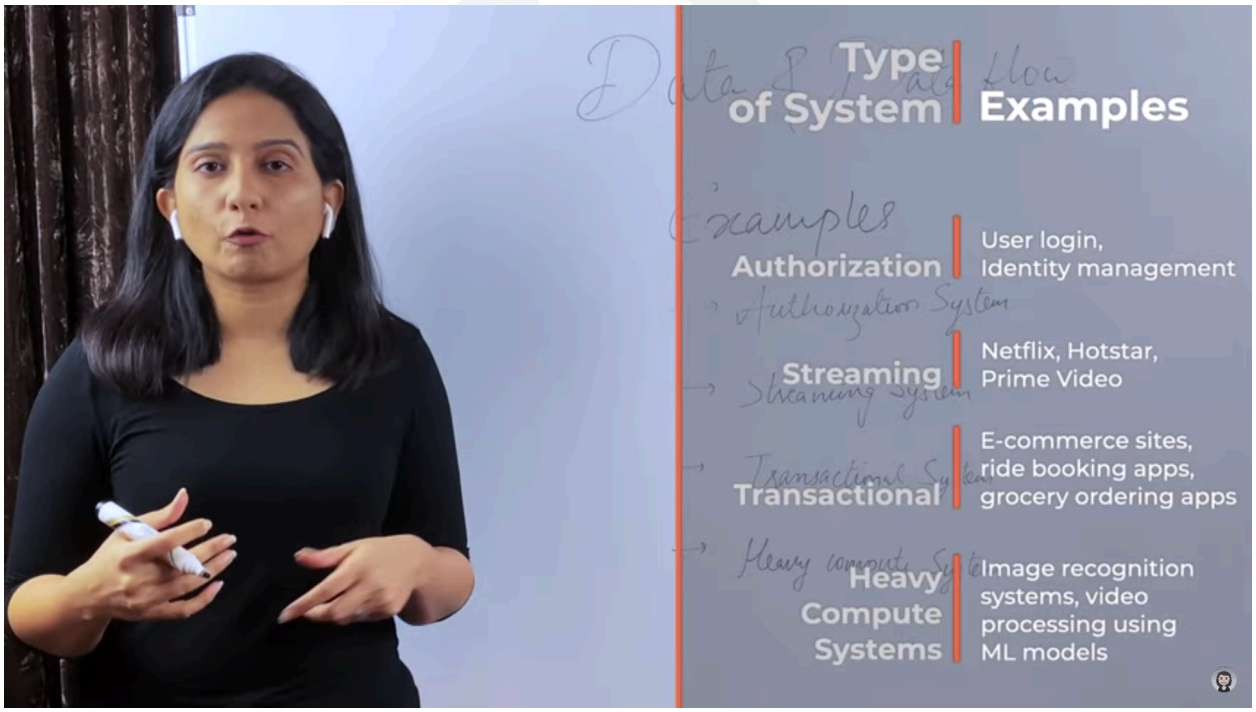
Data generation

1) User
2) Internal
3) Insights

Factors:

1) Type of data
2) Volume of data
3) Consumption / Retrievals (read & write)
4) Security

Case study for data :



Databases :

System Design

Relational :- Relational database is a database where data stored in rows and columns are related to each other. They have some constraints on the data that can be stored in tables.

Where data is stored in tables(relation)

Schema :- Structure of organization of data in database. Logical structure connection intertwining of different tables and the relation among them.

ACID : Atomic, Consistency, Isolation between transactions, Durability

We use relational DB's when we know that our data organization mechanism won't change.

Vertical scaling is easy
Horizontal scaling is tough. But it can be implemented.

So when we talk about relational databases we primarily mean the **relation between tables using foreign and primary keys.** We can also say that each column in a row is related to an instance of an entity. Every table is an entity.

Relationship between tables :
One to one : customer and Profile
One to many : customer and order
Many to many : product and order

Non-relational

1) **Key value : (**Stored in key value pair)(redis, **memeCache**).
    a) They are in a memory store.
    b) Cache implementation
2) **Column based :**
    a) Combination of relational and Documents
    b) Doesn't supports ACID but there is a Table to structure the data
    c) Good support of distributed databases
    d) Distributed DB is a database where data is available at different Node.
    e) Eg : Cassandra, HBase, Scylla
3) **Document based :**
    a) Used when we are not sure about schema, this is heavily unstructured.
    b) Supports heavy read and write
    c) Collection , documents (where collection is table and documents are row)
    d) We don't require you to join since every field is at a place.
    e) Problem : No fixed schema, No Acid transaction.
    f) Benefits : scalable, Sharding, Dynamic data flexibility , special query optimization / aggregation
4) **Search DBS :**
    a) They are used for searching the database or querying.
    b) Example = Elasticseach, Solr etc
    c) This is not a primary database.

