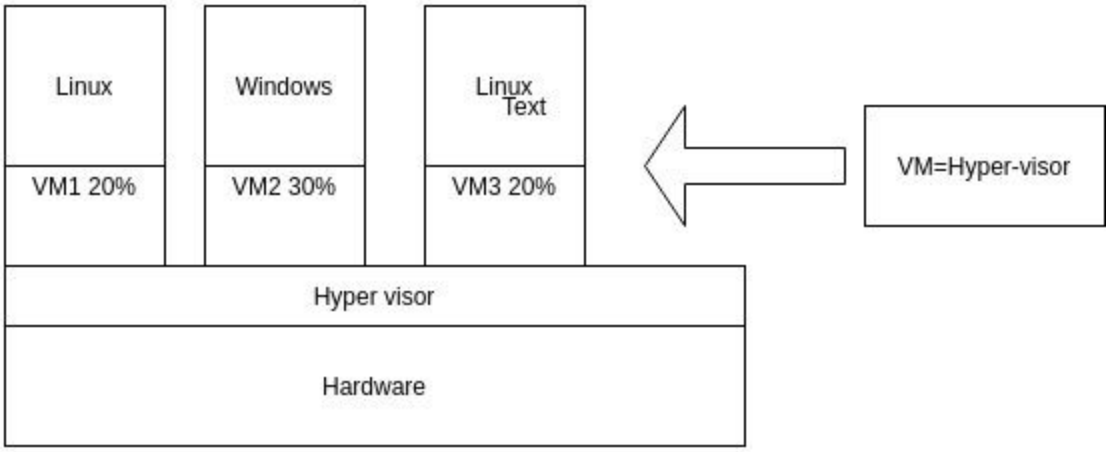
Microservices

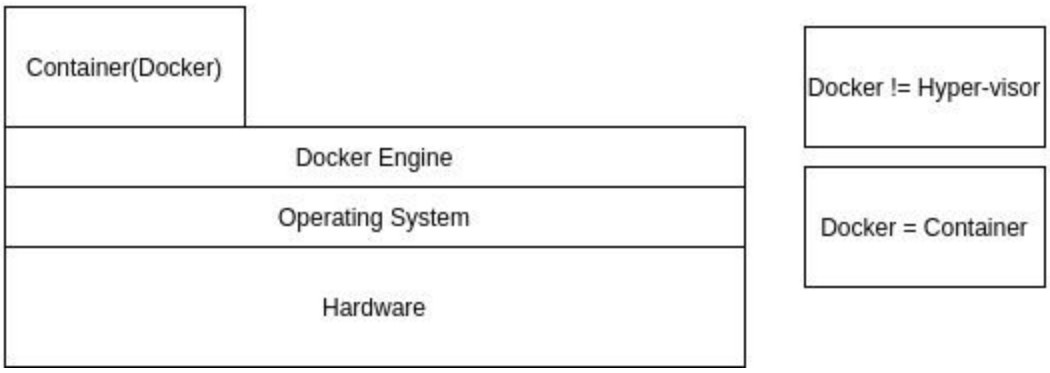
1.Docker

a) Physical Servers

b) Virtual Servers



c) Docker



1. Docker Engine is not in the docker project it is use as registry,ocestration,service and security

2. Registry - Place store your docker images (docker hope-largest repository),customize existing images and push it back
3. Ocestration - is the process all instances get together and go for a common goal . ex: If we have to deploy some services(auth,login) in different host what ocestraion does get responsibility move those services to host and controlling steps - Kubernetes
4. Docker is a kind of persistence or like a VM
5. Legacy applications doesn't support docker architecture - half true , better migrate
6. Docker fit with micro level service architecture
7. OCI - open container initiative (Docker + RKT)
8. Use principal call allocate and share(this much of time from container) - can do process allocation for container(allocating processing power)/limit the cpu CFS(completely fair scheduler) quota / limit cpu real time period in microseconds(out 100000ms need much 50000ms) / if allocated time exceed it won't affect other containers/ex :Unix -Nice - you may have 32 cores it is not you have to run 32 processes.
9. No of containers can exceed no of cores(CPU)/same as memory
10. Docker is a container which is another process in operating system
11. Docker toolbox - include linux image to process docker command(windows ,mac)
12. Dockerfile -
    FROM ubuntu:latest - command docker repository go and download : is version
    CMD ["echo","hello docker"] - command for print
    RUN - command for run
    WORKDIR - command for change working directory
    EXPOSE - expose localhost current port
    ENTRYPOINT - when start docker this is the thing you have to do
    ADD - command copy file given location add it into container given location
13. Docker commands
    docker build . - need to build location

