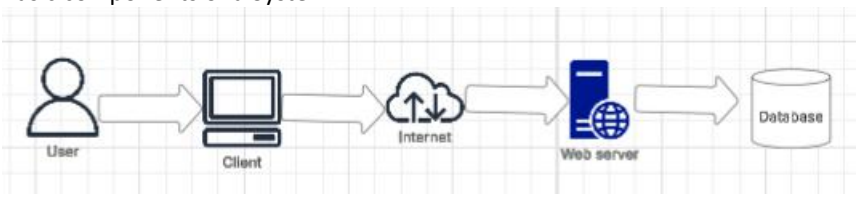
FaultTolerantTechniques - part II

3 majors concerns of System design:
1. Reliability
2. Scalability
3. Maintability

Scalability:
Scaling means adding resources to a system to we can cater to a larger audience
LOAD and LOAD PARAMS like Memory usage, Request per time, Network calls

Basic components of a System



Scaling is adding more resources or power to our system

We cant tell our Users to upgrade their system. We can scale our webserver and database

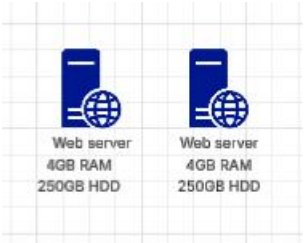
Horizontal scaling and Vertical scaling
Vertical scaling:

- Single machine being upgraded (CPU --> 4 core to 16 core, RAM --> 8GB to 16GB, HDD --> 500GB to 1TB)
- Easier to upgrade single machine because our code is already there on the machine
- For a small organization or application, Vertical scaling is better

Problem is, there is a limit, we cant upgrade beyond that limit.
Limitation is, Vertical system has limit and it could get expensive over time. Single point of failure, Tedious to scale up on emergency times

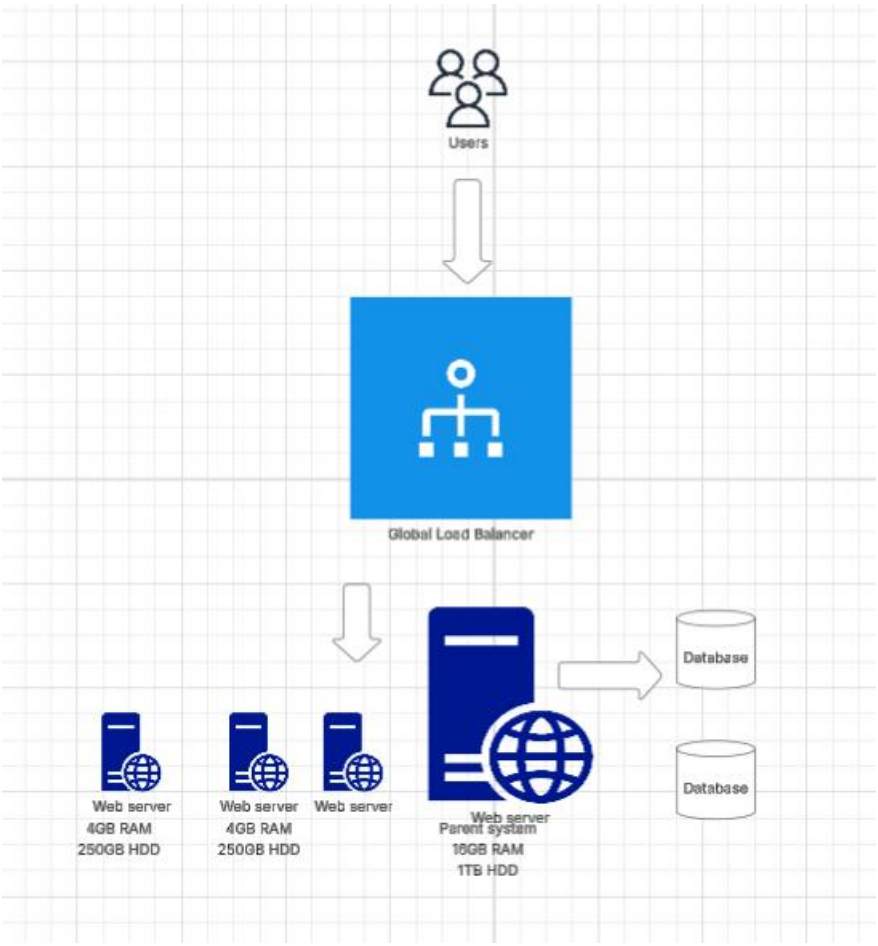
Horizontal scaling:

- We are adding more systems to one system. Instead of upgrading one Webserver, we are adding more webservers



Instead of one Webserver, we purchase another Webserver

- In some case, Horizontal scaling could be cheaper than Vertical
- We could have a bunch of smaller Webservers then one Parent system with a larger capacity that could talk to Databases



A load balancer will say where the user requests should go. Which server, the user requests should go to. If any of the webservers go down, Loadbalancer can see and direct the requests accordingly. That means, there is a low chance of downtime. If we have a huge application, we could divide the huge code and put different code chunks on different webservers (Microservices)

Pros:

