Agenda.

→ API Gateway

→ Load balancer.

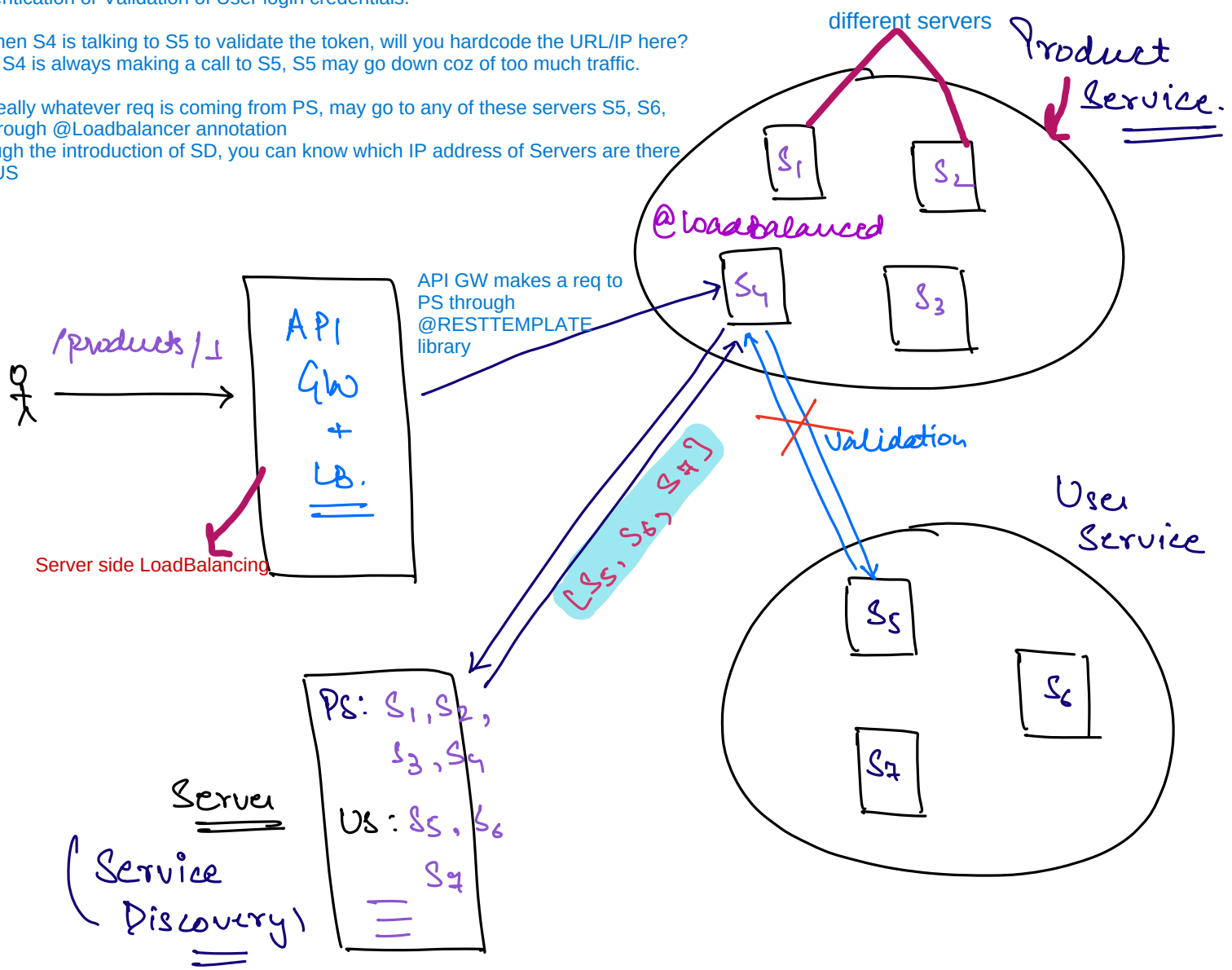
→ Logging & Monitoring

PS, US, PytS will run on different instances/machines/servers

When a user makes a Get Req -> it goes to the PS and PS goes back to the US for Authentication or Validation of User login credentials.

So when S4 is talking to S5 to validate the token, will you hardcode the URL/IP here?
No, if S4 is always making a call to S5, S5 may go down coz of too much traffic.