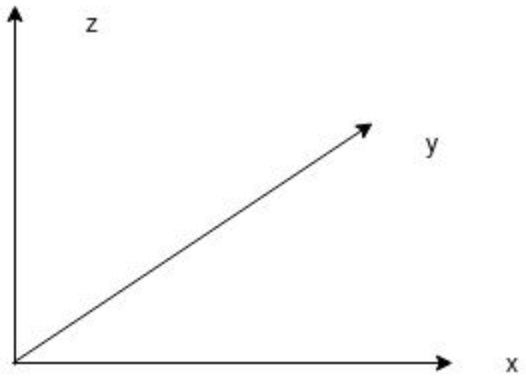
2. Container less application development - Spring Boot

1. Web Server - Handle Http requests and responses,in terms of java web server doesn't have any kind of capability to handle java EE related specifications ex : Tom cat
2. Web Container - Capable to handle part of java EE specification - Servlet
3. Application server - Capable to handle full spectrum of java EE specification (RMI) - Jboss,weblogic
4. Spring boot - Spring for DI,Jersey/Jaxson for REST,jetty for http servers , embedded container

3. Web application to Microservice.Go or No Go

1. Monolithic applications - all code base inside the single/one wrapped content(single jar or war file) - ex: rent car application(UI,Business layer,car registration) / disadvantages - any minor change need to deploy all single file, may be application send to offline ,debug process very hard because no one know end to end in this kind of huge application/ advantages - test process is easy/easily mocking ,easily monitor the application
2. **Microservice** - suggestions - SOA2,some level of SOA
    a) Has dedicated purpose of living/**domain driven development/exact(well) define scope**
    b) Run own processes - own web container
    c) Communicate with other services via lightweight way mainly use HTTP
    d) Scale and deploy as individual services -
    e) Should maintain decentralized control as much as possible
    f) Do one thing do it well
    g) Implement using different languages
    h) Two family meal feed team (10 -12)
    i) You build it you own it - have responsibility to strong service
    j) Availability friendly - microservice are loosely coupled,so it will not create bigger load on single server.also failures are separated so there is not single point of failure any more

    k) Scalability -in system one module may have high demand than the other one .in such case we can scale only and only particular call .since it loosely coupled it is not required scale unnecessary module when you need to scala .Also if one service has problem that does not case to cascade failures.(example - ref : Book - The art of Scalability)



x - axis - scalability up and down/ not enough spawn up ex - clustering /adding more data centers

Y - axis - functional decomposition
Z -axis - sharding ex : european customer separate server

I) Microservice is a component deploy as a service,so each service can communicate each other get the final job done

Spring Cloud

1. Released in 2015
2. Open source framework
3. Run anywhere
4. Build on industry standard patterns and best practices
5. Include Netflix OSS technologies

Spring Boot

1. Started early 2013
2. Version 1.0.0 released on April 2014
3. Convention over configuration - no tons of config file but there is convention property file should be this location
4. Opinions can be override (i.e. Change port whatever you want)
5. Dependency management made simple
6. Container less (embedded server into jar)
7. Most of required features are inbuilt as various frameworks

Basics of Spring Boot Application

1. **Controller** - Responsible for handling the traffic/route the traffic for required service/last layer handle the exception /controller method call from service(HTTP) url .

   Annotations-
   @RestController/@RequestMapping(value,name=Request Method.POST)

2. **Service** - hold the business logic/generate session/handle transactions
3. **Repository** - suppose to deal with database/CRUD operations

Rest client - to deal services ex: postman,insomnia

Resources/config dir - when running the application it consider as a classpath/take as property file

4.**Actuator**

- Give predefined endpoints from actuator service :ex-health,environment,configuration,beans - actuator/health ,actuator/threaddump
- threaddump - movementally state what are the all the threads running on your application
- actuator/beans endpoint not working(2.0) because need to config in application.yml to available it in endpoint
- **Create own actuator for enable log level dynamically ,enable or disable searching features dynamically,dynamically change configurations,log level - extending actuator endpoint**
- Using actuator ,can **change value at runtime**
- Ex: level and threshold value - need read and change dynamically
- @Component/@Service/@Repository-extends from component : create a bean
- @Endpoint(id) : automatically expose this component for webflux,jms-id means this is the one use after the actuator-actuator/stage
- @ReadOperation - http get call come here
- @WriteOperation /@DeleteOperation- http post call come here
- @Selector - need given value from function like path param ex: http://localhost:8080/actuator/{name}

Note :

