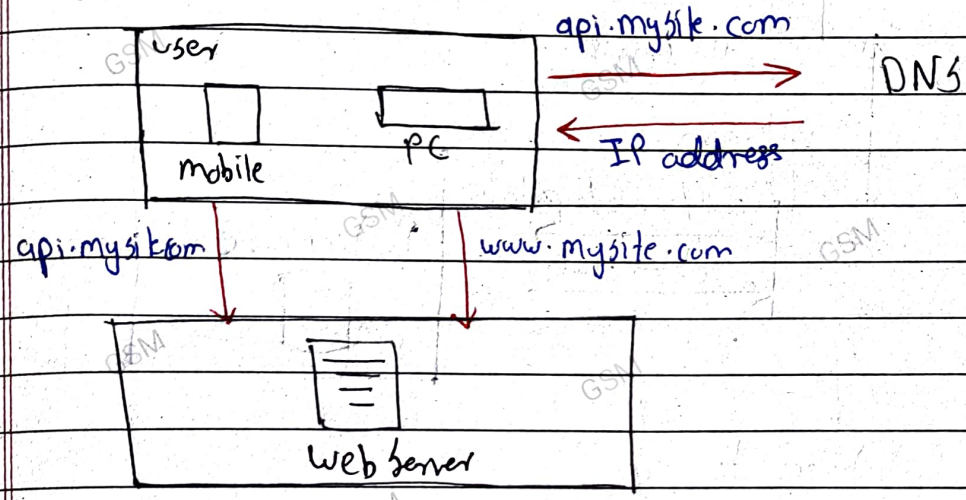
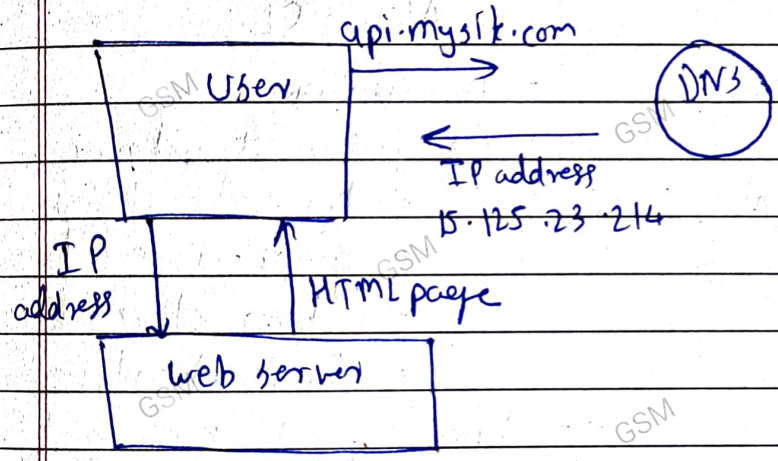
System Design.

SUBODH Page No.
Date

Chapter - 1

☆ Single Server Setup

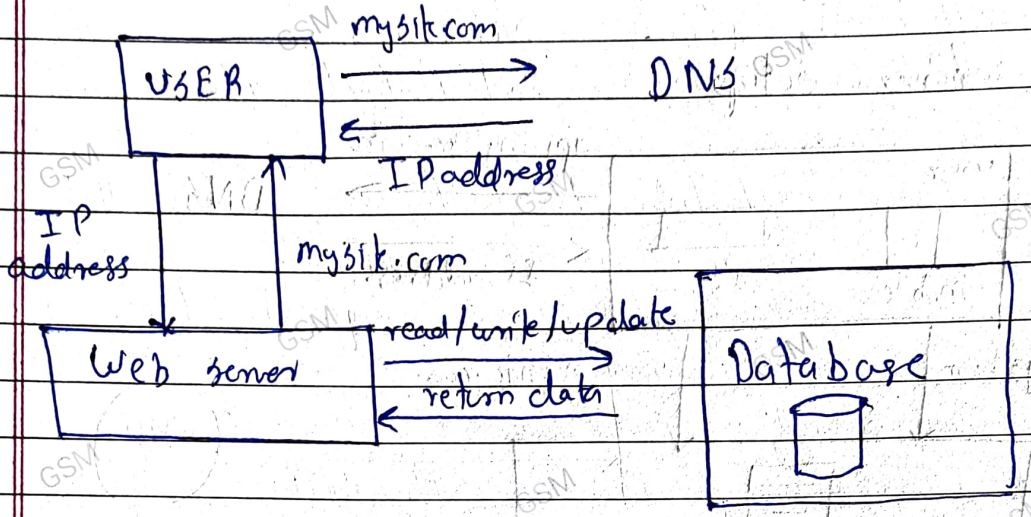




- API response in JSON

eg:- {
"id": 12
"FN": "Jhon"
"address": {
"city": New York
"stak": "NY"
}
}

☆ Data base



- Which Database to use?

RDBMS	NO-SQL
- SQL query based	- no query
- Tables & rows	- Grouped in 4 category
- Perform database Table join operation	• Key-value stores • graph stores • column stores • document stores
	- Join operations are not supported.

- NO-SQL is right choice when
  - App needs super low latency
  - data is unstructured
  - no relation or non-relational data
  - only need to serialize & deserialize data [JSON, XML, YAML, etc]
  - Need to store Massive amount of data.

SUBODH Page No.
Date

Vertical scaling Vs Horizontal scaling.

- Vertical scaling
  - Scale-up
  - Adding more [CPU, RAM etc]
  - Simple to scale up
  - great option when traffic is low
  - Limitations
    - hard limit to scale-up
    - impossible to add unlimited CPU & memory power
    - No support to failover & redundancy
    - Single point failure
    - No load distribution to manage traffic / thus slower response.

Horizontal scaling
- Scale-out
- Adding more servers into pool of resource
- Benifits
  - Solves single point failovers.
  - improved the availability of web-tier
  - if one server goes offline traffic is routed to another
  - Uses load balancer to manage traffic overloading thus improved responce time.
  - if in case of rapid traffic grauth just need to add one more server to the load-balancer.

- Load balancer [What is it?].

→ A load-balancer evenly distributes incoming traffic among web-servers that are defined in a load-balanced set.

