Microservice (different module)

Eg: Smart contact manager. {code with Durgesh}

Exam portal

Blog application

* monolithic architecture: multiple components are combined in single large app.

Eg: spring boot examples

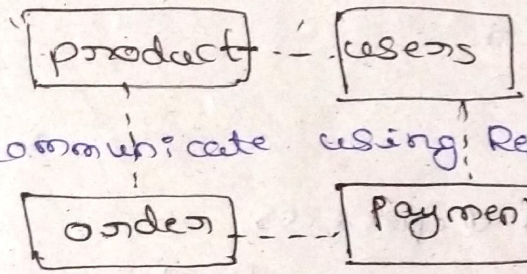
Disadvantages of springboot:

→ Change in one service then whole app is redeployed.

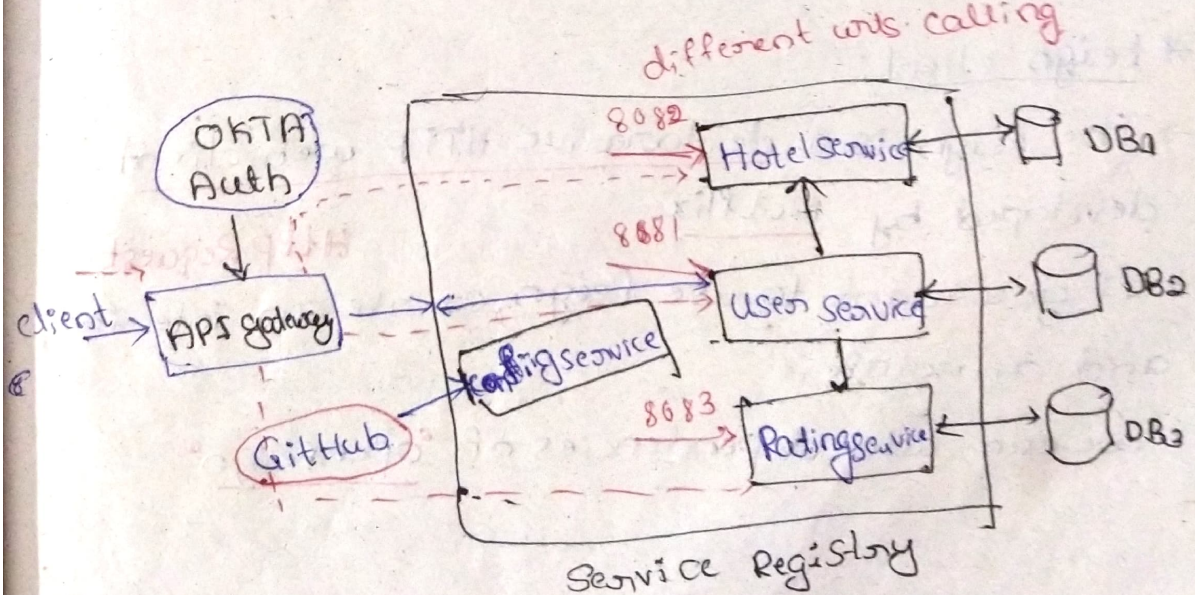
→ Problem is scale.

* Microservice: (Independent service & own database)

