:- Config Server :-

10-8-18

In every Application (Microservice) properties/yml file contains setup related keys like

→ DataSource, ORM, Batch, Gmail, AOP, Security, Eureka, Cas Gateway --- in the form of key = value.

→ In case of multiple microservices, these are repeated for every project.

→ In this case, Config server concept is used, for:

→ Writting common properties only one time outside of all projects.

→ Easy to modify common properties (one place to modify properties file that effects all projects).

→ Even service instances not required to stop/re-start for effect for properties file modifications (Refresh scope).

:- Types of Config Server :-

Spring Cloud has provided setup and support for Configuration properties in two different ways. Those are

a. Native Configuration Client file

b. External Configuration Client file

a. Native Configuration file :-

In this case one properties file is created outside all microservices in a folder system (or drive).

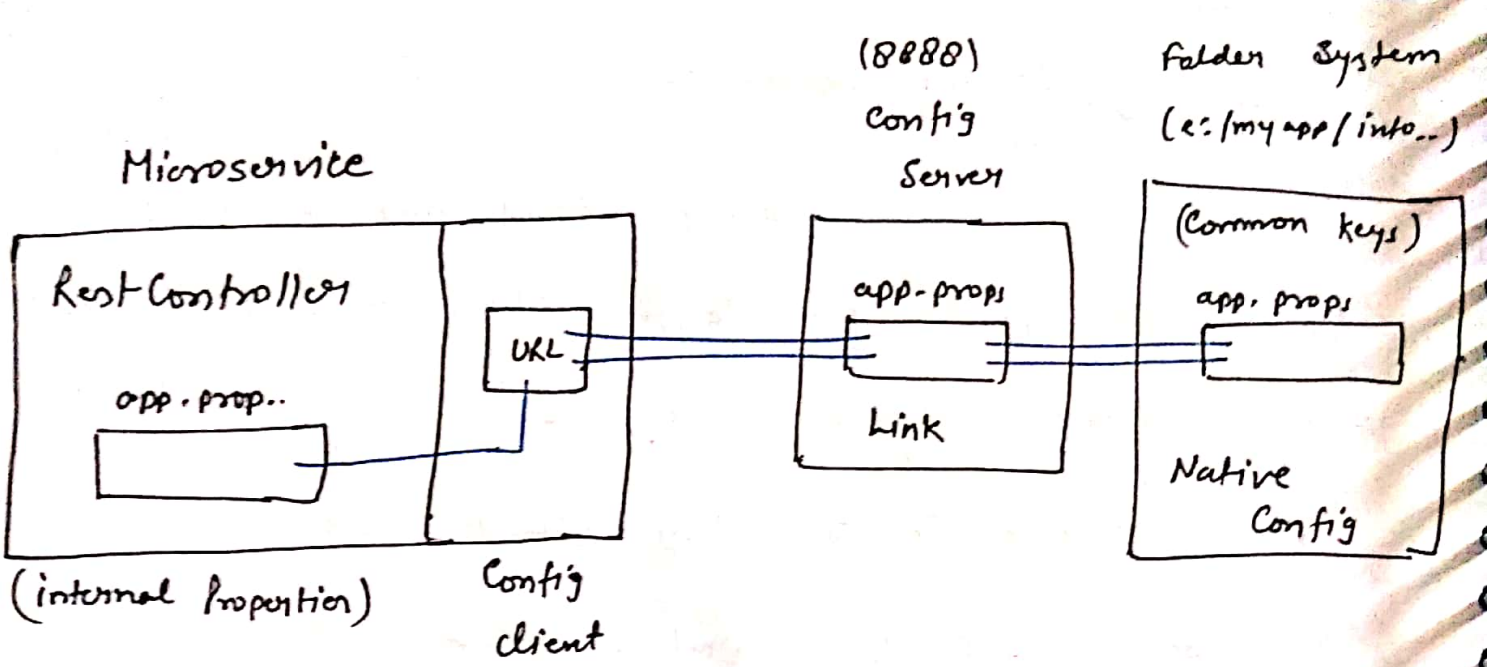
ex. d:/abc/myapp (or under config server location also)

