**Date**

**Eureka server 9 oct 1:20**

**Admin server 9 oct 1:35**

**How to stop firewall ?**

Control panel-> system and security-> window defender firewall-> turn off and click OK at bottom

**For working with microservice project 1)we first need to get idea about about yml file.2) Also we need to know how to start Eureka Service Registry server,Admin server, and zipkin server. 3)Then we need know about feign client to establish communication between RestAPIs**

**YML file**

Create as usual web n data jpa project, you automatically get application.properties..in this file do database configuration like below

**application.properties file**

**spring.application.name=yml\_WEB\_JPA**

**spring.datasource.url=jdbc:h2:mem:dhirajdb**

**spring.datasource.username=dhiraj**

**spring.datasource.password=12345**

**spring.datasource.driver=org.h2.Driver**

**Now lets convert application.properities to yml file..**

**rt click this file-> convert to yml.**

NOTE:-(1) spring key has two childs ‘application ‘and ‘datasource ‘. application key has one child ‘ name’ and datasource key has 3 childs ‘username,password,driver’.

(2)In application.yml file One key ‘sourses’ can have 3 values but in application.properties one key can have only one value

(3) don’t forget to delete configuration in application.properties after creating .yml file

otherwise firstly application.properties configuration is checked n then application.yml configuration is checked

(4)In application.yml file we can type the content the way we type in application.properties, ctrl+space will autoAdjust the contents

(5) all parents should be in one line. There should be single space after colon : otherwise gets error

**PROJECT-1)--------Microservice\_serviceRegistry project**

**1)create spring starter project**

**2) add dependency**

**Type eureka**

**<dependency>**

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

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

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-devtools</artifactId>**

**<scope>runtime</scope>**

**<optional>true</optional>**

**</dependency>**

**3) add annotation in start class**

**@SpringBootApplication**

**@EnableEurekaServer**

**public class Microservice1ServiceRegistryApplication {**

**public static void main(String[] args) {**

**SpringApplication.*run*(Microservice1ServiceRegistryApplication.class, args);**

**}**

**4)Now if you run the project , the eureka server will run on port 8080…**

**You can check it in browser. type url localhost:8080 it opens eureka dashboard.you wont see any client instance registered.**

**On port 8080 clients instances not get autoRegistered…**

**We can change port in application.yml file to 8761 to get clients autoRegistered by below code---**

**server:**

**port: 8761**

**You can check it in browser. type url localhost:8761 it opens eureka dashboard.you will see client(restAPi projects) instance registered automatically. Also if you want to stop autoRegistration on port 8761 … we can set register-with-eureka property to false----**

**eureka:**

**client:**

**register-with-eureka: false**

**5)application.yml**

**spring:**

**application:**

**name: microservice1\_serviceRegistry**

**server:**

**port: 8761**

**eureka:**

**client:**

**register-with-eureka: false**

**################################################################################################################################**

**PROJECT-2)--------Microservice2\_AdminServer project**

**1)create spring starter project**

**2) add dependency**

**Type admin**

**<dependency>**

**<groupId>de.codecentric</groupId>**

**<artifactId>spring-boot-admin-starter-server</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-devtools</artifactId>**

**<scope>runtime</scope>**

**<optional>true</optional>**

**</dependency>**

**3) add annotation in start class**

**@SpringBootApplication**

**@EnableAdminServer**

**public class Microservice2AdminServerApplication {**

**public static void main(String[] args) {**

**SpringApplication.*run*(Microservice2AdminServerApplication.class, args);**

**}**

**4) write port in yml file to 1111 .see below code---**

**server:**

**port: 1111**

**Now if you run the project , the admin server will run on port 1111…**

**You can check it in browser. type url localhost:1111 it opens admin server dashboard.**

**5)application.yml**

**spring:**

**application:**

**name: Microservice2\_AdminServer**

**server:**

**port: 1111**

**################################################################################################################################ PROJECT-3)--------How to start ZIPKIN server?**

**-type in browser zipkin jar download**

**-follow zipkin.io link. It will open zipkin dashboard**

**-scroll down you will see ‘JAVA’.. click on latest release.. it will download jar file zipkin-server-3.4.2-exec.jar copy this file**

