**Configuration Spring Boot Project:**

1.First create spring boot project using the spring.io or as in STS/Eclipse using spring stater project

2.Add the all the dependencies in the pom.xml

pom.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns=*"http://maven.apache.org/POM/4.0.0"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.5.2</version>

<relativePath />

<!-- lookup parent from repository -->

</parent>

<groupId>com.infy.employee.finance</groupId>

<artifactId>Infy-EmployeeFinance</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Infy-EmployeeFinance</name>

<description>Demo project for the microservice employee finance details</description>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-boot-starter-log4j2</artifactId>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

<exclusions>

<exclusion>

<groupId>org.junit.vintage</groupId>

<artifactId>junit-vintage-engine</artifactId>

</exclusion>

</exclusions>

</dependency>

<!-- Swagger -->

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger2 -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger2</artifactId>

<version>2.9.2</version>

</dependency>

<!-- https://mvnrepository.com/artifact/io.springfox/springfox-swagger-ui -->

<dependency>

<groupId>io.springfox</groupId>

<artifactId>springfox-swagger-ui</artifactId>

<version>2.9.2</version>

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Note: Simply copy paste and enter the only change is <groupId>org.springframework.boot</groupId> -> this is for the package name

<artifactId>spring-boot-starter-parent</artifactId> -> project name

<version>2.5.2</version> -> version of the project

3.Add application.properties file

spring.application.name=EmployeeAndTrainingManagementSystemApplication

server.port=8080

# Database properties

spring.datasource.url=jdbc:mysql://localhost:3306/infytel\_employee

spring.datasource.username=root

spring.datasource.password=7775

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

server.error.whitelabel.enabled=false

logging.level.org.hibernate.type=info

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

spring.output.ansi.enabled=always

#For actuator

#management.security.enabled = false

#management.port = 8765

General.EXCEPTION\_MESSAGE= Internal error please check the code

#----------------------------API/Controller Messages Environment ----------------------------------------

API.EMPLOYEE\_ADDED\_SUCCESSFULLY = Employee added successfully

API.EMPLOYEE\_UPDATED\_SUCCESSFULLY=Employee updated successfully

API.EMPLOYEE\_DELETED\_SUCCESSFULLY=Employee deleted successfully

4. Logging with the log4j

Lo4j.properties file

# Define the root logger with appender file

log4j.rootLogger = DEBUG, FILE, CONSOLE

# Define the file appender

log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender

log4j.appender.FILE=org.apache.log4j.FileAppender

log4j.appender.FILE.File=techTv.log

# Define the layout for file appender

#%5p - Priority of the logging event

#%t - Name of the thread that initiated the logging event

#%F- File name where the logging issue was requested

#%L - line number that caused the logging message to be generated

log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout

log4j.appender.CONSOLE.layout.conversionPattern=%5p [%F:%L] - %m%n

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%5p [%F:%L] - %m%n

name=LoggingFile

rootLogger.level=info

rootLogger.appenderRef.file.ref=LoggerAppender

appender.file.type=File

appender.file.name=LoggerAppender

appender.file.fileName=log/ErrorLog.log

appender.file.layout.type=PatternLayout

appender.file.layout.pattern=%d{dd-MMM-yyyy HH:mm:ss} %level - %m%n

# Console Appender

appender.console.name=ConsoleAppender

appender.console.type=Console

appender.console.layout.type=PatternLayout

appender.console.layout.pattern=%m%n

#Declaring logger for business logic

logger.infyacademy.name=com.infy.infyinterns.utility

logger.infyacademy.level=DEBUG

logger.infyacademy.appenderRef.file.ref=LoggerAppender

logger.infyacademy.additivity=false

logger.tester.name=com.infy.infyinterns

logger.tester.level=INFO

logger.tester.appenderRef.file.ref=ConsoleAppender

logger.tester.additivity=false

#name=LoggingFile

#rootLogger.level=info

#rootLogger.appenderRef.file.ref=LoggerAppender

#

#appender.file.type=File

#appender.file.name=LoggerAppender

#appender.file.fileName=log/ErrorLog.log

#appender.file.layout.type=PatternLayout

#appender.file.layout.pattern=%d{dd-MMM-yyyy HH:mm:ss} %level - %m%n

#

## Console Appender

#appender.console.name=ConsoleAppender

#appender.console.type=Console

#appender.console.layout.type=PatternLayout

#appender.console.layout.pattern=%m%n

#logging.pattern.console=%d{dd-MM-yyyy HH:mm:ss.SSS} [${spring.application.name}] [%thread] %-5relative %-5level %logger{35} - %msg%n