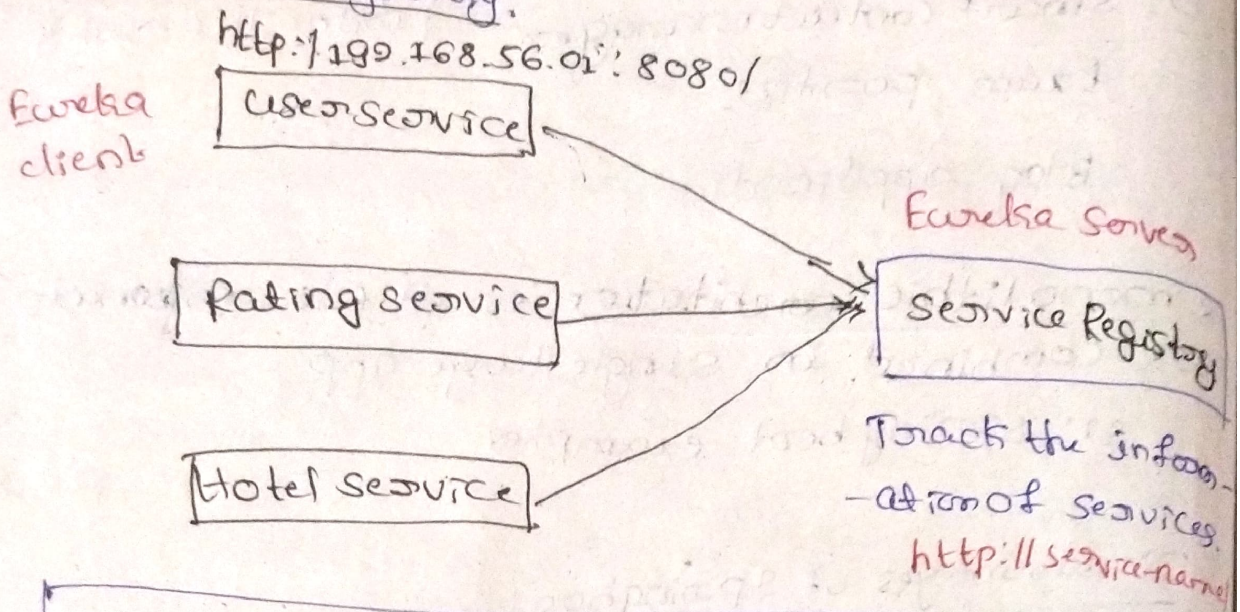
→ Large app are divided into single (small) parts.



microservice



* Service Registry:



⇒ Spring web → Rest apis.
mysql → DB
Data JPA → Hibernate support.
lombok → getter & setters.

* cloud Bootstrap (Spring cloud) dependencies
→ Eureka ~~client~~ server.
⇑
Service Registry

* For client add dependencies.
→ cloud Bootstrap, Eureka client.

* Feign client:
→ The Feign is a declarative HTTP web client developed by Netflix.
→ If you want to use Feign, create an interface, and annotate it. (Http Request)
→ we can add dependencies of "openfeign"

* API Gateway: Application programming interface gateway is a software which collect all will take all user input's or api's and route it into one or more backend services

→ add gateway of dependencies

→ add webflex (spring Reactive web) of dependencies

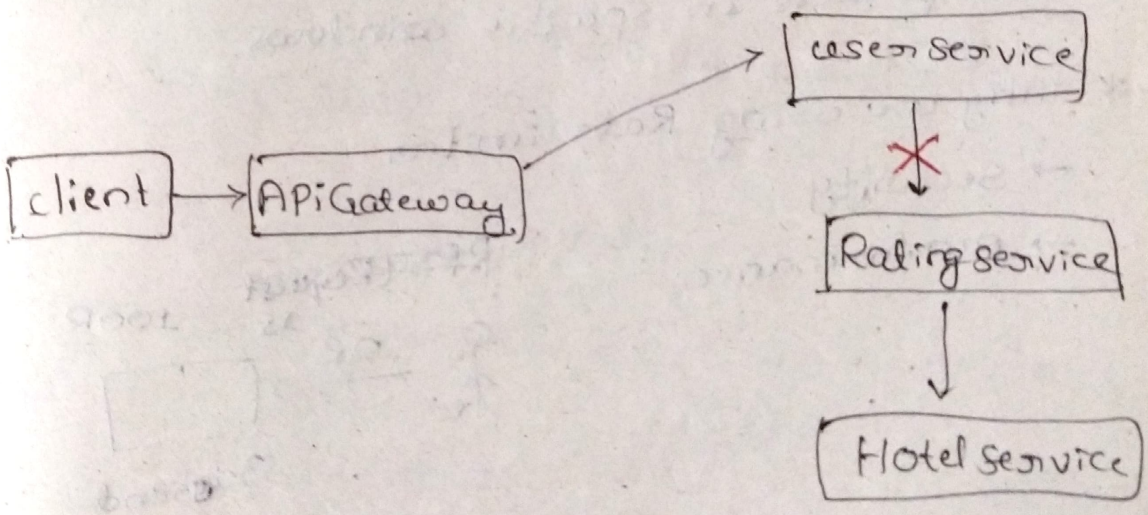
→ Eureka discovery client dependencies

* Config Server:

→ It provides client server architecture for using this for this helps make configuration externalize using distributed system

→ Config server, Eureka discovery client add these two soft dependencies

* Fault Tolerance: (Resilience 4j)



→ Rating service is down because of some reason then user service donot fetch the data from Rating service this senario is a fault.