In non relational databases it is easy to alter the structure without modifying the schema. It is used in a situation where we know that structure may change in future.
Horizontal Scaling and vertical scaling is easy.

File :

Eg : SQlite

Network : IDMS, IDS, UNIVAC

More

Applications :

A piece of code that helps us to meet our needs, it can be a presentation or a logic i.e. frontend or backend

Language are used for development of applications

Different applications communicate with each other using APIS;

Responsibilities :

1) Client :
    a) Rendering UI
    b) Handle interactions
    c) Collect data
    d) Communicate with the help of API
2) Server :

    Where logic is written

Types :

1) Backend
2) Frontend
3) CLI based

Factors while Designing application :

1) Requirements
2) Layers
3) Tech stack
4) Code structure, design pattern
5) Data store
6) Performance
7) Deployment
8) Maintenance

9) Operational excellence/ reliability

Architecture :

Monolith : Combine different functionality into the same application

Microservice : Device functionality into different applications

API :

Application programming interface. A programmatic approach of how different applications interact with each other.

Interface : Abstraction (Hiding the implementation details)

Advantages :

1) Communication
2) Abstraction
3) Platform Agnostic

Types :

1) Private
2) Public
3) Web
4) SDK/Library APIS

API Standards :

1) **RPC**
2) **SOAP**
3) **REST**

Cache :

Cache invalidation : Removing old data from cache and adding the new one. Adding the new one is called cache warming up.

Warming up the cache : Filling up the cache with new data.

Methods for cache invalidation :

a) **Expiry time (TTL)**
b) **Removing and adding (Updating by application)**
c) **Hybrid**

Cache Eviction :

When the cache is full and you have to add new value what will you do ? remove an entry and add the current entry right this is called cache eviction.

Methods :

a) **FIFO :** First in first out
b) **LRU :** Least recently used
c) **LFU :** Least frequently used

Cache Patterns :

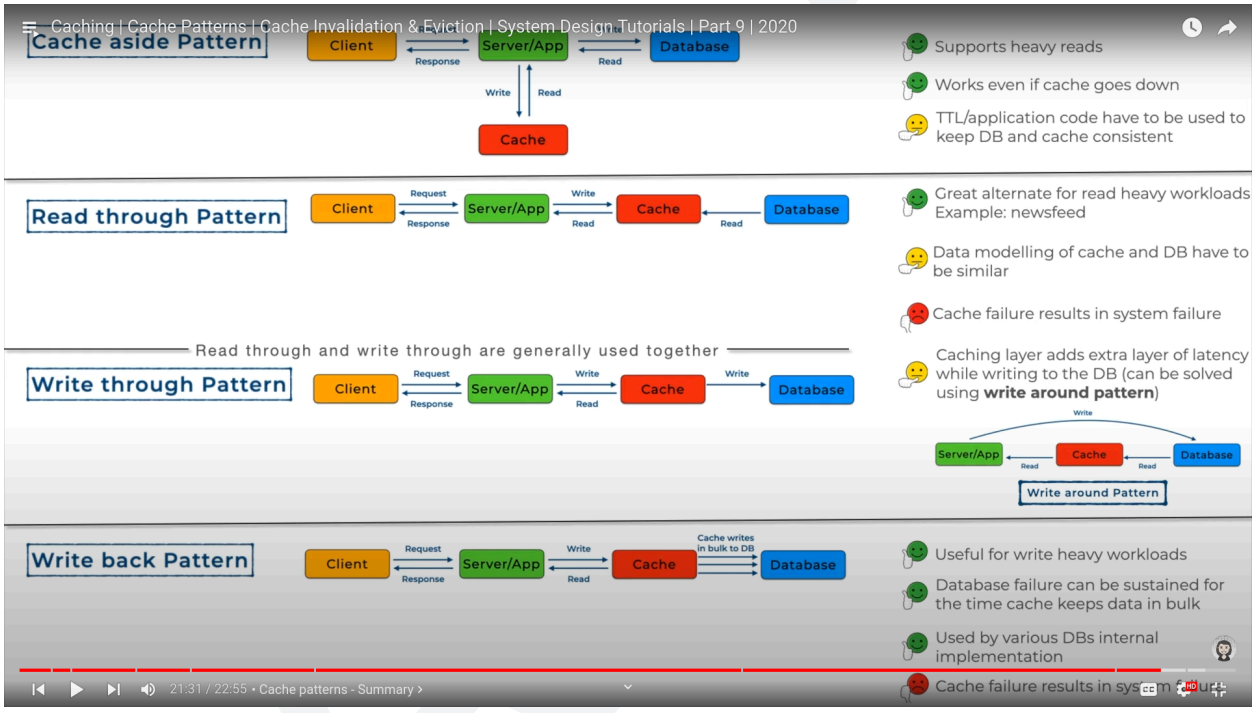
Cache Aside patterns : Cache always talks to applications not DB. Application talks to cache and updates it accordingly.