**-Now create new foler “jarFiles” on desktop. Paste the file into this folder.**

**-use cmd on this folder “jarFiles” ….**

**-run jar file by below command**

**java -jar zipkin-server-3.4.2-exec.jar**

**Zipkin server will start on port 9411 automatically**

**-type url localhost:9411 it opens zipkin dashboard**

**Application ports**

**Zipkin server 9411**

**AdminServer 1111**

**Eureka 8761**

**Greet API 9091**

**WelCome API 8081**

**MICROSERVICE PROJECT 4) :- RESTAPIproject microservice\_GreetAPI COMMUNICATING WITH RESTAPI project microservice\_WelcomeAPI USING FEIGN CLIENT**

STEP0)CREATING SERVER PROJECTS

create two spring boot projects 1)eureka server service registry project , 2)admin server project and run these servers on their ports by run as -> spring boot app.. then we can see their dashboard in browser by url localhost:port…

also run zipkin server in cmd directly and see its dashboard in browser . No need to create zipkin server project in spring boot.

STEP1)CREATING REST API PROJECTS

we need create two as usual restApi projects and run them on their ports. So these two projects run as clients( APIs) and will be detected by above 3 servers in dashboards in browzers)

1. **microservice\_WelcomeAPI (port 8081)**

**2)** **microservice\_GreetAPI(9091)**

so we need add dependency spring-web-starter in both projects . we can add also spring-dev-tools dependency in pom.xml to autoRestart the projects on saving them.

Note:- we say these two restAPI projects act as clients to above 3 mentioned servers .After running these two restAPI projects we also say the restApi is running on some server on particular port

Step2) ADDING DEPENDENCY ,DOING SPECIFICATION,ADDING PORT

as we making projects using microservice, we need make these two restApi projects as clients of Eureka service Registry server, Admin server, zipkin server , Actuators.

---So We need add these 4 dependencies to make them clients of above servers in pom.xml

---and also we need to do specification of 4 servers in application.yml file. (note:- ‘zipkin server ‘ and ‘Eureka server with port 8761’ automatically detects client restApis, so no need to do their specification in yml file , it means we need do specifications of Admin server and actuators only in yml file). We do specifications of 4 servers in yml file so that they can detect clients(two restApis)

----We need also mention on which port these two restAPI running in yml file).

Step3) ADDING ONE MORE DEPENDENCY

as RESTAPIproject microservice\_GreetAPI COMMUNICATING WITH RESTAPI project microservice\_WelcomeAPI USING FEIGN CLIENT, we need add feign client dependency in microservice\_GreetAPI project.

Step4)ADDING ANNOTATION IN START CLASS

We could make these two restApi projects as clients of mentioned 4 servers by adding theirs dependency in pom.xml. But to make client of Eureka server , besides dependency we also need to write annotation @EnableDiscoveryClient in start class of both API.

Also in GreetApi in start class we need add @EnableFeignClients as GreetApi communicating with WelcomeApi using feignclient

We need add below code in microservice\_WelcomeAPI

**In START CLASS add below annoatations**

**@SpringBootApplication**

**@EnableDiscoveryClient**

**public class MicroserviceWelcomeApiApplication**

**{**

**public static void main(String[] args)**

**{**

**SpringApplication.*run*(MicroserviceWelcomeApiApplication.class, args);**

**}**

**}**

**WelcomeRestController.java**

**NOTE:- environment object just prints which port particular API is using it has nothing to do with LBR(load balancing distribution)**

