BASICS OF SYSTEM DESIGN

↳ System design playlist.

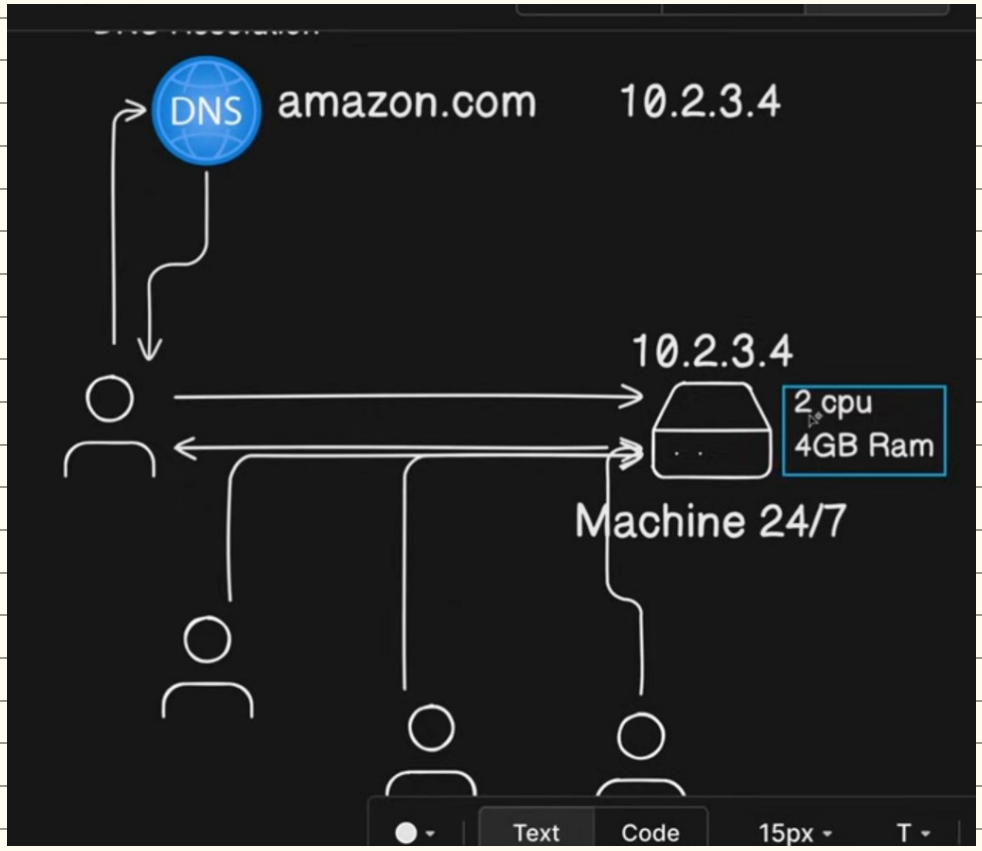
→ **Designing Amazon Ecommerce :-** A system must be fault tolerant.

Client → 10.2.3.4. machine (servers) 24/7

amazon.com.

DNS :- Domain Naming System. (Works in decentralized way).

∴ When the traffic increases, there are no enough physical resource to handle that much traffic. Ex :- 2 cpu 4 GB



- High CPUs and GPUs are not always required.
  - ↳ few times there's a hike in traffic and a few times the traffic is very low.

→ Vertical Scaling :- Adding more resources to the servers.

Vertical scaling, also known as scaling up, involves increasing the capacity or capabilities of a single hardware or software component within a system. This is done by adding more resources like CPU, RAM, or disk space to an existing server or VM, or by upgrading the server to a more powerful instance. The goal is to improve the performance and capacity of the system to handle higher loads without adding additional servers.

Key aspects of vertical scaling:

Disadvantages of Vertical Scaling :-

(i). When we upgrade the server/machine we cannot do it when it is on, we will face a downtime.

✅ **What is Vertical Scaling?**

Vertical Scaling means **increasing the capacity of a single server** by adding more:

- CPU (processor power)
- RAM (memory)
- Storage
- Faster networking components

👉 You keep the same machine but make it stronger.

It is like upgrading your laptop from:
8GB RAM → 32GB RAM
i5 CPU → i9 CPU

🔷 **Where Vertical Scaling is Used (Examples)**

Industry / System	How Vertical Scaling Helps
Databases (MySQL, PostgreSQL, Oracle)	Add more RAM/CPU to handle bigger queries
Application servers	More processing power for traffic spikes
Virtual machines (AWS EC2, GCP Compute Engine)	Upgrade instance type for better performance
Monolithic applications	All processing done in one big server

Example in Cloud:

- AWS: upgrading **t3.medium → m5.4xlarge**
- GCP: upgrading **e2-medium → n2-highmem**
- Azure: VM upgrade from **B-series → D-series**

⭐ **Advantages of Vertical Scaling**

Benefit	Explanation
Simpler	No need to modify code or split services
Easy to manage	Single server; fewer complexities
Less latency	All operations inside one system
Good for databases	DB consistency is easier in one node

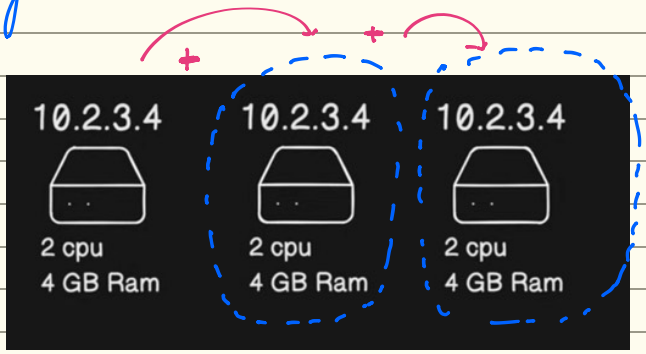
⚠️ Disadvantages of Vertical Scaling

Issue	Explanation
One single point of failure	If server goes down → whole system stops
Hardware limits	You cannot upgrade infinite RAM/CPU
Expensive at higher levels	Big machines cost much more
Downtime	Upgrading server may require shutdown