#

#

##Declaring logger for business logic

#logger.infyacademy.name=com.infy.infyinterns.utility

#logger.infyacademy.level=DEBUG

#logger.infyacademy.appenderRef.file.ref=LoggerAppender

#logger.infyacademy.additivity=false

#

#logger.tester.name=com.infy.infyinterns

#logger.tester.level=INFO

#logger.tester.appenderRef.file.ref=ConsoleAppender

#logger.tester.additivity=false

5. Validation for input DTO classes

**public** **class** EmployeeDTO {

@NotNull(message= "{employee.id}")

**private** Integer employeeId;

**private** String employeeName;

@NotNull(message= "{phoneNo.id}")

//@Pattern(regexp="[0-9]{10}",message="Phone Number Should be 10 digit number")

**private** Long phoneNo;

@NotNull(message= "{finance.id}")

**private** Integer financeId;

@NotNull(message= "{course.id}")

//@Pattern(regexp="[A-Z]{1}",message="Course Id must single capital letter from A-Z")

**private** Character courseId;

@NotNull(message= "{project.id}")

**private** Integer projectId;

}

Create the validation.properties

employee.id=Employee ID can not be null!.!Please Enter the employee ID....

phoneNo.id= phone number can not be empty!

project.id = Project id can not b empty!

course.id= course id can not ne empty!

finance.id= Finance ID can not be empty

Point to remember: Always the validation messages it takes the input from the separate file called as the validation messages.

we can not get the validation messages values if we created into the application properties

and how to know the spring boot project about validation.properties file as below:

@PropertySource("classpath:validationMessages.properties")

@SpringBootApplication

**public** **class** DemoApplication {

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

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

}

}

6. Creating the main source code package

Creating the packages like as below:

* com.infy.employee -> main application
* Demoapplication.class
* com.infy.employee.controller -> In this package all the controller ApI classes
* EmployeeController/EmployeeAPI
* ProjectController/projectAPI
* com.infy.employee.dto -> data tranfer object use to tranfer the data from the one layer to the another layer
* EmployeeDTO
* ProjectDTO
* com.infy.employee.entity -> this package is used for to interact with DB so this is called as the entity classes and we annotate it to match with the db
* Employee
* Project
* com.infy.employee.exception -> creating the user defines exception classes
* EmployeeException
* ProjectException
* com.infy.employee.repository -> this layer/ package tis used to interact with data Hibernate Query Lang and persisitenace query lang is used in this layer to fetech the data from the DB in this layer we use the interface which extending to another built in interface JparRepository<Entity,PrimaryKey\_DataType
* EmployeeRepository -> Interface
* ProjectRepository ->Interface
* com.infy.employee.service -> This layer contains all oue business logic to fetech the data from the database also do the some validation before inserting the data to the database
* EmployeeService->interface
* EmployeeServiceImpl ->class Business logic here
* ProjectService->interface
* ProjectServiceImpl->class business logic here
* com.infy.employee.utility -> this layer consist of the AOP -Aspect Oriented Programming (Three cross cutting concern = Transcation ,Logging,Seurity)
* ErrorInfo
* ExceptionHandlerAdvice
* AspectLogging

Problems faced during the Training:

Facing problem with initialization of the logger object and with their package name ?

**private** **static** **final** Logger logger = LogManager.getLogger(InfyAllocationApplication.**class**);

from package ->

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

Calling one Microservice to another Microservice:

Step-1: Declaring the RestTemplate-Basically we need to make our application lossely coupled so for getting the data from the one microservice to another microservice through RestTemplate by using REST url request

RestTemplate

Microservice-1-------------------------------🡪Microservice-2

RestURL

Inside the main SpringBootApplication:

@Bean

**public** RestTemplate getRestTemplate()

{

**return** **new** RestTemplate();

}

Method of the RestTemplate:

1. getForObject( URL, Class\_Name.class)

List<Employee> employee = (List<Employee>) restTemplate.

getForObject("http://localhost:8765/employee/CSE/0", Employee.**class**);

1. exchange(baseUrl,HttpMethod.***GET***,requestEntity,Employee.**class**);

ResponseEntity<Employee> reponseEntity= restTemplate.

exchange(baseUrl,HttpMethod.***GET***,requestEntity,Employee.**class**);

1. getForObject( URL, Class\_Name.class)

@GetMapping( value="/getEmployeeAndManager/{employeeId}")

**public** Employee getEmployeeAndManager(@PathVariable Integer employeeId) **throws** ManagerException

{

String baseUrl="http://localhost:8071/employee/getEmployee/"+employeeId;

//by using the method getForObject(url,class\_name.class)

Employee employee = restTemplate.getForObject(baseUrl, Employee.**class**);

**return** employee;

}

1. exchange(baseUrl,HttpMethod.***GET***,requestEntity,Employee.**class**);

