Hi Ashim,

**Class details:**

* Batch Name: **Academy Nov22 Intermediate 1**
* Topic to be covered: **HLD: Microservices 2**.
* Class Time: **09:00 pm (IST)**
* Link to join: [Classroom link](https://www.scaler.com/ahoy/messages/h5dG8XZ3lsgk3TdvUXjFDCA50Umbtklp/click?signature=ce125faf717ac3be585090181d44775d23236046&url=https%3A%2F%2Fwww.scaler.com%2Facademy%2Fmentee-dashboard%2Fclass%2F147405%2Fsession)

Following are the topics we shall be covering in the upcoming class.

* Topic 1: Summary of Last Class
* Topic 2: Consistency in Microservices
* Topic 3: Two Phase Commit
* Topic 4: SAGA Pattern
* Topic 5: Orchestration
* Topic 6: Choreography
* Topic 7: Choreography vs Orchestration
* Topic 8: Kafka for Event driven architecture
* Topic 9: CQRS Pattern
* Topic 10: Circuit Breaker Pattern - Problem
* Topic 11: Circuit Breaker Pattern
* Topic 12: Doubt Resolution

When product reach scale. It can bea collection of services together running in backend.. lets take a real life example..

Flipkart example, logical componnets:

A flipkart landing page, search, authentication, product page. Order, payment, delivery.

If all of these component/ functionalities are part of same Service… how will this look like… one version of deployable/ executable. This deployable will be deployed in one APP server..

This is monolith..

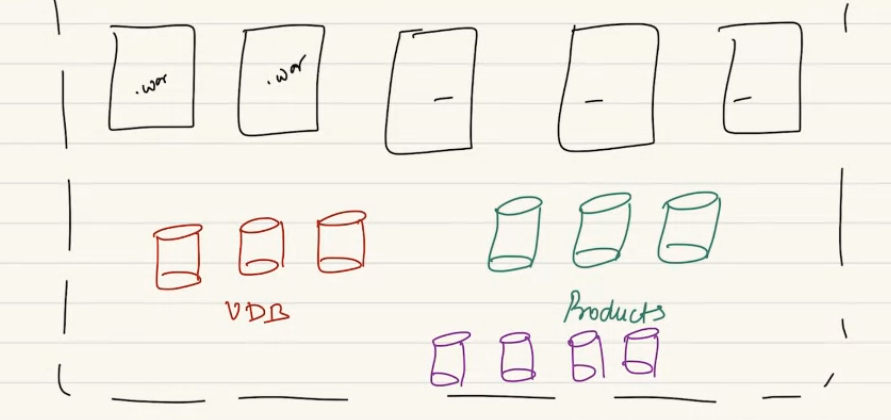
Backend look like:

Gateway will double up as a LB..

App server machines will run business logic. To run that they need executable to run on them.

All of these will run the same executable, running exact same code. Behind that, diff type of DB.

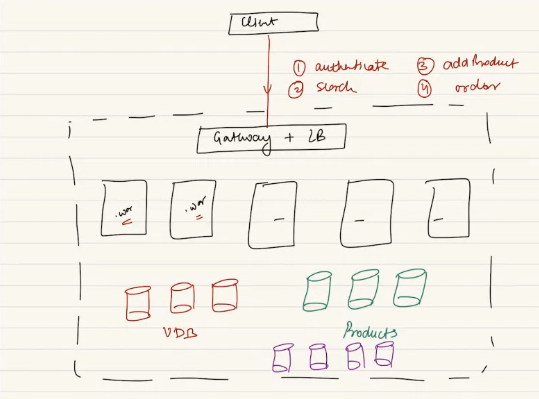
A User DB will have diff things stored of user. Same product, paymnets db etc. so I will have diff kind of DB. A file storage will have photos of product.



We are considering it as a single service. A single service wil have diff type of DB.

When client will reach out to you.. for multiple reasons like for authentication, searchinga product, adding product to cart, making a payment. So client will make request to backend for diff use case.

