SAD notes 26th april

Monolithic architecture

parallels and comparisons with microservices

- easy deployment
- development
- performance
- simplified testing and easy debugging –got to class at this point lol
  - service communication internally is easy,
  - decoupling through interfaces etc

monolithic vs microservice

microservices—(the ideal microservice)
- every microservice is in itself an independent application, each with its own database, coordinating with each other, some communication protocol is defined
- if a service has its own heavy load, it needs its own server, conversely, some small services with no relation to each other can be deployed on one server, har service ki deployment alag se defined hoti hai(communication is still not simple internal communication, protocol is followed)
- practically, monolithic applications hoti hein aur day one se microservice architecture pe nahi jaate-reason is that they are very complex, aur day 1 of development pe itni complexity hoti nahi hai, need rapid development, migrations ki zaroorat parti hai monolithic se microservices pe jaane mein, kyunke go-to-market time increase hota hai
- data redundancy because of separate databases (practically sometimes databases are shared too because of some use case or need, compromises are necessary in the real world)

monolithic—
- if two services are communicating with each other, you do not get to know this, pata nahi chalta humein, communicating is done internally
- (iss slide pe mostly abhi microservices pe hi baat huwi thi)

microservices—also known as microservice architecture
- every service is itself a separate project
- highly maintainable and testable—testing has a lot of coverage, more testing scope, may take more effort than in monolithic, have to test different services based on your workflows, maintainable because every project can be scaled and maintained separately
- loosely coupled—har service iss hadd tak loosely coupled hoti hein ke alag alag deploy bhi hosakti hein
- independently deployable—one of the main goals of this architecture, one of the main reasons for loose coupling
- organized around business capabilities/needs—how to decide what will be a service, kese decide karenge ke yeh meri aik service hogi, agar bohot choti choti services hongi tou-> jitni ziada interservice communication, utni ziada complexity aur size, aur agar bohot bari bari services bana di tou coupling barh jayegi, typically, one business unit should be a microservice----(domain-driven design is relevant here)-------this one area is the hardest thing in this architecture, since there is no one rule, no silver bullet, no size fits all

- every service has its own small team, a big prject w many services will have many small teams working on different services (diff teams can be working in different languages/frameworks)---again this is an ideal scenario, not always the case due to real world constraints
- migration example—you had a monolithic arch, in javascript, you had a lot of security issues. Client wants to fix this. Aik option yeh hai ke poora project migrate karein to Java for more security, or the better option, sirf payment wala module jahan yeh security concern hai usko aik separate service bana ke deploy kardein (another option is to use an sdk internal java library but that does not demonstrate this use case, I brought it up, can explain later, ask me)

Components of microservices—an example I think

1) Clients – sends all reqs to api gateway, wo baqi cheezein sambhalta hai
2) Identity providers (authentication) – are sometimes part of the api gateway, not made a part of services since har service ki authentication, tokens, caches, alag alag maintain karni paregi
3) API gateway – to prevent unnecessary redundancy, it knows which service to call, abstraction for services and cross cutting concerns
4) messaging formats
5) databases
6) static content -- directly serve hojata hai, uski tension nahi hai
7) management – to manage the service cluster (not sure about difference between management and service discovery)
8) service discovery – kind of like containerization in docker, konsi service down hai, kisse kab up karna hai etc

random code management talk—monorepo better or different repos?
Ans== no one answer

monorepo mein har cheez ka access miljata hai—less scope for access control, git conflict management may be more complex

Domain-driven design

- designing software based on underlying business needs
- zaroori nahi hai ke microservices mein hi ho, it just maps well to microservices, is an independent concept
- division and separation of modules and services must be done according to the requirement of the business
- also need to take care of your technical limitations

Assignment latest wali, submission on thursday, assignment tab on lms
extra class on some friday 330-530

API Gateway in microservices architecture

- microservice partition ko expose honay se bachata hai
- saves you from round trips (for eg in the case when api gateway is not there aur 3 services use horahi hon, tou 3 round trips lagengi faltu ki)
- apache kafka – a protocol for microservices* need to look into this
- Gateway offloading slide, just reading
- ip whitelisting—kis kis ip ko allow karna hai, api gateway nahi hoga tou har service pe karni paregi, lots more overhead

- api gateway is not really a part of microservices, lekin microservices ko implement karne ke liye api gateway is a very important part (lekin bilkul microservices api gateway ke baghair bhi microservices bhi rahengi)

Inter-service communication

- synchronous—http sync requests, client waits, everything goes on hold till the response comes back—advantages:low complexity, simple; disadvantages: thread blocking and performance issues
- asynchronous—increased complexity, request response wali kahani nahi rehti, koi aur protocol use hoti hai yahan, more efficient despite increased complexity, prefer this taake blocking na ho, EventBus – I.e apache kafka;   advantages: don't need connection oriented protocol, message brokers in every step of the way, low coupling since service doesn't need to know about the consumer, better failure isolation, better performance as response is immediate (this point needs clarity, does not mean ke actual kaam taiz hoga, bus client bloack nahi hoga, some message displayed like ok request received, and you can make the next api call while waiting for the first api call to return a response) ; disadvantages: greater complexity and difficult error handling, greater cost of implementation and monitoring

Benefits of microservices

- slide reading, simple enough
- easy to understand and modify for devs: in the context of a single microservice, overall complexity that a dev faces will decrease
- highly scalable—an example: an ecommerce platform, a basic flow→ search a product→ add to cart→ checkout→ payment→ order creation/place
  - 1000 users, all of the users will be on search, maybe 50% get to adding to cart, 20% move on to checkout, people get stuck on payment module perhaps 10% are doing the payment, 5-10% are having their order placed
  - in a monolith that supports exactly 1000 users, everything working fine, one server costs 100$. If 1000 extra users come on, we need an extra server to handle that 1000, cost doubled
  - in a microservice application, 5 chotay servers for each service, each server is 20$, 100$ total. Now users increase to 2000, but we realize that only search service is being affected, we get another server for 20$ so total cost is now 120$, rest of the servers are still under-utilized, since they were originally being used by less of the original percentage of users.
  - This way scaling works better in a microservices architecture.
- No long term commitment to tech stacks

Drawbacks slide – mostly reading

- case study possible: amazon prime moving back from microservices to monolithic architecture (I sent two links to the whatsapp group do check them out!!)