- DB transaction cycle - begin,commit,rollback
- Create own repository and extend with **Repository-** super,CRUDRepository or JPARepository(crud + pagination)
- createDatabaseIfNotExist,ddl-auto-create(data will lose when start the service but dont drop table)/create-drop(drop the table when start the service)/update(not delete the table and data also,if new field add model class that also update in table if it is not null)
- When multiple services involved that come service discovery also load balancing ,secure services need to put secure roles,resilience implementation(circuit breaker pattern),externalise configuration,deployment(run more than one instance -scale up/down,without replicating entire application should able replication individual components(individually deployable/scalable)),distributed tracing,deploy aws
- **Externalise Configuration(configuration store and server)** : This is a configuration server this give configurations through http.(

looking directories - classpath,resource,resource config,current directory ,current directory/config)

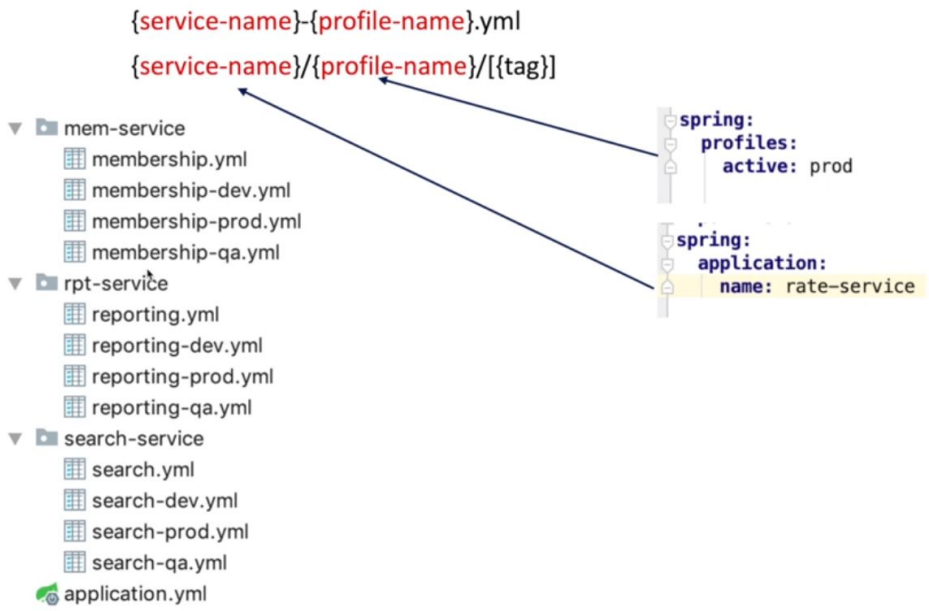
Local file system - file system backend - native profile(configurations are store in locally)

Github/hosted configuration

Reasons:

- Need to separate configuration from our code base
- Make sure each service have same consistent configuration working multiple services
- Change configuration when application run(on the fly)-log level/docker - each environment need separate container using profile configurations in code - ex:prod configurations,qa configurations
- application.yml is the default config file for all services and each services they have seperate profile level files
- ex:membership/default - will return membership.yml,application.yml | membership/prod - will return prod,default and application.yml | account/default - will return application.yml | membership-dev.yml - get exact file also we can get this as property or json file as well
- http://<host:port>/membership/prod/{} - can add version tag/commit id/ branch or new repo also here
- Need to create client service consumer this configurations
- 8888 - default port for the configuration server
- Configuration process before the application.yml file if not available config server it use default configurations for that we use bootstrap.yml file to move configurations in there
- Override server port details from config server
- When spring serialize it use bean naming configuration
- Fetch to ui use @Controller annotation
- Create separate maven project to keep common model classes which use in other services as a dependency - dependency on dependency(transitive dependency) and @EntityScan annotation use to get base packages from related project

- ❖ If we need to create bean have to have @Component,@Service or @Repository annotations



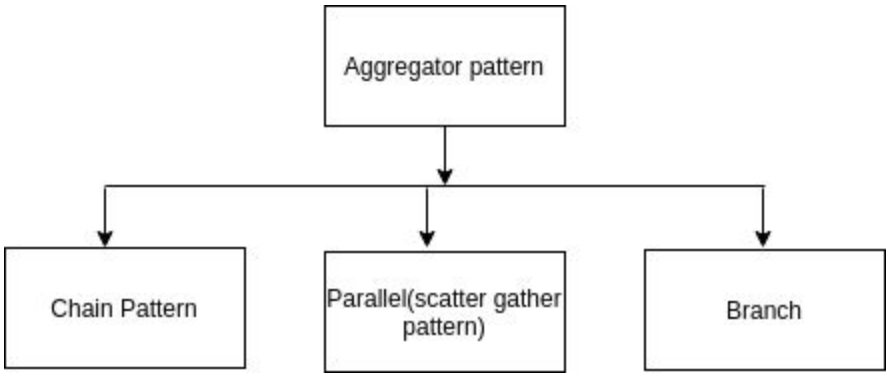
4. Best practices for the microservice development