**@RestController**

**public class WelcomeRestController**

**{**

**@Autowired**

**private Environment env;**

**@GetMapping("/welcome")**

**public String getWelcomeMsg()**

**{**

**String port = env.getProperty("server.port");**

**String msg ="welcome to ashokIt..!!(server port ="+port+"}";**

**return msg;**

**}**

**}**

We need add below code in microservice\_GreetAPI

**In START CLASS add below annoatations**

**@SpringBootApplication**

**@EnableDiscoveryClient**

**@EnableFeignClients**

**public class Microservice3GreetApiApplication**

**{**

**public static void main(String[] args)**

**{**

**SpringApplication.*run*(Microservice3GreetApiApplication.class, args);**

**}**

**}**

**GreetRestController.java**

**@RestController**

**public class GreetRestController**

**{**

**@Autowired**

**private WelcomeApiClient apiClient;**

**@GetMapping("/greet")**

**public String getGreetMsg()**

**{**

**String greetResponse = "Good Morning";**

**String welcomeResponse = apiClient.invokeWelcomeApi();**

**return greetResponse+ "'"+ welcomeResponse;**

**}**

**}**

**(5) WelcomeApiClient.java**

**@FeignClient(name = "microserviceWelcomeApi")**

**public interface WelcomeApiClient**

**{**

**@GetMapping("/welcome")**

**public String invokeWelcomeApi();**

**}**

**PROJECT-4-A)--------How to create rest API (client API) projectName:- microservice\_welcomeAPI**

**-create as usual spring starter web project**

**- add below dependency**

**(1) POM.XML**

**1) Type actuator**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-actuator</artifactId>**

**</dependency>**

**2)type web**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-web</artifactId>**

**</dependency>**

**3)type admin . this dependency makes API as client of ADMIN server**

**<dependency>**

**<groupId>de.codecentric</groupId>**

**<artifactId>spring-boot-admin-starter-client</artifactId>**

**</dependency>**

**4)automatically gets this dependency**

**<dependency>**

**<groupId>io.micrometer</groupId>**

**<artifactId>micrometer-tracing-bridge-brave</artifactId>**

**</dependency>**

**5)type zipkin**

**<dependency>**

**<groupId>io.zipkin.reporter2</groupId>**

**<artifactId>zipkin-reporter-brave</artifactId>**

**</dependency>**

**6)type eureka. this dependency makes API as client of EUREKA server**

**<dependency>**

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

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

**</dependency>**

**(2)application.yml**

**spring:**

**application:**

**name: microserviceWelcomeApi**

**boot:**

**admin:**

**client:**

**url: http://localhost:1111/**

**management:**

**endpoints:**

**web:**

**exposure:**

**include: '\*'**

**eureka:**

**client:**

**service-url:**

**defaultZone: http://localhost:8761/eureka**

**server:**

**port: 8081**

**NOTE:- 1)see in yml file we making the microservice\_welcomeAPI as client of ADMIN server and EUREKA server. So we wrote corresponding specification**

**no need to add eureka specification, bcoz we already running eureka on 8761 port.**

**Bcoz of 8761 port , all API which are eureka client gets automatically added to eureka server.But if you change eureka port to some 8054 then you need write above eureka specification for 8054 port. Admin server specification is mandatory. Actuators Endpoints specification is mandatory.**

**2) microservice\_welcomeAPI will run on port 8081. If we are going for LBR(load balance distributor) no need add 8081 port in yml.**

**We will add port through run as-> run configuration-> click on microservice\_welcomeAPI project-> click on arguments-> in VM arguments , type -Dserver.port=8081 -> apply -> finish**

**Add two more ports 8082, 8083 like this way.**

**You can check eurekaDashboard to see these ports.**

**If you hit localhost:8081/welcome url it will use 8081 port .**

**If you hit localhost:8082/welcome url it will use 8081 port .**

**If you made microservice\_GreetAPI project and /greet url to access /welcome url,**

**That time if u hit /greet it will hit /welcome ,welcomeAPI will use port 8081**

**if u again hit /greet it will hit /welcome ,welcomeAPI will use port 8082**

**so ONE API wont be running on only single port.**

**So WORKLOAD ON SINGLE SERVER PORT IS DISTRIBUTED AMONG DIFFERENT PORTS THIS IS LBR**

**(3) START CLASS add below annoatations**

**@SpringBootApplication**

**@EnableDiscoveryClient**

**public class MicroserviceWelcomeApiApplication**

**{**

**public static void main(String[] args)**

**{**

**SpringApplication.*run*(MicroserviceWelcomeApiApplication.class, args);**

**}**

**}**

**(4) WelcomeRestController.java**

**NOTE:- environment object just prints which port particular API is using it has nothing to do with LBR(load balancing distribution)**