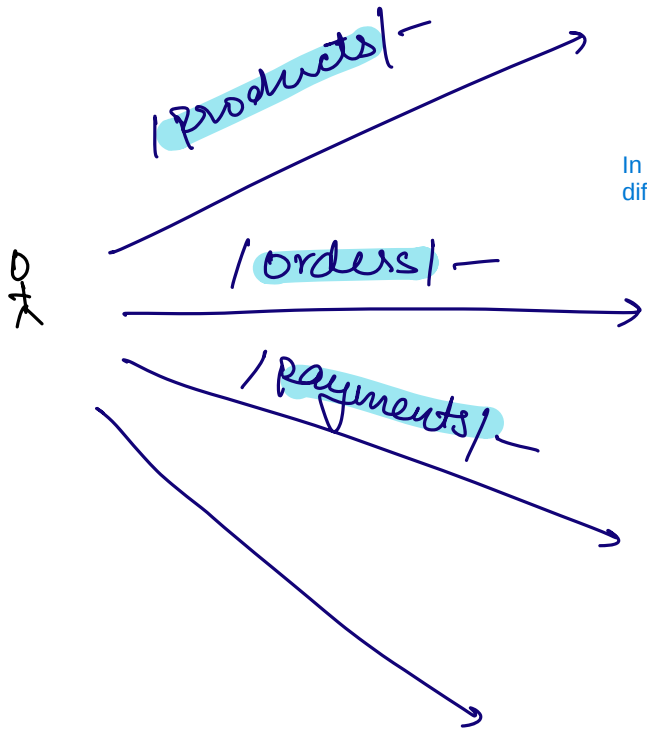
So ideally whatever req is coming from PS, may go to any of these servers S5, S6, S7 through @Loadbalancer annotation
Through the introduction of SD, you can know which IP address of Servers are there is in US



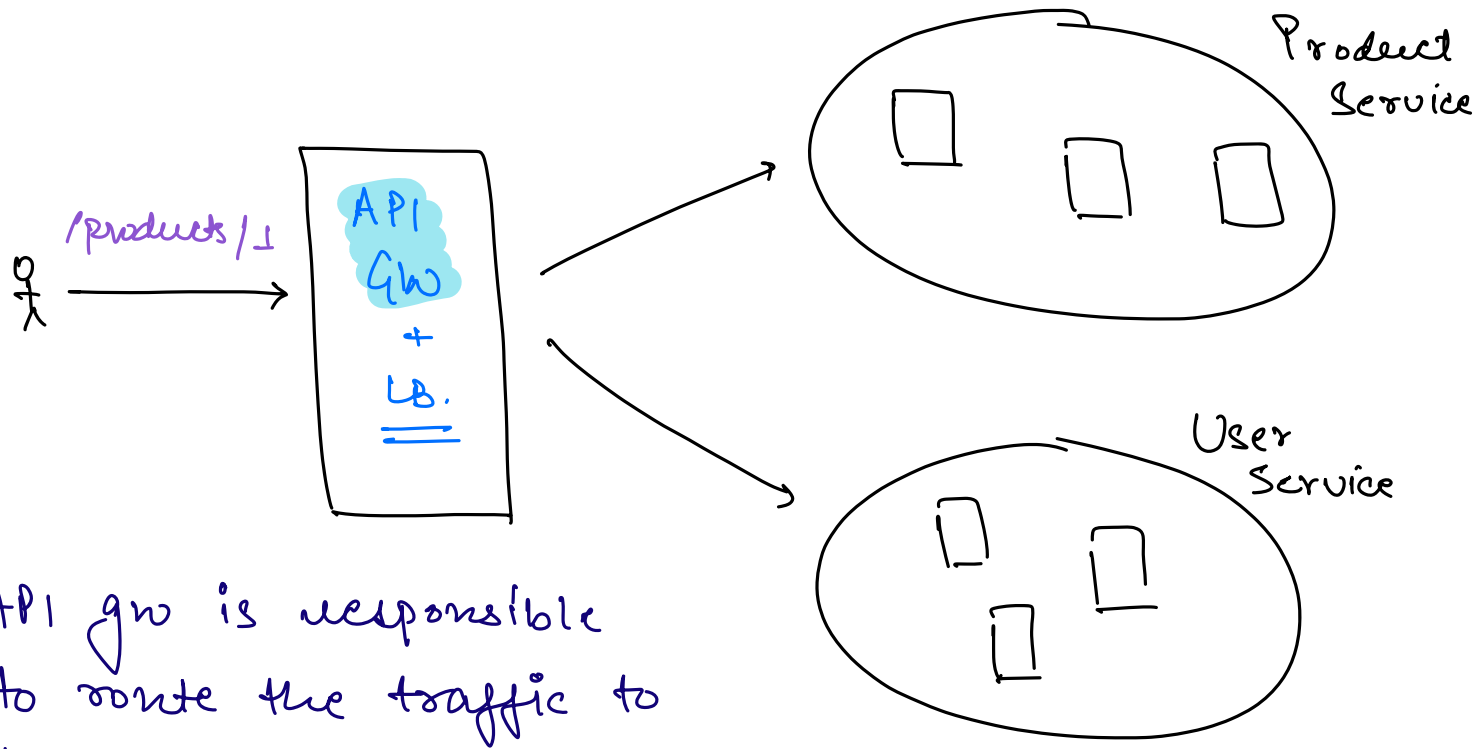
Here PS will connect with SD to get IP addresses of all servers of US and then with @Loadbalancer annotation, PS will send GET request to any of the servers in US in a balanced way. Since PS is a client who is sending req to US -> so this is client side LB

⇒ Client Side load balancing.