- Low downtime (almost negligible)
- Scale the system infinitely (Infinite scaling)
- Parallel computing
- Multiple backups of data
- Resource sharing
- Cost efficient (Reasonable pricing)
- Scability per need

Cons:

- Complex to handle (Loadbalancer, Distributed cache, Healthcheck)
- Debugging harder due to complexity
- Loadbalancer issues can bring down the entire system

If Loadbalancer is down, what will happen to the system? It will be discussed in the upcoming classes

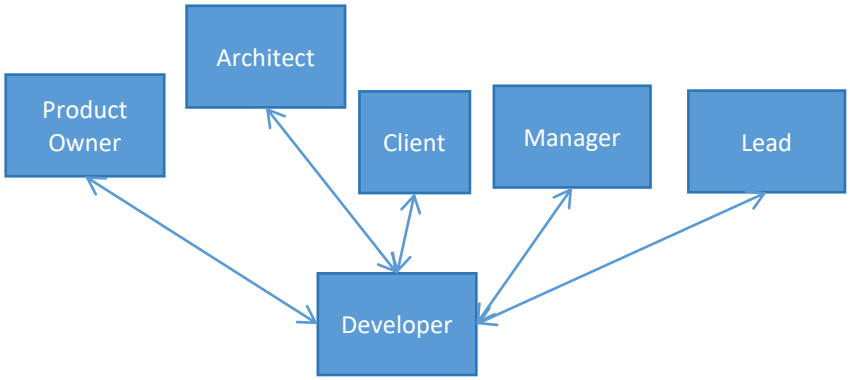
Maintainability:

Our system should be built in a particular form or structure so that it can be maintained

3 factors:

- Operability (if we want to perform any operations, our system should be able to handle those operations). for example, YouTube has two view options: 1. User and 2. Content Creator (where User can see Analytics, Views, Likes). Say if something doesn't work and if the User raises a ticket, it should eventually flow down to the Development team and that stream should not be broken. Our system should be working and open for Operations like checking metrics on Dashboard

- Simplicity (simplicity in Coding --> Developer that's going to work on our software should be able to understand the code easily)
- Evolvability (If we are creating a WhatsApp like application, say we have a chat, we should be able to evolve this application, say we want to add 'Reactions' to chat, we should be able to do. How ChatGPT application works is, it takes entire chat into database and eventually gives you the response. Only because it saves the entire conversation history into database, it is able to create next content based on that.



For every estimation, it goes down to the Developer. Developer is the center of Software development industry. SDLC revolves around the Developer. Every Developer should ask why do we have this Feature. Can we have it in a better way? So Developer got to know about other functions because if anything breaks all the Support systems will come to you only

Service Level Agreement (SLA)
Dominos pomises to deliver the pizza under 30 minutes.

Service Credits

Service Credits are calculated as a percentage of the following charges paid by you for Athena for the monthly billing cycle in which the Monthly Uptime Percentage for a given AWS region fell within the ranges set forth in the table below:

Monthly Uptime Percentage	Service Credit Percentage
Less than 99.9% but equal to or greater than 99.0%	10%
Less than 99.0% but equal to or greater than 95.0%	25%
Less than 95.0%	100%

According to AWS, if a service is available <99.9 and >99.0% then they will give a credit percent of 10%. How did they get to this point though? For SLA, there is an internal term: SLO (Service Level Objectives), collection of SLOs is SLA. SLA is for customers or end-users, SLO is for internal.
Say I can deliver pizza under 30 min then I know internally I can deliver pizza in 25 min. That means, P99 is 25 min, 99% of the data points are under 25 min. With SLO, another additional term, SLI (Service Level Indicator), Indicator is lik datapoints we delivered at 15 min, 10 min, 25 min etc. Based on the datapoints or indicators, we can say that majorly we are able to deliver in 25min.

Example:
We are building Instagram, the User can either create a Post or follow other users. That means other users also can follow this particular user. Say they are two users, one is a normal user with 50 followers and another is a celebrity with 50million followers.
User --> Schema(ID, Name)
Follow --> Schema(follower.ID, following.ID)

User

ID	Name
1	ABC
2	XYZ

Follow

Follower.ID	Following.ID
1	2

Say ABC is following XYZ. ABC is a follower of XYZ but not vice-versa

Post

ID	Type	Content	User.ID
1	Image	pic1.jpg	1

Say User ABC has posted a pic, it must be pushed to his/her 50 followers

Content

ID	Post.ID	Time
1	1	

Say ABC has 50 followers but he is following 100 people. that's means he gets multiple updates and popups whenever they post anything. If ABC posts something, 50 followers will get a Notification.
ABC ---> Post ---> Notification ---> Followers.
Say, we want to see real-time updates of only a few we are following but not all of them we are following. How do we selectively see contents among the ones we are following?
Two kinds of Notification:
1. Push --> Push notification to all followers. That means, whenever User creates a Post, it will go directly into the timeline of all the followers
2. Pull --> Whenever User opens an application or whenever they want, they will pull the Notifications. Instead of Pushing the posts to the timeline and sending a notification/popup, we will pull or fetch the posts only when we Log-in. We wont know about the posts until we log-in

System design will be covered after Databases