**@RestController**

**public class WelcomeRestController**

**{**

**@Autowired**

**private Environment env;**

**@GetMapping("/welcome")**

**public String getWelcomeMsg()**

**{**

**String port = env.getProperty("server.port");**

**String msg ="welcome to ashokIt..!!(server port ="+port+"}";**

**return msg;**

**}**

**}**

**PROJECT-4-B)--------Develope rest project microservice\_GreetAPI as clientAPI to communicate with microservice\_WelcomeAPI using FeignClient code. microservice\_GreetAPI will run on port 9091.**

**If u hit localhost:9091/greet url it will internally hit localhost:8081/welcome , localhost:8082/welcome, localhost:8083/welcome**

1. **POM.XML**

**Pom.xml is as same as pom.xml of microservice\_WelcomeAPI.**

**Just add one more dependency**

**feignClient**

**<dependency>**

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

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

**</dependency>**

1. **Application.yml**

**spring:**

**application:**

**name: microserviceWelcomeApi**

**boot:**

**admin:**

**client:**

**url: http://localhost:1111/**

**management:**

**endpoints:**

**web:**

**exposure:**

**include: '\*'**

**eureka:**

**client:**

**service-url:**

**defaultZone: http://localhost:8761/eureka**

**server:**

**port: 9091**

**NOTE:- 1)see in yml file we making the microservice\_GreetAPI as client of ADMIN server and EUREKA server. So we wrote corresponding specification**

**no need to add eureka specification, bcoz we already running eureka on 8761 port.Bcoz of 8761 port , all API which are eureka client gets automatically added to eureka server.But if you change eureka port to some 8054 then you need write above eureka specification for 8054 port. Admin server specification is mandatory. Actuators Endpoints specification is mandatory.**

**In START CLASS add below annoatations**

**@SpringBootApplication**

**@EnableDiscoveryClient**

**@EnableFeignClients**

**public class Microservice3GreetApiApplication**

**{**

**public static void main(String[] args)**

**{**

**SpringApplication.*run*(Microservice3GreetApiApplication.class, args);**

**}**

**}**

**(4) GreetRestController.java**

**@RestController**

**public class GreetRestController**

**{**

**@Autowired**

**private WelcomeApiClient apiClient;**

**@GetMapping("/greet")**

**public String getGreetMsg()**

**{**

**String greetResponse = "Good Morning";**

**String welcomeResponse = apiClient.invokeWelcomeApi();**

**return greetResponse+ "'"+ welcomeResponse;**

**}**

**}**

**(5) WelcomeApiClient.java**

**@FeignClient(name = "microserviceWelcomeApi")**

**public interface WelcomeApiClient**

**{**

**@GetMapping("/welcome")**

**public String invokeWelcomeApi();**

**}**

**PROJECT 5 ) writing whole project4 using load Balancing Distribution(LBR)**

**In project 4 , We had made “ microservice\_GreetAPI project and /greet url” to access /welcome url.**

**At present, if we hit /greet it will hit /welcome ,welcomeAPI will use port 8081 as we mentioned port 8081 in yml file of welcomeAPI.**

**So now microservice\_welcomeAPI running on port 8081. If we are going for LBR(load balance distributor) NO to need add 8081 port in yml. So remove 8081 from yml.**

**We will add port through-🡪 rt click welcomeAPI project🡪 run as-> spring configuration-> click on microservice\_welcomeAPI project-> click on arguments-> in VM arguments , type -Dserver.port=8081 -> apply -> finish. Port 8081 is added.**

**Add two more ports 8082, 8083 like this way.**

**Now we can say welcomeAPI running on 3 servers with different ports .**

**3 URls with different ports ( means 3 instances ) are registered with Eureka server.**

**You can see these URLs(instances with different ports) on eurekaDashboard .**

**If you hit localhost:8081/welcome url , welcomeAPI will run on server with 8081 port .**

**If you hit localhost:8082/welcome url, welcomeAPI will run on server with 8082 port .**

**If you made “ microservice\_GreetAPI project and /greet url” to access /welcome url,**

**That time if u hit /greet it will hit /welcome ,then feignClient will check url(instances) on Eureka server, then welcomeAPI will RUN ON SERVER WITH port 8081**

