Microservice

# *Why we use Microservice*

*Suppose we have monolithic application and contains 3 modules let’s say chatms,userms,orderms*

*Here our all three modules will be dependent on each other by compile time dependency that means suppose we want to release the new version of every module and our chatms is ready to release but others are not ready to release then we can’t release chatms separately because it has compile time dependency on other modules , similary if want to use different languages, databases, packages version etc for different modules but we can’t do in monolithic application*

*So now in microservice application every module will be complete application and will dependent on each other by runtime and we can compile independently and release and we can use different languages, databases, packages version etc for different modules*

*They all having runtime dependency so each module can use other module by http request*

***Payment***

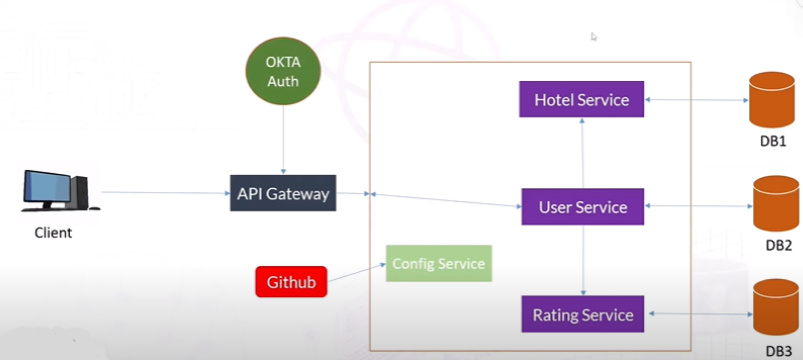
### *Product*

### *Orders*

## *Users*

*communicate Using Rest APIs*

*Microservices*



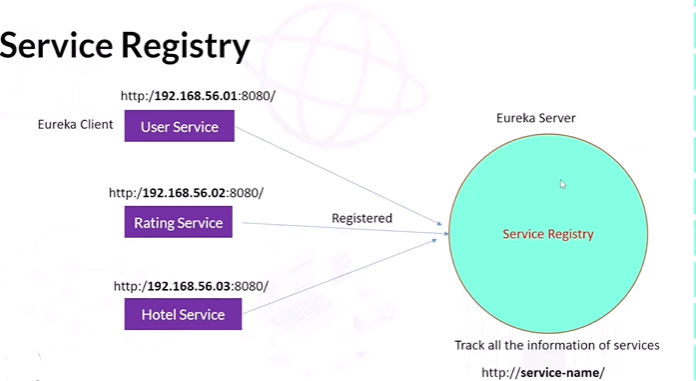
*Request will come to api gateway then it will call different services, we will use OKTA auth for authentication and if the services have some common configurations then we will keep that configurations in config server for that we will use github then we will call them in different services*

Service Registry

Suppose if our one service is running on one Ip, port and calling another service using http which obviously running on different Ip, port or different machine but due to some technical issue port or any other things like ip, path are not working so our one service can’t use other service because two services are calling each other on the basis of physical properties

*to get rid of it we will use Eureka server which obviously is one service and we will register our different services on the eureka server by name so now our different services are not dependent on each other by physical things like ip, port, now one service can call other service by http request using name*

*Eureka server/Discovery server will keep track of each registered service , we can easily see which services are up or down or the complete information of the services*



#### *Implementing Service Registry using Eureka SERVER*

*As our eureka server/discovery server is also service so we will create spring boot project*

*in the pom.xml file we will add two dependency*

**Cloud Bootstrap SPRING CLOUD**

**and**

**Eureka Server SPRING CLOUD DISCOVERY**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

*Now we for enabling the Eureka/Discovery server we will go inside main class and top of that we will mention @EnableEurekaServer*

*And now will go inside application.properties file or application.yml file and add some configuration to make this spring project as Eureka server instead of normal spring boot service*

server.port=9000(any port you can mention)  
  
#configure discovery server  
eureka.instance.hostname=localhost  
eureka.client.registerWithEureka=false  
eureka.client.fetchRegistry=false

*Last two properties we are adding because we want to prevent this service to be registered on eureka server or behave as eureka client because it is itself eureka server and other service will be registered here and will be eureka client for this service*

*Now we will run this service and will check on browser then Eureka UI will be rendered*

##### *Implementing Service Discovery Client*

*Now we will register our services on eureka server as eureka/discovery client*

*So for this first we will add dependencies inside our existing service pom.xml file*