Read through Strategy : Here cache sits between application and DB. First request is always missed. And it's used for reading heavy tasks.

Write through Strategy : Here also cache sits between DB and application. Data is written through cache in DB.

Around around Strategy : Here also cache is there between db and application but write is done around the cache. Used when we have a write heavy load.

Write back Strategy : First write is done in cache then it is written in the DB in a batch(bulk). Here if DB goes down for sometime it doesn't affect.



APIS :

Rest API's : Representational state transfer. Client and server agree upon a method to share the data. Like JSON or XML

Saying that state of the data is transferred to the client.

Stateless : server should not know about the client information.

Few guidelines :

1) Client server
2) Cacheable
3) layered
4) Stateless
5) Uniform interface
6) Code on demand : (server sends the code to the cline that executes)

HTTP Methods : CRUD

POST, **PUT(Entire update)**, **Delete**, **get**, **patch**(partial Update) , **Head**(like get but only body), **OPTION(** ask server for option method for communication)

Uniform resource identifier

URI : Generic term for character string that identifies the resource

URL : Specific Type of URI that defines the location of resource on internet

https://anuj.verma.com

Path : location of particular resource

Query parameter : Querying a specific instance

localhost/person?name="Anuj"&age="21"

Status code :

- **200 - Ok**
- **201 - created**
- **301 - shifted somewhere resource is available somewhere else**
- **404 - page not found**
- **403 - Unauthorized access**
- **400 - sending bad data**
- **409 - Creating resource which already exists**
- **412 - server is unable to process data**
- **500 - internal server error**

There must be some security for api to provide limited access.

There must be throttling and rate limiting on API.

SO HTTP is a protocol and Rest is a guideline.

Message Queue :

Queue can be a process that has data structure in a memory to store messages and that is called a message queue.

Some queues in complex systems are KAFKA, RabbitMQ, SQS dedicated queue structure in system design.

Messages are pushed in the queue and messages are some instructions that have to be executed asynchronously meaning later at some time.

Synchronous : at a time

Asynchronous : Later at a time

So there is a producer who produces the messages and a consumer who consumes the message.

Components :

- Messages
- Consumer
- Producer
- Order message

Types of queues :

Ordered : **FIFO**

1) If one message fails the rest will not be executed. Since all the messages are in order the next to fail one won't execute.

