System Design

M T W T F S S
Page No.:
Date: 26/01/25
YOUVA

How to start with Distributed Systems

↳ Pizza Parlour ⇒ 1 chef (How to handle more order??)

(1) [VERTICLE SCALING]

↳ As a manager ⇒ Do more hardwork → Get more Salary → 1 chef

Verticle Scaling ↳ optimise processes + Increase throughput → Scale Resources

Chef → computer

(Preprocessing & cornJob)

(2) Preparing Beforehand at non-peak hours → Optimising Processes

Making System Resilience (Recovering from faliure Quickly)

(3) If chef is sick for 1 day ⇒ There wont be any buisness

Single Point of Failure

↳ Hire a backup chef → Only for particular day + pay money/by

↳ Keep backups and avoid single points of failure.

(Backups)

Master → Slave Architecture

(master chef) → (slave chef) (To avoid single point of failure)

(4) If the demand is huge ⇒ Hire more chefs (eg 10 chefs)

↳ Hire more resources (HORIZONTAL SCALING)

2 types of order A & B we have 10 chef few good at A and few at B

(5) Microservice Architecture ↔ Random assignment of order (X)

↳ Build on the strength (✓)

well defined Responsibilities ⇒ Scalable to a large extend

(6) Distributed System (Partitioning) → [when having the backup of the shop incase we have no electricity in one shop, other can deliver

[↳ More fault Tolerant

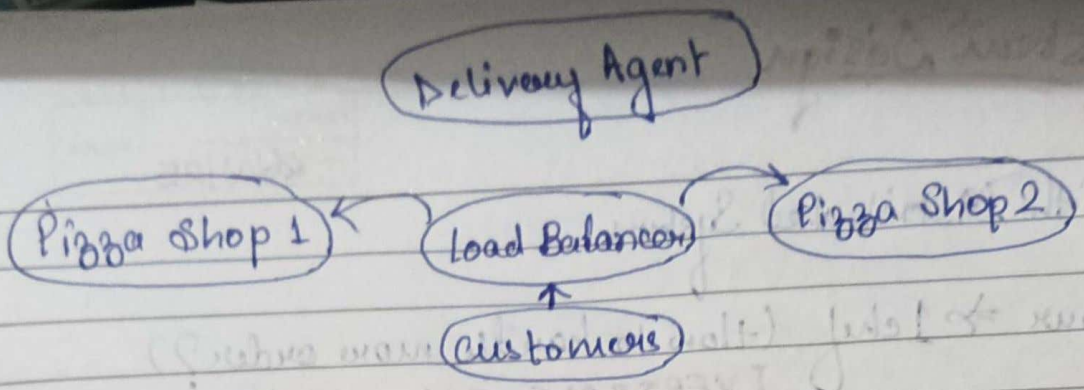
↳ Gives Quicker Response Time

* any order local to the 2nd shop can be delivered by it

(7) Load Balancer

→ * To give quick responses to the request globally

we need local servers serving/share



1. Every time a customer Makes a request
2. Pizza Shop needs to decide to send it to Shop 1 Or Shop 2
3. ~~Send~~ customer send it to central place that will route the request to either shop 1 or shop 2
4. Based on predifined parameter the the central placed route the request to either 1 or 2 [Eg. the parameter is how much time required to deliver the pizza]
5. This central authority that route the request is called a ~~5. This~~ LOAD BALANCER
6. The main moto of the delivery agent is to deliver the pizza to customer quickly

7. Seperation of Responsibility ⇒ Pizza shop manager is not delivering the pizza to customer. Different people has different roles.

(8) Decoupling The System ⇒ Seperating out concers so that you can handle seperate System more efficiently.

⁂ Let say → Pizza shop 1 → Faulty Oven → Cooking Time Increase
→ Bike issue with delivery Boy → Delivery Time Increase

(9) Logging & Metrics Calculation → To understand & fix the issue we have to log everything to see at what time Someting happen and what is the next event. Taking the events condensing them, finding sence out of them ⇒ Metrics
① Analytic ② Auditing ③ Reporting ④ Machine Learning

(10) Extensible :- To keep the system extensible.

As a backend engineer, you dont want to rewrite all the code again and again to serve a different purpose

Eg The delivery agent need not to know if we are delivering a bun

* The reason why we can scale out a business because we want to decouple everything to make sure that the system is extensible.

Summary [High Level Design]

1. Order Overload → Recruitment
2. Complexity → Seperation Of concerns
3. Mishaps → Fault Tolerance

LOW LEVEL DESIGN

High level → Deploying on servers → figuring out how2 system will interact with other

Low level design → Low level system has a lot more to do with how you are actually going to code

Eg making classes, making objects, functions, signatures

* How to write efficient code and clean code