→ For that how to tolerance these fault.

*Circuit Breaker: Circuit Breaker

Hystrix resilience4j

→ Circuit Breaker has 3 normal states:
closed, open, half-open

→ Add actuator, aop, resilience4j add dependency

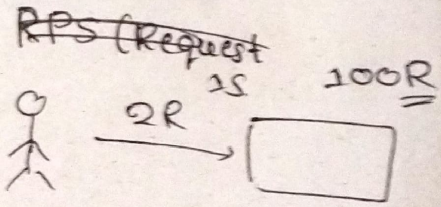
* Retry: Retry also has some limitation on maximum attempt

* Rate Limiter:

→ This functionally allows limiting access to some service

→ Rate Limiter make services highly available by limiting the number of calls we could process in specific windows.

* Why we using Rate limiter
  → Security
  → performance



RPS (Request per Sec)
RPM (Request per minite)
RPH (Request per Hour)

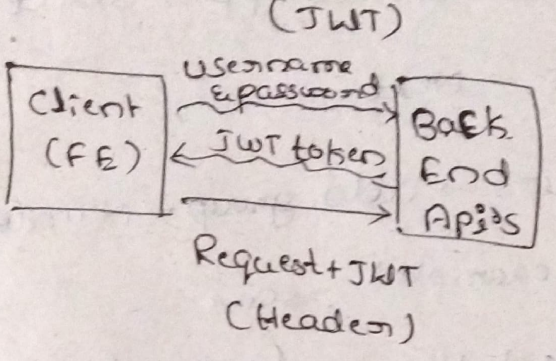
* For testing these we use JMeter software (jmeter.bat)

* Testplan → Add → Threads → Thread Group
* Thread Group → Samplers → Http Request
* Http Request → Add → Listeners → View Results Tree

* Security with OKTA:

→ we use JWT also for security

→ JWT is 'stateless' means no save any data on the server

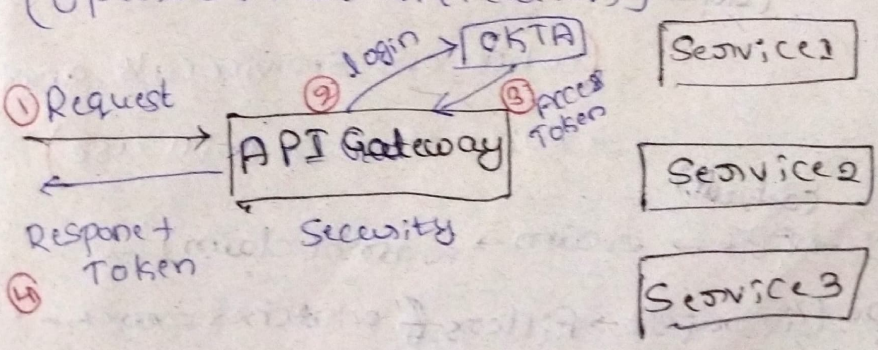


→ In JWT first give the username, password then only we can access the Backend apis and In the Header we can pass Request as. JWT token example Bearer ExABOC....

* OKTA is 'Third party Service'

* provides the authorization and authentication Server features.

{ OAuth: Authorization + openId: Authentication }



steps:

→ OKTA developers → Signin with google.

→ security → api → here we will get all api information.