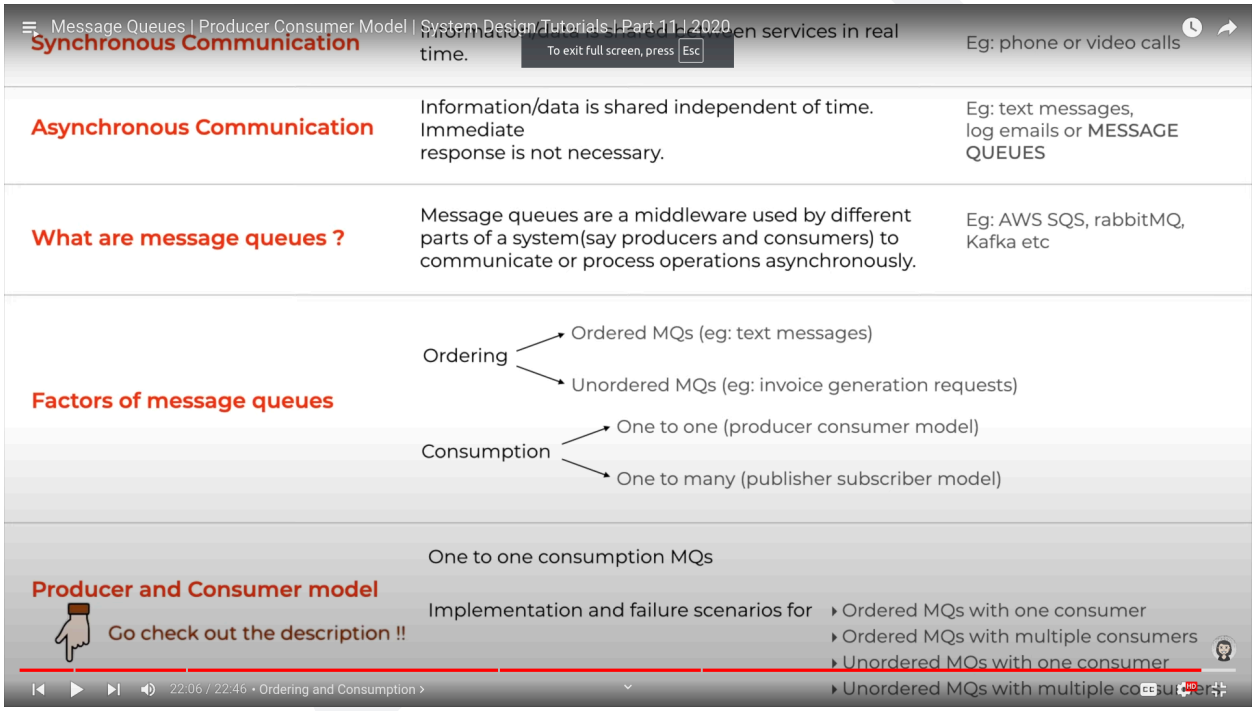
Unordered Queues :

1) If one of the messages fails it won't affect the other because that failed message will be pushed into the dead letter queue.
2) Then the failed message is pushed at the end.

Message queues are implemented to work with background jobs. This is good when one message executes just for once. After execution it will be removed from the queue. If we want that message to be executed by many consumers we will use the publish subscribe method in the next part.

One to one : Producer consumer

One to many : Publish subscribe



Pub Sub : (publish, subscribe)

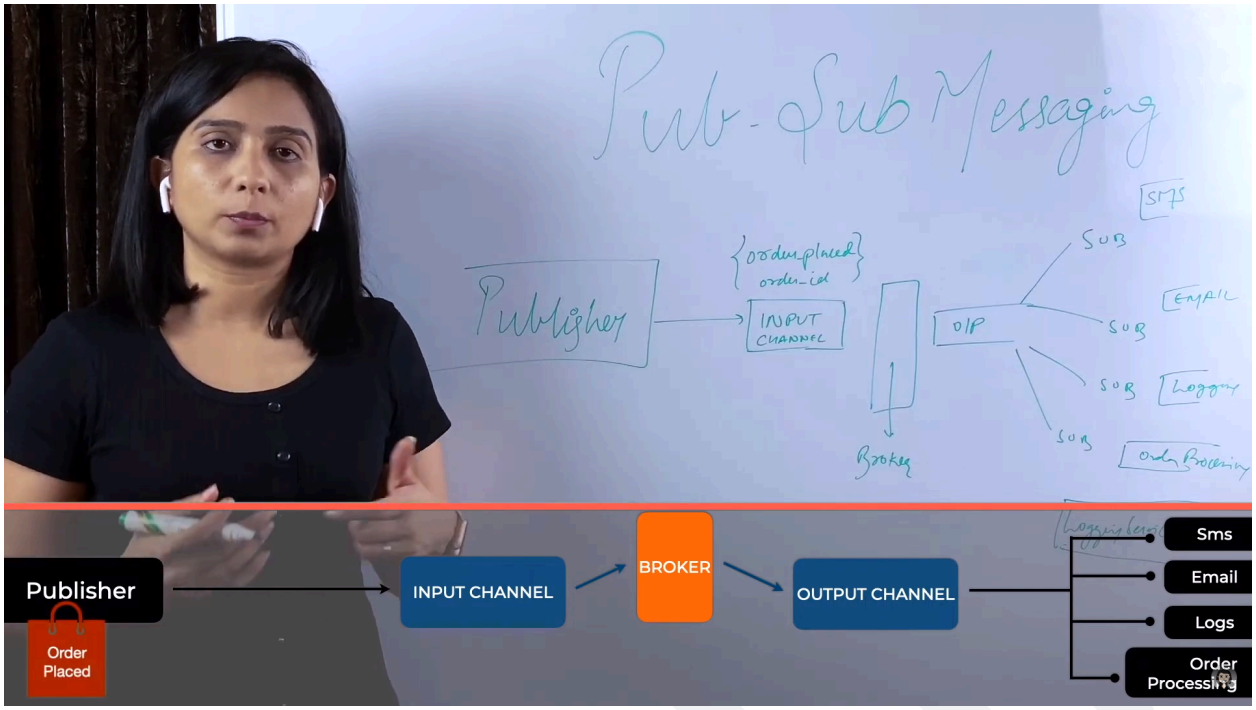
Publisher or producer(only one) : Produces the message or notice