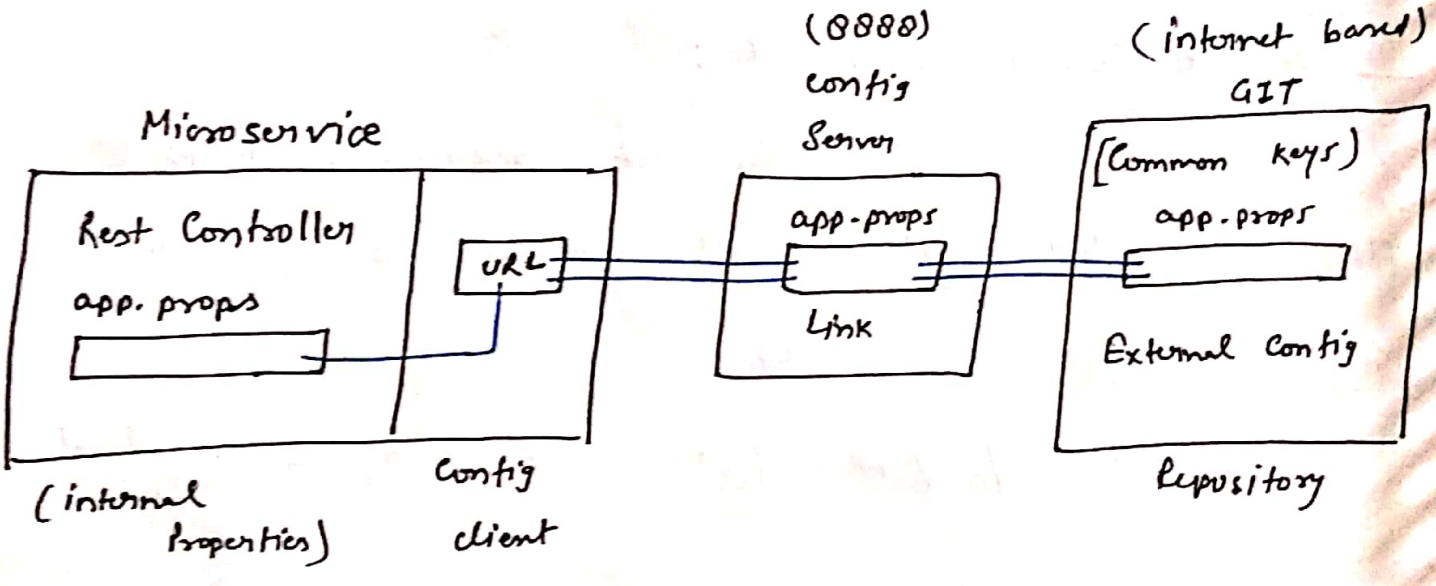
b. External Configuration file :-

Here properties file is placed in internet and accessed using its URL.

→ It is also called as Global location

** GitHub is used in this process.





#1 → Every Microservice (our App) must have below dependency in pom.xml

```
<dependency>
    <g.id>  org.sf.. cloud  </g.id>
    <a.id>  spring-cloud-starter-config  </a.id>
</dependency>
```

→ It behaves as Config Client (Any Component) connects to config server.

→ Above config client dependency will search for Config server in default location given as : (key=val

```
spring.cloud.config.uri = http://localhost:8888
```

* To modify above location (IP or Port) provide this key in config client setup using bootstrap.properties file.

Q What is the difference b/w application.properties and bootstrap.properties file?

Ans. Application.properties file is to provide input to our location,

Where bootstrap.properties file is used to provide input to parent project (or configuration) setup) which gets run before our app.

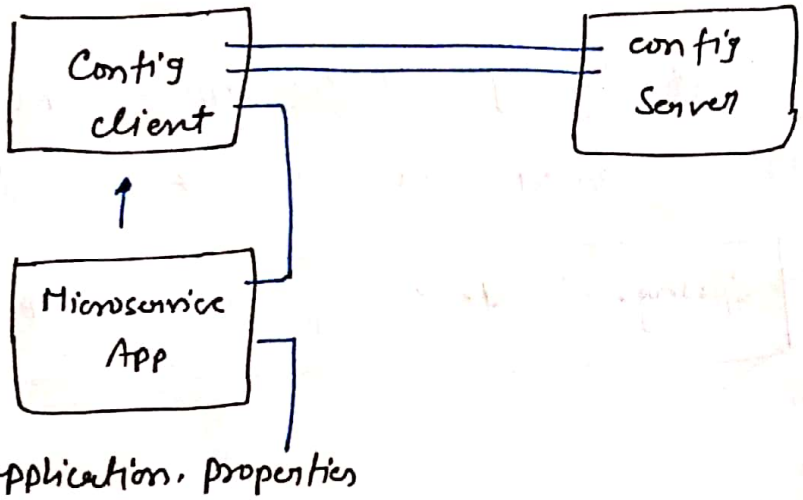
Execution Order :

bootstrap.properties

application.properties

(spring.cloud.config.uri = ——)

bootstrap.properties



→ ** By default config client will not get updates, modifications from config server after starting client + microservice.