1. Design should be domain driven design
2. Need to use service discovery method because some one can use hard coded values(host:port) to call one service other service
3. Fail fast log later
4. Generate unique id when first hit for server - core retaliation id
5. Service versioning - best option is semantic versioning
6. Deploy microservice - docker (host - operating environment)
7. Authentication and Authorization mechanism
8. Maintain IAM service to validate services - first hit service then need to send IAM to validate for further steps
9. Dependency - independency is must

10. API contract /Don't break consumer - CI use for test to identify if there any issue in newly developed service before it deploy
11. Fall tolerance
12. Documentation - swagger

5. Design patterns for microservices

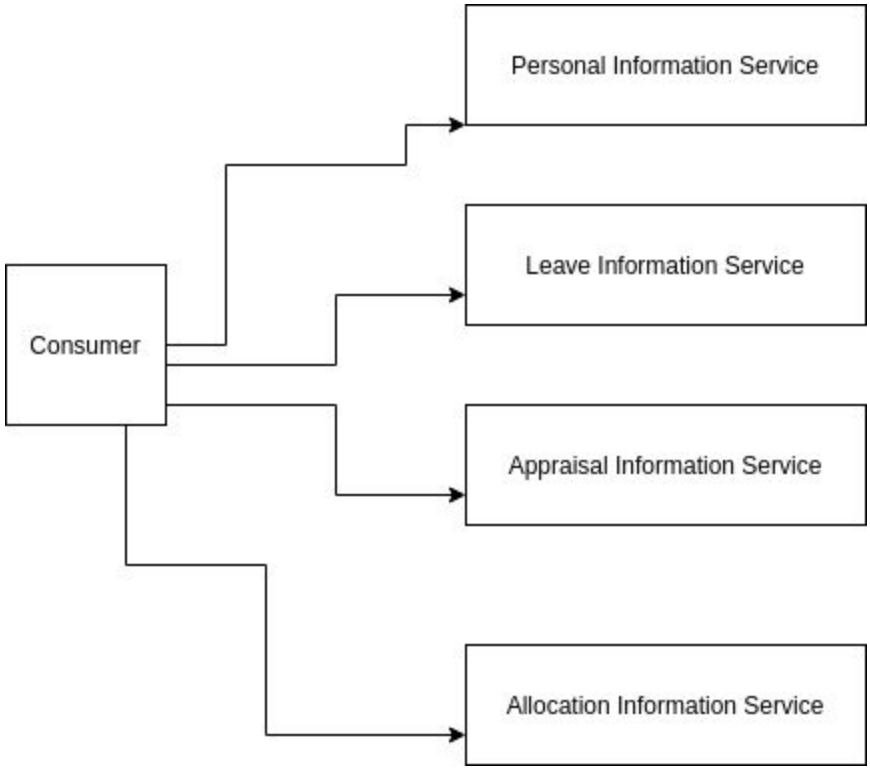
1.Aggrigater Pattern



Note:

- One of EIP(Enterprise Integration Pattern)
- Service chaining is invoke one service and get the response take one property of that response and use to that property to invoke second service
- Scatter gather pattern - ex:Invoke airline ticketing api and want to travel colombo - los angeles and want know airline prices (sri lanka,quarter) for that send request to all airline service and get back the response merget it and get back(scatter it and gather it)
- Better one database per service - basics is independently deployable independently scalable :ex customer service writing one db,vehicle writing one db . when we scale up the application evering service data include one db is bottleneck for required output

- Example Microservices:
  1.Get personal information
  2.Get Leave information
  3.Get appraisal information
  4.Allocation information



- Note 1:

  ESB has a concept call 'intelligent pipe' and 'dump backend' but microservice has dump pipe and intelligent backend

- Parallel :

  Call services parallelly and aggregate all results and send it to requested consumers

- Chain :

  Call one service and get result from their service pass that result to next service