When API GW sends req -> it is Server side LB



In MS architecture, there are different endpoints. And API GW sends the req to different services based on the endpoints



⇒ API gw is responsible to route the traffic to the correct Microservice based on the API endpoint.

⇒ Rate limiting.

If you are getting lets say more than 1000 req in 1 sec from one IP address, you can block that IP address through API GW-> illegal activity

⇒ Authentication.

API GW can authenticate at its level. Suppose a person sends an order req to API GW, if the login credentials is not there, the req will be rejected, else if its there, API GW will take the token with login req to US for validation and then only it will route the req to order service

Logging & Monitoring

LOGGING.

1234.

```
signUp(String email, String password){
```

You should give some kind of print statement which will reflect in the logs so that this user if does some malicious activity on the a/c lets say in scaler, you could track when the user with this mail id, the a/c was created, and all of the activities can bee seen in the logs. should

```
    print("User with email:" — " + "signed up");
}
```

```
changeClassName(long classId, String name){
    Print("User with Id: —" is — ");
    ~~Print~~("User with Id: —" ————————");
}
```

Log

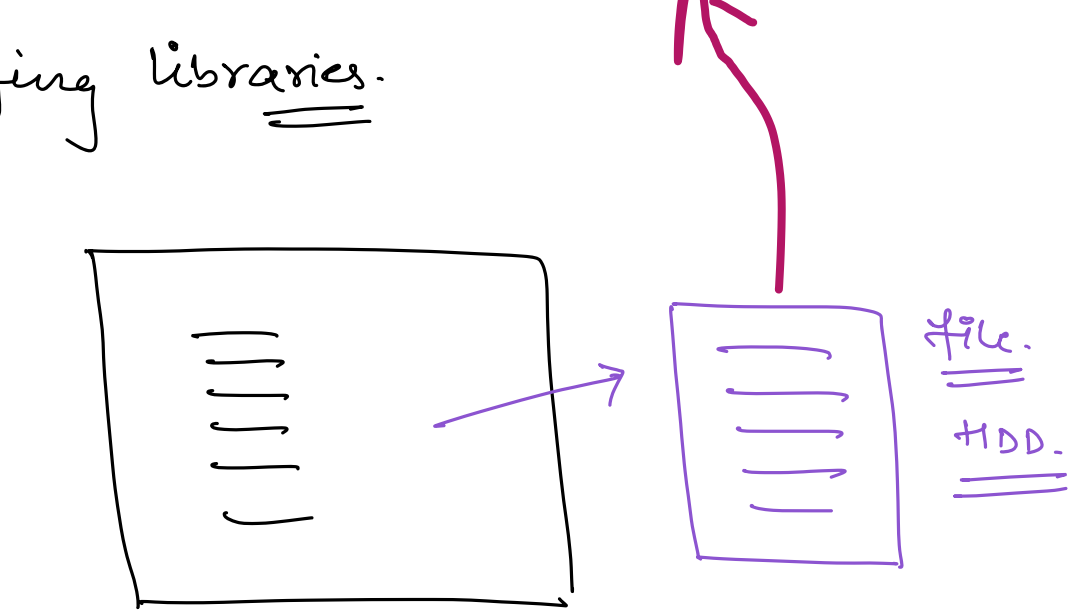
Log are used for :

→ Auditing

→ Debugging.

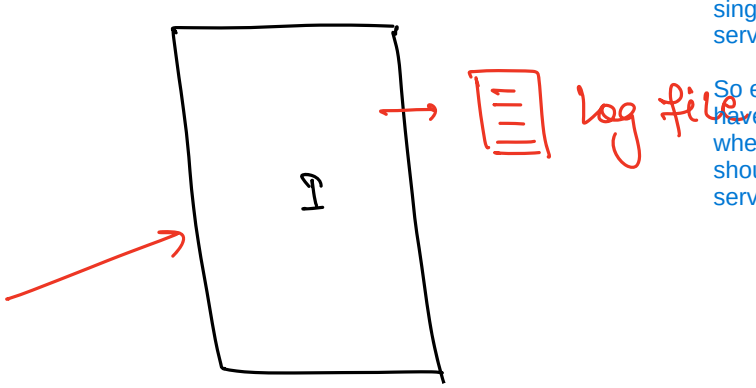
But the print statement gets printed in the console, so if in production you restart the application, you cannot see the older lines on the console. Thats why in production, you use some kind of logging libraries, so instead of print(write log) and all of this log statements will get accumulated in a file which gets stored in the hard disk of your server

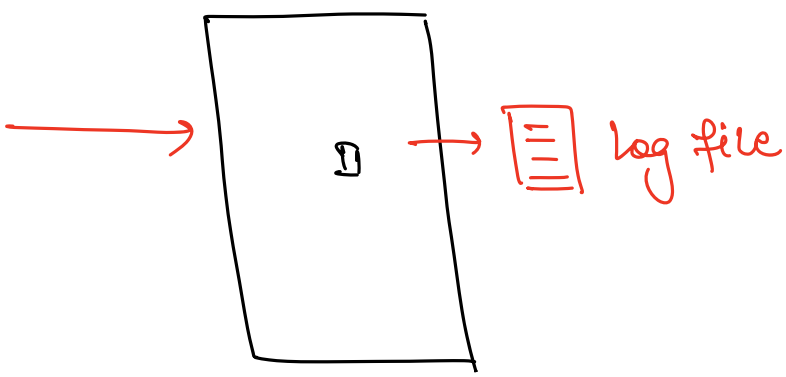
⇒ Logging libraries.