Idea of monolith is no matter for request all of themw ill be handled by the same WAR,



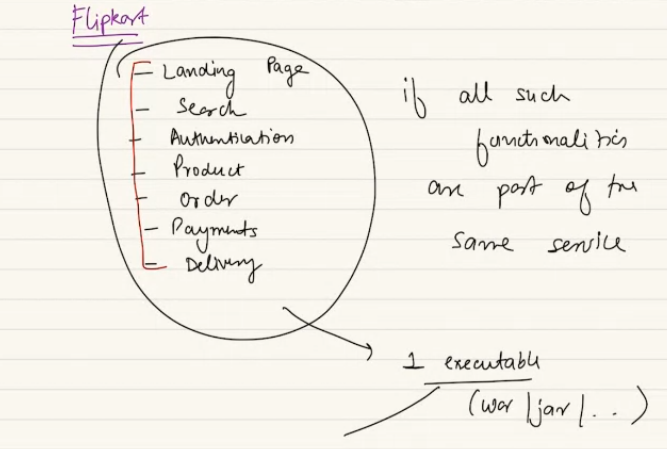
A great idea of what we have been covering in our classes..

App sever can be statefull// less.

Running diff DB, we learnt DB federation.. u might not store all data in same DB. Diff db for prouct diff for users. Diff DB for diff use case.

What are the problem for doing this:

This is great at starting. Problem occurs now you are trying to solve so many logical distinct problem with one WAR. One war for many use cases.



If I have same code deployable. If an engineer try to change a logical code.. “how my products will be mnaged in backend..” if there is a code change in this part. All code will be deployed together. So will rebuild the entire project. For a single change, rebuild, repackaged, new version of war. New deployment.. we redeploy as we have only one deployable. Everything gets deployed in same package.

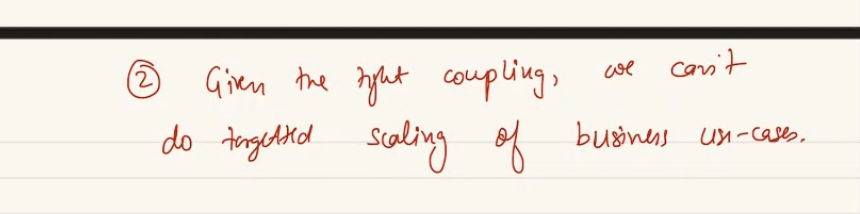
Problem of Monolith:

* 1. Very tight coupling in code deployment, even a small code change in one logical use case. Will require us to rebuild the entire code and redeploy on all machines.
  2. Related side-effect: rebuilding, unit test, deploy in all machine will take time.

**First problem**: If I make any mistake, blust radius will be higher… if u make wrong, it can effect somewhere else as well.

When entire thing run as one deployable, u make any change, have to combine everyting and repackage. Its tightly coupled.

**Second problem:** If I realize the search utility taking time..so when I make a search request…if u think from Distributes system pers.. you add more system in cluster. When we realize our utility is high we do horizontal scaling. If my search taking time.. search handled by same war.. I have to add new machines. They will be for entire use case. Cant do targeted scaling. One cluster I am giving more resources for every use case. Cant target just search. Cant do targeted scale up.



3rd problem: there might be use case: for managing payment if I wanna use different tech stack and writing authentication I want to use python.. if all in one deployment I cant use diff tech stack for different use case.. given the tight coupling, I cant select different tech stack for different use case.

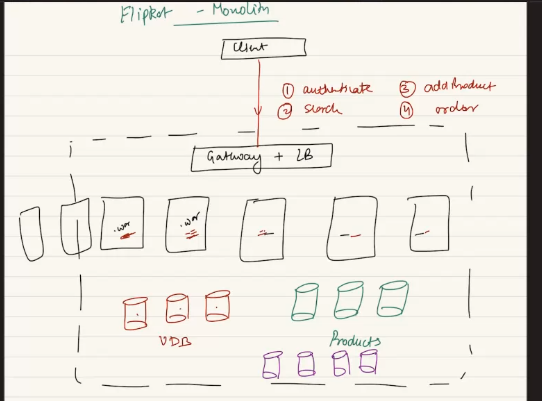
if mistake happened, rollback will also take time..

**4th Problem:** Developer onboarding wil be a challenge: I have single deployable. Developer is joining in my team, 1071th developer he is. I told him to do make some change. A new guy will find it difficult. Understanding how the entire web of very diff use case working together, will find very difficult to make change. Making in change in search. It can effect that users cant even make a login. Mindfull about all use case. Blast raidus high.