VS Vertical Scaling vs Horizontal Scaling

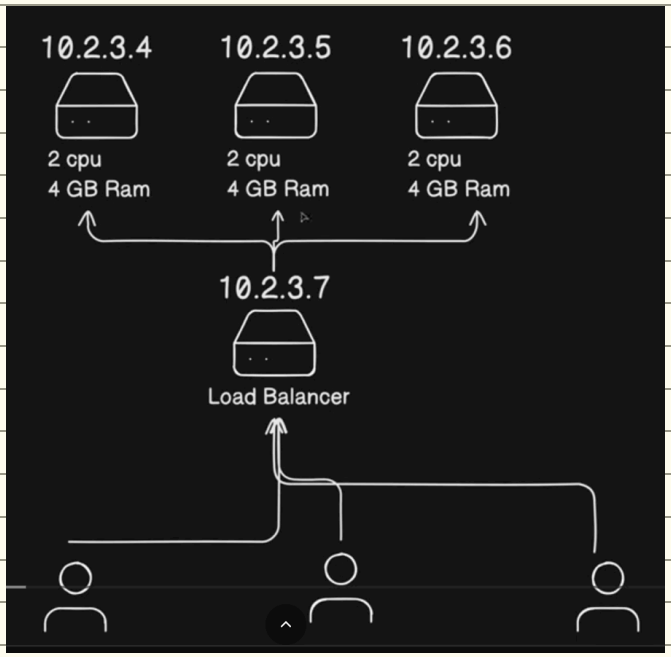
Feature	Vertical	Horizontal
Meaning	Scale "up" single machine	Scale "out" by adding more machines
Cost	Cheaper initially	Long-term more scalable
Fault tolerance	Low (1 server)	High (many servers)
Max capacity	Limited	Almost unlimited

→ Horizontal Scaling :-



- Whenever, the traffic increases, in Horizontal Scaling we spin up new machines side by side, hence facing zero down time.

- When we add new servers, we need to route the user traffic to the servers IP, and that's why we use Load Balancers.



→ Load Balancer :-

A **Load Balancer** is a device or service that **distributes incoming network traffic across multiple servers** so that no single server becomes overloaded.

Think of it as a **traffic police** for the internet:

- Users send requests → Load balancer decides which server will handle it.

🎯 Why are Load Balancers Used?

Reason	Explanation
Avoid server overload	Split traffic so all servers can handle requests well
Increase availability	If one server fails, traffic is sent to healthy ones
Improve performance	Faster response time by choosing the best server
Scalability	Easily add more servers behind the load balancer
Better security	Load balancer can hide internal servers from users

➡️ This ensures **High Availability & Reliability**.

⚙️ How Do Load Balancers Work?

1. User sends request → Example: www.google.com ↗
2. Load balancer receives it first (not the server)
3. It picks the **best** server using specific rules (algorithms)
4. It forwards the request to that server
5. Response goes back to user (directly or through LB)

🔁 Load Balancer Mechanisms (Algorithms)

Algorithm	How it works	Use case
Round Robin	Sends requests one by one to each server in order	Balanced equal-load servers
Least Connections	Chooses server with least active users	When server loads vary
IP Hash	Uses client's IP to always route to the same server	Session consistency required
Resource-based	Chooses server with free CPU/RAM	High-traffic dynamic apps
Weighted Round Robin	Stronger servers get more requests	Non-identical servers

🔍 Types of Load Balancers

Type	Layer	What it balances
L4 (Transport layer)	TCP/UDP	Based on IP & port
L7 (Application layer)	HTTP/HTTPS	Based on URLs, cookies, headers

Example:
Amazon ALB (Layer 7)
Amazon NLB (Layer 4)

🌐 Examples in Real Use

Scenario	Benefit
E-commerce like Amazon	Traffic spikes during sale
YouTube/Netflix streaming	Handles millions of requests per second
Banking apps	High security + session consistency
Cloud apps on AWS/GCP	Scale out easily

→ There are different Algorithms in Load Balancers for distributing load.

(1) · Round - Robin Algorithm :-

🔷 What is Round Robin in Load Balancing?

Round Robin is a **simple** load balancing technique where incoming requests are distributed to servers **one by one in a circular order**.

📌 Imagine servers standing in a circle.
Requests go: **Server1 → Server2 → Server3 → Server1 → Server2 ...**

No server gets too many requests compared to others (if they all are equal).

🧠 How It Works (Step-by-Step)

1. Load balancer receives a user request.
2. It sends that request to the **next server in a list**.
3. When it reaches the last server → it starts again from the first server.

📌 No performance check — all servers treated equally.

🔁 Example Scenario

Servers:

- S1
- S2
- S3

Requests come: R1, R2, R3, R4, R5, R6

Distribution:

```nginx
R1 → S1
R2 → S2
R3 → S3
R4 → S1
R5 → S2
R6 → S3
```

➡️ Fair distribution for equal-power servers

🔄 Variations

Variant	Explanation
Pure Round Robin	Simple circular order
Weighted Round Robin	Stronger servers get more requests