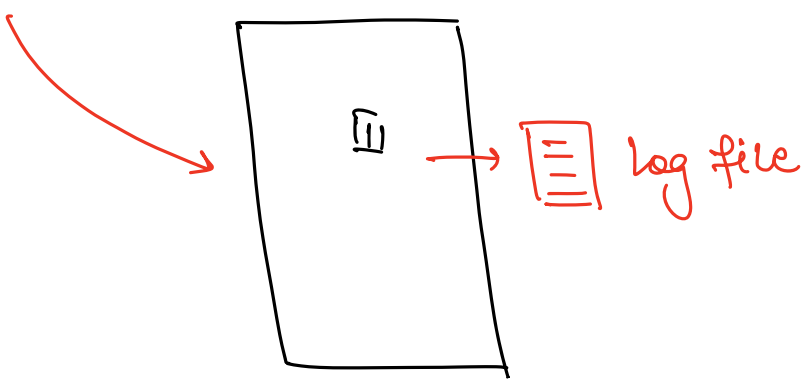


But this scenario is okay if your PS will be running on a single server but if in real, PS will run on multiple servers.

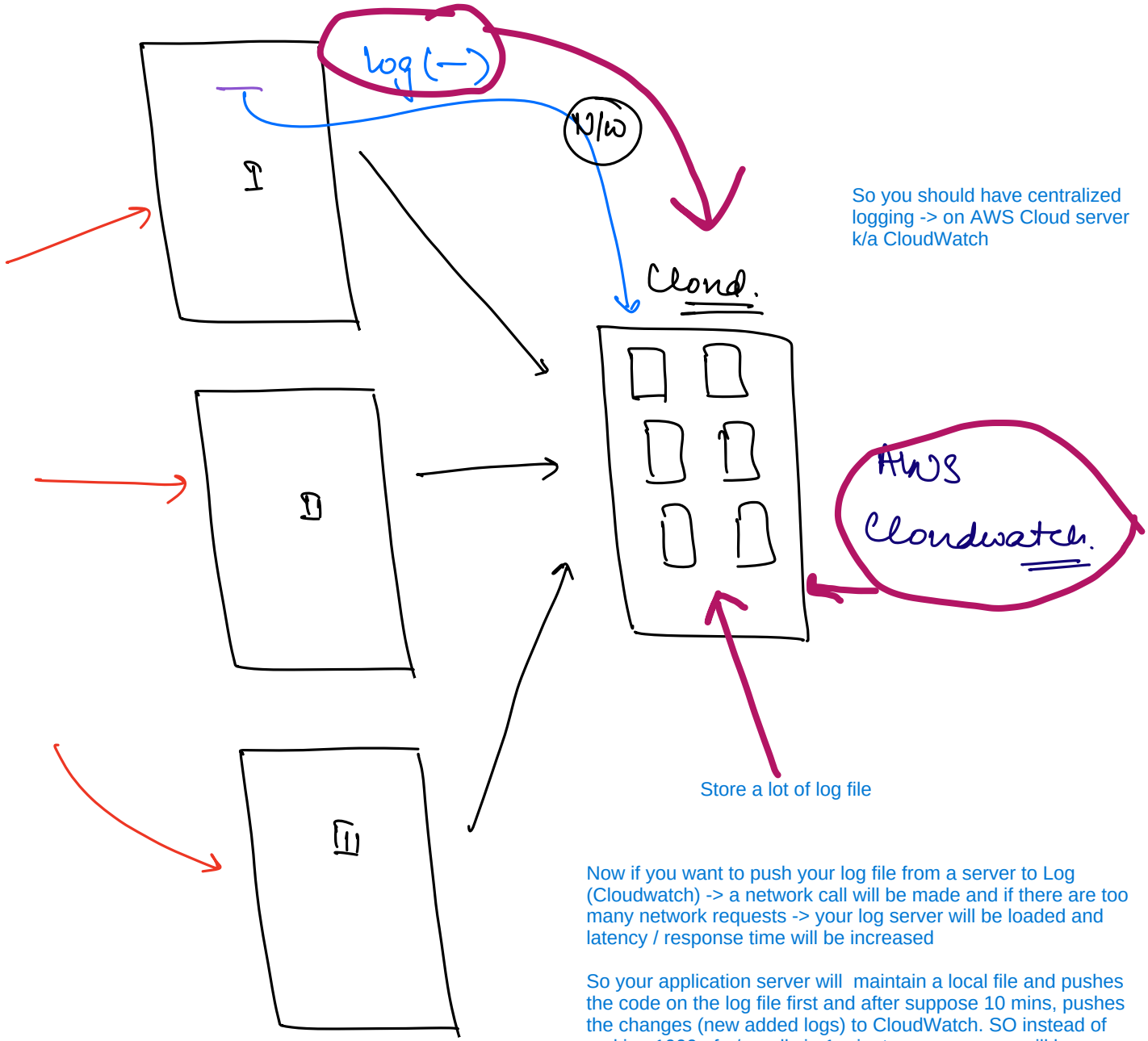
So every server will have its own log file. But if you have to debug, will you check on every server's log file where the issue happens? No, not a good way. So you should have centralized logging -> on AWS Cloud server k/a CloudWatch