Also more coordination between different working groups is needed in case of a monolith. And one person does a wrong checkin - whole code needs to be withdrawn making other people wait to test their change.

**Lets see comparison in microservice vs monolith.**

U can use all low level design it still can be monolith, if u are deploying as one deployable.. its monolith. Code quality can be beautiful using all LLD principle, it can be monolith. There is nothing explicitly right or wrong .



One build = monolith.

How this diagram will change if I build flipkart not as monolith but as multiple services… we are not suing the term microservice here..

I have user service: user do sign up.. login, change password. Change profile details. Change address. Etc. these all come in user service.

Product service: independent: add a product to be sold, change desc of product, search product..

Payment service: only manage payment for flipkart..

How does the diagram will look like..

LB+Gwy can be built out of same machine but use case is different.

Client will interact with Backend, nehind dotted line. First point of contact is Gateway. AKA LB. client will make a call to gwy/ LB. if I have 3 FK build as 3 diff service. So inside backend I will have 3 different layers of app server. Cluster of app server for user service, for product & for payment service.

Each service here has own clusters. 3 layers of service. Each of service has won app server. Has diff deployable. Diff jar for usert, product and diff jar for payment service. Diff-diff deployable for each service. Each one will have build pipeline, deployment pipeline…

Before all these a set of machine who look at request and decide to which service this particular request will go to.. user service can expose API like, login, logout, register API.

Product: add, remove, add to acrt, place order

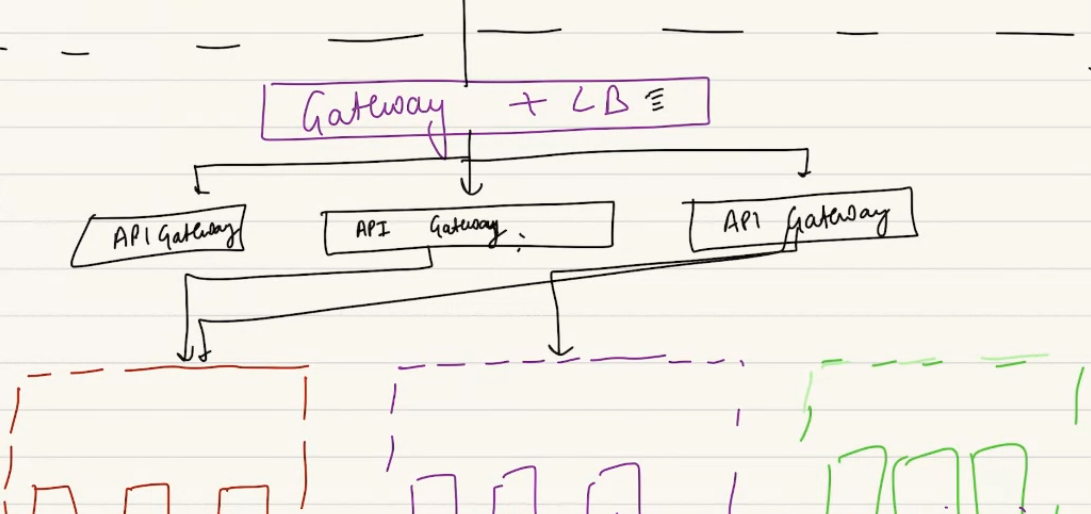
Payment: API, make payment, claim refund..

Each has diff set of API exposed..

Hence after LB+GWY will have a API gateway who decides which service this request goes to. Client don’t know diff service diff app server. Clint will make call to LB. LB will now forward the call to API gateway: machine who fwd the request to correct services based on whereever the request can go…

API gwy job can be done by the first machine (gwy+LB)

People have diff cluster of API machine, so GWY can fwd the request to nay API gateway in round robin .