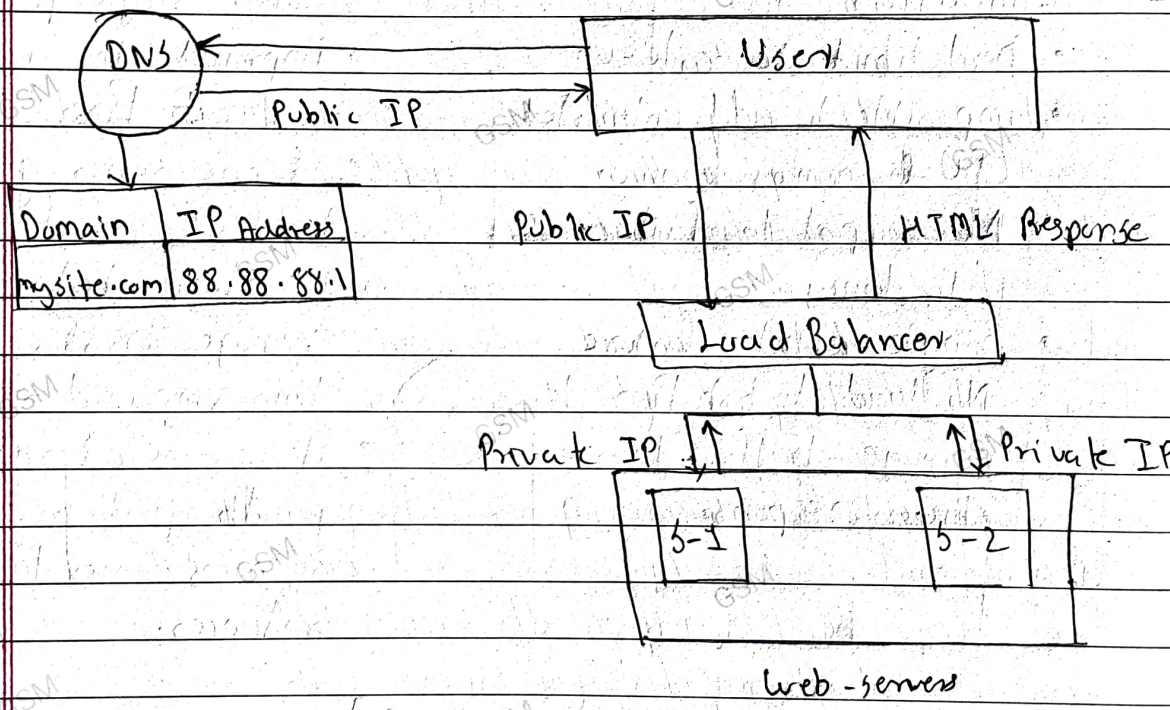
→ How it works ?

→ The user connects to the public IP of the load-balancer directly. Thus web servers are not directly available to the user, Now the load balancer uses private IP to communicate and get response from the web servers. A private IP is only reachable to the

load balancer & in between the web servers. Thus, it enhances the security. Now the request's are evenly distributed between web-servers.

☆ Still there is a lack of detailed

Current design.



☆ The current Design have only one database integrated into web server, so it does not support data failover & redundancy

☆ Data Replication

- Database Replication can be used in many database management systems, usually with a master/slave relationship between the original (master) & the copies (slaves)

SUBODH Page No.
Date / /

- A master database supports only write operations
- A slave database gets the copies of the data from master
- All data modifying commands like INSERT, DELETE, MODIFY, UPDATE are sent to master data base
- Most applications requires much higer reads to write ratio, thus there exist large number of slave database than the number of master data base.

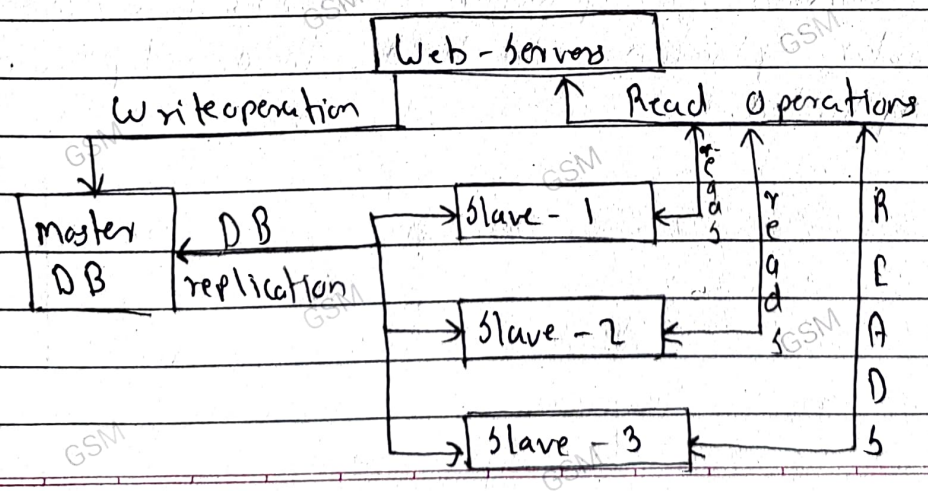
- Advantages of data replication.

i) Better Performance :- process multiple queries in parallel.

ii) Reliability :- If we lose one database, still due to data replication we have database secure across multiple locations.

iii) High Availability :- Due to replication even the nearest Database to the user is shut-down still the app/site is active as the data is accessed from an another location.