→ Applications → Applications → here we will create the applications

~~(1)~~ ~~create~~

(1) Application → create a new app → select openID → web applications → Next

(2) Directory → Groups → add group → Name → (admin as example) → save

(3) Application → select the application (mywebapp) → assignments → Groups → clickon assign → assign to groups → assign

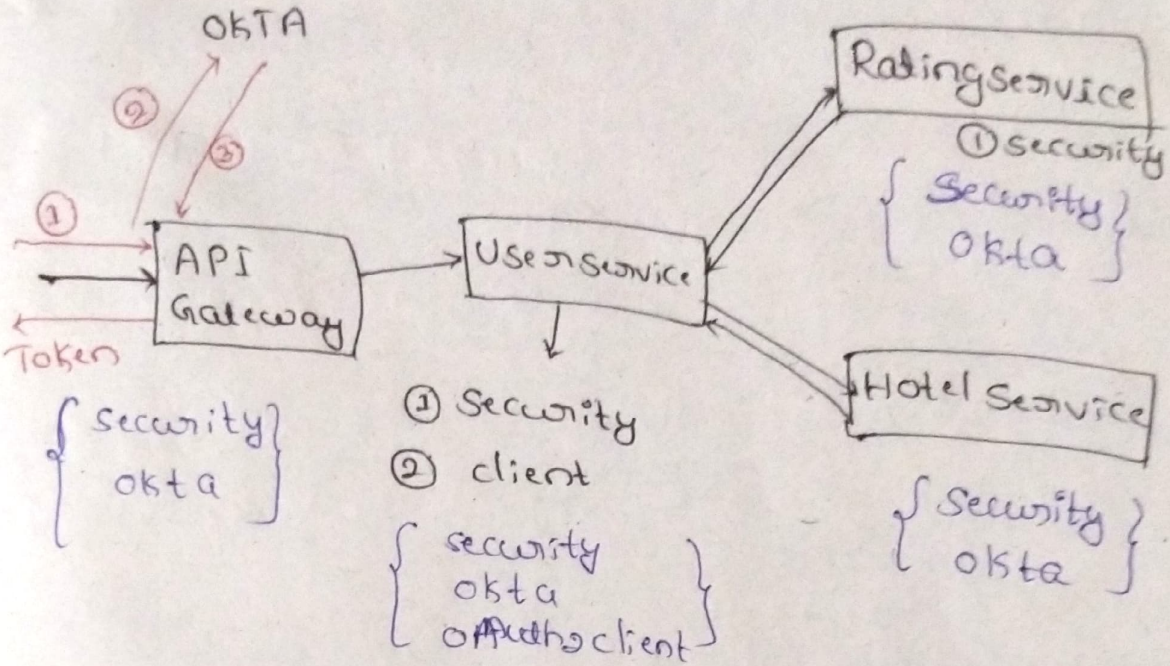
OKTA → peoples
Groups
Applications
API
scope
claims

(4) Security → API → scop → Internal → create.
(default)

(when one service call another service then we will use)

(5) Security → API → (default) claim → Name (claim) → valuetype (Groups) → Filters → Matches regex - .* → create.

* API Gateway → security, okta dependency we add



* For auth login

→ First login with auth is required for that login with google select → login → done.

→ Then we will get access token.

→ Applications

→ Directory

→ security-api

Microservices (Udemy)

* Docker:

* Deployment: How we deploy all the tiny 100s of microservices with less effort & cost?
* portability: How do we move over 100s of microservice across environments with less effort, configurations & cost?
* Scalability: How do we scale our applications based on demand on the fly with minimum effort & cost?

→ we can achieve based 'docker'

* creating Docker image:

→ Run open spring boot project in cmd.

→ Run mvn clean install

→ If build success it will show the .jar file inside target file

→ mvn spring-boot:run → spring boot will run

(or)

java -jar target/filename.jar

* steps to create docker image:

1. open spring boot project on cmd.
2. Run docker images → no images found
3. docker build. -t eazybytes/accounts.
   - -t → tag
   - eazybytes/accounts → name
4. docker images → image found.

eazybytes/accounts	latest	706ca53...	2min	464MB

5. docker image inspect 706 → Details of container
6. docker run -p 8080:8080 eazybytes/accounts
   - -p → port
7. after these if we hit appln url on postman we getting the o/p
8. docker ps
   - ↓ show whatever containers runing ~~on backend~~
9. docker run -p 8081:8080 eazybytes/accounts
   - ↓ another container will run
10. docker ps.
11. docker logs fc → container ID starts with fc
    (or)
    docker logs -f fc
    - For checking logs.
12. docker stop fc. → stop the container
13. docker ps -a → show containers which are present in the containers means whether which are running or stop.