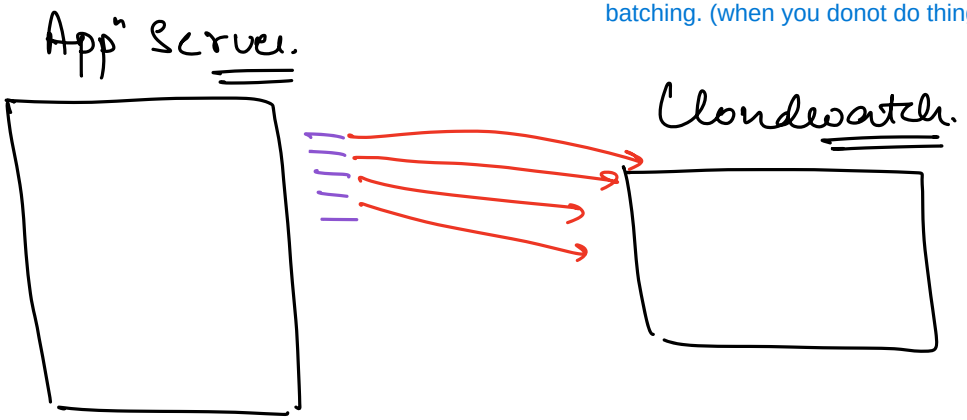
#



So you should have centralized logging -> on AWS Cloud server k/a CloudWatch

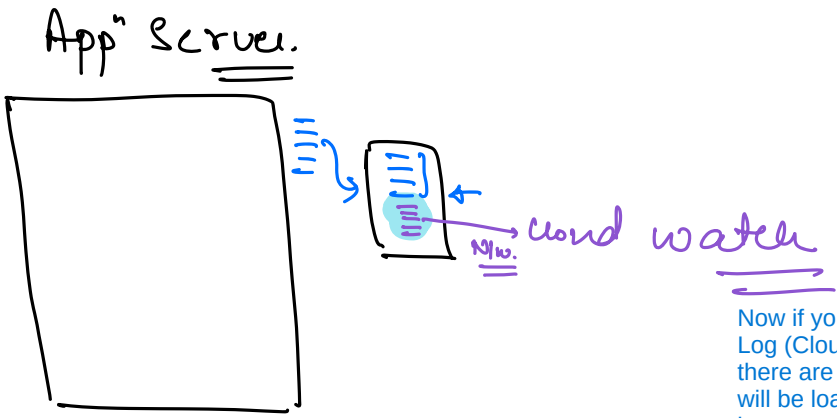
Now if you want to push your log file from a server to Log (Cloudwatch) -> a network call will be made and if there are too many network requests -> your log server will be loaded and latency / response time will be increased

So your application server will maintain a local file and pushes the code on the log file first and after suppose 10 mins, pushes the changes (new added logs) to CloudWatch. SO instead of making 1000 of n/w calls in 1 minutes, your server will be making single n/w call in 1 min. This way of programming is k/a batching. (when you donot do things singling but in batches)



Too many N/w requests.

Batching.



Now if you want to push your log file from a server to Log (Cloudwatch) -> a network call will be made and if there are too many network requests -> your log server will be loaded and latency / response time will be increased

So your application server will maintain a local file and pushes the code on the log file first and after suppose 10 mins, pushes the changes (new added logs) to CloudWatch. SO instead of making 1000 of n/w calls in 1 minutes, your server will be making single n/w call in 1 min. This way of programming is k/a batching. (when you donot do things singling but in batches)