- Database Architectural design



- In this DB design

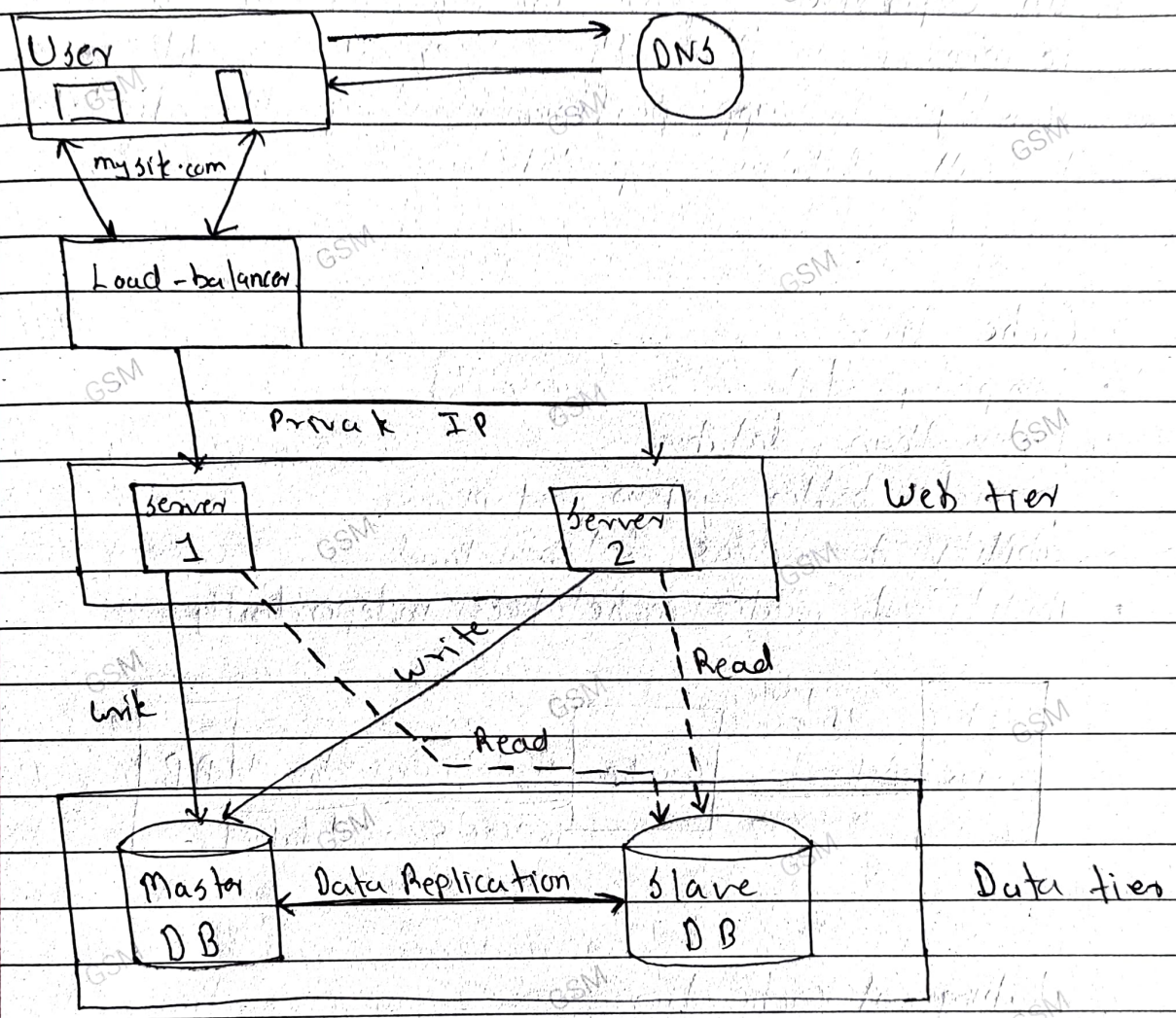
CASE - 1 :- If only one slave is available & it goes offline then all the read traffic is ~~read~~ routed to the master DB, until the new slave replaces the old one. In case multiple slave the reads are re-directed to the healthy slave server db.

CASE - 2 :- i) If the master db goes offline, then a healthy slave will be promoted to the new master. All the database operations thus are directed to the new master db & the new slave db is added to replace the old one immediately to perfor data replication

ii) In production promoting the slave as a new master is more complicated as the data in the slave db might not be up to date Thus the missing data needs to be recover using data recovery scripts

- Although some other replication methods like multi-master and circular replication could help, those setups are more complicated

- The current System Design after adding the load-balancer and data replication.



- Let us take a look at the design
  i) A user gets public IP address of the load balancer from DNS
  ii) A user connects the load balancer with this IP address
  iii) The HTTP request is routed to either Server 1 or Server 2
  iv) A web server reads user data from a slave database
  v) A web server routs any data modifying operation through master DB that includes write, update, delete, modify etc.

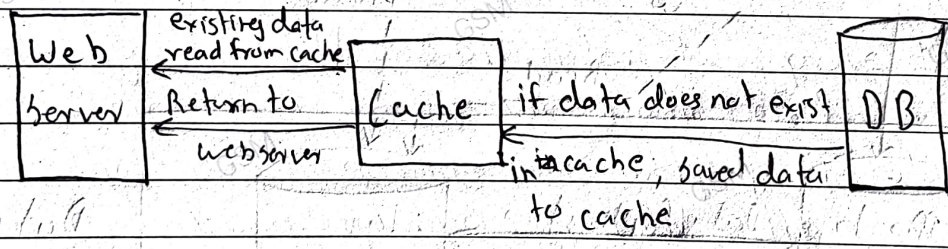
☆ As we now understand web & data tier let us improve load/response time. This is done by adding cache layer and shifting static content to CDN.

[JavaScript, CSS, image, video, files]

Cache

- A temporary storage area that stores the result of an expensive response or frequently accessed data in memory so that the subsequent request are served quickly. The app performance is greately affected to call database repeatitively

- Cache tier
- temporary data store layer
- faster than database
- provides better system performance
- ability to reduce database load
- ability to scale cache tier independently



Working of cache tier

- As web server receive the request, a web server checks for the response in cache if the response is available it is send to user else the cache layer gets the data from database and stores it in cache tier with a response to the webserver hence when the web server asks for similar response the response is available in the cache tier.
This strategy is called as read-through cache
Other caching strategy are also available depending upon size, data type and access pattern.

SUBODH Page No.
Date

Consideration for using cache

- Use cache when data is read frequently but modified infrequently. as cache server stores data in volatile data persisting data into it is not ideal. Thus if cache server suffers powerloss & restarts the data stored is lost.

- Expiration Policy. As a good practice it is to be followed with an implementation of an expiration policy. Once the cached data expires it is removed from the cache tier else it remains until the tier restarts.
  Also it is advisable not to make expiration time too short or else the system will reload the data from the database too frequently thus increasing the database load. Meanwhile, it is also not advisable to make expiration time too long else the data becomes stale.

- Consistency! This involves keeping the data store & cache in sync. Inconsistency can occur when data modification on data store & cache are not in single transaction.

- Mitigating failures: A single cache server represents a potential of single point of failure [SPOF]
  "A single point of failure is a part of system, that if it fails then the entire system stops working."

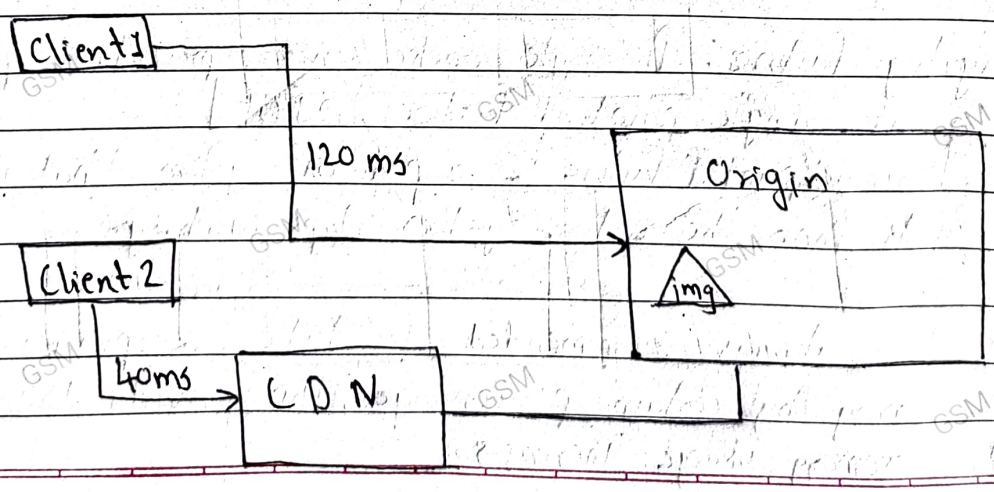
Thus it is always recommended to take overprovision approach for memory by certain percentage, this provide the buffer as the memory usage increases.