@GetMapping( value="/getEmployeeAndManager" )

**public** Employee getEmployeeAndManager() **throws** ManagerException

{

String baseUrl="http://localhost:8765/employee/CSE/0";

//List<Employee> employee = (List<Employee>) restTemplate.getForObject("http://localhost:8765/employee/CSE/0", Employee.**class**);

HttpHeaders headers= **new** HttpHeaders();

headers.setContentType(MediaType.***APPLICATION\_JSON***);

HttpEntity<Object> requestEntity = **new** HttpEntity<>(headers);

ResponseEntity<Employee> reponseEntity = restTemplate.exchange(baseUrl,HttpMethod.***GET***, requestEntity,Employee.**class**);

**return** reponseEntity.getBody();

}

Note:For more details refer infyallocationProject present in the git

Complete Exchange method declaration:

**package** com.infy.infyallocation;

**import** java.util.List;

**import** org.apache.logging.log4j.LogManager;

**import** org.apache.logging.log4j.Logger;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.context.annotation.PropertySource;

**import** org.springframework.http.HttpEntity;

**import** org.springframework.http.HttpHeaders;

**import** org.springframework.http.HttpMethod;

**import** org.springframework.http.HttpStatus;

**import** org.springframework.http.MediaType;

**import** org.springframework.http.ResponseEntity;

**import** org.springframework.web.client.RestTemplate;

**import** com.infy.infyallocation.restclass.Employee;

**import** springfox.documentation.swagger2.annotations.EnableSwagger2;

@EnableSwagger2

@PropertySource("classpath:validationMessage.properties")

@SpringBootApplication

**public** **class** InfyAllocationApplication {

**static** RestTemplate *restTemplate*= **new** RestTemplate();

**static** String *baseUrl*="http://localhost:8765/employee";

**private** **static** **final** Logger ***logger*** = LogManager.*getLogger*(InfyAllocationApplication.**class**);

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

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

// exchangeMethodRestTemplate();

}

@Bean

**public** RestTemplate getRestTemplate()

{

**return** **new** RestTemplate();

}

**public** **static** **void** exchangeMethodRestTemplate()

{

// List<Employee> employee = (List<Employee>) restTemplate.getForObject("http://localhost:8765/employee/CSE/0", Employee.class);

HttpHeaders headers= **new** HttpHeaders();

headers.setContentType(MediaType.***APPLICATION\_JSON***);

HttpEntity<Object> requestEntity = **new** HttpEntity<>(headers);

*getSingleEmployeeExchangeMethodRestTemplate*(requestEntity);

*getListEmployeeExchangeMethodRestTemplate*(requestEntity);

Employee employee = **new** Employee();

employee.setEmployeeId(127);

employee.setCourseId("CSE");

employee.setEmployeeName("Charls");

employee.setCourseName("CDC");

employee.setHoursSpent(162);

employee.setScore(85);

requestEntity= **new** HttpEntity<>(employee,headers);

*addEmployeeExchangeMethodRestTemplate*(requestEntity);

*updateEmployeeExchangeMethodRestTemplate*(requestEntity);

*deleteEmployeeExchangeMethodRestTemplate*(requestEntity);

}

//getSingleEmployee from another Employee RestAPI running on differnet port Number

**public** **static** **void** getSingleEmployeeExchangeMethodRestTemplate(HttpEntity<Object> requestEntity)

{

***logger***.info("\*\*\*\*\*\*\*\*\*\*\*\* GET Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

//In String format

ResponseEntity<String> reponseEntity = *restTemplate*.exchange(*baseUrl* + "/101", HttpMethod.***GET***, requestEntity, String.**class**);

HttpStatus statusCode = reponseEntity.getStatusCode();

***logger***.info("Sataus Code:"+statusCode);

String emp = reponseEntity.getBody();

***logger***.info("Response Body:"+emp);

HttpHeaders httpheader = reponseEntity.getHeaders();

***logger***.info("Response Header:"+httpheader);

//In Object format

ResponseEntity<Employee> reponseEmployee = *restTemplate*.exchange(*baseUrl* + "/101", HttpMethod.***GET***, requestEntity, Employee.**class**);

HttpStatus statusCodeEmp = reponseEmployee.getStatusCode();

***logger***.info("Sataus Code:"+statusCode);

Employee employee = reponseEmployee.getBody();

***logger***.info("Response Body:"+"--->"+"Employee Name:"+employee.getEmployeeName()+"Employee ID:"+employee.getEmployeeId());

HttpHeaders httpheaderEmp = reponseEmployee.getHeaders();

