Bipin Kumar

Shael.dhn88@gmail.com

Spring Cloud Config

Spring cloud config

Using Spring cloud config, we can centralize all the config file related to different microservices. In this case what we can do we can keep all the common config file inside a GIT repo/ SVN repo/ DB following some naming convention.

e.g. Suppose we have S1, S2… S20 service we can may name the config file like S1-dev.yml, S1-prod.yml etc. Similarly, if we have common property we may keep those property in application.yml file.

Now when we starting our application we want to fetch all the configuration before initializing the beans. Now for this we can configure config server which will expose API to fetch all the configuration property before startup. So now suppose S1 is starting it will send rest call to config server like /S1/prod then it will fetch all the configuration based on these parameters and return to S1.

1. Add below dependency in your pom.xml file.

<dependency>

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

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

</dependency>

1. Add @EnableConfigServer annotation on main config class.
2. Add below property in yml file.

spring:

cloud:

config:

server:

# jdbc: for db

# sql:

git:

uri: https://github.com/shaelraj/ms-properties

server:

port: 4000

1. Now start server and postman. When you hit below URL you will get property related to it.

localhost:4000/springcloudconfigclient/default

{

"name": "springcloudconfigclient",

"profiles": [

"default"

],

"label": null,

"version": "7d2d10b0a012f0c0d508e3fe07e64b5c8cab0414",

"state": null,

"propertySources": [

{

"name": "https://github.com/shaelraj/ms-properties/springcloudconfigclient.yml",

"source": {

"test.message": "CCCCC",

"server.port": 2233

}

},

{

"name": "https://github.com/shaelraj/ms-properties/application.yml",

"source": {

"A": 1

}

}

]

}

How to use config server in microservice.?

1. Create microservice. And add below dependency in it.

<dependency>

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

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

</dependency>

1. Instead of application.yml file add bootstrap.yml file.

spring:

application:

name: springcloudconfigclient

cloud:

config:

uri: http://localhost:4000

fail-fast: true

Here we can see the uri is pointing to the cloud server and fail-fast is set to true because if it don’t get the properties file it should fail to start.

1. Now start the server and in server startup logs you can see the changes. You can see tomcat started on server which we have mentioned in property file.

Fetching config from server at : <http://localhost:4000>

Located environment: name=springcloudconfigclient, profiles=[default], label=null, version=7d2d10b0a012f0c0d508e3fe07e64b5c8cab0414, state=null

Located property source: [BootstrapPropertySource {name='bootstrapProperties-configClient'}, BootstrapPropertySource {name='bootstrapProperties-https://github.com/shaelraj/ms-properties/springcloudconfigclient.yml'}, BootstrapPropertySource {name='bootstrapProperties-https://github.com/shaelraj/ms-properties/application.yml'}]

No active profile set, falling back to default profiles: default

BeanFactory id=14736867-57e6-3745-9c25-269bb37581f2

Tomcat initialized with port(s): 2233 (http)

Now suppose we changed the file in github. But still it can’t refresh the property. Because by default all the beans are singleton. So, what we can do, we can add actuator dependency in our service which will expose /actuator/refresh url.

<dependency>

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

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

</dependency>

When above api called what it will do it will call context.refresh() which will refresh the configuration but don’t create the singleton bean again. After 2.x spring boot after adding actuator dependency it will not work we need to enable it using property.

management:

endpoints:

web:

exposure:

include: refresh,metrics,beans

Now restart the server, change the property and hit post request to below URL.

localhost:2233/actuator/refresh

But still you can’t see the changes get refresh because we have singleton bean and above changes only refresh property value. To reflect changes we need to annotate our bean with @RefreshScope then only it will refresh value.

Now suppose we have 10 instances of this service then to refresh each instance with new value we need to hit actuator url in every instance which is very difficult. So instead of this what we can do we can add a dependency called spring cloud bus which by default implement rabbitmq and its property in yml file. What it will do while startup it creates a mq with exchange name spring cloud bus. And its also create queues specific to each instance. So now whenever changes are done in git we need to call endpoint /actuator/bus-refresh.

<dependency>

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

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

</dependency>

And also include bus-refresh in actuator property.

Instead of calling bus-refresh what we can do we can use webhook concept of GIT. For this we need to add below dependency on spring-cloud-config-server project.

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

Define the property related to mq. And put /monitor url to github webhooks.