14. docker start fc ee → again starts the docker

15. docker container pause fc → pause the container

16. docker container unpause fc

17. docker container inspect fc. → Details Details of container

18. docker kill fc → not gives any logs after runing command.

19. docker stats → gives details of cpu, memory usage

~~20.~~

~~20. docker stop ee~~

20. docker rm fc → ~~re~~move containers
↓
remove
from stoped containers also

21. docker ps -a → now no containers are available

22. docker run -d -p 8081:8080 eazybytes/accounts

→ container docker while run but not any logs.

* Build packs: (pakeeto-build packs)

→ Detect what all the dependencies that you have inside your application, what is your business logic so based upon that, they will create you an image.

→ open Spring boot project on cmd.

→ mvn Spring-boot:build-image

which is create image which internally use build pack to generate docker image

→ docker images

→ docker run -p 8090:8090 eazybytes/loans.

→ after these we can use from postman

* Pushing docker images from local to remote docker hub repository:

→ hub.docker.com ⇒ open url.
→ open docker images.
→ docker push docker.io/easybytes/accounts

project name

* Docker compose:

→ If we have 50 microservice then we have give command 50 times for 50 microservice as previously discussed.

→ But 'docker compose' will gives single command to run all these things

→ Docker compose is a tool that was developed to help define & share multiple containers applications.

→ To these you can create yml file to define what role the service that you want to create with a single command.

Steps:

1. docker-compose version
2. create .yml file for all the services
3. docker-compose up → run all services
4. docker ps
5. docker-compose stop → stop services

* Add extensions → Logs Explorer → Install

↓

Docker Desktop. we can install these because it will help to see logs.

* cloud native applications:

→ Collection of small, independent & loosely coupled services.

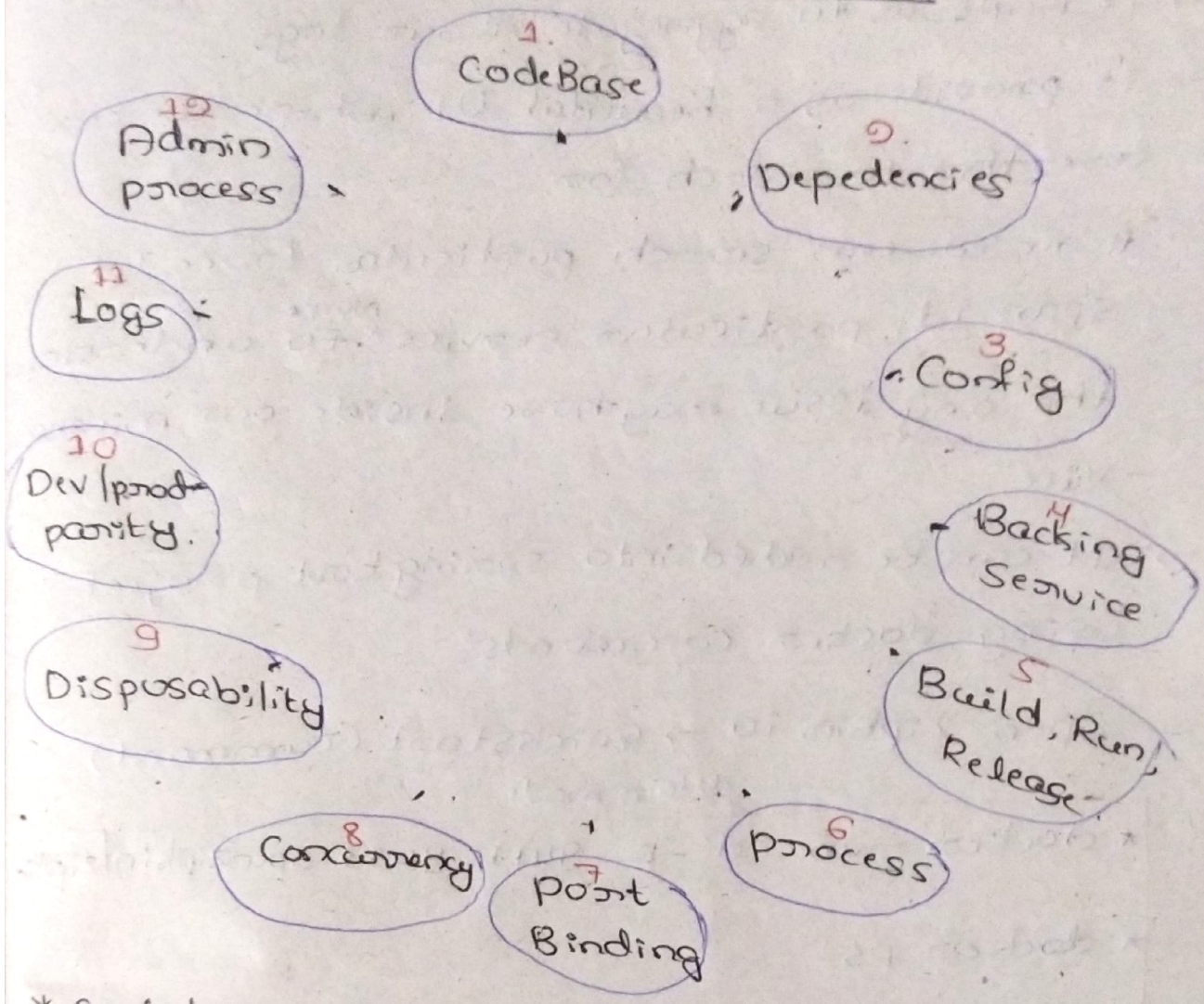
* principle of cloud native:

1. Containers
2. Microservice
3. DevOps
4. Continous delivery

→ To build and develop cloud native applications, ~~we need to follow~~ we used "twelve factor app methodology"

cloud native applications	Traditional enterprise appln
* predicatable behaviour	* Unpredicatable Behaviour
* OS abstraction	* OS dependent.
* Right-sized capacity & independent	* Oversized capacity & dependent
* Continuous delivery	* Waterfall development
* Rapid recovery & automated scalability	* slow recovery

* Twelve factor app methodology:



* codebase → Each microservice have own microservice & these data base

* Config → different different environment have different configuration. (dev, test, prod)

* Concurrency:

Vertical scaling (scale up) : increase in cpu or RAM

Horizantal scaling (scaleout) : adding more instances of the applications

→ always go with Scale out not up.

* Zipkin: (zipkin.io)

→ It helps us to aggregate all our logs

→ It provide us a beautiful UI ~~where we can use that to search for~~

→ Here we can search particular trace Id, Span Id, particular service Name to understand the any issue may have inside our micro-service.

→ It can be added into spring boot project using 'docker commands'.

→ open zipkin.io => Quickstart (command)

* docker run -d (detach mode) -p 9411:9411 openzipkin/zipkin
* docker ps
* open localhost:9411/zipkin/ in the browser
* add zipkin dependency in spring boot project
* add configuration in appln.properties file.
  spring.zipkin.baseUrl = http://localhost:9411/
  spring.sleuth.sampler.percentage = 1 (100%)
* ~~open~~ Run SB project

→ * Search service Name then we can see all details which method call, controller, start time, end time, also get visualization of microservices.

*Rabbit MQ:

→ push all sleuth info into Zipkin server like Rabbit MQ

→ Rabbit MQ is one of the implementation of Java messaging service mechanism, like where you can push logs asynchronously into queue.

* add Rabit dependency in SB project
* Run command on command prompt as

=> docker run -it --rm --name rabbitmq
-p 5672:5672 -p 15672:15672
rabbitmq:3-management

(accept UI; 15672 → management UI)

=> docker ps

* open localhost:15672 on browser, by default username & password will be guest
* First setup the Queue so dfe
  * click on Queue → add a new queue → give the name → add queue
* add configuration into appln. properties

spring.zipkin.sender.type = rabbit
" " .rabbitmq.queue = zipkin
spring.rabbitmq.host = localhost
" " .port = 5672
" " .username = guest
" " .password = guest