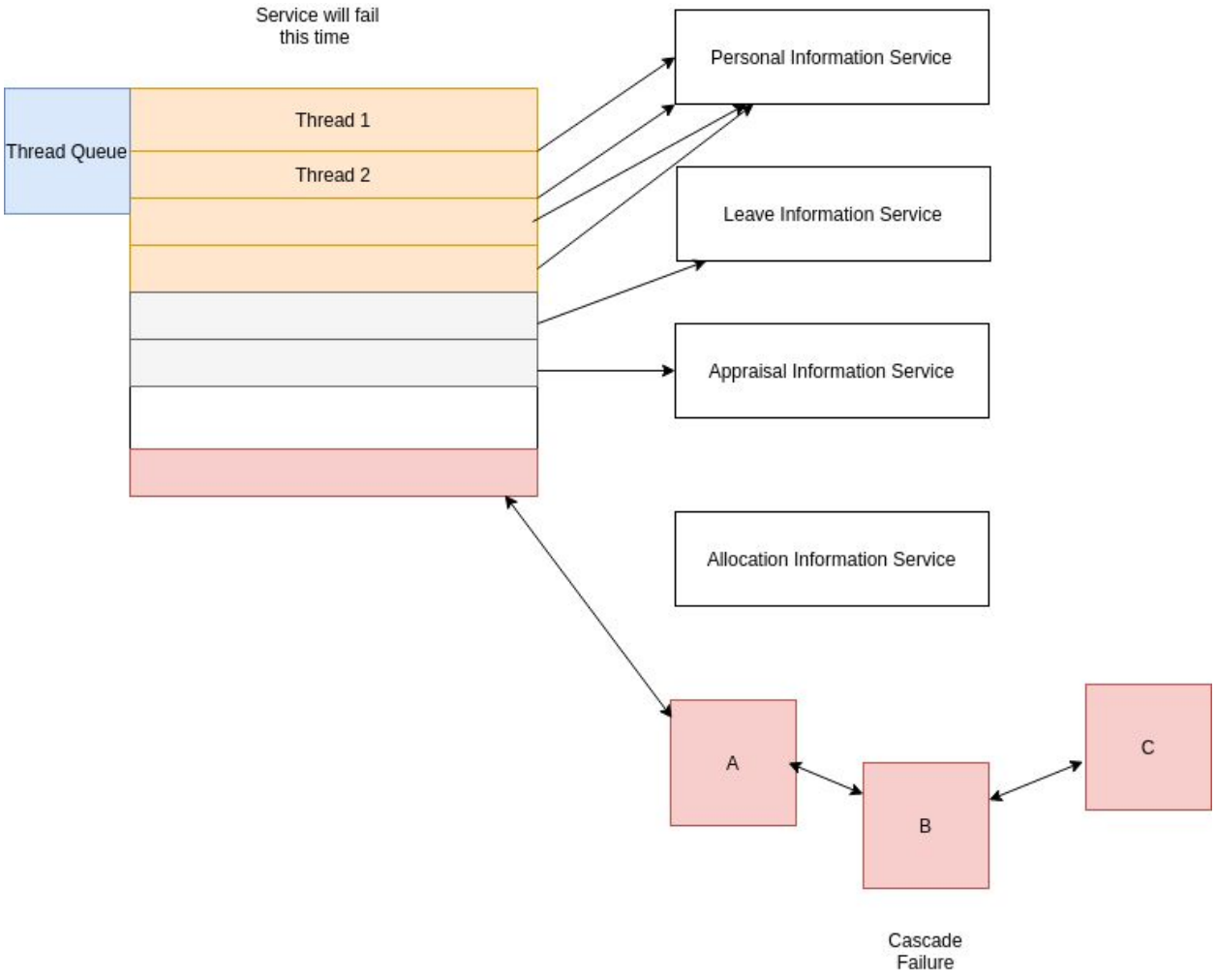
- Branch:

  Call services Based on a decision.ex: user need to get employee details from marketing branch

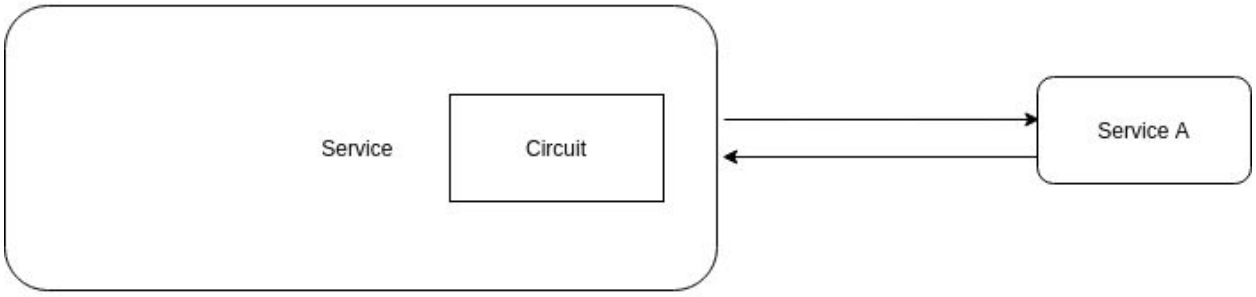
- Note 2
    - Best deploy our services and consumer application same time but microservice perspective we can deploy our services without any consumer response breaking
    -
    - Need to keep separate IAM service to validate the consumer request before hit the our
    - Aggregation service