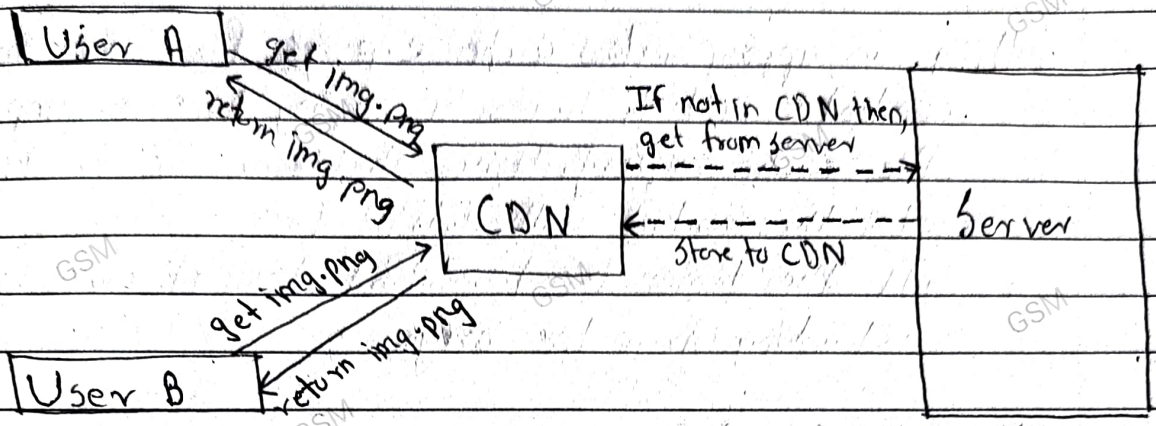
- Eviction Policy: Once the cache is full, any request to add items to the cache might cause existing items to be removed. This is called cache eviction.
The cache eviction policies are
  i) Least Recently Used [LRU]
    ↳ This is the most popular cache eviction policy
  ii) Least Frequently Used [LFU]
  iii) First in First Out [FIFO]

☆ Content Delivery Network [CDN]

- A CDN is a network which is geographically dispersed servers which are used to deliver the static content. CDN server cache the static content like images, videos, CSS, Javascript files etc.

Working of CDN: CDN helps to improve the load time as it caches the static content, thus when the user requests some webpage the static content is delivered by CDN. the user will recieve this static content from the server closer to them. thus as close the CDN server be the faster the load time for the user.





Let's Understand the working of the above system.

- User A tries to get image.png from an URL
- If CDN server does not have image.png in the cache, the CDN will request the server for the file from the origin and once the origin returns the image as response the image is then stored in CDN and same response is transfered to the User A.

- When the server gives the image response it includes an optional HTTP Header Time-to-Live (TTL) which describes the duration of image to be cached in CDN.

- User B requests the same image thus the image will immediately be transfered from CDN as long as TTL has not expired.

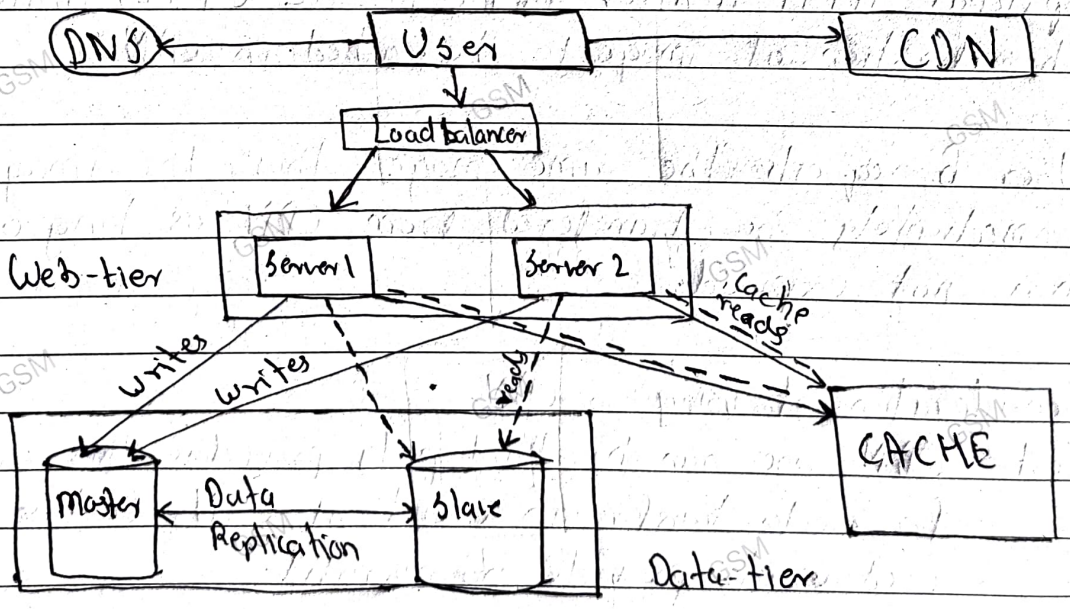
☆ Consideration of using a CDN

- Cost: CDN are ran by third-party providers thus we are charged for data transfer in & out of CDN. Caching infrequently used assets provide no benifit.

Setting an appropriate cache expiry: For time sensitive content, setting a cache expiry time is important. The cache expiry time should not be too long else the content will not be fresh.
If the time is too short the it cause repetatively reloading from the origin.

- CDN fallback: App should know how to handle CDN fallback. If there is temporary outage of CDN then client should be able to access files from origin.
- Invalidating files: We can remove files from cache before their expiration by using the Invalidate cache API provided by vendor or by using versioning method to creat multiple versions by adding a simple parameter to the query.

1. The current system design after adding CDN and cache



- Static assets [JS, CSS, images, etc] are no longer served by data fires or web servers. They are fetched from the CDN for better performance.
- The database load is lightened by caching data.