Example: if S1 is twice as powerful:

```
S1, S1, S2, S3, S1, S1, S2, S3 ...
```

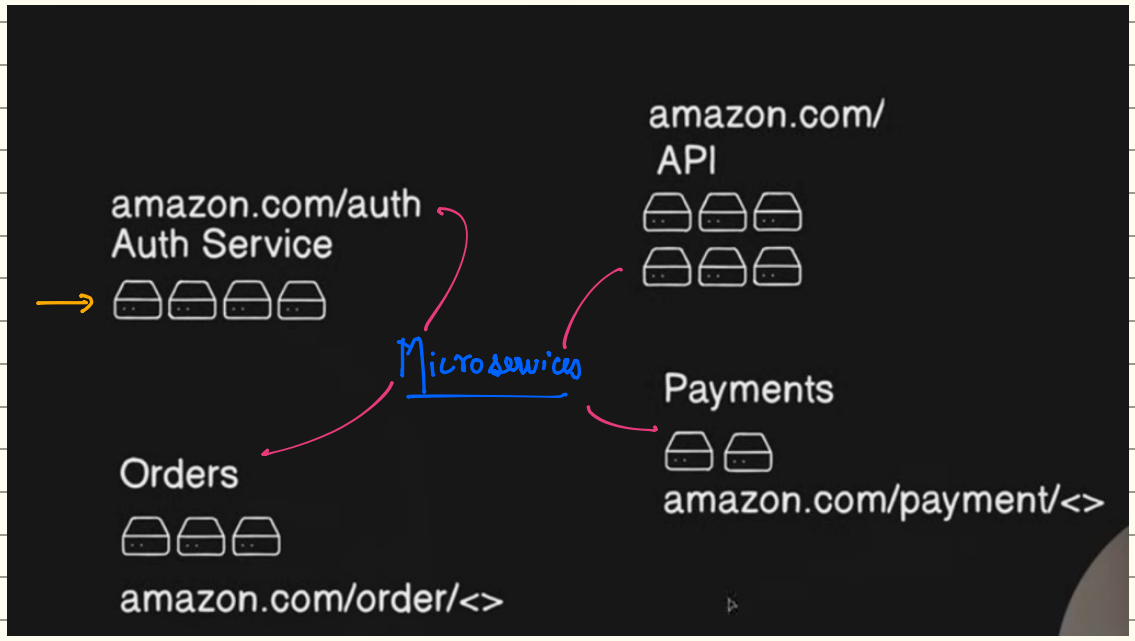
⭐ Advantages

Benefit	Why it is good
Very easy to implement	No complex calculations
Good for equal servers	Each gets fair load
Efficient performance	Distributes traffic quickly

⚠️ Disadvantages

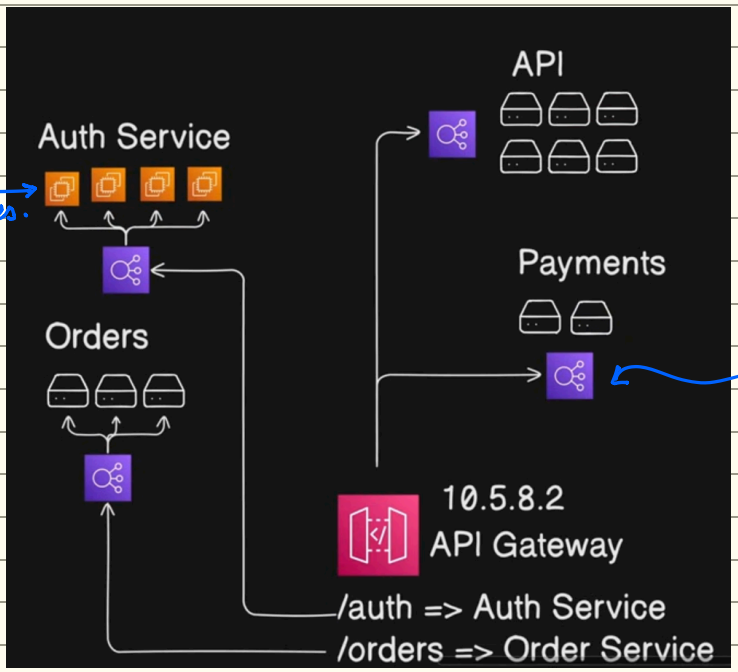
Issue	Why it matters
Does not check server load	Busy server may still get traffic
Not good for dynamic requests	Some servers may overload faster
Sessions may break	User may move to another server each request

👉 In apps where users need to stay on the same server (login session), additional technique needed:
✨ **Sticky Sessions**

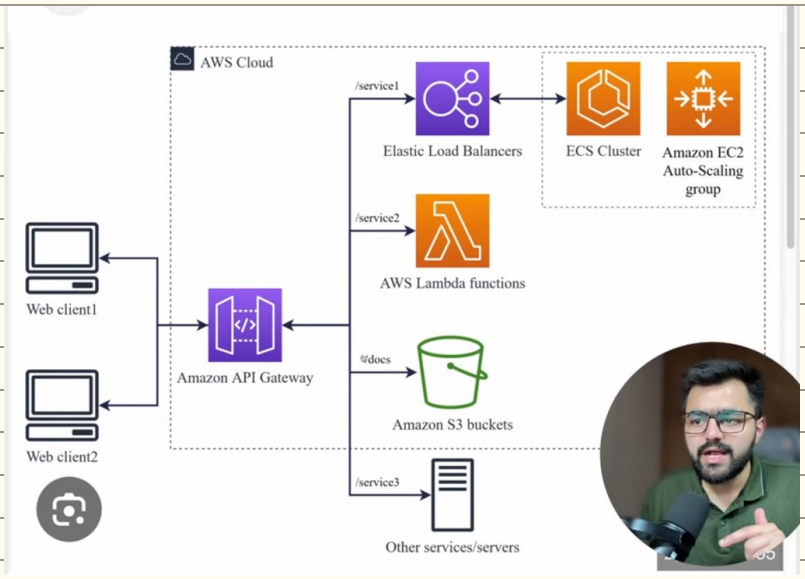


Servers dedicated for different service.

→ Now the problem arises of distributing different requests to their route specific services.
to do that we will add different load balancer to different microservices.



