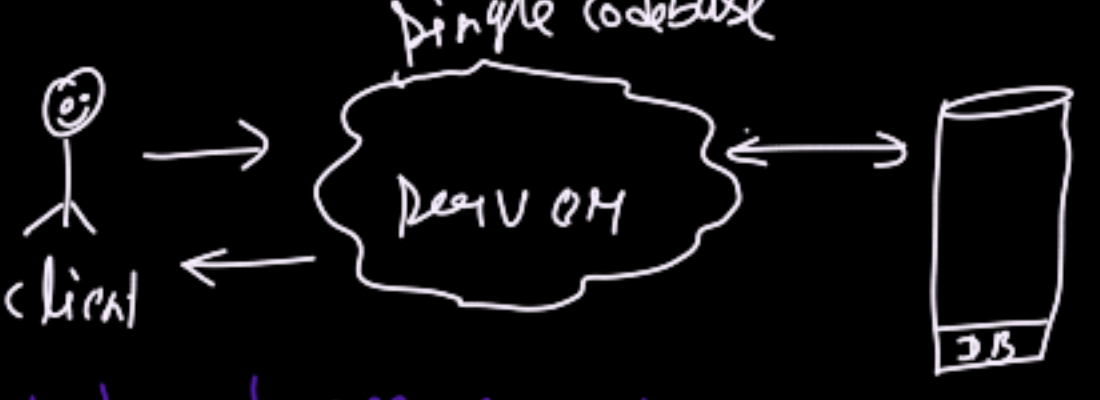
Types of Architecture

① Monolethic Architecture

→ All features and logic and integerations in the single codebase



Advantages :→

(i) perfect for startups
(ii) low latency : No need to call external microservices becaux all are in single codebase
(iii) Easy integerations
→ single DB
→ ACID are too easy

Disadvantages :

(i) small changes can break unrealated features
(ii) scaling is All-or-nothing
→ scale login only
→ you must have to scale the Entire application
(iii) slow development at scale
→ many developers touching same code.
→ merge conflicts
→ Hard ownership
(iv) Technolog locked-in
- same language
- same framework
- same run time

Modular Monolethic

— /auth
— / payment
— / orders

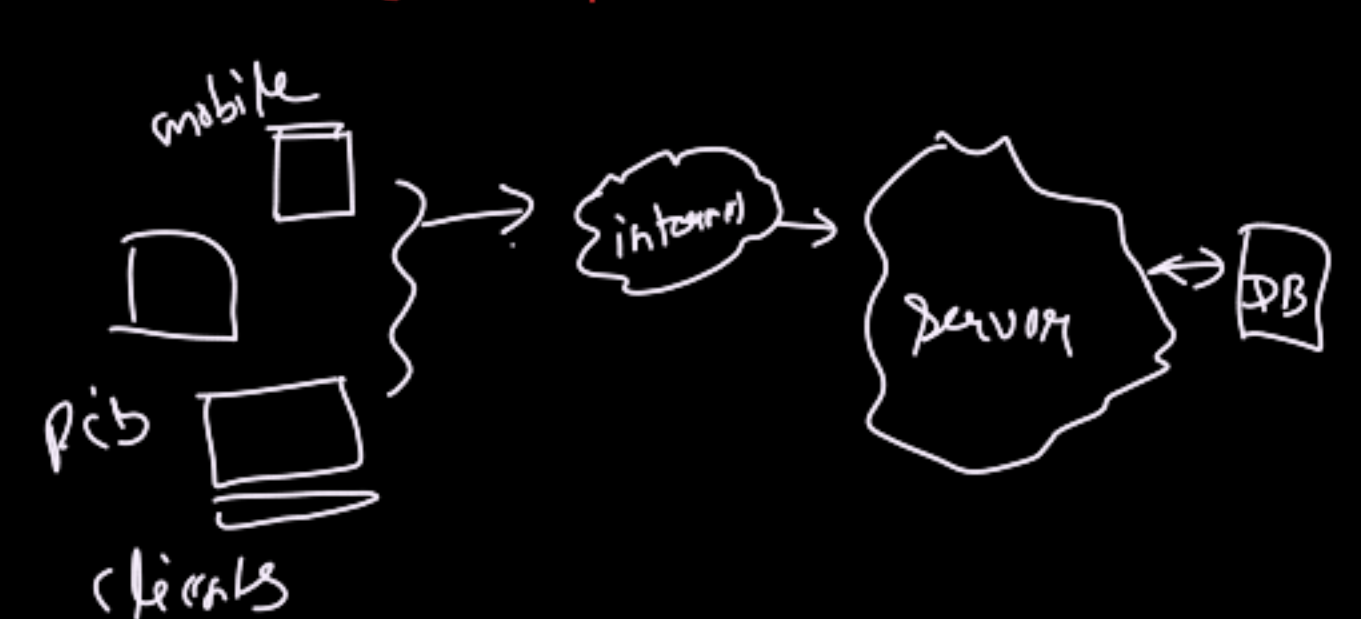
* still one deployment
* But clear internal modules
* Easy failure migration to microservices.