☆ Stateless Web-tier

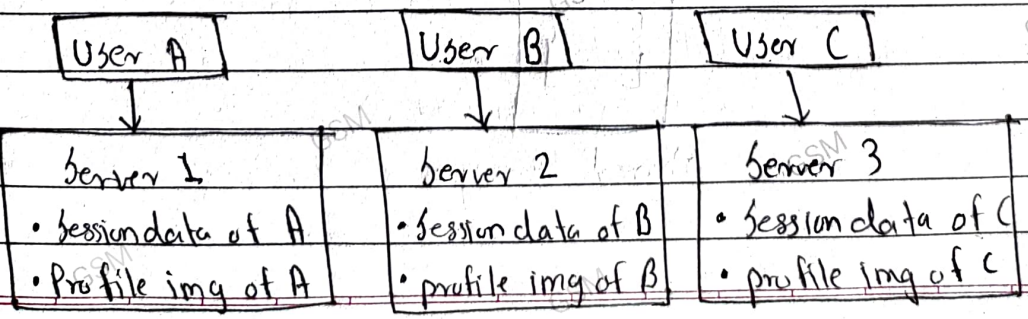
Now we scale our web tier horizontally. For this we need to move out the state of the web tier.

State:- generally the state is the user session data.

As a good pratice we should store session data into a persistent storage such as RDB i-e Relational Database or ~~NO-SQL~~ NoSQL. Thus each web cluster can access state data from database. This is called stateless web-tier.

☆ Stateful Architecture

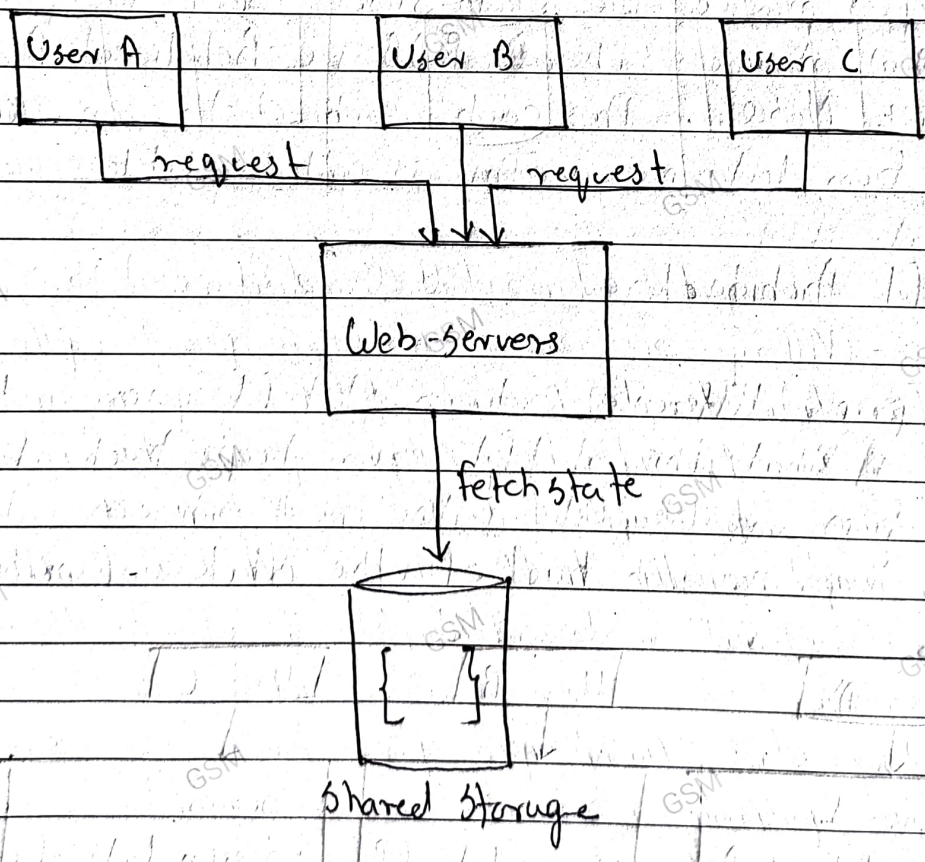
The prime difference between stateful server and stateless server is that the stateful server keeps track of user data from one request to the next whereas stateless ~~sent~~ server keeps no such track of the state information.



User A session data and profile image is stored in server 1 so to authenticate user A the request from A should be routed only to server 1 if is is routed to any other the it will fail to authenticate.
Similarly the request from User B should be routed to server 2 and all from User C to be routed to server 3

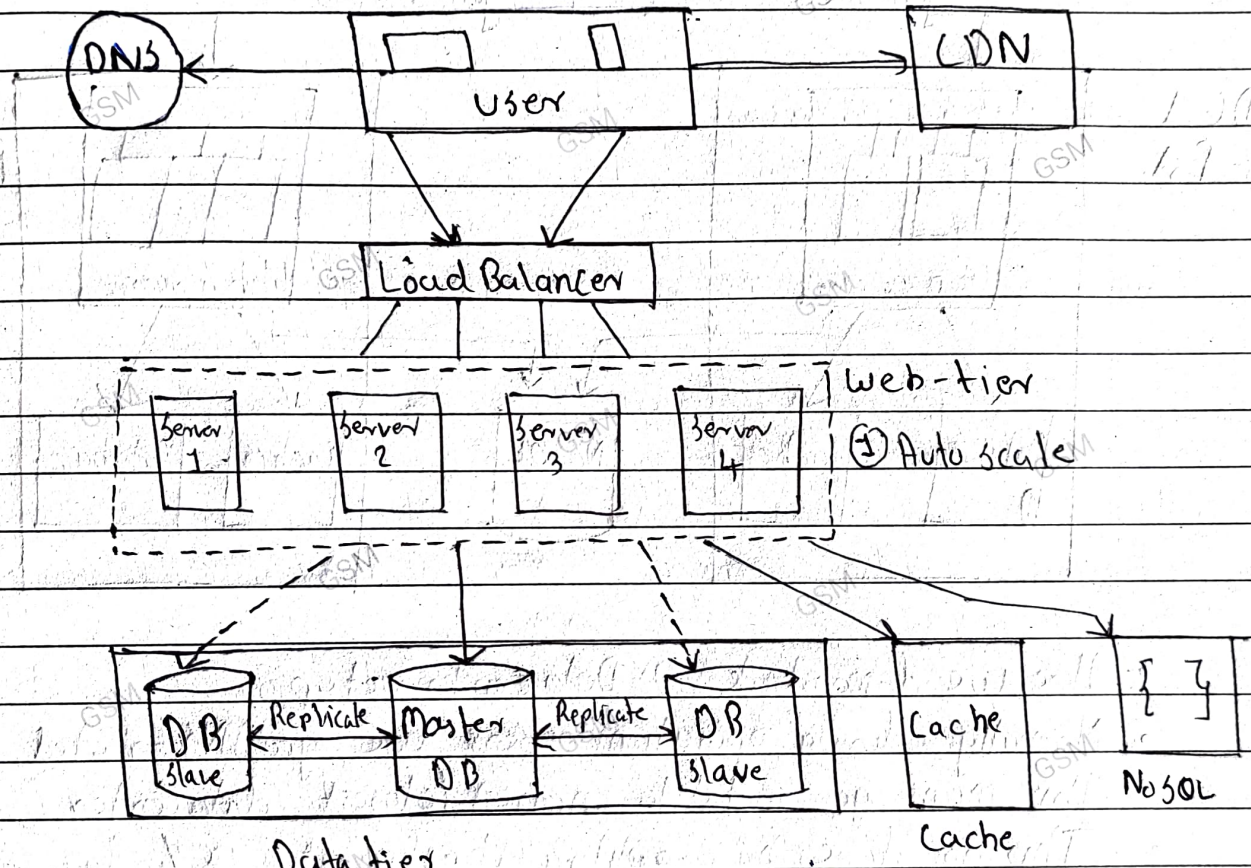
The issue is to route same client to the same server this can be done with sticky sessions in most load balancer however this adds the overhead. Adding or removing server becomes more difficult also it is difficult to handle server failure.