***logger***.info("Response Header:"+httpheaderEmp);

}

//getALLemployeeByCourseCode Employee RestAPI

**public** **static** **void** getListEmployeeExchangeMethodRestTemplate(HttpEntity<Object> requestEntity)

{

***logger***.info("\*\*\*\*\*\*\*\*\*\*\*\* LIST GET Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

//In String format

ResponseEntity<List> reponseEntity = *restTemplate*.exchange(*baseUrl* + "/CSE/0", HttpMethod.***GET***, requestEntity, List.**class**);

HttpStatus statusCode = reponseEntity.getStatusCode();

***logger***.info("Sataus Code:"+statusCode);

List emp = reponseEntity.getBody();

***logger***.info("Response Body:"+emp);

HttpHeaders httpheader = reponseEntity.getHeaders();

***logger***.info("Response Header:"+httpheader);

//In Object format

ResponseEntity<List> reponseEmployee = *restTemplate*.exchange(*baseUrl* + "/CSE/0", HttpMethod.***GET***, requestEntity, List.**class**);

HttpStatus statusCodeEmp = reponseEmployee.getStatusCode();

***logger***.info("Sataus Code:"+statusCode);

List employee = reponseEmployee.getBody();

***logger***.info("Response Body:"+"--->"+employee);

HttpHeaders httpheaderEmp = reponseEmployee.getHeaders();

***logger***.info("Response Header:"+httpheaderEmp);

}

//addEmployee from Employee RestAPI

**public** **static** **void** addEmployeeExchangeMethodRestTemplate(HttpEntity<Object> requestEntity)

{

***logger***.info("\*\*\*\*\*\*\*\*\*\*\*\* POST Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

ResponseEntity<String> reponseEntity = *restTemplate*.exchange(*baseUrl*, HttpMethod.***POST***,requestEntity,String.**class**);

HttpStatus statusCode=reponseEntity.getStatusCode();

***logger***.info("Status Code:"+statusCode);

String reponseBody =reponseEntity.getBody();

***logger***.info("Response Body:"+reponseBody);

HttpHeaders httpheaderEmp = reponseEntity.getHeaders();

***logger***.info("Response Header:"+httpheaderEmp);

}

**public** **static** **void** updateEmployeeExchangeMethodRestTemplate(HttpEntity<Object> requestEntity)

{

***logger***.info("\*\*\*\*\*\*\*\*\*\*\*\* PUT Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

ResponseEntity<String> reponseEntity = *restTemplate*.exchange(*baseUrl*+"/777/MVC", HttpMethod.***PUT***,requestEntity,String.**class**);

HttpStatus statusCode=reponseEntity.getStatusCode();

***logger***.info("Status Code:"+statusCode);

String reponseBody =reponseEntity.getBody();

***logger***.info("Response Body:"+reponseBody);

HttpHeaders httpheaderEmp = reponseEntity.getHeaders();

***logger***.info("Response Header:"+httpheaderEmp);

}

**public** **static** **void** deleteEmployeeExchangeMethodRestTemplate(HttpEntity<Object> requestEntity)

{

***logger***.info("\*\*\*\*\*\*\*\*\*\*\*\* DELETE Method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

ResponseEntity<String> reponseEntity = *restTemplate*.exchange(*baseUrl*+"/127", HttpMethod.***DELETE***,requestEntity,String.**class**);

HttpStatus statusCode=reponseEntity.getStatusCode();

***logger***.info("Status Code:"+statusCode);

String reponseBody =reponseEntity.getBody();

***logger***.info("Response Body:"+reponseBody);

HttpHeaders httpheaderEmp = reponseEntity.getHeaders();

***logger***.info("Response Header:"+httpheaderEmp);

}

}

Microservices:

We have created the 4 microservices as follows:

1) Employee-Training-and-Management-system ->8071

2) InfyAllocation->8072

3) Infy-Employee ->8073

4) Infy-Finance ->8074

http://localhost:8074/finance/getFinance/

<http://localhost:8071/employee/getEmployee/>101

"http://localhost:8072/manager/getEmployeeAndManager/"

Microservice-4

Infy-Finance

Microservices-2

Microservice-3

Microservices-1

ETMS----RESTurl--->InfyAllocation ----RESTurl---> Infy-Employee <------RESTurl---Infy-Finance

**For the configuration on the cloud server side:**

1)

Create a new spring boot stater project (InfyCloudConfigServer) with three dependency 1)spring-cloud-config-server, 2)spring-boot-starter-actuator 3)spring-boot-starter-test

Pom.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.6.1</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.infy.cloud.config</groupId>

<artifactId>InfyCloudConfigServer-1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>InfyCloudConfigServer-1</name>

<description>This project is for configuring the properties file on git or github on the cloud and this project act as the server connect to the all other microservices as client .this project route all the properties file present on the git or github to all microservices</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.0</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<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>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

2)

In the main application add the depencency as @EnableConfigServer

3)

Create the GitHub/git new Repository and create or upload the files on the github repository as public

Github URL: <https://github.com/shubham7775881631/InfyCloudConfigServer>

Note: if we create the repository inside the folder or with a different repository name it will get the invalid remote so

Best Practice: Create the new repository with same name of the spring cloud server project name and add all the properties immediate to repository

4)

application.properties:

server.port=1111

spring.cloud.config.server.git.uri=https://github.com/shubham7775881631/InfyCloudConfigServer

spring.cloud.config.server.git.cloneOnStart=true

management.security.enabled=false

QUE ) How to check the our cloud config server working or not ?