Layered (N-Tier) Architecture

- controller → logic → data → DB
- seperation of concerns
- used inside monoliths and microservices.

It is only a way to organized the internal code base. It's not a real architecture like the monoliths.

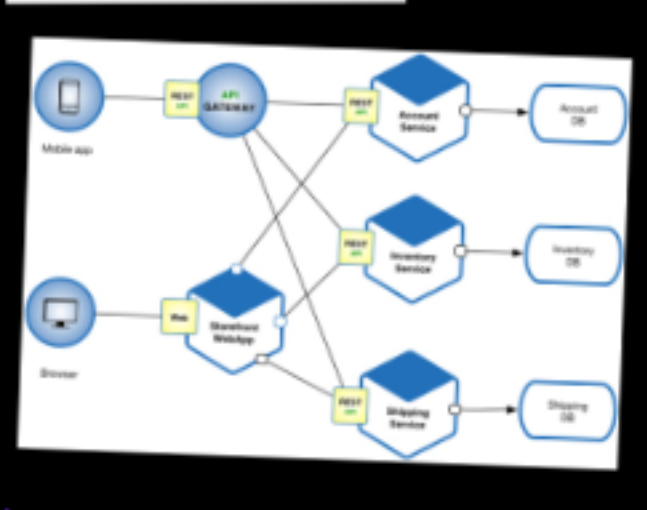
Client-server Architecture.



Microservices Architecture

* The application is broken into multiple independent service

* Each service :
- own logic
- own database
- Independent deployment



Advantages :

(i) scales independtly
(ii) fault isolation
→ It means when something fails, keep limited to that part only.

→ key Idea :
- failure is allowed
- failure must not cascade

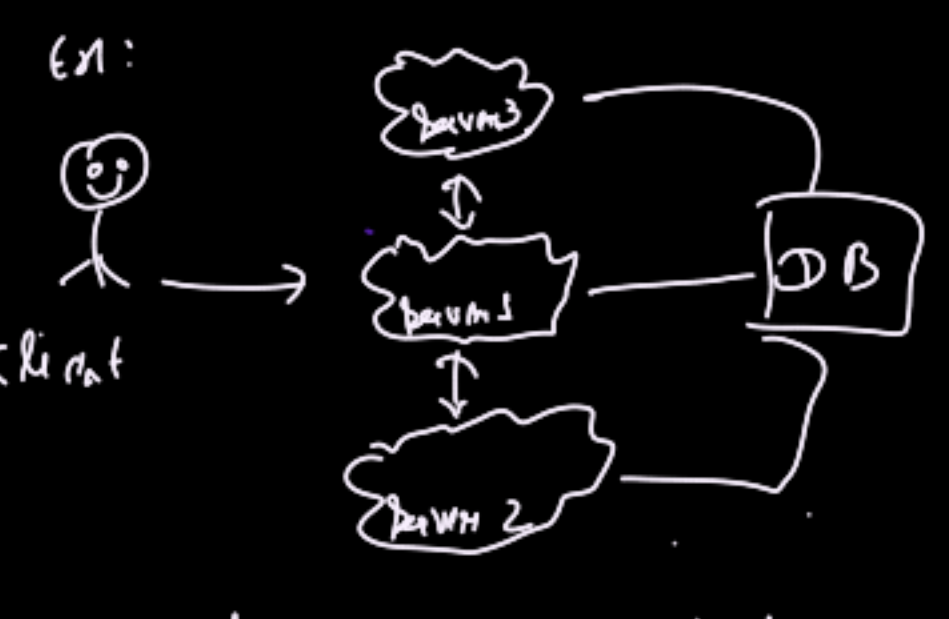
Ex:
- payment services crashed
- user login → still work