2.Curcuit Breaker Pattern

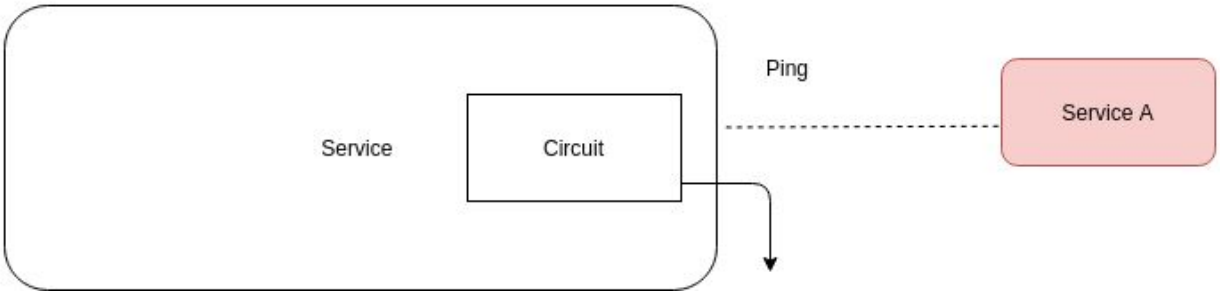
- Sustainable pattern ,keep your service without dying/keep your service alive
- If no of request occurrences missing given upper threshold(0-150ms -delay/150-200ms -risky,200ms < - cut off services) value service is failing


Service will fail
this time
Thread Queue
Thread 1
Thread 2
Personal Information Service
Leave Information Service
Appraisal Information Service
Allocation Information Service
A
B
C
Cascade
Failure

When Service in good response time



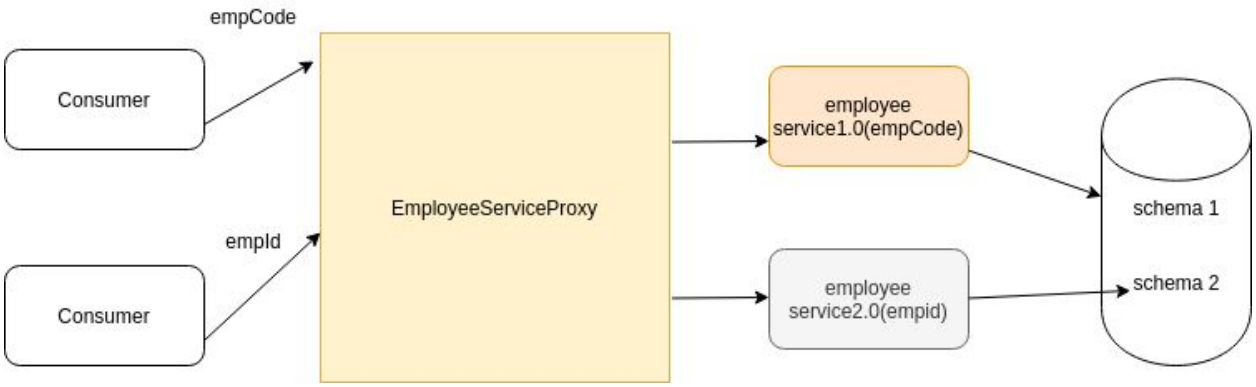
When Service failed



Note :

- Use circuit breaker in the caller service (ex:Rent service - ita call the othe service such as customer and vehicle)
- @HystrixCommand - to call this annotation the method will be the first method in the controller - not recommended (lot of limitations in the code)
- Separate Hystrix common for each services
- Use CommonHystrixCommand

3.Proxy Pattern



- Semantic versioning has major,minor and patch versions ex: 2.5.9 - 2 -major,5 - minor,9-patch (Major version -need to increment whenever your current service not compatible with previous service,Minor version - need to increment when ever you have new functionality however you can use remain functionalities ,Patch version - can up need)

- Service Discovery- whether infrastructure change need to change hostname,ip address everything can change So when connect proxy with service discovery it will ask from service discovery,where each service locations ex :Wos2 governors registry,consul
- Avoid cascade failures in service chaining in proxy pattern need to use message queues
- Thread pools