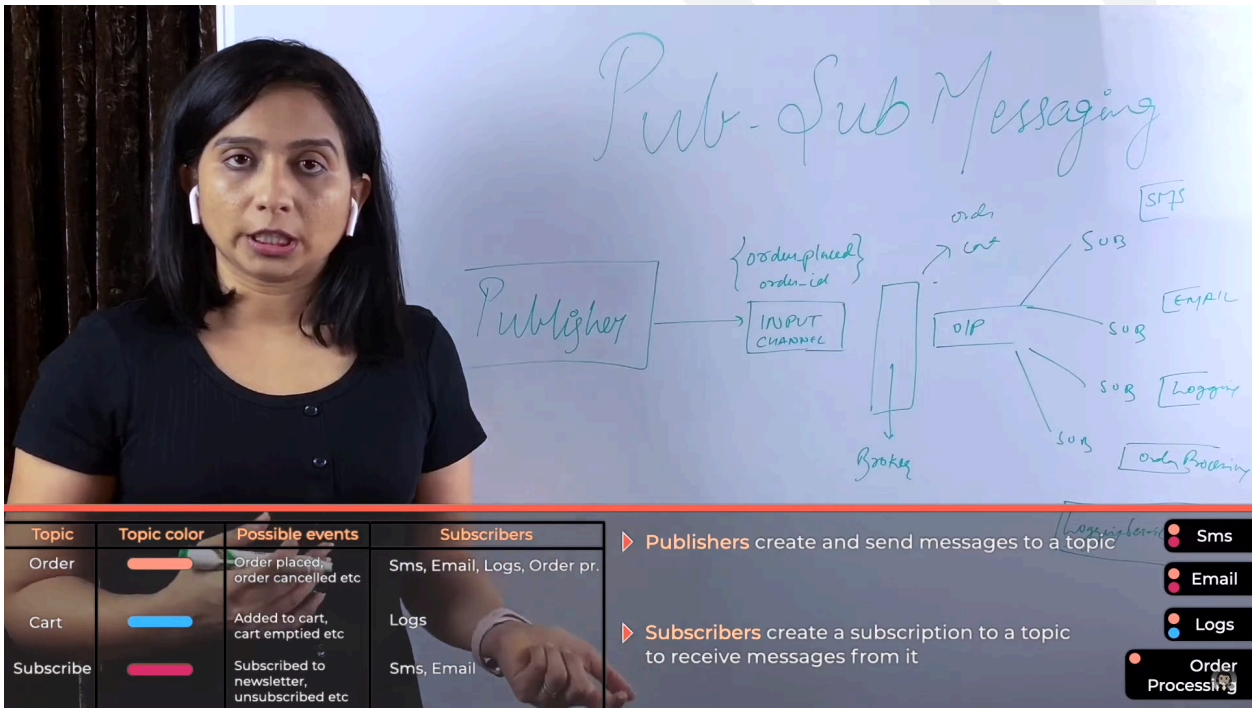
Input channel : Publisher produces message here.

Output channel : subscriber subscribe over here

Message broker :

System Design





Pub sub helps in decoupling and scaling of the system.

Factors of PUB-SUB

1) Message ordering : No order required but you can add priority and becomes priority queue
2) Duplicate message

3) Message consumption
4) Passion messages

In **Pub-Sub** also there is a queue where publisher publishes the message that is input channel

Use Case :-

1) Async Workflow
2) Decoupling
3) Load balancing
4) Deferred processing (Similar to async workflow) Do it at free hours not at peak hour
5) Data streaming

We should not use this in very simple tasks because it is meant to handle complex problems and also don't use in sync problem solving.

Performance metrics :

Throughput : Amount of work done per second. **Load / time taken or work done / time taken. Relevance : No of api calls being served per Unit Time. Unit time can be a second minute hour etc**

Bandwidth : The amount of data that can be transferred in a network. It is generally considered when data is being transferred through networks. **Throughput** is always less than **Bandwidth.**

Throughput and Bandwidth is balanced by Good amount of resources.

