Microservices :-

* Network protocols :- Helps comm. system. Application Layers

Client server | Peer 2 Peer

Client server protocols :

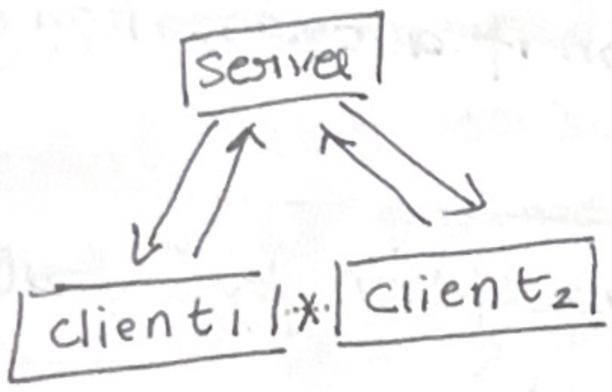
a) HTTP b) FTP c) SMTP d) Web Sockets

Peer 2 Peer : WebRTC → 2 connection

→ 1 connection

Client server → only client starts the conversation by sending request to server and server sends a response.

In websockets, the connection is bidirectional. That means even the server can start communication with client.



This is a web socket, not P2P as client, and client2 cannot talk to one another.
generally used in a chat app

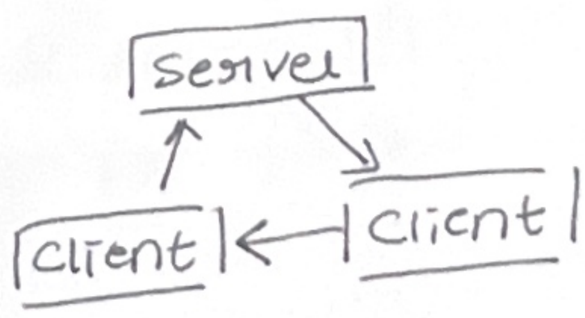
FTP maintains 2 connections, Data connection and control connection

SMTP is always used with IMAP or pop.

SMTP → send a mail, IMAP to receive a mail.

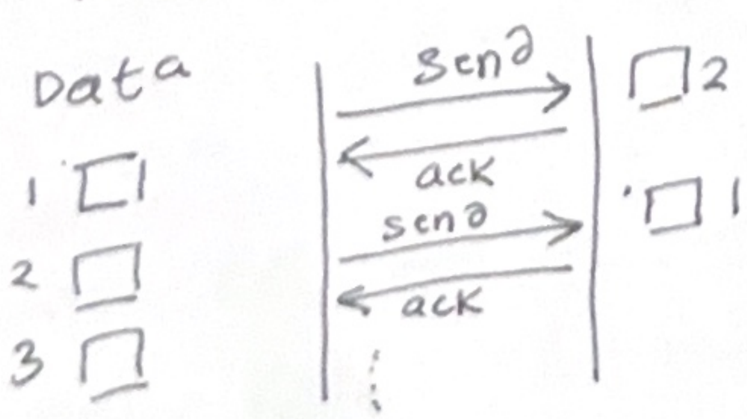
POP3 is not used as it downloads the mail.

In P2P, everyone can talk to everyone.



Transport layer → TCP/IP
→ UDP/IC

TCP/IP ⇒ maintains order and take acknowledgment for each packet it sends. If acknowledgement is not received for any packet it is send again.



UDP/IP :- divides data into data grams. Does not maintains order. Does not gives acknowledgement. Thus is faster than TCP/IP.
used in for example, video calling.

WebRTC is using UDP.

FTP is not secure, thus we use HTTPS.

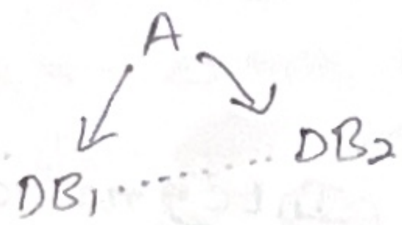
CAP Theorem :- Desireable property of a distributed system with replicated data.

C → consistency, A → availability, P → Partition tolerance.

** all the 3 properties cannot be used together.
can be used like CA, CP, AP.

if 2 or more services somehow are not able to talk to each other.

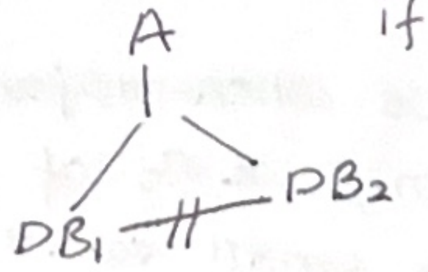
Consitency → No matter a user is fetching data from any where it should always be same



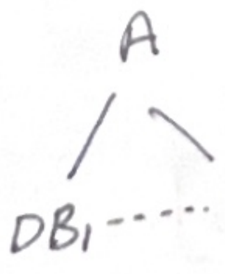
example if A is making some change in DB_1 then it should also reflect in DB_2. Now if A fetches same data from DB_1 then it should fetch same data from DB_2.

Availability :- all nodes should response. Doesnt matter if the response is correct or wrong. All services should be up.

Partition Tolerance :- example connection issue b/w DB_1 and DB_2, even then if A is able to query DB_1 and DB_2 then that means. system if partition tolerant.



CA → in this case all the services should always be available.



the connection between DB_1 and DB_2 break, then both of them will go out of sync. Thesiy by losing consistancy. and consistance. Now if

similary for CP, AP. (refer net)

** P should always be present, so choices are CP, AP

Microservices designs patterns :-

Problems with monolith :-
① overloads IDE. ② scaling is hard ③ Tight coupled ④
④ Making changes is time taking. ⑤

Advantages with monolith :-
① Easy debugging ② simple development ③ Easy deployment.

Advantages of microservices :-
① Maaging is easy ② scaling is easy.

Disadvantage of microservice :-
① If not created properly, any many communication b/w services then it leads to latency.
② Monitoring is difficult. ③ Transaction management is difficult. as different services are involved.

Phases of Microservices :-
Decomposition, Database, Communication, Integration....

Decomposition :- by business capability, by subdomain
Database :- same/common DB, individual DB
communication :- API, events...

Strangler Pattern :- Use when refactoring a monolith to a microservice. Send only x% of request to microservice
x will start from a small value and then go to 100%.
If there is any issue in the microservice then you can delegate all the request to monolith.

DB management in MS :-

① Shared Database :-
disAdvantage :-
a) If service S_2 is sending more data to the database, then we cannot scale just S_2's DB, we will have to scale the entire DB.

b) some service S_3 cannot delete any row or column as some other service might be dependent on that.

Advantages :- Maintain transcation, easy to join

Individual DB :- disadvantages :- Maintaining transaction ⇒ SAGA
Take joins ⇒ CQRS

SAGA :- Main idea is that for every successfull / fail DB transaction the service will create an event. This event will be consumed by subsequent service to perform next set of operations.

If the event is a success event, then next service will continue the process, if some error occurs in some service, it will roll back its changes and also create a failure event. Previous service will consume this event and roll back their changes and will create a failure event and this cycle will continue. untill it reaches the first node.

There are 2 ways of achiveing SAGA. ① Choreography ② Orchestration.

① Choreography ⇒ We create multiple queues. Problem :- cyclic event can trigger leading to unending loop.

② Orchestration ⇒ There is an orchestrator which handles all such events of success and failure.

CQRS :- We create a new DB (view DB)
All create, insert (write operations) will happen in primary DB.
All read operation will happen in view DB.
Challenge :- sync all the view DBs