**Eureka Discovery Client SPRING CLOUD DISCOVERY**

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

*And where dependencies tag is ending after that we will add*

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

*Now inside the service main class we will add @EnableEurekaClient on the top of the class*

*Next thing we will do is to add the configuration in the application. properties file or application.yml file*

eureka:  
 instance:  
 prefer-ip-address: true  
 client:  
 fetch-registry: true  
 register-with-eureka: true  
 serviceUrl:  
 defaultZone: http://localhost:9000/eureka/

and also

spring:  
 application:  
 name: serviceName

*This service will be registered on the Eureka Server with this name*

*Till now we are using our services on the basis of IP*

Now our all services are ready , now we will do communication between services but before doing that let’s understand what is our requirement

*I want request will come to user service with userId then it will get the user information from user Service then for rating it will get data from rating service then get the data from hotel service for which user is giving the rating then we will send back the details to the client*

###### Microservices Communication

Now first we will do How USER SERVICE Communicate to RATING SERVICE

Now here we will get rating list for the particular user from rating service for that we will call the rating list by userid api using http from the rating service inside user service for the particular userId method

Now we know one service call other by http client so here calling rating service Api we need http client for that we will use RestTemplate , so for getting the RestTemplate object first we will declare the bean of RestTemplate in main or configuration class then autowired in service layer then we can call the other service by using http client called RestTemplate

*we will call the api by using*

*restTemplate.getForObject(url, return type);*

*For here return type will be Rating[].class*

*Then we can set the rating by using setter and pass the value of return type*

*For example we are calling the api like "http://localhost:8080/ratings/users/"+user.getUserId();*

*Here there is a problem because we are calling the service api by hostname and port which is not recommended , At the later time we will remove hostname and port and will call by service name registered on eureka server*

*Calling Two Microservices Together*

*Till now we have got the ratings for the user , now we call the hotel service for getting the hotel information by hotel id for which the user has given the rating*

*Now inside userService we will create one more entity Hotel and inside existing Rating class we will add one more field Hotel*

*Removing Host and Port of Microservices*

*Now while calling the service from other service we will remove hostname and port and provide the serviceName by which service is registered on Eureka but we are calling service by using RestTemplate so need to inform the RestTemplate to use the service name instead of hostname and port so for that we will mention @LoadBalanced where we have declared the bean of RestTemplate i.e in the configuration class*

*now link will be*

[*http://serviceName/ratings/users/"+user.getUserId()*](http://serviceName/ratings/users/%22+user.getUserId())*;*

*API GATEWAY*

Currently client is calling different different services like hotelService, userService, RatingService for CRUD operation but let’s suppose our application have 50 urls then client will have to call all these urls if required but now using Api gateway client will only have to request to api gateway and inside api gateway we will add configuration that will internally call all these services on the client need basis

And api gateway is also a microservices it means we will create spring-boot project for api gateway

*Implementing API GATEWAY*

*We wil create spring boot project and we will register this service also on Eureka Server*

[*learn more*](https://cloud.spring.io/spring-cloud-gateway/reference/html/)*🡪 most important to visit before implementation*

*we will add below mentioned dependencies in pom.xml file*

**Cloud Bootstrap SPRING CLOUD**

**Gateway SPRING CLOUD ROUTING**

**Spring Reactive Web WEB**

**Eureka Discovery Client SPRING CLOUD DISCOVERY**

*Now in the application.yml or application. properties file we will add the configuration to register this service on eureka server and also add the configuration for the services so that when client request will come to api gateway then internally it will call the services on the basis of client request*

*Suppose we have three 3 services hotelService,userService and ratingService then we will add configurartion like that*

server:  
 port: 8084(can mention any port)  
  
spring:  
 application:  
 name: API-GATEWAY  
 cloud:  
 gateway:  
 routes:  
 - id: USER-SERVICE (service name)  
 uri: lb://USER-SERVICE  
 predicates:  
 - Path=/users/\*\*  
  
 - id: HOTEL-SERVICE  
 uri: lb://HOTEL-SERVICE  
 predicates:  
 - Path=/hotels/\*\*  
  
 - id: RATING-SERVICE  
 uri: lb://RATING-SERVICE  
 predicates:  
 - Path=/ratings/\*\*  
  
eureka:  
 instance:  
 prefer-ip-address: true  
 client:  
 fetch-registry: true  
 register-with-eureka: true  
 serviceUrl:  
 defaultZone: http://localhost:9000/eureka