Response time : Amount of time taken to respond to the request. Suppose 100 requests in 1 sec so response time = 100/60 = 0.6 millisecond. Response time should be very less. To maintain good throughput. Response time is made up of delay and processing time.

Resource : The amount of infra we have to serve the user with their needs.

Latency : we can say it is the delay in the transmission of the response. Which makes up a part of response time.

Analysis of the Application performance :

Api response time
Throughput of the APIS
Error occurrence
Bugs in system

Analysis of the database performance :

Time taken by various queries on database (less join more performance)
No of queries per unit time (or throughput)

Analysis of the cache performance:

Latency of writing to case
Number of cache eviction and invalidation
Memory of cache instance

Analysis of the message queue performance:

Rate of production and consumption
Fraction of stale or unprocessed messages
No consumers and producers.

Analysis of the workers performance:

Time taken for job processing
Resource used in processing

Analysis of the Instance component performance :

Memory / Ram Utilization
CPU Utilization

We can have PM tools performance management tools in instances to manage the system performance. Datadog, Nagios, New Relic, Zabbix etc. cloud providers have their own performance management tools that we can use to improve the functionality.

Fault vs Failure:

Faults are the error full condition or event which cannot be controlled and which leads to failure. Types of faults : **network faults, machine crash, hardware issue, high CPU utilization, Human error programming bugs**. Fault is the cause and failure is the effect.

1) Understanding faults
2) Tolerating faults
3) Making system fail safe

1) Hardware faults : Create a replication of a hardware machine like multiple db and multiple servers.
2) Bugs : Testing a code or UNIT testing code.

Fault tolerance in distributed systems is a big topic.
We have to handle failure in a graceful manner. We have to implement some fail safe mechanisms.

Types of faults :

1) Transient Fault : Occurs for very small duration and hard to locate
2) Permanent Fault : Continues until gets fixed easily identifiable

Scaling :

Scaling is the ability of a system to be able to serve my customer as required when the requirement increases without affecting the performance and business. Type are given below :

Increased load
Not complexity in managing
No performance issue

Horizontal : Adding more machines.

1) Scale out
2) Required load balancing
3) No single point of failure
4) Data inconsistent
5) Network communication
6) No hardware limitation

Vertical : Adding more resources to the system.

1) Scale up
2) Single point of failure
3) No load balancing required
4) Data consistent
5) Hardware limitation
6) IPC communication

The concept of scaling came from the operating system. Scalable systems are always planned ahead of time.

Database replication :

Replica is an exact copy of something. So database replication is the process of creating an exact copy of the database instance for fault and failure safe mechanisms as well as fast response time.

Replication lag : The amount of time taken to copy data from master to slave is called replication lag. If more replication lag data will be inconsistent.

So there are consistency algorithm to solve this problem let's see:

1) **Read after write consistency model :**
    a) In this method until every replica is not written to its updated value the write is not completed.
    b) This is called synchronous replication.
    c) The master has to inform the writer that data has been written.
2) **Asynchronous replication :**

a) Master asks every replica to update in asynchronous mode and acknowledgement is sent to the creator.
b) May lead to inconsistency.

3) **Hybrid approach (semi synchronous)**
   a) Update value to every replica but waits only for one acknowledgement.

Snapshot : Taking a state of the database at a different time. It is used to roll back to different states.

Time replication , update replication etc etc

In Most cases we use hybrids rather than just going with one or the other approach.

CAP Theorem: Applicable to Distributed systems which are connected by network.

Here, Theorem says all three cannot be achieved at a time, at least one will be missing. Partitional tolerance is something which is going to occur due to network uses since networks are prone to fail.
Since partition is inevitable the system can either be available or consistent.

C: Consistency (Data is consistent through different system)
A: Availability
P:Partitioning & partition tolerance(Not broken but integral) When the communication channel between two nodes is broken but still system is available is system tolerance.

Types of system :

CA : Consistent & Available
CP : Consistent & partition tolerance
AP : Available & Partition Tolerance
PC : Partition & Consistent (If system not available then no need of consistence)

But we can make our system consistent and available despite having partitions to some extent. So while solving the problem we have to think about the use case of what we actually need, either high consistency or high availability. LIke high consistency in financial cases and available in others like youtube.
We can also avoid partition by adding the backup networking in our system.

Eventual consistency is the case where a system eventually becomes consistent.