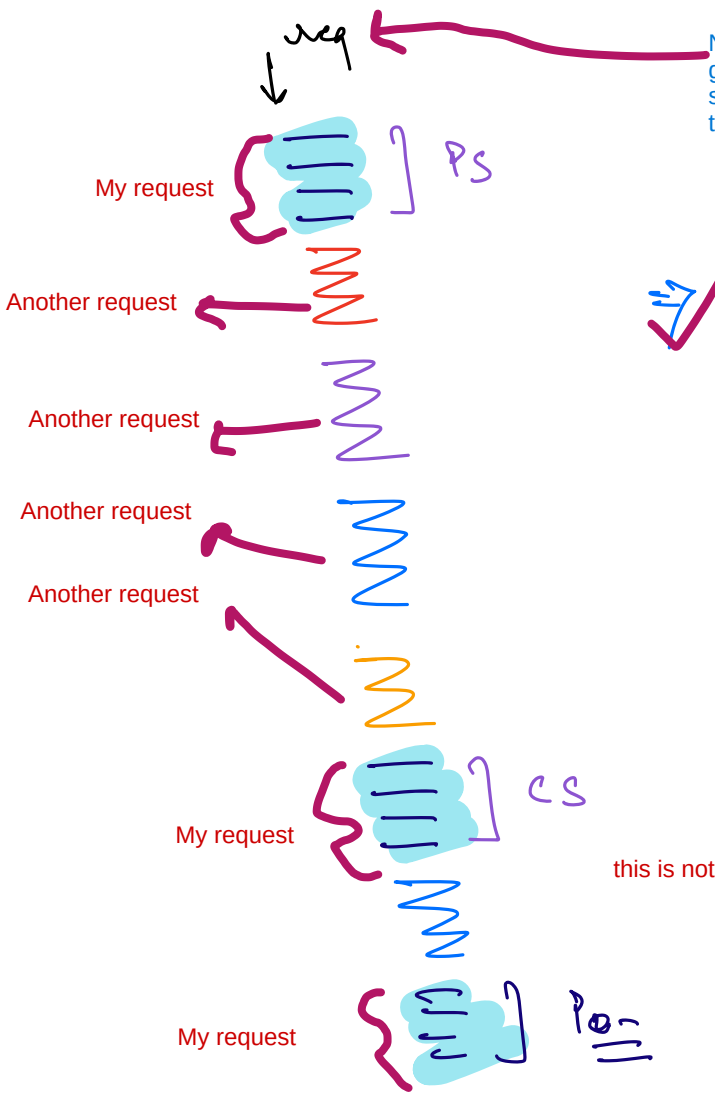
One most imp property of log server -> should be "search" since you are storing logs to go through it -> so searching should be very fast

⇒ Search

This search used ElasticSearch

↳ ElasticSearch.

↳ Inverted Indexing.



Now, when a user orders on Amazon, he places a lot of requests and req goes through 50 different MSs. At the same time, multiple users also are sending req to the same server concurrently. So the log will be saved in that way

⇒ We need a way to search the logs within the complete log file for a particular request.

this is not a good way of storing the logs

⇒ traceId / requestId / correlationId / ...

When your req will 1st enters into the system, a unique ID will be generated and that Id will be attached to all the request going forward. For eg-> PS will create a unique Id, lets say a reqID now when the PS will call the Cart Service, it will send the unique ID and when card Service will call Order Service, it will send the unique ID to Order Service so that all of these logs are easily searchable with that unique request ID

✓ get all the logs for a particular traceId & sort it on the basis on timestamp.

in increasing or decreasing order

⇒ 3 months.

Amazon maintains a log of 3 months in the servers and then they compress them and store in cold storage

MONITORING.

✓ Keeping a check on how our App" is working in Production after being deployed.

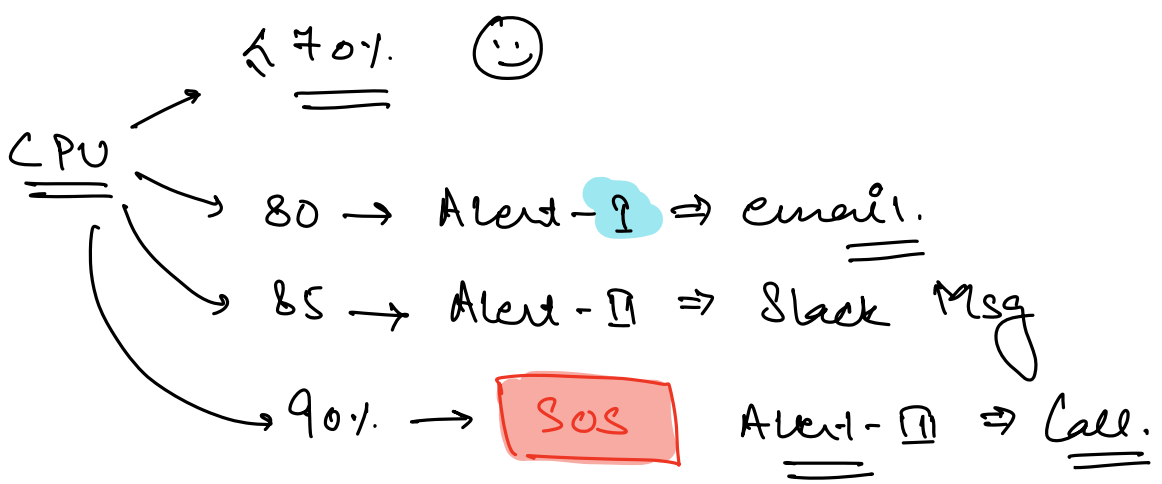
If your server lets say can handle 100,000 req per min and now you are getting 10* 100,000 of the traffic- > don't you think your server might crash -> your applications will go down and users will not be able to use your service -> if you donot monitor