☆ Stateless architecture.



- In this stateless architecture, HTTP request from user can be sent to any of the server which will further fetch the state from the shared storage.
- A state data is kept out of web servers thus this system is simpler, more robust and scalable.

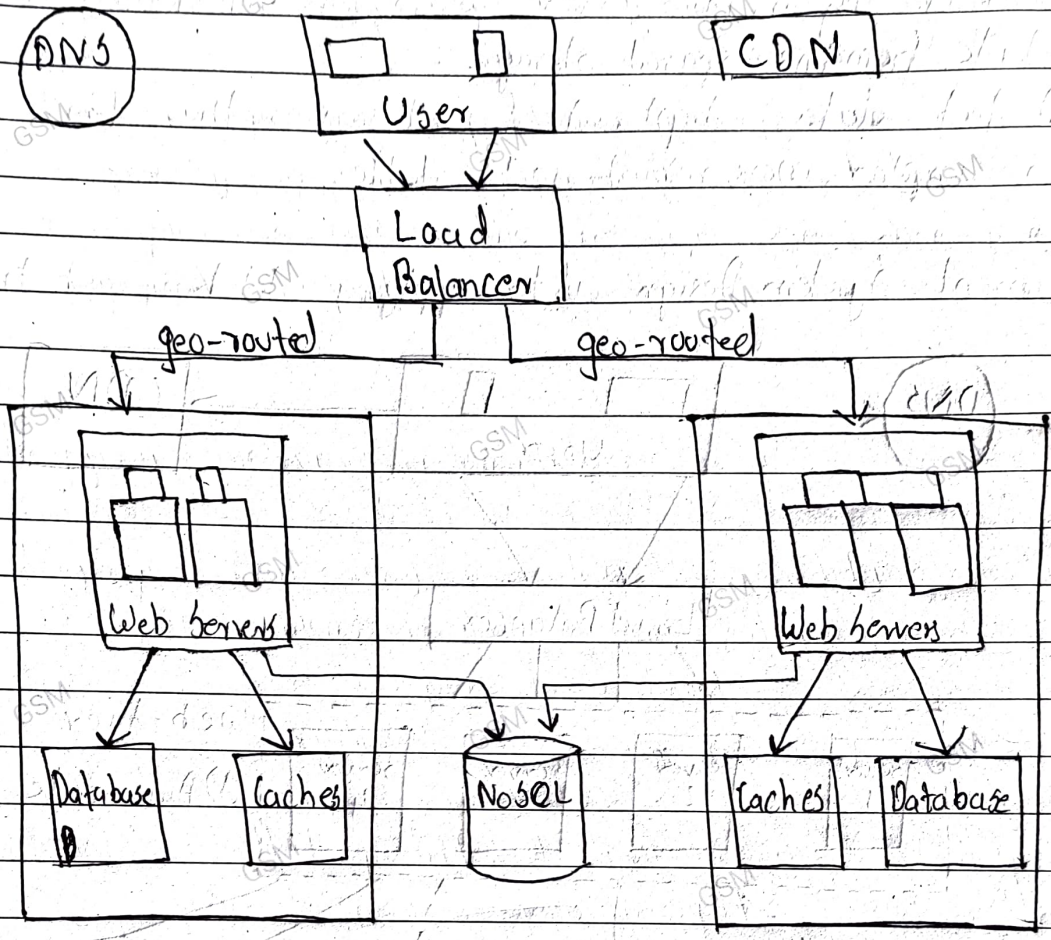
Current System Design after applying stateless web tier.



- Here we move data from web-tier & save it in persistent datastore
- Shared datastore can be relational or NOSQL. The NoSQL can be choosen as it is easy to scale.
- Autoscalling means adding or removing web-server depending upon the traffic load. As state data is shifted auto-scalling is more easier.

★ To improve availability and user experience for wider geographical area supporting multiple data center is crucial.

☆ Data centers



- Here, we demonstrate 2 Data center setup
- Normally during operation users are geoDNS-routed, this means geo-routed to nearest/closest data center
- This is done by spliting traffic of x% to US-East and (100 - x%) to US-West.
- geoDNS is a DNS service that allows domain name to be resolved to IP address based on the location of the user.
- In the event of any significant data center outage all the traffic is routed to a healthy data center.

Challanges to be resolved to achieve multi-data center setup

i) Traffic redirection:- GeoDNS is used to direct right traffic to the correct and nearest data center

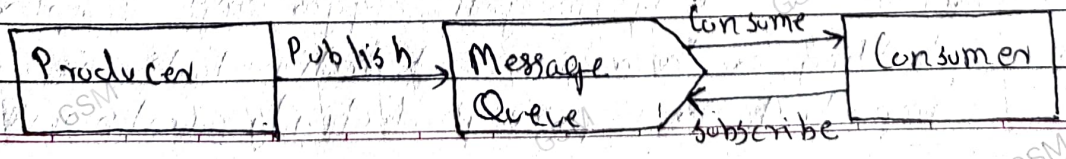
ii) Data synchronization:- Users from different location can use different local database. In case of failover the traffic might be routed to the datacenter where data is not available. A common solution is data replication across multiple data center

iii) Test and Deployment:- With multiple datacenter setup it is important to test whether the app/site is accessible through different locations.
Automated deployment tools are vital to keep system & services consistent throughout all data centers.