Working of API Gateways:-



→ API Gateway :-

An API gateway is a centralized entry point for API calls, acting as a reverse proxy that routes requests from clients to backend services and back. It handles tasks like authentication, authorization, load balancing, and rate limiting, ensuring secure and efficient communication between clients and services.

→ Batch process and Bulk workers:-

✅ Batch Process

A **batch process** is a type of processing where multiple tasks or data items are collected and executed **together as a group** rather than one by one in real-time.

🔹 **Key Characteristics:**

- Processes data **in chunks** (batches)
- Usually runs at **scheduled times** (e.g., every hour, nightly job)
- Good for **large volume** tasks that do not need instant results
- Uses system resources efficiently

🔹 **Examples:**

- End-of-day billing in banks
- Monthly salary payroll processing
- Aggregating analytics data every 24 hours
- Batch upload of 10,000 customer records

🚢 Bulk Workers

Bulk workers are **background workers** that handle **large workloads** asynchronously (not blocking user operations).

You can think of bulk workers as **engines** that run batch jobs.

They are used when:

- Processing tasks might take longer (e.g., generating reports)
- Request should not make users wait
- Tasks can be retried if failed

Where are bulk workers used?

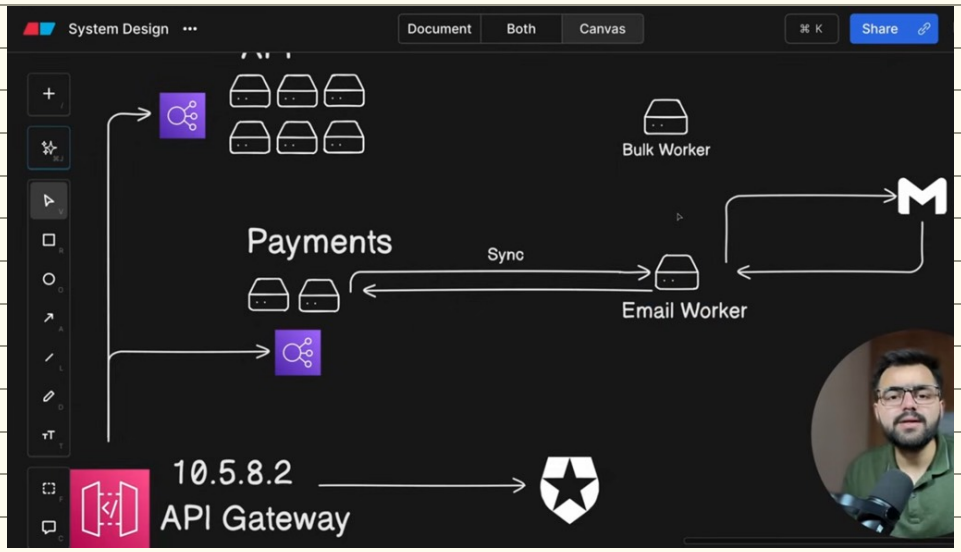
- **Message Queue + Worker model**
  - Example: **RabbitMQ, Kafka, AWS SQS, Celery, Sidekiq**
- Worker pulls jobs from a queue in bulk and processes them

Difference (Simple Comparison)

Feature	Batch Process	Bulk Worker
When data is processed	At scheduled times	As soon as tasks arrive in queue
User wait time	No immediate result needed	No waiting, task handled later
Operational Type	Group processing	Background job processing
Typical Use	Reports, payroll, data aggregation	Emails, notifications, file processing

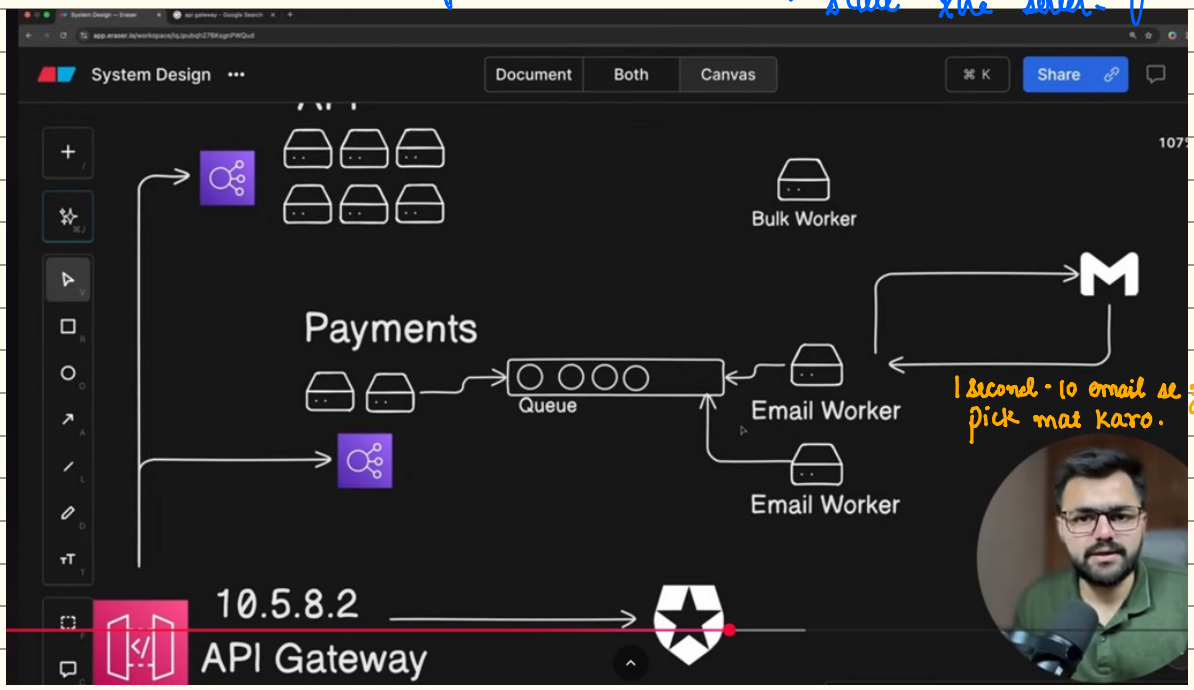
→ Now, suppose we want to bulk mail users.