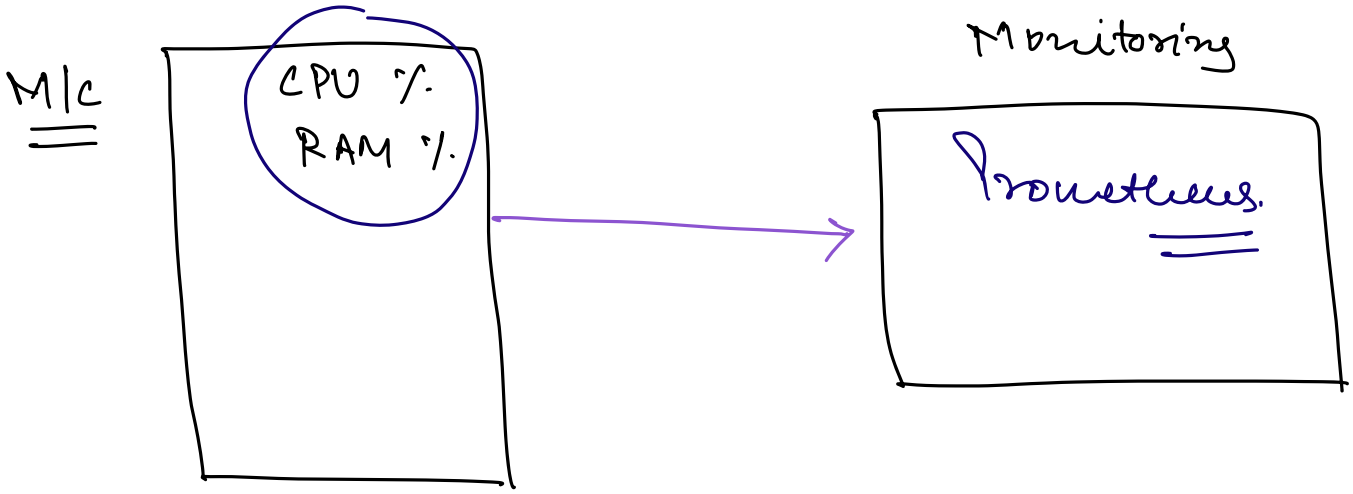
→ Performance of important API's.