**if u again hit /greet it will hit /welcome ,then feignClient will check url(instances) on Eureka server, then welcomeAPI will RUN ON SERVER WITH port 8082.**

**IF u again hit /greet it will hit /welcome ,but now welcomeAPI will RUN ON SERVER WITH port 8083.**

**SO SERVERS ARE USED IN ROUND-ROBIN FASHION**

**so ONE API wont be running on only single SERVER .**

**So IF API RUNNUNG ON SINGLE SERVER , WHOLE BURDEN WILL BE ON SINGLE SERVER.BCOZ OF LBR (LOAD BALANCER) WORKLOAD ON SINGLE SERVER IS DISTRIBUTED AMONG DIFFERENT SERVERS . THESE DIFFERENT SERVERS ARE USED IN ROUND-ROBIN FASHION.EACH SERVER HAS ITS OWN PORT.**

}

**Git**

Git is version control system tool which helps you to track changes in the code .

**What is Github?**

GitHub is website that allows developers to store and manage the code using Git

Git is available at github.com.

We create folders in github.com website which are called repository

**What is GITBASH?**

GitBash is command prompt to control working of GitHub from your computer

How to Proceed?

Step1) create account in github.com , we need to set username, email , password for GitHub. Learn how to create repository, how to upload files in repository, how to delete file or whole repository. These all process are explained here only in this word file.

Step2) download gitbash and vscode ( follow git tutorial on apna college shradha parka) . we need to set username, email for GitBash. For GitBash all commands and complete knowledge about Git (follow git tutorial on apna college shradha parka) –saved in watch later in youTube

Step 3) learn how to connect STS with Github

NOTE:- we need to set username, email, password for GitHub and need to set username, email GitBash account separately .

For GitBash commands and complete knowledge about Git (follow git tutorial on apna college shradha parka) –saved in watch later in youTube

For error “git repository cloning error pemission denied” , follow BiggiDroid channel) --saved in watch later in youTube—to rectify this error, (from gitBash) you need to unset username , password, old files and then reset them again

**Advantages of Git**

1. to track history
2. we can discard changes done from particular part of coding… it means we can revert the changes and retrieve the code till particular part of coding
3. collaborate code developed by different developers and which developer’s code to accept or discard
4. free and open source. Fast and Scalable

**Step1) GITHUB**

**HOW TO LOG IN Github ?**

Type in browser <https://github.com/>

Username dhirajdhone

userEmail [dhirajdhone442@gmail.com](mailto:dhirajdhone442@gmail.com)

password Stranger123@#$

Type <https://github.com/>

Click on NEW or + sign to create new Repository

Enter name of new repository, then click on Public, click on READMe, then click create Repository

Once you create any repository , you automatically gets one default file README.md, in this file we can write description about Repository… there is pencil icon – we can edit README.md file---, then click COMMIT CHANGES

Now on HOME page you can see name of all the repositories you created..

We can upload any number of files in the particular repository…

**HOW TO ENTER HOME PAGE?**

By clicking following url…

<https://github.com/dhirajdhone>

dhirajdhone is username

**HOW TO UPLOAD FILES IN A REPOSITORY?**

Go to Home page, then click on particular repository

Click on add files, click on add file , then click upload, then click COMMIT CHANGES..

You will see all all the uploaded files inside particular Repository

**HOW TO SHARE REPOSITORY URL TO SOMEONE?**

<https://github.com/dhirajdhone/myRepository>

dhirajdhone is username

myRepository is name of Repository

in order to share a repository url… share <https://github.com/dhirajdhone/myRepository>

in order to share all repository url… share <https://github.com/dhirajdhone>

**HOW TO DELETE FILES IN A REPOSITORY?**

Click on the file, it opens file, you will see ICON TO DELETE file

**HOW TO DELETE REPOSITORY?**

Click on the repository, you will SETTING WITH ITS ICON in top menu click on setting icon, you will see option TO DELETE repository in bottom, you have to eneter passoword

**Step2) GITBASH**

GitBash is command prompt to control working of gitHub

**Downloading GitBash (follow git tutorial on apna college shradha parka**

**15 min onward)**

<https://git-scm.com/downloads/win>

then click --- Click here to download for 64 bit

For GitBash commands and complete knowledge about Git (follow git tutorial on apna college shradha parka) –saved in watch later in youTube