By hiting the url we will let you know it gives us the JSON file

<http://localhost:1111/InfyCloudConfigServer/default>

**Changes in the client side i.e all other microservices:**

1. Add the dependency in pom.xml which is act as the client

<dependency>

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

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

</dependency>

1. Add the bootstrap.properties in all the microservies with property url of the cloud config server

spring.cloud.config.uri=http://localhost:1111

OR

We can skip the bootstrap properties we can add url of the cloud config server inside the application.properties file

spring.config.import = optional:configServer:http://localhost:1111

1. The properties present on the client side is only port number and application name other properties file will be present on the cloud

Example of Local properties file

spring.application.name=Infy-Employee

server.port=8073

Remote application properties will have higher precedence than the local properties file

Remote application properties will load first then load local

**Cloud Configuration properties of the Microservices:**

**For Config the properties on the cloud i.e. on git or git-hub or bitbucket**

**Step1: Create the new spring boot project name as InfyCloudConfigServer which is acting as the Cloud Configuration Server**

**This InfyCloudConfigServer -----🡪 Act as server with dependency Cloud-Config in the pom.xml**

<dependency>

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

<artifactId>spring-cloud-config-server</artifactId>

</dependency>

**All the Microservices ----🡪 Act as the Client with spring-cloud-starter-config dependency in the pom.xml**

<dependency>

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

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

</dependency>

1)Inside this server we have to add this dependency mention above

spring-cloud-config-server

2)Add the properties application.properties

spring.application.name= **InfyCloudConfigServer**

server.port=1111

//For git

spring.cloud.config.server.git.uri=https://infygit.ad.infosys.com/spring-microservice/infytel.git

//For GitHub

spring.cloud.config.server.git.uri=https://github.com/shubham7775881631/SpringBootMicroservices/tree/main/DEMO\_2\_CloudConfigMS/InfyPropertiesFiles.git

Note: There are many properties present for initialInterval, multiplier, maxInterval, maxAttempts

3 ) Add the annotation @EnableConfigServer in the main class

@SpringBootApplication

@EnableConfigServer

**public** **class** InfyCloudConfigServer {

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

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

}

}

Step-2) Changes in the all other Microservices

1. Add the dependencies all microservice

<dependency>

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

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

</dependency>

1. Changes in the application.properties files

#only these properties present inside the

spring.application.name=MicroserviesProjectName

server.port=8400

1. Add bootstrap.properties -> This context will helps to load all the properties present on the cloud

spring.cloud.config.uri=http://localhost:1111

FlOW OF THE ENTIRE PROJECT NOW:

Run the all the project: InfyCloudConfigServer + all other microservices

When we start the project from the microservices bootstrap.properties will run the first

This properties or context will helps to load properties present on the cloud this

spring.cloud.config.uri=http://localhost:1111 -> call the server

and server will fetech the config details form github and get to the its client i.e all the microserivoces

spring.cloud.config.server.git.uri=https://github.com/shubham7775881631/SpringBootMicroservices/tree/main/DEMO\_2\_CloudConfigMS/InfyPropertiesFiles --> Fetch properties from the github and servers to its client i.e to respective microservice

which application.properties will run the first?

Remote application.properties or Local application.properties

-🡪 Remote application.properties > Local application.properties

Remote application.properties -> will run the first then local

But it load only once means if InfyCloudConfigServer if already run and then we make the changes into the Remote application.properties then it will not directly reflected to the all other ms so for that we need to restart the InfyCloudConfigServer then it will load the updated properties

Note InfyCloudConfigServer load the properties file only once

1)To run dynamically we have to add the annotation @RefreshScope bean

From dependency spring-boot-starter-actuator

2)In order to access the endpoints using HTTP we need to both enable and expose then by adding the below property in the relevant microservice

management.endpoints.web.exposure.include=\*

1. Send the POST request to the refresh endpoint service -> this refreshes the ms without restarting or redeploying it

In this case we would add the @Refreshscpe to on the Infy-Employee ms on its controller layer by adding actuator dependency and send a post request to the <http://localhost:8071/actuator/referesh> where our Infy-Employee ms is running