*in the – id we mention the service name , in the uri we will write base url name , lb means load balancer because we are already using load balancer because we are using service registry , in predicates we will mention the path matcher*

*Now we will run the application and will call all the services api by api gateway*

*for example if we want to call the user service for getting all the users information then*

*localhost:8084/users/findall in the postman or by any client*

*here we client is using the api gateway port but internally it’s calling user service because we have added configuration in the api gateway application.yml file*

*API Gateway configuring multiple url of microservice*

*Suppoe in the hotel service we have different urls I mean some url that is not starting with /hotels then we can call that api by api gateway , let’s suppose that api url starts with /staff not with /hotels then we can configure for /staff also in the application.yml file*

- id: HOTEL-SERVICE  
 uri: lb://HOTEL-SERVICE  
 predicates:  
 - Path=/hotels/\*\*,/staffs/\*\*

*Config Server in Microservices*

*Till now whatever services we have created in that service application.yml file we have added the configuration but the scope of configuration is local but if we want to externalize that configuration then we we can use config server that provides user the client-server architecture with the help of this we can externalize our service configurations on the distributed cloud based system and get that configuration in our services using this config client server, for example here we are using github as an external location , another usecase can be when our services have common configuration then we can externalize that and with the help of client-server we can use that configuration in our services*

*We will create github repository there we can write our configuration*

*Implementing Config Server in Microservices*

*config server is a microservices so we will create the spring boot project for the config server also*

*we will register this service on the Eureka server*

*Now in the pom.xml file we will add the below mentioned dependencies*

**Eureka Discovery Client SPRING CLOUD DISCOVERY**

**Config Server SPRING CLOUD CONFIG**

*Now we will go to github and create the repository there we will create application.yml file and write the configuration that we want*

*Now to make this service as config server we will go to main class or configuration class and mention @EnableConfigServer*

*Now inside this service application.yml file we will write the configuration*

*now we want to externalize the common configuration i.e here in this case eureka client configuration is same for all the services so we will got to github and inside application.yml file we will store the configuration*

*and in the config server application.yml file we will add the configuration*

server:  
 port: 8082  
spring:  
 application:  
 name: CONFIG-SERVER  
 cloud:  
 config:  
 server:  
 git:  
 uri: (repository link here)  
 clone-on-start: true  
  
eureka:  
 instance:  
 prefer-ip-address: true  
 client:  
 fetch-registry: true  
 register-with-eureka: true  
 serviceUrl:  
 defaultZone: http://localhost:9000/eureka

*Now we will run the config server and will get the data from the github application.yml file on the clicking on the eureka server UI page CONFIG-SERVICE link or using postman or any browser*

*Reading config from github, Implementing config client microservices*

*Till now we are getting the data from the github using config server now we want to implement this configuration in our services , so we can do this using config client*

*Let’s suppose we want to add that eureka client configuration in our USER-SERVICE so we can go inside USER-SERVICE pom.xml file and add one dependency*

**Config Client SPRING CLOUD CONFIG**

*->It is the Client that connects to a Spring Cloud Config Server to fetch the application's configuration.*

*And in the USER-SERVICE application. properties or application.yml file we will add the below mentioned configuration and will delete the eureka client configuration from the application.yml file because now we are getting the configuration from the github using config client*

spring:

config:  
 import: optional:configserver:http://localhost: port of config server

*Fault Tolerance and Circuit Breaker*

*Suppose if our one service is calling other service for example if user service is calling rating service and rating service is down for let’s suppose for 10 minutes then user service can’t get data from rating service that’s is the fault*

*Now how to tolerate from the fault is using Circuit breaker*

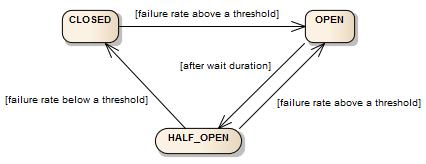
*Circuit breaker basically works on the basis of states and there are 3 important states of circuit breaker*

*Close,Open,Half Open*

*close state means there is no fault means services are communicating*

*and open means service is down , it means suppose if threshold is 5 so it means user service will try to connect with rating service for 5 times and get failure response from raring service it will not do further request to the rating service and state will change from close to open and suppose we have set the time frame for x seconds for example 5seconds then after 5 seconds it will go to half open state and user service will again do some request to rating service and if it gets high success rate and low failure request from the rating service then it will assume that rating service is now up and go to close state otherwise go to open state*