For “cloning error pemission denied” , follow BiggiDroid channel) --saved in watch later in youTube

**Step3) connecting STS with Github**

we will cover this part later

**================================**

**What is Cloud Config Server**

**================================**

=> We are configuring our application config properties in application.properties or application.yml file

Ex: DB Props, SMTP props, Kafka Props, App Messages etc...

so that our application.properties or application.yml file will be packaged along with our application (it will be part of our app jar file)

its disadvantage is If we want to make any changes to properties then we have to re-package our application and we have to re-deploy our application.

Also Note: If any changes required in config properties then We have to repeat the complete project build & deployment which is time consuming process.

=> To avoid this problem, we have to seperate our project code and project confg properties files.

=> To externalize config properties from the application we can use Cloud Config Server.

=> Cloud Config Server is part of Spring Cloud Library.

Note: some part of yml file is presnt in application only and some part of yml file will be maintained in git hub repo and config server will load them by using gitHub url and then our application will access them by using url of configServer

**Project Config Server and Config Client**

We will make 1 config server application microservice\_configServer.java and 2 APIs microservice\_config\_client\_greet.java and microservice\_Config\_client\_welcome.java

**Step1)** in git Repository, upload greet.yml and welcome.yml

### their Git Repo url : <https://github.com/ashokitschool/configuration_properties>

Note: -1) in actual application.yml file in 2 APIs , the name of ‘spring.application.name ‘ must be as same ‘name of yml files uploaded in git repository’

Note-2)some configuration of 2 yml files is present in ‘yml files present in git repository’ and some configuration will be present in ‘actual application.yml files of 2 APIs’

Note3) 2 APIs have some configuration in 2 application.yml and for remaining configuration , they will contact configServer to retrive data(msg key data) from configserver by using url optional:configserver:http://localhost:8080

And for this data(msg key data) , configServer will contact gitRepository to retrive the data(msg key data) from gitRepository by using url

<https://github.com/ashokitschool/configuration_properties>

---yml file of greet API in git repoistory

msg: Good Morning.....

---yml file of welcome API in git repoistory

msg: Welcome to Ashok IT (software training institute), hyderabad.

---application.yml of greet API

spring:

application:

name: greet

config:

import: optional:configserver:http://localhost:8080

server:

port: 9090

**AND**

----application.yml of welcome API

spring:

application:

name: welcome

config:

import: optional:configserver:http://localhost:8080

server:

port: 9091

**step2)** create config server application microservice\_configServer.java

A) add below dependencies

- config server

-spring dev tools

B) Write @EnableConfigServer annotation at boot start class

C) pass url of ‘git repository of yml files’ in application.yml file of config server, so that config server can retrieve yml files from git Repository

spring:

cloud:

config:

server:

git:

uri: https://github.com/ashokitschool/configuration\_properties

clone-on-start: true

management:

security:

enabled: false

**Step3)** create 2 config client application (2 APIs ) microservice\_config\_client\_greet.java and microservice\_Config\_client\_welcome.java

A) add below dependencies

-spring web

- config client

-spring dev tools

B) no need to add any annotations in start class

C) now inside application.yml of 2 APIs , we will write some configuration of 2 APIs and some configuration of 2 APIs will be present in config Server( as ConfigServer has url of git Repository of 2 yml files) pass ConfigServer url in application.yml file of 2 APIs so that 2 APIs can retrieve configuration from configServer

server:

port: 9090 (or 9091)

spring:

config:

import: optional:configserver:http://localhost:8080

application:

name: greet (or welcome)

step4) run server applications and 2 APIs applications .

microservice\_configServer.java will automatically run on 8080 port

microservice\_config\_client\_greet.java and microservice\_Config\_client\_welcome.java will run on 9090 and 9091 respectivelly

D) Create Rest Controller in both 2 APIs

@RestController

@RefreshScope

public class WelcomeRestController {

@Value("${msg}")

private String gitYmlmsg;

@GetMapping("/")

public String getWelcomeMsg() {

return gitYmlmsg;

}

}

@RestController

@RefreshScope

**public** **class** MsgRestController

{

@Value("${msg}")

**private** String gitYmlMsg;

@GetMapping("/data")

**public** String getMsg()

{

**return** gitYmlMsg;

}

}

Note:- @Value("${msg}") takes value from yml files in git repository

step5)

now hit url <http://localhost:9090/data>

we get output---

Good Morning.....

now hit url <http://localhost:9091/data>

we get output---

Welcome to Ashok IT (software training institute), hyderabad.