To further scale our system, we should decouple different components of the system so they can be scaled independently. Messaging queue is a key strategy employed by many real-world distributed system to solve this problem.

Message Queue

- A durable component stored in memory, that supports asynchronous communication. It serves as buffer & provide distributing asynchronous requests.



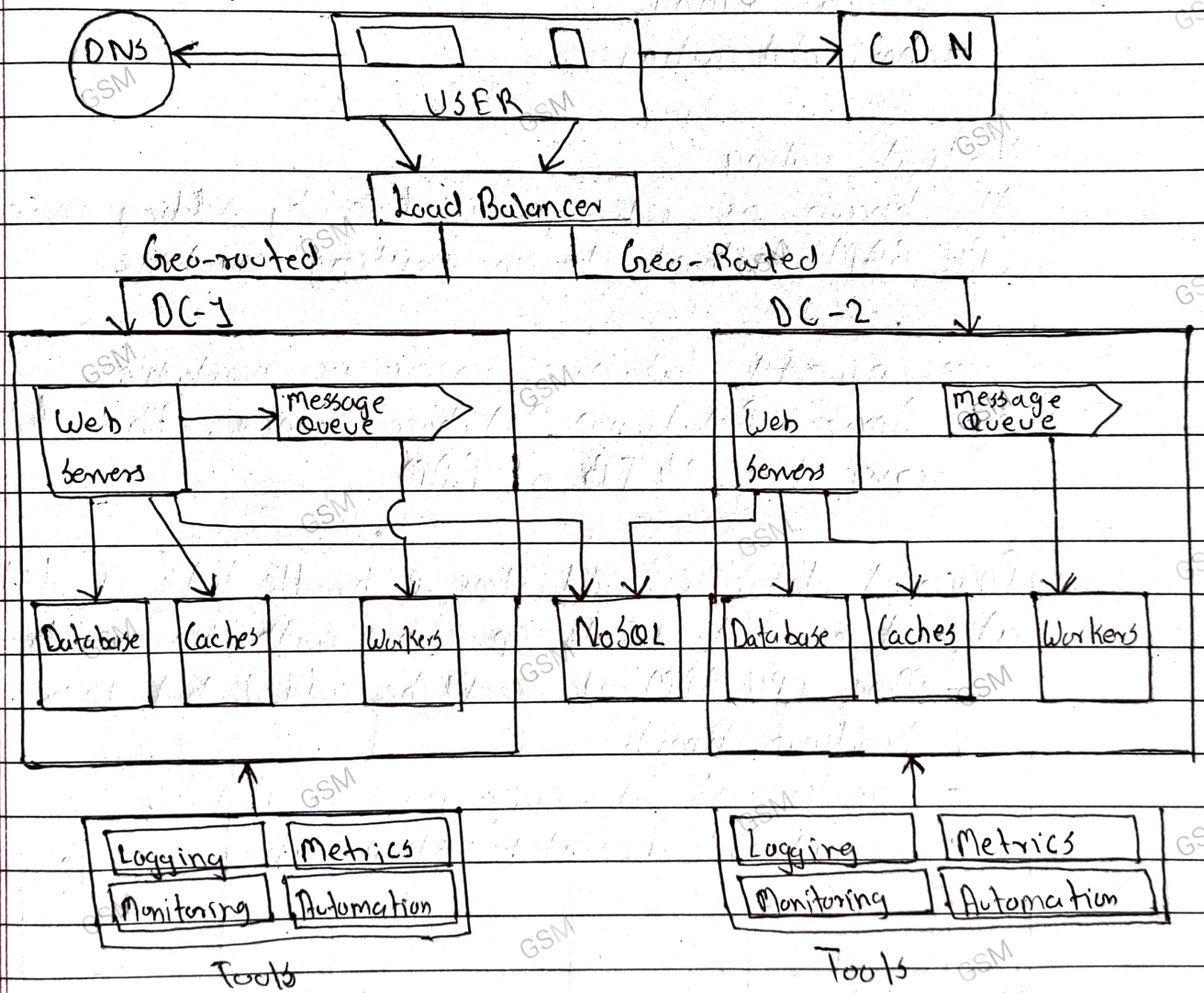
- Decoupling makes the message queue a preferred architecture for building a scalable and reliable application.
- With message queue, publisher can publish message to the queue when consumer is unavailable & consumer can access the message even when publisher is not available.

Logging, metrics, automation

- Logging :- Monitoring error logs to identify errors and problems in the system. Logging can be done at per server level or we can centralized service for easy search & viewing.
- Metrics : Collecting different types of metrics helps to gain bsiness ~~insigths~~ insights and health status of our system.
  Some important metrics are :-
  i) Host level metrics :- CPU, Memory, disk I/O etc
  ii) Aggregated level :- performance of the entire database tier, cache tier etc.
  iii) Key business metric :- daily active users, retention, revenue, etc
- Automation : As a system gets bigger and complex, building or leveraging automation tools is important to improve productivity. Continious

integration is a good practice, in which each code checks-in is verified through automation, allowing teams to detect problems early. Automating build, test, deploy process etc. can improve developing productivity.

☆ Adding message queues and different tools



- A design includes message queue which makes system more loosly coupled and failure resillent.
- Logging, monitoring, metric and automation tools are also included.

As now the system is well-suited for failure handling. As the database gets more overloaded. It's time to scale the data tier.

Database scaling

There are two broad approach for database scaling.
- Vertical scaling
- horizontal scaling