** → Only first time config client fetch the data. Even to get new modification after starting client (without re-start)

use annotation : @RefreshScope

→ At config server Application add below dependency in pom.xml.

```
<dependency>
    <g.id>  org.sf...cloud  </g.id>
    <a.id>  spring-cloud-config-server  </a.id>
</dependency>
```

→ At config server starter class level add annotation

@ Enable Config Server

→ In application.properties (in config server) provide Native or External configuration def details.

→ ☆ This file also called as link file.

application.properties ---

server.port = 8888 [recommended]

for git { spring.cloud.config.server.get.uri = ——

for native only { spring.cloud.config.server.native.search-locations = ——
spring.profile.active = native

12-8-19

For Native 12-8-19

Step #1 - Create Eureka Server and setup → eureka

#2 - Create Config Server Application → config server

→ Open Starter class, Add Annotation

@EnableConfigServer

→ application.properties

server.port = 8888

spring.profiles.active = native

spring.cloud.config.server.native.search-locations
= classpath:/myapp/config/

spring.cloud.config.server.native.search-locations
= file:///myapp/config

→ Create folder 'myapp-config' under src/main/resources

→ create file "application.properties" under "myapp-config" folder.

~~→~~ → myapp/application.properties [external file]

my.msg = Hello from Native

3- create any microservice app [all will be config client]

eureka.discovery client, config-client, web

→ create a file bootstrap.properties

bootstrap.properties. [to communicate with server]

spring.cloud.config.uri = http://localhost:8888

→ application.properties

server.port = 9900

spring.application.name = CART-APP

eureka.client.service-url.default-zone =

http://localhost:8761/eureka

4 RestController Code -

```
package com.app.rest;

@RestController
@RequestMapping("/cart")
public class CartRestController
{
    @Value("${my.msg : Hello default one}")
    private String code;
```

* First try to get from config server, if not present then default will taken @Value("${key: default value}")

```
@GetMapping("/show")
public String show()
{
    return "from cart *" + code;
}
}
```

→ Execution Order →

1- Eureka Server
2- Config Server
3- Cart Application

for External server [Spring cloud Config Server External]

1- Eureka Server, Client App are same as previous.

2- Create Config Server App.

3- ~~Crea~~ In application.properties file provide details of git uri

~~application.properties~~ -

application.properties.

server.port = 8888

spring.cloud.config.server.git.uri = http://github.com/abcdsample2019/springcloudconfig

→ Github setup →

→ Create new repo. - springcloudconfig [anyName]

→ create a new file application.properties

application.properties [~~config~~ External File]

my.msg = Hello from git

→ copy uri for config server propatics file.

Note→ If we change something in external file [int github file] then changes will not be reflected in our application. When first time when app is loading then data is fetching from server.

To reflect changes in our app. ~~out~~ without reload we need to apply an annotation @RefreshScope on controller class.

Refersh is a service of actuator so we need to add this dependency in our app.

→ To enable Refresh concept for Microservice application (Cart-App). follow below steps

→ Inside Config client [Cart App]

→ Open pom.xml & add dependency - actuator

```
<dependency>
    <g.id>  org.sf.boot  </g.id>
    <a.id>  spring-boot-starter-actuator  </a.id>
</dependency>
```

→ application.properties add -

management.endpoints.web.exposure.include = *

→ At controller class level add annotation @RefreshScope

→ Run → Eureka Server, Config Server, Cart App

→ Enter url for CartApp

→ Now modify value in github file.

→ Open Postman & make POST Request

POST	http://localhost:9900/actuator/refresh	SEND

→ After showing success App Homage, goto cat cart app url and refresh

Note →

actuator/refresh :- It is a ready made service given by spring boot, that fetch data from Config Server location to our App.

→ It must be made as POST Type request only using any Http Client (ex. Postman)

** first bootstrap.properties [input to parent] will be executed then application.properties [input to app] will be loaded.

→ Whenever external application.properties changes then just make a post request to actuator using. postman or restTemplate.