Synchronous job.



To improve parallelism, we queues the emails, and again to scale this process, we horizontally scale the servers.

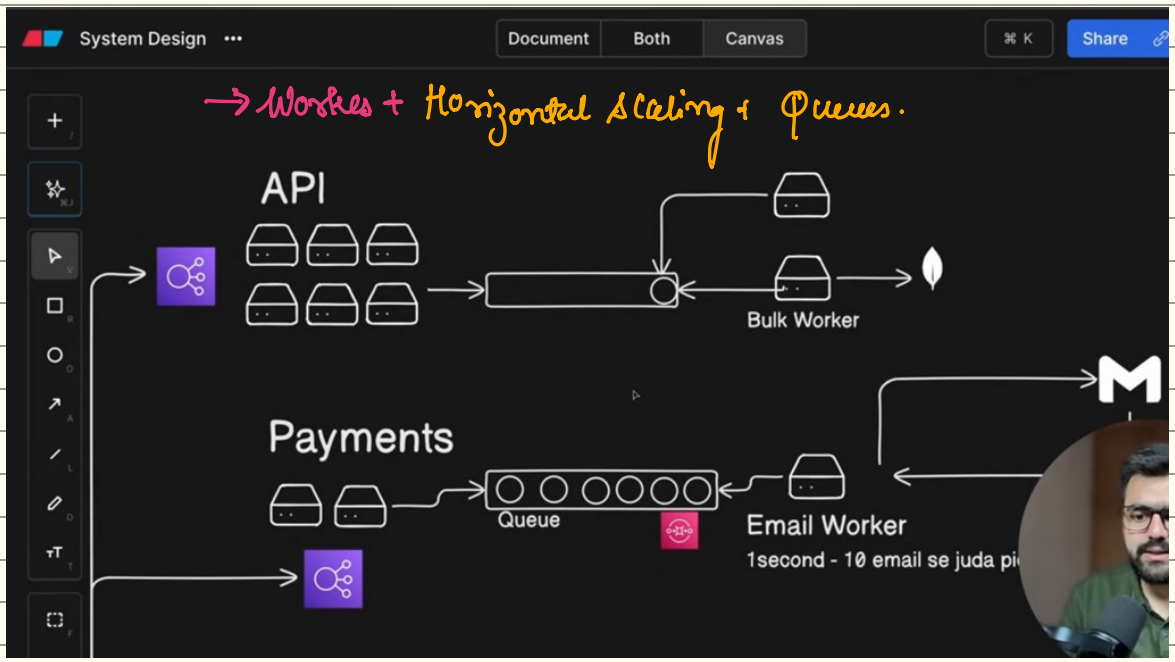
Asynchronous server



in Amazon for queue, we have SQS:- Simple queue system.

SQS mechanism :- (SQS is a polling mechanism).

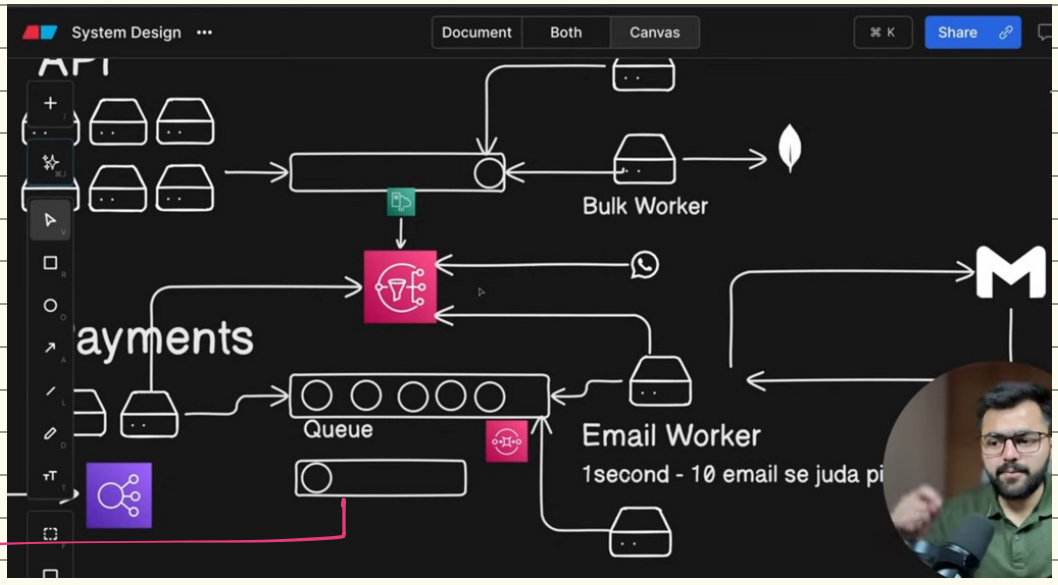
- pull mechanism
  - uses polling to check for new events
    - long poll
    - short poll.
- push mechanism.
  - new events are automatically pushed.



→ SNS :- Simple Notification Service :-

↳ uses PUB/SUB mechanism.