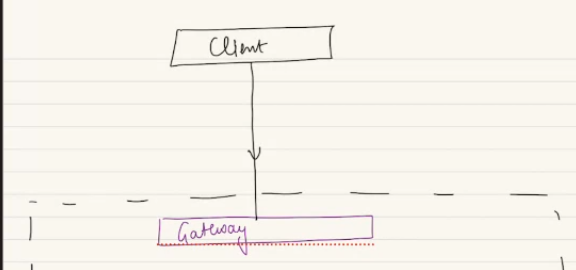
Re explaining:

1st point of contact of any outside req is Gateway machine. LB is 2nd use case of this machine. So correct answer is gateway. Its the entry point for outside request. It also known as reverse proxy. gWY machine also doubles up as LB. it fwd the request to one of the app server. For statefull= correct, stetaless = any in round robin manner.

When you have more than one services, its not just important to find out correct app server, we would find out which app server.. we need to go to correct service. Its not service discovery (its in ZK, who is master, who is slave)..

When request from outside.. API GWY will fwd a req to correct service based on API exposed by that service.

We also call it LB as it takes us to correct app service. When ultple app server its not a LB anymore as it don’t know which server to go to. We just call it a gwy.

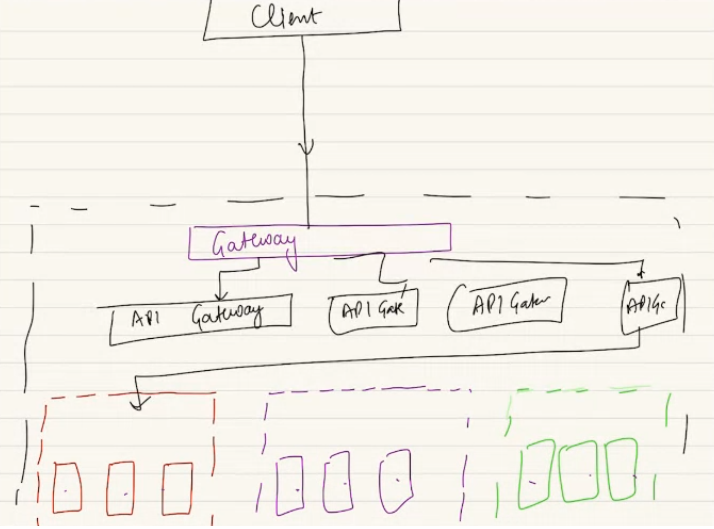


Now I ned a machine to know which service do I need to go to… look at API call and find which service to goto.. login= user service, making payment = payment service.

This machine knows which PAI call go to which service that machine call as API gateway. Knows which API is mapped to which service.

Gwy now no longer does load balancing, it will just fwd to 1 API gateways….of multipe API gateways.. we have many API gwy running same code, have sam mapping.

Why multiple API Gateways? For availability? Higher availability and scale.



Once we request to the service , we need to find which app server to go to.. I have just reached to a service don’t know which server… so we need to have a LB.

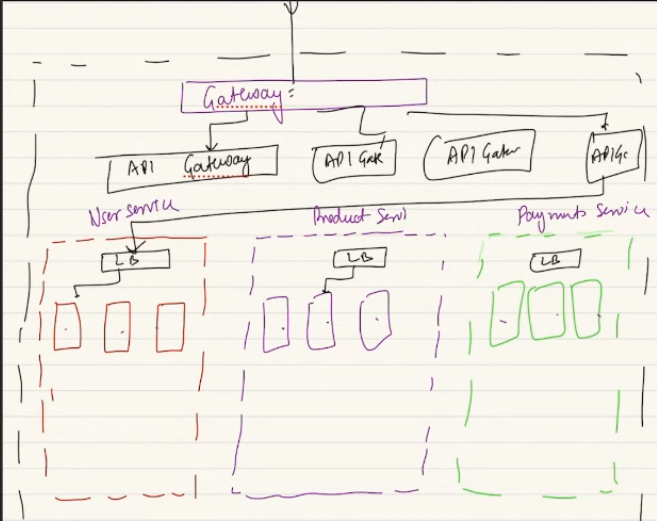
stateless = round robin, Stateful = consistent hashing.

Gateway = I don’t want to expose least resource. One gwy for outside world. This gwy convert the req from outside world. Act as 1st pint of contact. Single gwy allow to run security consideration.

API gwy need not running at that high security. Run on internal IP.

I don’t wanna GWY to overburden to all responsibility. Can act as firewall. Can decrypt encrypted data check security.

Monolith me we don’t need to decide..