This above @RefreshScope add only when wee are using @Value if we are feteching values from the environment then there is no need to add this @ RefreshScope

The problem with <http://localhost:8071/actuator/referesh> endpoint is that every time we have to manually hit the url of the respective ms if we have 1000 ms then it too complex and not good pactice then we use

Spring-Cloud-Bus + RabbitMQ/ IBM MQ / ActiveMQ -🡪 This will trigger dynamically when there is refresh even needs

Note: In this project we are using .properties it will also support .yml also

If the InfyCloudConfigServer is fail then by default run in port:8888 to avoid this there is properties

spring.cloud.config.failFast=true

A config server is running on at local port 8888 with the below config

CustMS-dev.properties

Hello=world

What will be the output when we access the URL <http://localhost:8888/CustMS-dev.properties>

--🡪 Display the properties file content in JSON

**Load Balancing – Ribbon:**

**Scenario:**

**If the number of user on the application is suddenly increases so the more number of hits on the Infy-Employee ms (**[**http://localhost:8073/infytelemployee/getFullEmployeeProfile/101**](http://localhost:8073/infytelemployee/getFullEmployeeProfile/101) **)**

**As we can see from the diagram the Infy-Employee microservice hitting multiple times so the load on this ms is more as compare to the other ms so there are chance to failed the Infy-Employee ms**

**So, the solution of this problem is load balancing we run the Infy-Employee in multiple instance manage the load on the both ms if one of the port number is fail to load the it balance the and try to run on other port number**

**Client-server for Load Balancing:**

**Server side Clint Side**

**Ex. Infy-Employee ---🡪 Infy-Allocation-----🡪ETMS**

**Infy-Employee --🡪Infy-Finance**

**Two types Load Balancing:**

**Client Side Load Balancing -> Adding the Load balalncer onn client side i.e. on multiple instance of InfyAllocation and ETMS**

**Is natural way of the load balancing client will responsible for the to whom will send the request**

**Best Practice: Use Client Side Load Balancer than server side LB**

**Example:**

**Spring Cloud Load Balancer**

**Ribbon**

**NaNX**

**Varnish**

**HAProxy**

**Mod\_athena**

**Server Side Load Balancing: -> adding the Load balancer on server act as Infy-Employee ms**

**Example :**

**F5 BIG IP Load Balancer**

**CISCO system catalyst**

**Braacuda LB**

**CoytePoint LB**

**Step-1) Adding dependency where we have RestTemplate**

<dependency>

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

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

</dependency>

**Step-2)Add the properties in the application.properties**

spring.application.name=CustomerMS

server.port=8200

empribbon.ribbon.eureka.enabled=false

empribbon.ribbon.listOfServers=http://localhost:8301,http://localhost:8300

spring.jackson.default-property-inclusion: NON\_NULL

**Step-3)create the class InfyEmployeeRibbonLBConfig and annotate the RestTempalte with @LoadBalanced**

@Configuration

**public** **class** RestTemplateConfig {

@Bean @LoadBalanced

**public** RestTemplate restTemplate()

{

**return** **new** RestTemplate();

}

}

**Step-4)Annotate the Controller ms class of server LB (the class which has RestTemplate ) with @RibbonClient as below**

@RestController

@CrossOrigin

**@RibbonClient (name="empribbon",configuration=InfyEmployeeRibbonLBConfig.class)**

**public** **class** EmployeeController {

@RequestMapping(value ="/getFullEmployeeProfile/{employeeId}", method = RequestMethod.***GET***, produces = MediaType.***APPLICATION\_JSON\_VALUE***)

**public** ResponseEntity<RestEmployeeDTO> viewFullEmployeeProfile(@PathVariable Integer employeeId) **throws** EmployeeException {

String allocationMsUrl = "http://empribbon/manager/getEmployeeAndManager/"+employeeId;

//feteching the details from ms-Infyallocation

RestEmployeeDTO restEmployeeDTO = restTemplate.getForObject(allocationMsUrl, RestEmployeeDTO.**class**);

//feteching employee details from our class details f

EmployeeDTO employeeDTO = employeeService.viewEmployeeProfile(employeeId);

//fetevhing the employee finance detials from the ms-Infy-Finance

String financeMsUrl = "http://empribbon/finance/getFinance/"+employeeDTO.getFinanceId();

FinanceDTO financeDTO = restTemplate.getForObject(financeMsUrl,FinanceDTO.**class**);

restEmployeeDTO.setEmployee(employeeDTO);

restEmployeeDTO.setFinance(financeDTO);

**return** **new** ResponseEntity<>(restEmployeeDTO,HttpStatus.***OK***);

}

**}**

**Step5) creating the new class as methion in the above annotation -> This class is used when if after adding the load balalacing the service using the two port if one of the port fail and user hiting the down url then this class get down the url for some time again allow to hit the url**