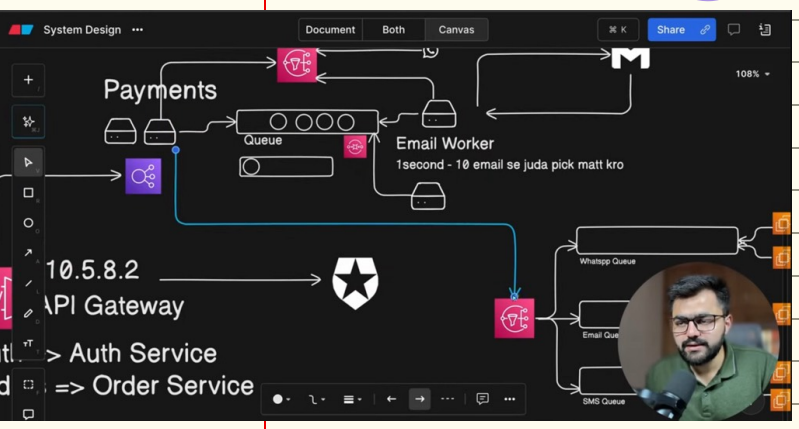
↳ These type of architecture is called event-driven architecture.



Dead later Queues :-

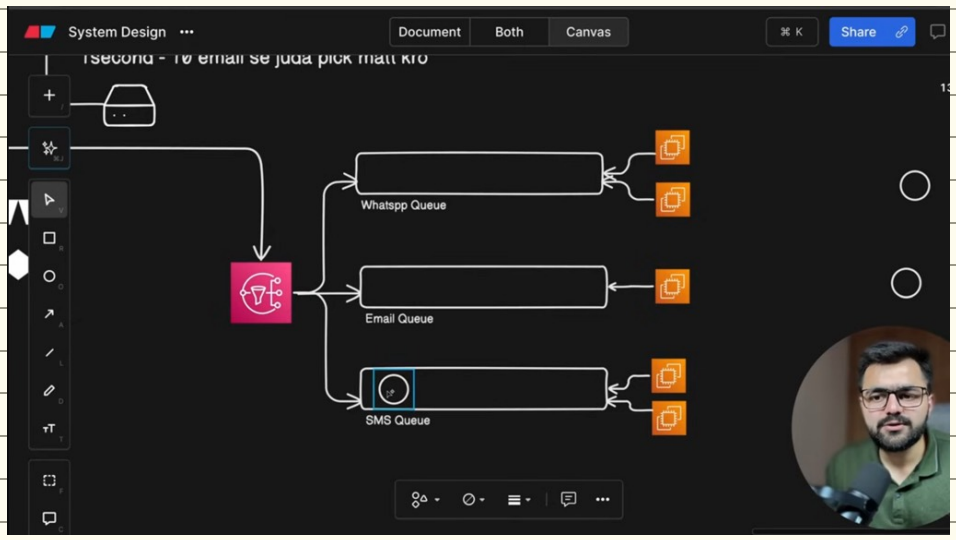
if the event fail keep it in a DLQ and retry it after sometime.

→ Event Fan out Architecture :-

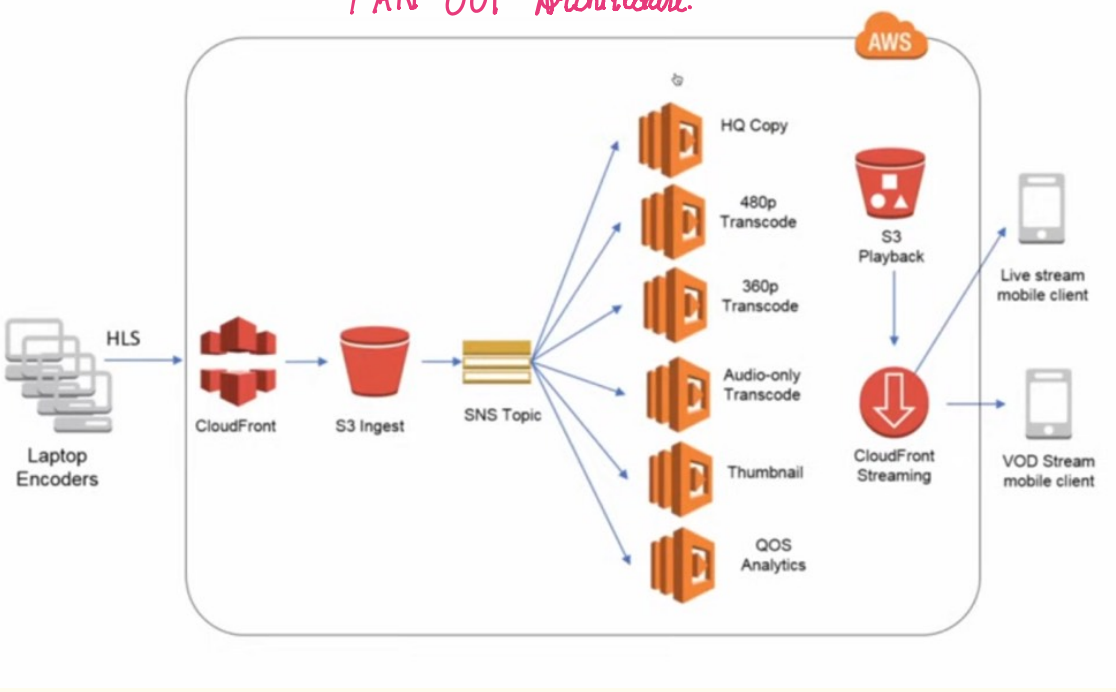


(i) Using FOA, we were able to notify multiple mutiple services/user.