If I realize the the product service, we can add ore resource to product service. Can do targeted scaling.. this service cant scale well so I can add more machine here. 2nd point is resolved when we have multiple services.

For user service I would need a user DB… USB.. is gonan be a SQL DB.

Product DB will be part of product service..

If payment handles the orders. The orders DB can be in product service..

Each service can inter related comminucation with each others DB. And can do inter service communication.

These service will have to talk to each other….

Create service which will logically solve one problem. Just not create insanely so many services. Few services not a small service for each use cases.

SOA: service oriented arctiture. SOA used to be a earlier. Years ago.

Problems: 1: tight coupling: if I make change in user service I will not have to redeply whole thing.

Build time will be less, less code, file, use case. Deployment less as less no of machine.

Now we can do targeted scaling. I can improve caching layer in user service.

Resource: app server, cach server, DB etc.

Now I have 3 service I can have different tech stack. Each service in diff tech stack.

Developer onboraidng has become simplified. One dev can worry about payment service. He handles the service team he is joing. don’t worry everyting in the world.

Blust radius less. Regression is less.

No free lunch in world. Everythijg is trade off here in HLD

**DIsadv of microservices…**

1. U have more no of layers, higher latency..now its not diretly gwy🡪 app server, its gwy🡪 api gwy🡪 service. Diff service diff use case I will keep having talk to diff service, send request get. More inter communication. More layers, more network calls…jumping btwn machnes. More layers. Hops.
2. Spend more: spend some resources uniquesly for each services.. more n of machines.. usually costlier to manage a MS. Require more resource to maintain this.  virtualization helps here . create container, virtual box where many virtual box can run on same machine. When u have containerization u add overhead on top of base case.

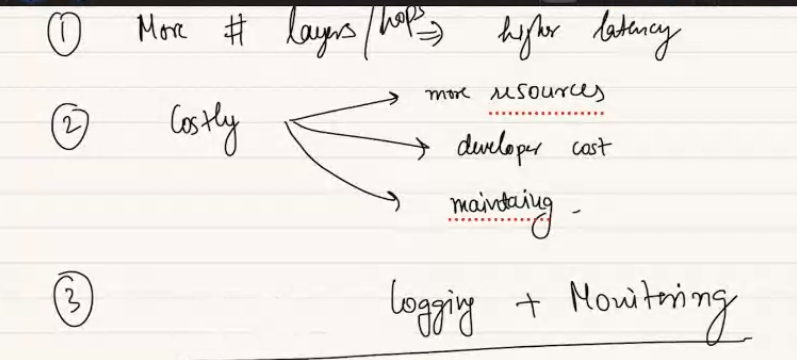
Also costly hwne more MS need more developer as well….team of 3 cant handle 20 MS.. need more hands on deck. more jobs for developers.

1. I am building swiggy as MS, I will go and talk to auth service. Select resturent, req will go to resturent service. Tne order serveice, then payment service. Req come to order . then go to logistic service. Also go to resturrent service.. for a single request many service involved. Passing the ball type. Many communication, flow of req. if u don’t invest in good logging, tracing, matrix. Your request can get lost in between.

Difficult to find where did the error happen, as many prices involved. Many machine, own architure, own resource, how do u trace an error.

Also we have devops team, check s server health, Q size, latyency on service. If I have one service take care of that. When 300 service, spend time on all 300. None of them overburden, out of memory. Etc… problem is to invest in tracing and logging and monitoring….. set up good monitoring to keep track of health o feach service. Track of log to find which service have issue..

ZK helps on service desicovery, who is mater, who is slave, IP of master, slave. We need some way like ELK stack.. SPLUNK, data DOG, cloud watch, grafana. These product help us to keep eye on our systems. When we have districuted system my headache to manage so many stsem. When yu have muktple services together it becomes DS on sterioids. Even tuffer…



Also we need interservice communication, along with that there will be problem of data consistency. WHY??? Order and payment service, order need to track of payment done or not, then it will proceed. Sam eway payment service need to know payment was successful or failed. When you have requirement of storing same idea in multiple places problem of consistency always arises. So MS has problem of ensuring data consistency..

We create some servce need to create. As every new service add lot more load, worry to my system.

Event driven arch…