1. Enabling Discovery Server –Eureka
2. Create new spring boot project with Eureka server dependency. In POM.xml it will look like below

<dependency>

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

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

</dependency>

1. Go to your main java file and add @EnableEurekaServer annotation below @SpringBootApplication.
2. Default port in 8761. You can change if required. Also in application.properties file add below properties if this is only one discovery server in your project. This property will stop this server to register as client with itself/other servers and also will not try to fetch any registry entries from other servers.

eureka.client.registerWithEureka = false

eureka.client.fetchRegistry = false

1. All your other microservices should have below dependency added.

<dependency>

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

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

</dependency>

1. Add @EnableEurekaClient annotation below @SpringBootApplication in all microservices.
2. Enabling API Gateway in Microservices
3. Create new spring boot project with gateway dependency. In POM.xml you find entry as below.

<dependency>

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

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

</dependency>

1. Add application.yml file in resources folder. It should have entry like below:

spring:

cloud:

gateway:

routes:

- id: MovieService

uri: lb://MOVIECATALOGUESERVICE

predicates:

- Path=/catalogue/\*\*

- id: MovieInfoService

uri: lb://MOVIEINFOSERVICE

predicates:

- Path=/movie/\*\*

- id: UserRatingService

uri: lb://RATINGDATASERVICE

predicates:

- Path=/rating/\*\*

1. If you are getting domain name issues then add below property in application.properties file

eureka.instance.prefer-ip-address=true

1. Adding Hystrix to your project
2. Create new spring boot project with Hystrix dependency. In POM.xml file you will see as follows.

<dependency>

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

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

</dependency>

1. Add annotation in your main java class @EnableHystrix
2. Now in application.yml file add below settings:

spring:

cloud:

gateway:

filters:

- name: CircuitBreaker

args:

name: MOVIECATALOGUESERVICE

fallbackuri: forward:/movieCatalogueFallback

And

hystrix:

command:

fallbackcmd:

execution:

isolation:

thread:

timeoutInMilliseconds: 4000

1. Reading property file using @Value
2. You can have default value from property file for some fields in class. So you can annotate field with @value as below. Suppose you have application.properties file entry as

*my.propertyfile.value=Welcome to Spring Boot*

@Value("${my.propertyfile.value}")

private String someText;

1. If property is not defined in property file, spring container will fail to start up. If you want to avoid this failure you can have default in value annotation as below

@Value(“${my.propertyfile.value1: Welcome to Spring Boot1}”)

private String someText;

1. You can have comma separated multiple values in single property as can assign it to list in java using @value

*my.propertyfile.listOfMovies=Raaz,Krish,Darr*

@Value(“${my.propertyfile.listOfMovies}”)

private List<String> movies;

1. You can also have maps kind of key value pair in property file and read it and directly assign to Map type in java.

*my.propertyfile.mapOfmovies={Raaz: ’1’, Krish: ‘3’, Darr: ‘2’}*

@Value(“#{${my.propertyfile.mapOfmovies }}”)

private Map<String,String> moviesMap;

Note : #{ } is used to evaluate whatever is inside braces.

1. Reading configuration using annotation @ConfigurationProperties

Suppose you have bunch of properties which needs to be made available to entire application then we can map them to pojo and mark it as a bean.

In property file you have below properties

db.connectionUrl=http://localdb:8090/

db.username=admin

db.password=pass

db.hostname=devDB

You need to create POJO class with same field names (Note that case can be different) as in property class. At class level just add annotation @ConfigurationProperties("db") and mark this as bean using @Configuration

@Configuration

@ConfigurationProperties("db")

**public** **class** DBSetting {

**private** String connectionURL;

**private** String username;

**private** String password;

**private** String hostname;

//Getters and setters here

**}**

1. Spring Profiles
2. You can have set of properties which are specific to environment clubbed together is a profile. For this you need to create separate yml or properties file besides yours application. properties or yml file. Name of file should be like below.

application-<profilename>.properties or application-<profilename>.yml

e.g: application-dev.properties

1. Now in your application. properties or yml file you need to mention name of your profile like below.

e.g : spring.profiles.active=dev

1. Also only properties which are environment specific can be copied in new file. No need to replicate whole file. Spring will check property in profile related property file and if property not found then it will pick value from default profile i.e application.properties or yml file.
2. Reading properties from external system or service
3. Create a new spring boot project with config server dependency. In pom.xml file it will be as below

<dependency>

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

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

</dependency>

1. In your main class add annotation @EnableConfigServer
2. Add application.yml file and add below entry

server:

port: 8888

spring:

application:

name: MoviesConfigServer

cloud:

config:

server:

git:

default-label: main

uri: https://github.com/Vivek-Sawant1/SpringMicroservices/

clone-on-start: true

1. Once server boots up you can check properties using URL as below

http://<serverIP>:<serverPort>/<FileName>/<ProfileName>

e.g : <http://localhost:8888/application/default>

1. Configure Config Client
2. You need to add dependency in your spring boot application to make it as client

<dependency>

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

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

</dependency>

1. Create bootstrap.yml file and add below entry, as it’s loaded before application context is formed.

spring:

application:

name: MovieCatalogueService

cloud:

config:

uri: http://localhost:8888

enabled: true

1. Auto Refresh properties

For having any properties auto refreshed in your microservice, following steps needs to followed.

1. Add actuator dependency in POM.xml file

<dependency>

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

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

</dependency>

1. Add @RefreshScope annotation on main class file.
2. When you change any property in repo and you want latest version to be read, then hit POST request as below URL

http://localhost:8081/actuator/refresh