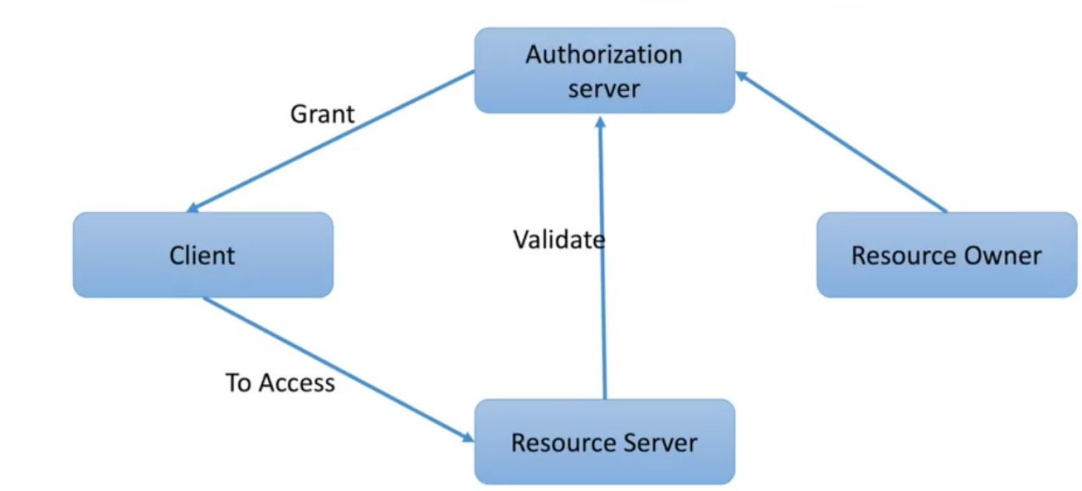
6.Service Authentication

For Microservice authentication JWT and OAuth2 use as the best practices

OAuth2

- It’s a protocol for token based authorization
- It’s a way of delegate access without sharing credentials ,It can grant a client to perform certain action on behalf of user
- Technically oAuth is not authenticate user, User has already authenticated to get access token however most providers implement both together lead for some confusion because oAuth2 is authorization framework
- We can have 3rd party oAuth providers such as google,facebook,github or own oAuth server
- Authentication is Who you are ,Authorization is What can do

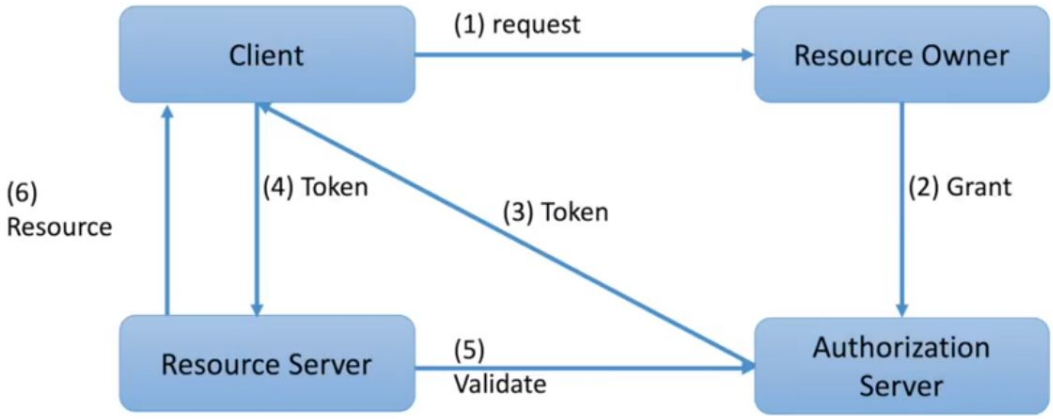
Actors



Terms

- Access Token - A random string which is not make sense on human reading
- Refresh Token - Same as access token but use to renew access Token once its expired and within allowed refresh time
- Client id/Secret - given app to identify the app(client)
- Scope - allowed permission(Read/write/or anything you define)
- JWT - widely use mechanism to pass information between services this support encryption also

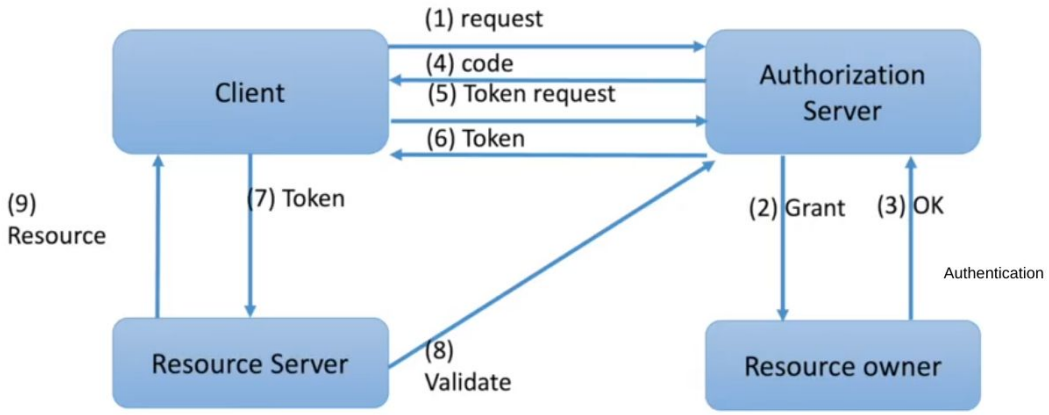
General flow



Grant types

- Authorization code
- Client credentials
- Implicit
- Password - Client take your password and send that password to the authorization server-ex:simple login
- Device code

Authorization Code flow



** Client does not see user credentials

Notes:

- Cash token for one hour using expiration time
- WebSecurityConfigurationAdapter - In new spring security version don't have access to AuthentcationManager
- 401 - unauthorized access
- Spring boot look two different files in classpath(create file in the resource directory it considered as the class path) those are schema.sql or data.sql and also define profile(prod,dev,qa) as config file
- Need put encrypted algorithm before the token{bcrypt} https://www.browserling.com/tools/bcrypt
- 'role_admin' some kind of practice in spring using _
- Spring security provide class call UserDetailsService and UserDetail
- Model is the one who carrying the data service is the one who handle the data,repository is the one who connect the database
- @EnableGlobalMethodSecurity - enable use to method level security
- Give access base on the role or permission
- Give access base on the Role(user must have role) or permission
- When user will not trust the client / hesitate the client use Authorization code flow : ex client(web ui) ask fb/google passwords to get the resources and when user can trust the client use password flow ex: office application web ui ask office credentials for get the resource server details
- For UI perspective works need to annotate with @Controller annotation
- @EnableOAuth2Sso - to inform UI service to enable authenticate from Authentication server
- Single Sign on - when you goto secure page it will redirect you to authorization server you need to enter user name and password then come back to secure page in their can invoke profile service without taking new token(once you sign in authorization server you can access any services if you have permission to access that)
- @Autowired RestTemplate - rest template is the way import the remote service

7.Asynchronous short lived microservices

- HTTP - synchronize protocol
- To make asynchronous processors so can make reduce dependencies between services - usually REST service use HTTP as protocol and which is synchronous and it cause wait until process complete(suggest:messaging)
- Reduce blocking calls can increase latency and deliver high throughput
- Support for event driven model can trigger external services beyond usual request response architecture
- Can avoid cascading failures due to thread or resource lock/block

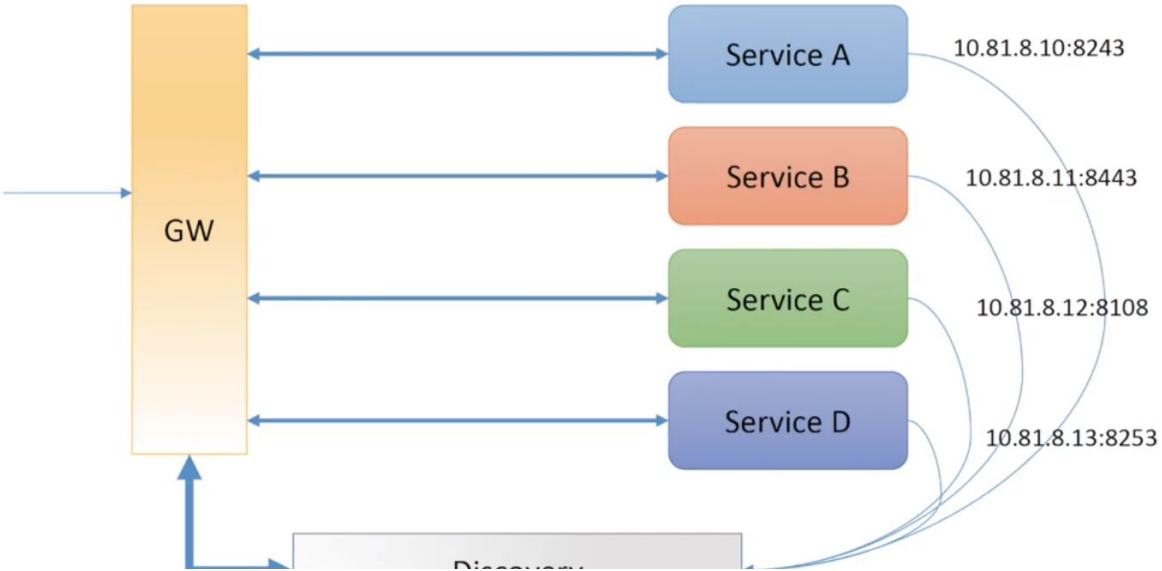
Important

- Sometimes we have services which not frequently used
- Though its not use frequently we have to keep it online to serve when it need
- Solve problem of scaling problem as task work on demand no coupling

Note:

- Serverless computing - Spring task implement function and deploy that function
- Asynchronous - one service should not wait until other service complete.non blocking calls
- Born it self do the work and die it self
- Independently deployable independently scalable
- Whatever parameter comes it takes it and executes. Ex:if completed some submission task need to send email or message

8.Discovery Service(Netflix Eureka)



- Service A(Customer Service) want more instance to expose when traffic is more high for that there will be separate ip address for same customer service.when client request the customer service through Gate way service what discovery service does it passes the appropriate customers service when it available(up and running) as well as all services registered in the service discovery service