(iii) Team autonomy

Disadvantages :

(i) complex communication

Ex:



" means all server communicate by the internet. "

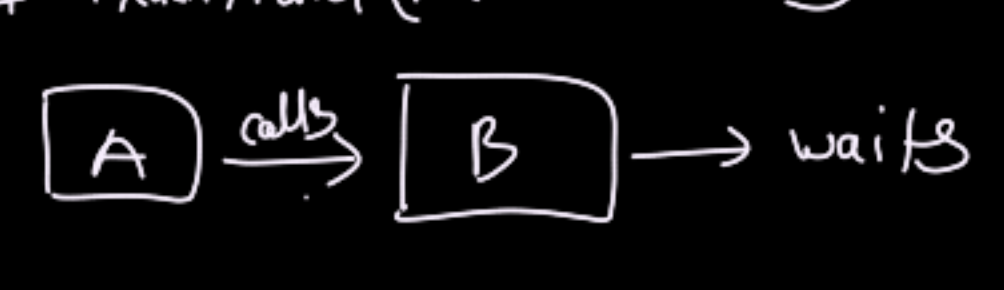
(ii) Needs DevOps, monitoring
(iii) overkill for beginners

→ used by : large industries

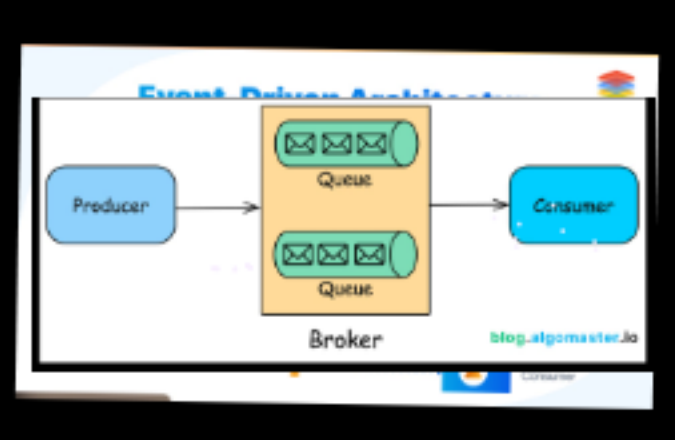
Event driven Architecture

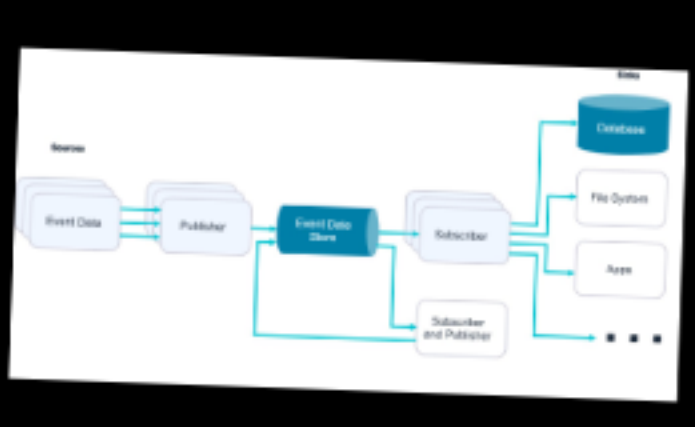
* In this Event-driven Architecture is a system design where services do not call each other directly.
Instead, they communicate by producing and reacting to events.

Traditional (Request → Response)



Event-driven





Note: more things have to cover about this architecture

Serverless Architecture

" serverless does not mean no servers , It means you don't mange servers. "

* It means servers exist, but you don't take care of them
→ The cloud company (WS, Google, Azure) handles everything.

* cloud provider handles;
- server provisioning
- scaling
- patching
- Availibility

you only write the functions

core Idea ;
instead of :
client → server → DB
you do :
client → API gateway → function → DB

Each function ;
- Is stateless
- Runs only when trigerred
- Dies immadiatly after execution

Function as a Service (FaaS)