Database sharding:

Database sharding is the process of splitting a large amount of data across multiple database server instances called shards. Since a single machine or database server can store

only a limited amount of data, database sharding helps us to overcome the issue. All of the shards have the same underlying architecture and they work together as a whole.
Database sharding helps us to process large amounts of data parallel reducing the overhead.
This is a horizontal scaling facility. In sharding each shard contains the unique data.

Logical shards : The part of a large database which is splitted and stored at different database server instances is a logical shards.

Physical shards : The database server instance containing the part of the database is a physical shard. Physical shards are also called nodes. Physical shards can contain one or more logical shards.

Benefits of sharding:

1) **Reduced latency , availability in case of replication, Query optimization.**

Type of sharding :

Alternative of Database sharding :

1) **Replication :** Designing the fault tolerance system. When one of the systems fails the other is available. Exact replicas of the database are made and stored somewhere else may or may not be in different geographical locations.

2) **Vertical Scaling :** Adding more computation power in a single machine. But it has the limitation that it cannot grow much.

3) **partitioning :** Data is divided into separate servers and collection of all is represented as a single data.
    1) **Horizontal partitioning :** Dividing the database in row wise. It is similar to sharding where data is divided into different groups uniquely. But in partitioning the groups are stored in the same node or computer.
    2) **Vertical partitioning :** Divide the database column wise.

Methods of sharding :

Key based sharding :

Range based sharding :

1) Sharding is done on the basis of range.
   No evenly distributed data.
   But it is more scalable.
   When data is unevenly distributed the shard containing the huge data is called a hotspot.
   Algorithmic sharding

Hashed sharding :

Another concept : Consistent hashing

It is a key based sharding.

Hash function maps the shard key and separates the data.
Hash function is always consistent.
Hash function always gives a consistent result.
IN case of datachange the hash function changes and we have to manage it.
Gives evenly distributed data
Algorithmic sharding

Directory sharding:

Dynamic sharding : Lookup table.
Here key in data is mapped with shard key in lookup table then a query is made.
There will be a hotspot
There will be one more operation of viewing a lookup table.
If the lookup table is down it's a single point of failure. It's good to have replica

Geo sharding :

TYpes of sharding :

Dynamic sharding :
Algorithmic sharding : No of shards are static

Shard Key : Shard key is a column that is used to divide the data separately. Primary key can be a shard key, not every shard key is a primary key. We can also choose multiple columns to be a shard.

Drawbacks :

1) Costly
2) Uneven distribution because of poor sharding strategy
3) Sharding is one way.
4) It is only done when nothing works like cashing, load balancing, scaling etc

Consistent hashing :

Consistent hashing is a technique in a distributed system which is used to distribute the data to the set of nodes such that minimum reorganization is needed when no of nodes are changed.

Here both the data values and node values are hashed and then they are placed on a circular hash ring and further the value of data is mapped to the nearest node.

IN this approach when there are less no of nodes the reorganization factor is 50% and when we use replica to save from overload the reorganization factor is low

Key factors :

1) **Hashing Ring or circle (360*)**

2) **No of nodes**
3) **Mapping to nodes.**

Approaching to interviews :

1) **Understand the function requirements**
    a) **What a system is supposed to do. Functions**
    b) **How : API, Workers, Events, Messaging**
    c) **Outcomes: System design , Component and architecture design**
    d) **About Product and objective**

2) **Understand the non-functional requirements**
    a) **To help system perform their functions in efficient way**
    b) **Resource analysis**
    c) **Throughput , bandwidth , speed, availability, no of users etc**
    d) **We have to do capacity estimation.**
    e) **Outcomes : Tech choices, Resource utilization , data storage, Server and hardware**
    f) **About performance**

SLA : Service level Agreement
SLO : service level objective

Capacity Estimation : NO of transitions, no of read and write, no of request, no of servers.

Preparing how much capacity is needed to serve the requirements.

Thumb rules :

1) We have to approximate the estimation like you can round of for easy calculation eg 17 to 20 etc or to the power of 2's or 10's
2) No the table till 12
3) KNow metric system (million, billion, trillion)
4) Million 10^6 billion 10^9
5) Storage capacity (gigabytes = gig)
6) Some calculations
    a) 1million/day = 12/sec request
    b) 1million/day = 700/min
    c) 1million/day = 4200/hr
7) Memorize the latency numbers baba

Back of the envelope estimation : It is the estimation technique that gives the idea of how much resource should be considered for the system.