**In this time it removes the load from that port again for specific time duration completed it again start hitiing if again still the it doewn the again it stop hiting the url get hitting for some duration ….. until the url hit is not completed**

**package com.infosys.config;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.context.annotation.Bean;**

**import com.netflix.client.config.IClientConfig;**

**import com.netflix.loadbalancer.\*;**

**public class** InfyEmployeeRibbonLBConfig **{**

**@Autowired**

**IClientConfig clientconfig;**

**@Bean**

**public IPing changePing(IClientConfig clientconfig){**

**return new PingUrl();**

**}**

**}**

**Check the url:**

[**http://localhost:8073/infytel-employee/getFullEmployeeProfile/101**](http://localhost:8073/infytel-employee/getFullEmployeeProfile/101)

**if it is working fine the it will run**

**How to check the maven the dependency tree**

**Command:**

mvn dependency:tree

**Let’s understand the pom.xml:**

spring-boot-starter-parent:

<parent>

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

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

<version>2.6.3</version>

<relativePath />

</parent>

**Uses of the spring-boot-starter-parant:**

1) **Default Configuration: provides default configuration for Maven plugins such as maven-failsafe-plugin, maven-jar-plugin, maven-surefire-plugin, maven-war-plugin.**

2)**Managing Dependencies:**

<dependencies>

<dependency>

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

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

</dependency>

<dependencies>

It helps in pull out the maven dependencies from the local/remote/cemtram maven repository without mentioning the version it pull the version according to the spring-boot-starter-parent version 2.6.3

3)**Dependencies Management Tag:** it helps to manage the dependency explicitly (Manage the version,type,scope)

Example-1:

<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>

In above the dependencyManagement for all the spring-cloud-depencencies that we are added inside the <dependencies></dependencies> that are mention explicitly according to the spring boot version

3)**Manage Properties:**

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.0</spring-cloud.version>

</properties>

we can manage the properties inside this tag for the all the dependencies

**To change the value of any property defined in the starter parent, we can re-declare it in our properties section.**

**Also we can override the properties**

4)spring-boot-maven-plugin:

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

Maven users can inherit from the spring-boot-starter-parent project to obtain sensible defaults. The parent project provides the following features:

* Java 1.8 as the default compiler level.
* UTF-8 source encoding.
* Compilation with -parameters.
* A dependency management section, inherited from the spring-boot-dependencies POM, that manages the versions of common dependencies. This dependency management lets you omit <version> tags for those dependencies when used in your own POM.
* An execution of the [repackage goal](https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/#goals-repackage) with a repackage execution id.
* Sensible [resource filtering](https://maven.apache.org/plugins/maven-resources-plugin/examples/filter.html).
* Sensible plugin configuration ([Git commit ID](https://github.com/ktoso/maven-git-commit-id-plugin), and [shade](https://maven.apache.org/plugins/maven-shade-plugin/)).
* Sensible resource filtering for application.properties and application.yml including profile-specific files (for example, application-dev.properties and application-dev.yml)

For more details: <https://docs.spring.io/spring-boot/docs/current/maven-plugin/reference/htmlsingle/>

If we are using the release as the version the spring boot starter project then we need to add this following dependencies

<repositories>

<repository>

<id>spring-milestones</id>

<name>Spring Milestones</name>

<url>https://repo.spring.io/milestone</url>

</repository>

</repositories>

In pom.xml the spring boot version for various project Table

Description automatically generated

<https://spring.io/projects/spring-cloud>

<parent>

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

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

<version>2.5.2</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version>

<spring-cloud.version>2020.0.3</spring-cloud.version>

</properties>

If spring-boot-starter-parent version is 2.4.x/2.5.x then the spring-cloud.version properties would be 2020.0.3

Similarly if If spring-boot-starter-parent version is 2.1.x then the spring-cloud.version properties would be Greenwich.RELEASE as ssown in the above image

<parent>

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

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

<version>2.1.2</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<java.version>1.8</java.version>

<spring-cloud.version> Greenwich.RELEASE </spring-cloud.version>

</properties>

<dependencies>

**Service Discovery And Registration:**

When we create the eureka server the with java version 11 the we got error of jaxpy because fro, java 9/10 onwards jaxpy is not supported . so for this we need to add the extra dependency t0 resolve this error

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-core</artifactId>

<version>2.3.0</version>

</dependency>

But in the new spring boot version no need to add

Eureka server comes with inbuild ribbon

Eureka Service Registration:

Step-1) Create a new Spring boot project and the following pom.xml one this need to take spring-boot-starter version(2.6.3) must be matched with spring clous version(2021.0.0) otherwise we get the error

