Problems with Refresh Endpoint

In the real world application, there will be multiple micro services that are serving the enterprise application and its difficult to manually triggers the refresh event for all services whenever a property is changed I the config server.

The solution is to trigger the refresh event for a single service, which is broadcasted to other services using Spring Cloud Bus.

Spring Cloud Bus

- The **Spring Cloud Bus** provides a mechanism to refresh configurations across multiple instances without knowing how many instances there are, or their locations.
- This is done by connecting all service instances through a single message broker. Spring cloud bus links the independent services in the microservices environment through a light weight message broker like RabbitMQ, Kafka, etc. Thus,**Each instance subscribes for change events, and refreshes its local configuration** when required.
- Here message broker is used to broad cast the configuration changes and events across the microservices And ***the bus is like a distributed actuator*** for spring boot application.

How Spring Cloud Bus Works

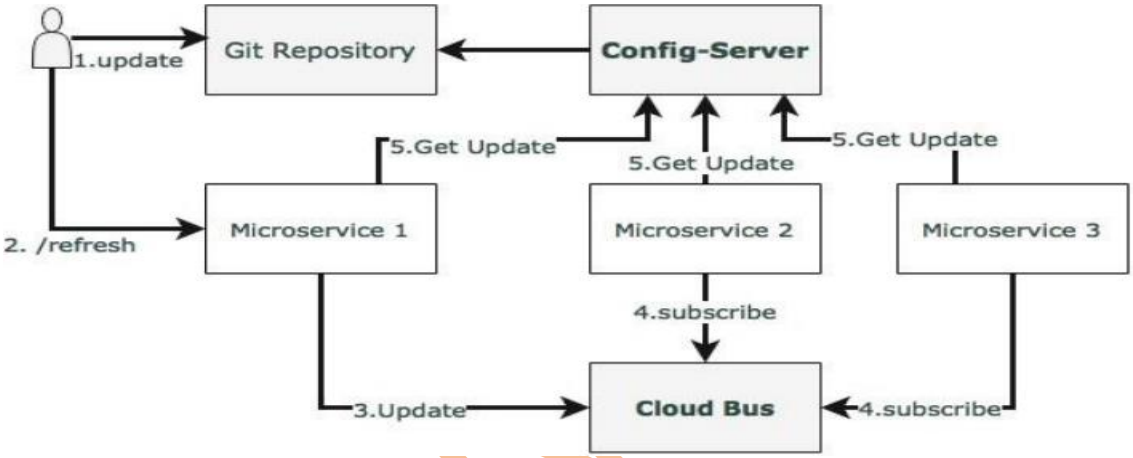
So far we externalized the Configuration for two micro services, I.e. driver-service and vehicle-service by keeping the properties files in remote Git repository.

The microservices, requested the Configurations on startup , from the cloud config server that fetches the properties changes from Git repository. This way the configurations were externalized at the central repository that was consumed by multiple microservices.

Any change in the configuration required, the individual microservices refresh endpoint need to be invoked to reflect the change. This was the drawback which we will overcome with the help of spring cloud bus.

To do so, we will connect the spring cloud clients via the spring cloud bus. Spring Cloud bus, uses a light weight message broker such as RabbitMQ, Kafkka, etc to broad cast the events across all connected clients.

We Can change the properties, and push the change to Git Repository. Then trigger the ***bus-refresh*** event for any of the micro services(say driver-service). The Spring Cloud bus receives the refresh event and broad casts the refresh event across all the connected microservices through the underlying message broker.



Note:

Each Config Client(microservice), should have the spring boot actuator endpoint in its class path, So that the refresh event, will be handled properly and all the beans annotated with @RefreshScope will be refreshed when **bus-refresh** event is triggered for one of the microservices.

Implementation

Lets take the Message Broker as RabbitMq in our applications. Cloud ready RabbitMq we will get from the **https://customer.cloudamqp.com/instance**.

We should have to register ourself here, the we will get the login credentials. Once we login, we can create a queue where we can send and receive messages from it.

Lets configure the Bus on the config-server and all other micro services which required the updates from the config-server.