*we will manage the circuit breaker with the help of Resilience4j, previously we used to achieve with the help of Hystrix*

**

[*learn more*](https://resilience4j.readme.io/docs/circuitbreaker)

***Implementing Circuit Breaker using Resilience4J***

*Now for example user service is calling rating service then we will implements circuit breaker with the help of resilince4j library*

*we will go inside user service pom.xml file and will add below mentioned dependencies*

*<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>*

*With the help of actuator dependency we can monitor the health of our service*

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-aop</artifactId>  
</dependency>

*With the help of this aop dependency we can send the matrix or updates to the actuator*

*and*

*<!-- https://mvnrepository.com/artifact/io.github.resilience4j/resilience4j-spring-boot2 -->  
<dependency>  
 <groupId>io.github.resilience4j</groupId>  
 <artifactId>resilience4j-spring-boot2</artifactId>  
</dependency>*

*Now we will go inside user service controller and will search which Api method is calling other service , in my case findByUserId method is calling method ofrating service and hotel service so on the top of that method we will mention*

*@CircuitBreaker(name = "circuit breaker name(any name you can keep) for example ratingHotelBreaker",fallbackMethod = "any custom method name that you want to execute when any service fails for example ratingHotelFallback ")*

*so we will create ratingHotelFallback method inside controller and response type should be same as controller api method that is calling service i.e findByUserId and parameter should be same and one extra parameter will be there i.e Exception*

*Now this fallback method will run if our any calling service is down*

*Now we go inside application.yml file and add the configurations of Resilience4j to set the threshold limit, duration timer etc*

management:  
 health:  
 circuitbreakers:  
 enabled: true  
 endpoints:  
 web:  
 exposure:  
 include: health  
  
 endpoint:  
 health:  
 show-details: always  
  
  
  
resilience4j:  
 circuitbreaker:  
 instances:  
 ratingHotelBreaker(circuit breaker name that we have mentioned):  
 registerHealthIndicator: true  
 eventConsumerBufferSize: 10  
 failureRateThreshold: 50  
 minimumNumberOfCalls: 5  
 automaticTransitionFromOpenToHalfOpenEnabled: true  
 waitDurationInOpenState: 6s  
 permittedNumberOfCallsInHalfOpenState: 3  
 slidingWindowSize: 10  
 slidingWindowType: COUNT\_BASED

*and now we will run the user service and if our both the rating service and hotel service are up and calling the api method findByUserId method then it will give the result and if any service is down then fallback method will get executed and return the response that our fallback method is returning*

*now if we hit* [*http://192.168.1.5:userService port*](http://192.168.1.5:userService%20port) *number /actuator/health*

*Then we can see how our circuit breaker is working and calling the services , threshold states etc that we have discussed in the starting of circuit breaker concept*

*Logger*

*if we do system.out.* *println(), this is not preferable way because print method is the input output method and write on the character stream and we know reading from the stream and writing to the stream is very very slow process So it will reduce our application performance so we will use Logging framework so we can easily move from Log in development and in production environment and in dev more logs will be there and in prod less logs will be there*

*There are lots of logging framework , so in the logging there are multiple logging levels like WARN,DEBUG,INFO,ERROR*

*Let us say debug logging level has priority 1, Info has 2 and ERROR 3*

*And you have set up your application for debug level so then whatever you're logging in with debug will be log whatever you're logging within Info will be log because this is higher level, and Error will also be logged*

*Now at the later time your application, you have configured for the Info level*

*Now, whatever you're logging with debug level will not get logged*

*What you're logging with Info level will be logged and what you're logging with error level will be logged*

*Similarly, at a later time let’s suppose from info you have changed the level to error level Now what you log with debug will not be logged , for info also will not be logged and only for error will be logged*

*in spring boot by default the logging level is configured for Info*

*Now we go to any class and write private static final Logger log= LoggerFactory.getLogger(ClassName.class);*

*it is of slf4j*

*if we want to change the configuration then inside application.properties file we can mention the level by*

*and if want to set to the DEBUG then*

*logging.level.applicationName=DEBUG*

*similarly for hibernate*

*logging.level.org.hibernate=DEBUG*

*by default is Info*