Estimations :

1) No of servers

2) Ram capacity
3) Storage capacity
4) Trade-offs (CAP) consistent or available
5) Processing power

Matrics :

Thousand = 3
Million : 6
Billion = 9
Trillion = 13
quadrillion = 15
Kilo = 3
Mega = 6
Giga = 9
Tera = 12
Peta = 15
T* k = Mega 3+3 =6
M*K = Giga 6+3 = 9
M*M = Tera 6*6 = 12
M*G = Petabytes

Estimation traffic :

1 Billion users 250 active user
Read = 5 , write = 2 = 7
250 million / total no of seconds per day(86400) ~ 100000 = 18k request per second

Storage assumption = each user 2 post size of one post is 5 mb = 500 million posts * 5 mb = 2500000000000000 ~ 2.4 petabytes of storage

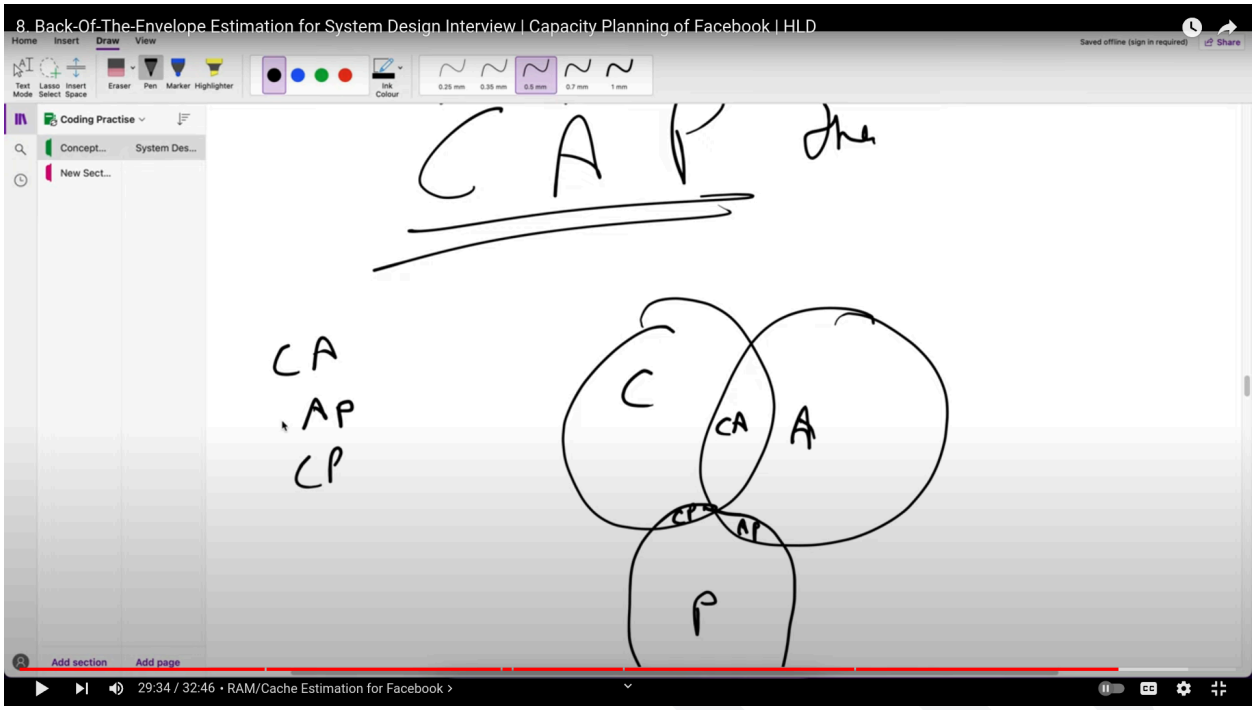
Ram estimation : Last 5 post for each user where each post is of 3kb then total space required for 250 million = 250*3 = 750gb
Each server holds 75gb of ram then we will need 7 machines

Latency : 1 request takes 500 ms to server
1000 ms to 1 sec. 18k request per sec
1 server = 50 thread = 100 requests
So we wil need 18k/100 requests = 180 machines

So now we have

180 servers
10 cache nodes with 75gb each
Storage = 2.4 petabytes image
Trade-off = here ill choose CP

System Design



Never take more than 10 min in capacity estimation or back of envelope estimation.

NOTE :

1) **First understand the requirements like functional and non-functional.**
2) **Always talk and estimate the capacity of the system.**
3) **Trade-offs (Like CAP theorem)**
4) **Understanding the faults of the system.**
5) **Do not talk about something you don’t know about e.g. tech stack.**