Configuring Bus at Config-Server

Step-1: add amqp and cloud bus starter in **config-server** as shown below:

```
<!-- Dependencies for Cloud Bus Configuration Start-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-bus-amqp</artifactId>
    </dependency>
    <!-- Dependencies for Cloud Bus Configuration End-->
```


```
<!-- Dependencies for Cloud Bus Configuration Start-->

    <dependency>

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

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

    </dependency>

    <dependency>

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

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

    </dependency>

    <!-- Dependencies for Cloud Bus Configuration End-->
```

Step-2:

Add @Refresh annotation on Application.java file as shown below:

```
@SpringBootApplication
@EnableConfigServer
@RefreshScope
public class ConfigServerApplication {

    public static void main(String[] args) {
        SpringApplication.run(ConfigServerApplication.class, args);
    }
}
```

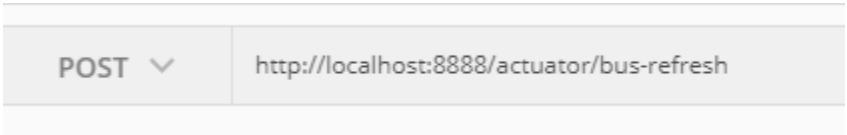
Step-3:

Add the ActiveMq Broker I.e. RabbitMQ server details in **application.properties** file to which the config-server should subscribe to send the configuration changes from where all other subscribed micro services will receive the configuration changes.

application.... | driver-flee... | application.... | vehicle-fle... | application.... | DriversCont... | DriverFleet...

```
1 server.port=8888
2 spring.application.name=config-server
3
4 spring.cloud.config.server.git.uri=https://git-codecommit.us-east-2.amazonaws.com/v1/repos/fleet-management-platform
5 spring.cloud.config.server.git.username=admin-at-102172735049
6 spring.cloud.config.server.git.password=LKfISmCVPec60pPBayrMEwza9q1Nsq0Lyw38SiV+6Pw=
7
8
9 spring.cloud.config.server.git.search-paths=driver-service-config/*,vehicle-service-config/*
10
11 spring.rabbitmq.host=grouse.rmq.cloudamqp.com
12 spring.rabbitmq.virtual-host=ddfrrrdd
13 spring.rabbitmq.username=ddfrrrdd
14 spring.rabbitmq.password=5w8gTwhmnVKTUPr5bW04XV5D_d-HeaJ2
15
16 management.endpoints.web.exposure.include=*
```

Step-4: restart config-server and hit the url as below:



Note:

1) /actuator/bus-refresh end point will be post the request not GET

2) By default this end point will be in secured mode, so that if directly we could access it, we will get "Access denied-401" exception. Hence disable the security mode as below:

Configuring Bus at Microservice

Lets an example of patient-service which is configured with high availability.

Step-1: add amqp and cloud bus starter in config-server as shown below:

```
<!-- Dependencies for Cloud Bus Configuration Start-->

    <dependency>

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

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

    </dependency>

    <dependency>

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

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

    </dependency>

    <!-- Dependencies for Cloud Bus Configuration End-->
```

Step-2: Add @Refresh annotation on Controller.java file as shown below:

```
@EnableResourceServer
@RestController
@RefreshScope
@RequestMapping("/api")
@Slf4j
public class DriversController {
```

Step-3:

Add the ActiveMq Broker I.e. RabbitMQ server details in **application.properties** file to which the config-server should subscribe to send the configuration changes from where all other subscribed micro services will receive the configuration changes.

```
spring.rabbitmq.host=grouse.rmq.cloudamqp.com
spring.rabbitmq.virtual-host=ddfrrrdd
spring.rabbitmq.username=ddfrrrdd
spring.rabbitmq.password=5w8gTwhmnVKTUPr5bW04XV5D_d-HeaJ2
```

Step-4: restart the micro service:

Note:

1) /actuator/bus-refresh end point will be post the request not GET and can be issued at config-server and micro-service side.It is recommended to hit this end point on config-server so that all the micro service will get the changes at the moment when the above endpoint triggered.

2) As it is recommended to issue the /actuator/bus-refresh end point on config-server, we no need to disable the security on each micro service end.

Now to see the working effect of the bus, change the property

```
novelhealth.date.format=dd-MM-yyyy HH:mm:ss
```

In patient-service properties file of config-server, then hit the end point /bus/refresh on config-server as shown above then these changes should automatically picked up by patient-service where they also connected to the same bus I.e.amqp to which the config-server has been connected.