Eureka Server pom.xml for new spring boot 2.6.3

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<!-- Spring Boot Starter Project With version(2.6.3) must be match with

respective to the spring cloud version(2021.0.0) -->

<parent>

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

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

<version>2.6.3</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<!-- Properties Management -->

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.0</spring-cloud.version>

</properties>

<!-- Dependency Management -->

<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>

<!--Maven Build Plugin -->

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

<!-- Project Name -->

<modelVersion>4.0.0</modelVersion>

<groupId>com.infy.eureka.server</groupId>

<artifactId>InfyEurekaServerWithRibbonLB</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>InfyEurekaServerWithRibbonLB</name>

<description>Demo project for Spring Boot</description>

<!-- Dependencies Management -->

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

</project>

Step-2) Add the following the properties

Try to run the eureka server on default in local if you are getting errors

register-with-eureka : Eureka comes with the cluster of eureka by default configuration so the current server act as client another eureka server as server so to as of now we are using the only one eureka server so the make it as false

it calls the only single instance of the eureka if this

register-with-eureka =false

if there are multiple/cluster of the eureka server present then this current eureka instance act as client other as server

if the this current server fails to up then automatically the request get forward to another eureka server unless and until current instance gets fail

so by default it is true

register-with-eureka =true

fetch-registry: this properties will fetch registry from the eureka server at startup time and will cache it. It will check the euraka server at regular interval of time (default to 30sec)

if there is any update in the service registry it will fetch the changes at his regular interval of the time it takes only changes part and update it other part already present in the cache.

eureka.client.fetch-registry=false

application.properties

spring.application.name=InfyEurekaServerWithRibbonLB

server.port=8761

eureka.client.fetch-registry=false

eureka.client.register-with-eureka=false

eureka.client.service-url.defaultZone=http://localhost:8761/eureka

if we are using the single instance of the eureak server then we need to use

eureka.client.fetch-registry=false

eureka.client.register-with-eureka=false

steps-3) add the @EnbaleEureka Server in the main application

**Eureka client Microservice :**

Step-1)

Pom.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>2.6.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.infy.employee.finance</groupId>

<artifactId>InfyDemo-1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>InfyDemo-1</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>11</java.version>

<spring-cloud.version>2021.0.0</spring-cloud.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<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>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Step-2)

Add the annotation @EnableEurekaClient in the main spring boot application

Step-3) Now we do not need the @RibbonClient in the controller remove it because now the the load is handled by the eureka so whatever details required for ant other microservice will take it from the eureka server

Remove the @Ribbon client

And where is the port put it Microservice name

Before:

String baseUrl="http://infymanager/employee/getEmployee/"+employeeId;

Application.properties

infymanager.ribbon.eureka.enabled=false

infymanager.ribbon.listOfServers=http://localhost:8071

spring.jackson.default-property-inclusion: NON\_NULL

After:

Remove the properties and in the url put the name localhost as application name:

String baseUrl="http://EmployeeAndTrainingManagementSystemApplication/employee/getEmployee/"+employeeId;

**Eureka Without Ribbon :**

we need the same configuration for eureka server once eureka server is up lets go for changes for the other microservice

Eureka Client without ribbon for the microservice:

Step1 )Add the dependency in pom.xml

<dependency>

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

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

Step-2)In the main spring boot application @EnableDiscoveryClient .

Remove the @EnableEurekaclient

Step-2) we are not using the by default ribbon so we do not need load balanced the rest template. Remove rest load Balanced templated and add normal RestTemplate

Step-3) In the controller class add the DiscoveryClient from

**import** org.springframework.cloud.client.discovery.DiscoveryClient;

@Autowired

**private** DiscoveryClient discoveryClient;

@GetMapping( value="/getEmployeeAndManager/{employeeId}")

**public** Employee getEmployeeAndManager(@PathVariable Integer employeeId) **throws** ManagerException

{

List<ServiceInstance> instances = discoveryClient.getInstances("EmployeeAndTrainingManagementSystemApplication");

ServiceInstance instance =instances.get(0);

URI url = instance.getUri();

String baseUrl=url+"/employee/getEmployee/"+employeeId;

//by using the method getForObject(url,class\_name.class)

Employee employee = **new** RestTemplate().getForObject(baseUrl, Employee.**class**);

ManagerDTO manager = managerService.fetechManagerDetailsByCourseId(employee.getCourseId());

employee.setManager(manager);

**return** employee;

}

Problems and Solution:

While importing the project if you are getting error first try below steps:

1. Right click on the project->Run as ->Maven build… ->Goal:clean install
2. If build is successful then we ready to install then problem will be other
3. Always try Maven Clean-> Maven Build -> Maven Force Update
4. If nothing working then problem will be other.