Monitor all of these →

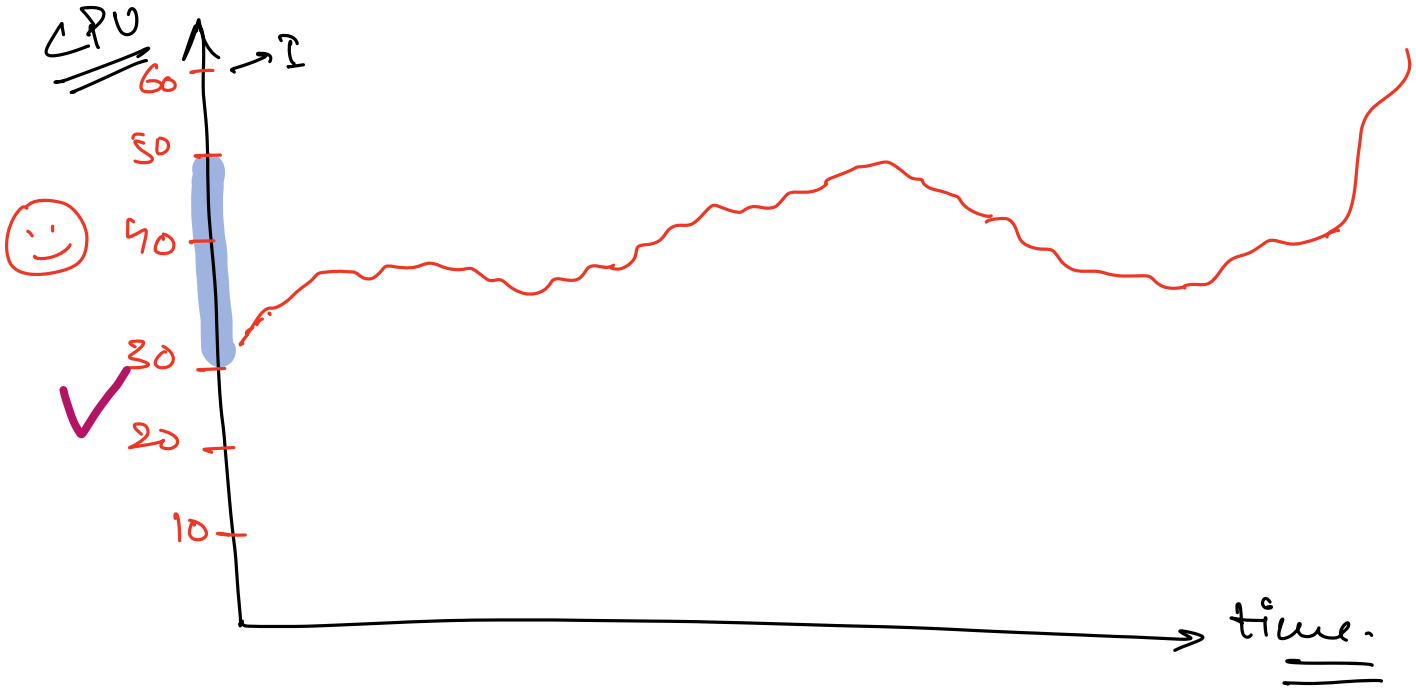
- CPU
- RAM
- Disk

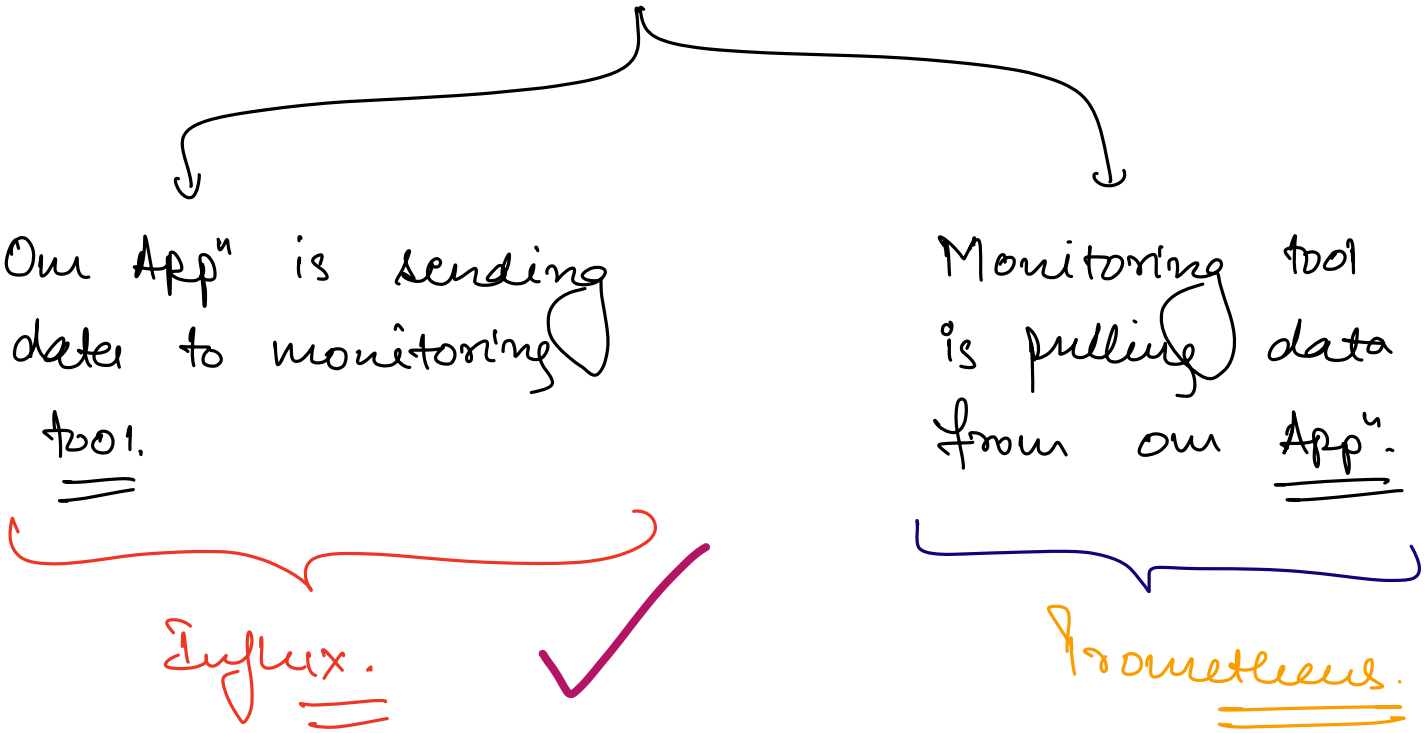
Prometheus is one such tool through you can monitor CPU performance in Production. And if CPU Utilization is <70% then everything is okay, if it goes 89% then you get an email, if it reaches 85% -> you get a slack msg and 90% is SOS and you may get a call





Monitoring tool needs to get data about different metrices (CPU/RAM...) from our server.





Add Spring Boot Starter Actuator dependency in Maven and run the PS file

Go to "localhost:3030/actuator/metrics/system.cpu.usage" for seeing the cpu usage

For Monitoring you have the below libraries:-

1. Prometheus
2. New Relic

For Tracing logs:-

1. CLoudWatch
2. LogDNA

a log management tool that helps users track, debug, and troubleshoot applications