**Multiple yml files in git Repository for single API**

We can have multiple yml files in git Repository for single API microservice\_config\_client\_greet.java

Suppose greet.yml is actual yml file present in applications so we generally have sameNamed greet.yml file in git Repository too..

But now we can have multiple yml files in git repository with different profiles(environments) for single application

Greet-dev.yml

Greet-sit.yml

Greet-production.yml

Greet-pilot.yml

.dev,sit,production,pilot are called various profiles of greet.yml

**How to execute particular profile of yml file ?**

From actual greet.yml file we can specify particular profile so corresponding yml file from git Repository will execute…

spring:

application:

name: greet

config:

import: optional:configserver:http://localhost:8080

profiles:

active:

- dev

server:

port: 9090

compression:

excluded-user-agents:

Note:- for two different API , we don’t have one yml file in gitRepository

**Circuit Breaker (CB)**

**Circuit Breaker Project**

1. **pom.xml**
2. <dependency>
3. <groupId>org.springframework.boot</groupId>
4. <artifactId>spring-boot-starter-actuator</artifactId>
5. </dependency>
6. <dependency>
7. <groupId>org.springframework.boot</groupId>
8. <artifactId>spring-boot-starter-web</artifactId>
9. </dependency>
10. <dependency>
11. <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

1. </dependency>
2. <dependency>
3. <groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot3</artifactId>

1. <version>2.0.2</version>
2. </dependency>
4. <dependency>
5. <groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

1. <scope>runtime</scope>
2. <optional>true</optional>
3. </dependency>

**2)Application.yml**

spring:

application:

name: Circuit\_Breaker\_Resilence4j

management:

endpoints.web.exposure.include: '\*'

endpoint.health.show-details: always

health.circuitbreakers.enabled: true

resilience4j.circuitbreaker:

configs:

default:

registerHealthIndicator: true

slidingWindowSize: 6

minimumNumberOfCalls: 5

permittedNumberOfCallsInHalfOpenState: 4

automaticTransitionFromOpenToHalfOpenEnabled: true

waitDurationInOpenState: 10s

failureRateThreshold: 50

eventConsumerBufferSize: 10

**3) MyActivityRestController.java**

@RestController

**public** **class** MyActivityRestController

{

**int** i;

**int** m=1;

**int** f=1;

@GetMapping("/")

@CircuitBreaker(name="boredApi" ,fallbackMethod = "doWork")

**public** String invokeBoredApi()

{

System.***out***.println(m+") main-Logic executed "+ LocalTime.*now*());

//String url = "https://www.boredapi.com/api/activity";

String url = "https://jsonplaceholder.typicode.com/todos/1";

//String url = "https://jsonplaceholder.typicode.com/todos";

RestTemplate rt = **new** RestTemplate();

ResponseEntity<Activity> forEntity = rt.getForEntity(url, Activity.**class**);

Activity body = forEntity.getBody();

String title = body.getTitle();

m++;

// i = 10/0;

**return** title;

}

**public** String doWork(Exception e)

{ System.***out***.println(f+") fallback-Logic executed "+ LocalTime.*now*());

f++;

**return** "Exception Occured in Main Logic, so fallBackLogic is executed ";

}

}

**4)Activity.java**

**public** **class** Activity

{

**private** String userId;

**private** Integer id;

**private** String title;

**private** String completed;

**public** Activity() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Activity(String userId, Integer id, String title, String completed) {