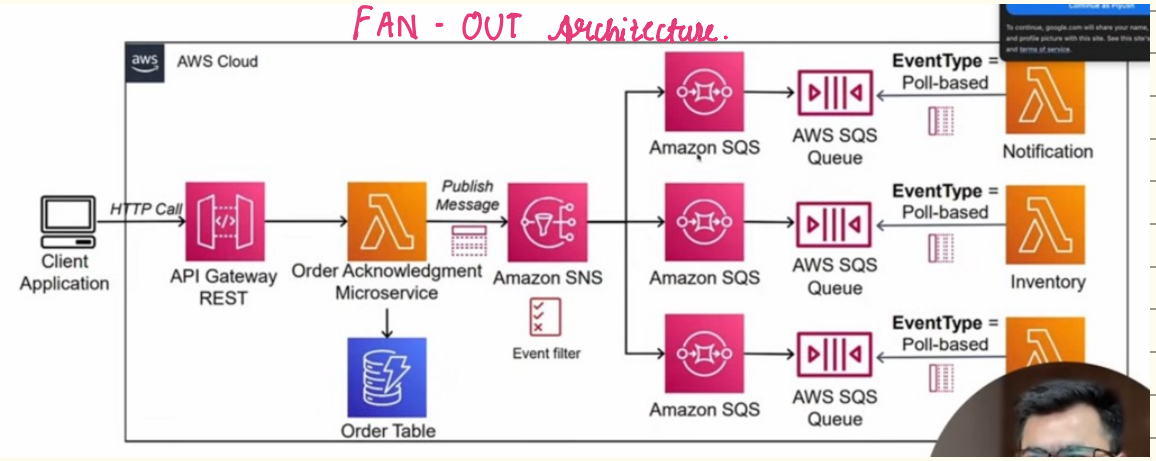
(ii). We can track down events due to retry acknowledgement mechanism.



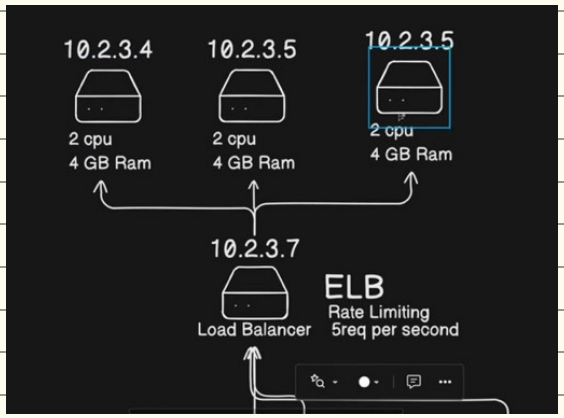
FAN-OUT Architecture.



FAN - OUT Architecture.



→ Rate Limiting :- Rate limiting is a strategy used to control how many requests a user/system/client can make to a service within a specific time period.



Strategies in Rate Limiting:-

(i). Leaky Bucket.

(ii) Token Bucket.

🔹 Why is Rate Limiting needed?

- Prevent **DDoS or spam attacks**
- Protect backend from **overload**
- Ensure **fair usage** among users
- Prevent **API abuse**
- Manage **resources efficiently** (CPU, DB queries, network)

Example:

> "Only 100 requests per minute allowed per user."

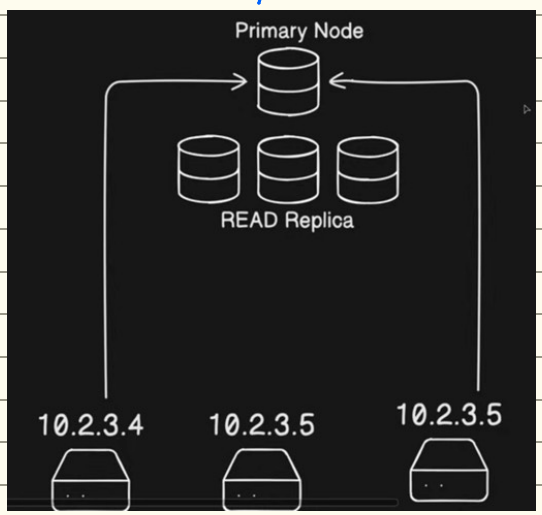
If limit is crossed → block, delay, or reject further requests with HTTP status **429 - Too Many Requests**.

⚙️ Common Rate Limiting Strategies

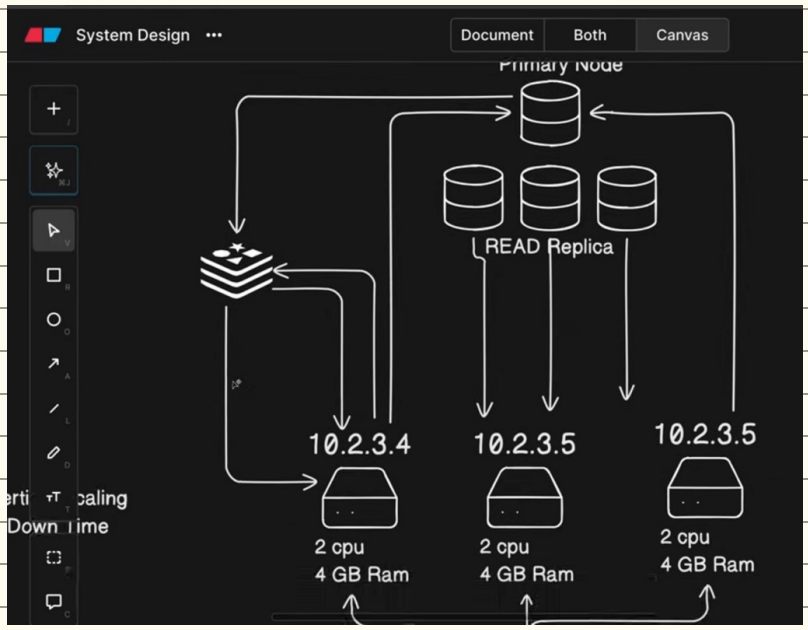
Strategy	How it works	Pros	Cons
Fixed Window	Count requests in fixed intervals (e.g., every 1 min)	Simple	Bursts can exceed limits at window edge
Sliding/Moving Window	Uses timestamps to check requests in rolling time window	Fair & accurate	Costly → needs timestamps storage
Token Bucket	Tokens fill in bucket at a fixed rate; each request uses a token	Allows bursts, commonly used	Complex to implement
Leaky Bucket	Requests go into a queue & leak out at constant rate	Smooth request flow	Burst requests may be dropped
Rate Limiting at Load Balancer / API Gateway	Global threshold across infrastructure	Centralized control	Gateway cost & dependency
User/Client-based Throttling	Different limits for each user tier (Free vs Premium)	Supports monetization	Needs authentication tracking