Vertical scaling

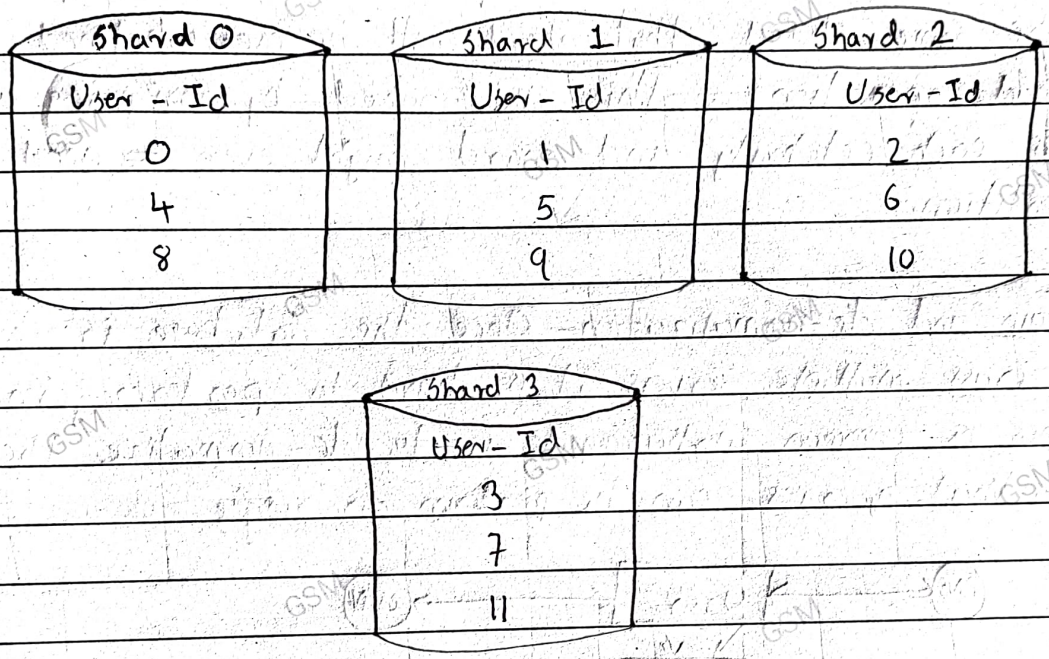
- Also known as scale-up, is done by adding more CPU, RAM, Disk etc to an existing machine
- Some powerful database servers are available eg:- Amazon Relational Database service, the database server has 24 TB of RAM
- Powerful database could store & handle lot's of data at a time but scale-up comes with serious drawbacks
  - More CPU, RAM etc could be added but there are hardware limits
  - Greater risk of single point of failures
  - Overall cost of ~~restrict~~ vertical scaling is high.

Horizontal scaling

- Also known as scale out or sharding is a pratice of adding more servers. Sharding seperates large database into smaller, easily managed ~~parts~~ parts. Each shards, shares the same schema, though actual data is unique on each shard.

SUBODH Page No.
Date

- Important factor while implementing shrading is to choose the shrading key.
- Shruding key also known as primary key consist of one ~~or~~ or more columns that determines how the data is distributed.



- Here the user-Id is the sharding key.
- It helps to efficiently retrive and modify data by routing database queries to correct database.
- Sharding key should be the one which is evenly distributed

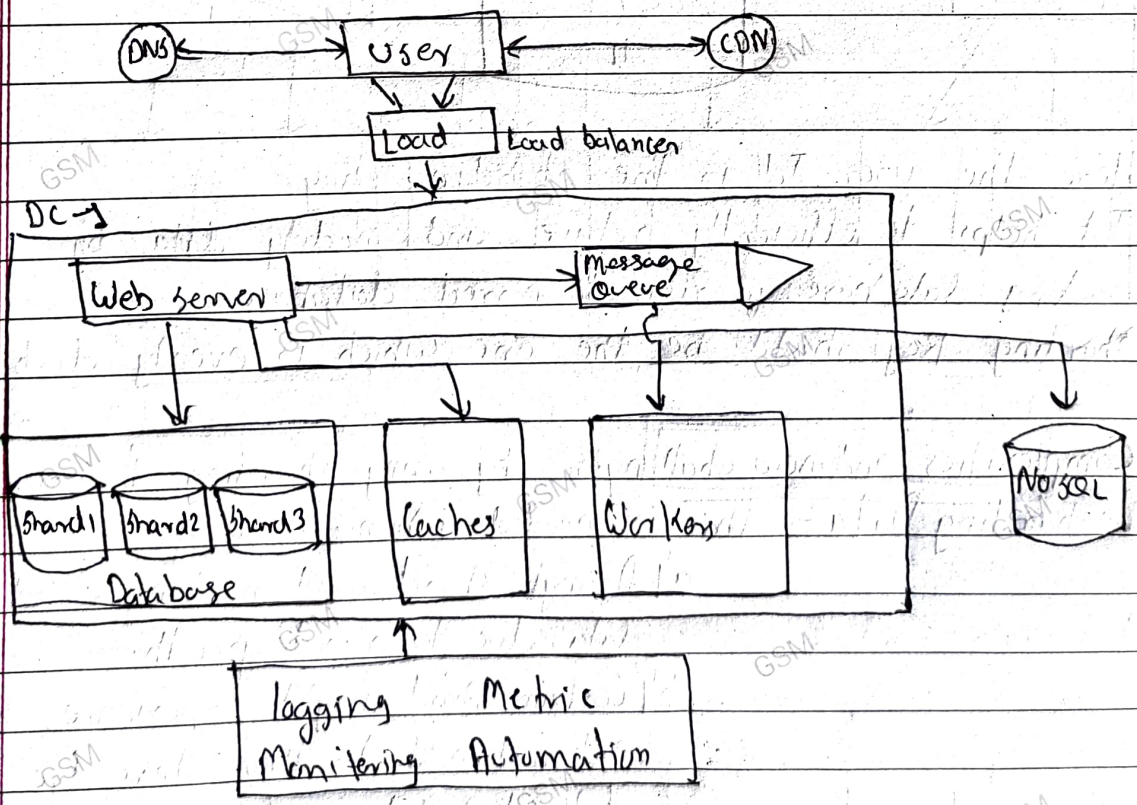
Complexities and new challanges by using sharding technique

i) Resharding Data :- This is needed when
   a) A single shard could no longer hold more data due to rapid growth.
   b) Certain shard might experience shard exhaustion faster than others due to uneven data distribution.

- When shard exhaustion happens, it ~~requires~~ requires updating the sharding function and moving data around. Consistent hashing, is a commonly ~~use~~ used technique to solve this problem.

ii) Celebrity problem: This is also called hotspot key problem. Excessive access to a specific shard could cause server overload. If data of multiple celebrity ended up on the same shard, that shard will be overwhelmed with read operations. This is ~~a~~ solved by assigning a shard to each celebrity and shard might even ~~ee~~ need further partition.

iii) Join and de-normalization- Once the database is sharded across multiple servers it is hard to perform join operation. This a common workaround is to de-normalize the database so that queries can be perform on single table.



SUBODH Page No.
Date / /

☆ Millions of user and beyond.

- System scalling is an itterative process of what we have learned

- More fine tuning and new strategies are required to scale beyond millions of users.

- Optimize the system and decouple the system into even smaller services.

- Key takeaway

  - Keep web-tier stateless
  - build redundancy at every tier
  - Cache data as much as possible
  - Support multiple data centers
  - Host static ~~ass~~ assets in CDN
  - Scale data tier by sharding
  - Split tiers into individual services
  - Monitor the system and use automation tools.