**super**();

**this**.userId = userId;

**this**.id = id;

**this**.title = title;

**this**.completed = completed;

}

**public** String getUserId() {

**return** userId;

}

**public** **void** setUserId(String userId) {

**this**.userId = userId;

}

**public** Integer getId() {

**return** id;

}

**public** **void** setId(Integer id) {

**this**.id = id;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

**public** String getCompleted() {

**return** completed;

}

**public** **void** setCompleted(String completed) {

**this**.completed = completed;

}

@Override

**public** String toString() {

**return** "Activity [userId=" + userId + ", id=" + id + ", title=" + title + ", completed=" + completed + "]";

}

Explanation:-

Note:- cycle of steps 1 to 4 will keep repeating

0)suppose you just run spring-bootAPI A .and you didn’t hit request

[localhost:8080/](http://localhost:8080/actuator/health) to other API B.

this time the initial state of circuit breaker is UNKNOWN and status is blank and it is demonstrated by url [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

1) after you started hitting requests, The state of ciruitBreaker remains CLOSE as long as you getting successful response.

Also The state of the CircuitBreaker remains CLOSE till {minimumNumberOfFaliedCalls-1}( i.e.5-1=4) . It means at 5th  falied request , circuitBreaker moves to OPEN-STATE.

After 10s, circuitBreaker automatically moves to HALF-OPEN-STATE

3 failed calls in HALF-OPEN-STATE takes circuitBreaker to OPEN-STATE

cycle of OPEN->HALFOPEN will keep repeating until CircuitBreaker goes to CLOSE state(means until it gets successful response)

2) minimumNumberOfCalls= 5 means minimumNumberOf faliedCalls = 5 . It takes circuitBreaker to OPEN-STATE. After 10s, it goes automatically to HALF-OPEN-STATE

3) waitDurationInOpenState: 10s means ciruitBreaker will remain in OPEN state for 10s

4) automaticTransitionFromOpenToHalfOpenEnabled: true means ciruitBreaker will automatically move to HALF-OPEN state after 10s

5) permittedNumberOfCallsInHalfOpenState: 3 means 3 faliled calls to api B in HALF-OPEN-STATE will take circuitBreaker to OPEN state otherwise if calls successful then CB goes to CLOSE state.

It means HALF-OPEN-STATE can take CB to CLOSE or OPEN STATE dependending on response

Note:- cycle of OPEN->HALFOPEN will keep repeating until CircuitBreaker goes to CLOSE state

WORKING ILLUSTRATION:-

Step -0)suppose you just run spring-boot-API having no exception,

Now to check health of api, hit url [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

You will get

circuitBreakers":{"status":"UNKNOWN"}

circuitBreakers":{"state": not printed

Step1)suppose you hit url [localhost:8080/](http://localhost:8080/actuator/health)

If Your API get successful Response from other API

you hit url [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

circuitBreakers":{"status":"UP"}

circuitBreakers":{"state": CLOSE}

if you keep hitting same url , you will get same successful response with both url. And the state remains CLOSE

but if API throws exception , it means response will fail. 5 failed calls will take circuitBreaker to OPEN state. circuitBreaker will remain in OPEN state for 10s and then automatically goes to HALF-OPEN-STATE.

Now lets hit 5 failed calls by throwing exception. Now throw exception in API code.

Now hit url [localhost:8080/](http://localhost:8080/actuator/health) for 5 times

1stto 4th failed calls- it will execute fallbackLogic

now check health by [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

circuitBreakers":{"status":"UP"}

circuitBreakers":{"state": CLOSE}

5t fail call:-

it will execute fallbackLogic

now check health by [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

circuitBreakers":{"status":"UP"}

circuitBreakers":{"state": OPEN}

now circuitBreaker will remain in OPEN-STATE for 10s. lets wait 10s. after 10s circuitBreaker will move to HALF-OPEN-STATE. it can be demonstrated by url [localhost:8080/actuator/health](http://localhost:8080/actuator/health)

permittedNumberOfCallsInHalfOpenState= 3 means 3 faliled calls to api B in HALF-OPEN-STATE will take circuitBreaker to OPEN state otherwise if calls successful then CircuitBreaker goes to CLOSE state.

It means HALF-OPEN-STATE can take CircuitBreaker to CLOSE or OPEN STATE dependending on response

Note:- cycle of OPEN->HALFOPEN will keep repeating until CircuitBreaker goes to CLOSE state(means if response successful)