→ DataBase Scaling :-

(i). Maintain Read Replicas :-

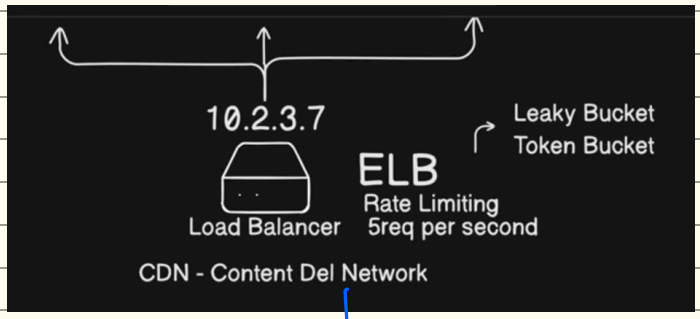


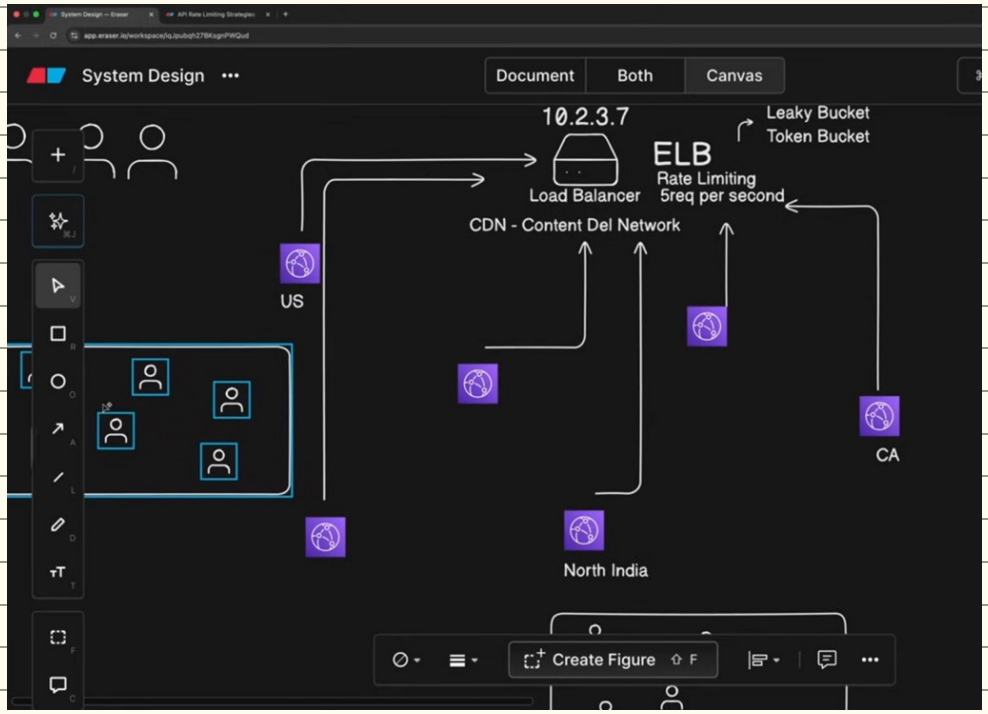
(ii). Caching :- Used Redis (in App memory) caching.



→ CDN - Content delivery Network.

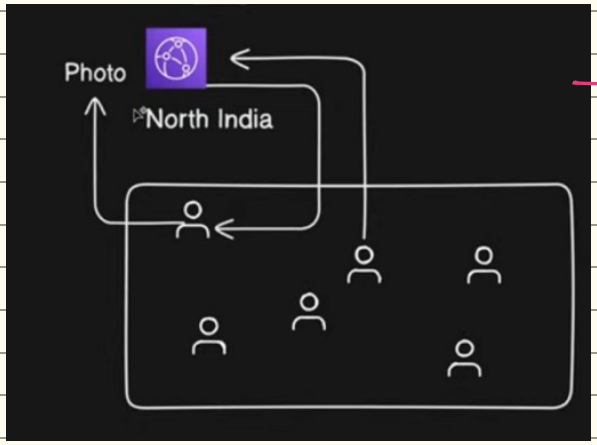
We generally do not expose the load Balancer directly, we use CDN instead





Users from different regions request to their cloudfront (CDN)

→ As the IP of diffrent CDNs differ, we use ANYCAST



→ The user from a region, requests some media, and for the first time the load balancer serves the request and the S3 caches it.

Later if someone from the same region requests the same media, the CDN won't request the load balancer.

→ ANYCAST :- Anycast is a network routing technique where multiple servers share the same IP address, and traffic is automatically routed to the nearest or best-performing server based on routing protocols.

📌 In short:

> One IP → Many servers → Client is routed to the closest one

🎯 Why Anycast is used?

Benefit	Description
Low latency	Users get connected to the closest server geographically
High availability	If one server fails, traffic automatically reroutes to others
Load distribution	Traffic naturally spreads based on closeness and network health
DDoS mitigation	Attack traffic gets distributed across several servers

Uses **BGP (Border Gateway Protocol)** to advertise the same IP address from multiple locations.

arduino — Copy code

```
Client
  ↓ (nearest route)
┌─────────┐
│ Anycast │ Same IP everywhere
└─────────┘
  ↙      ↓      ↘
Server A Server B Server C
(USA)   (India)   (Europe)
```

Client from India → Server B
Client from USA → Server A
Client from Germany → Server C