* Enabling Ribbon

1] Dependency

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

2] Registering instances to Ribbon Server [i.e CES instances in our Case]

```
@Configuration
class RibbonConfig
{
    @Bean
    Public ServerList<Server> ribbonServerList()
    {
        StaticServerList<Server> l = new StaticServerList<>
            ( (new Server("localhost", "8100")),
              (new Server("localh", "8101")));
        return l;
    }
}
```

3] Proxy Class

```
@Ribbon("spring.application.name")
@Feign(" - 1 - ") // no use as it is already
interface CESProxy                registered with ribbon
{
    // mtd definitions
}
```

* Eureka Server [Naming Server] : For dynamic up & down instance based on usage or load

1] Dependency : Eureka Server
Config-Client

2] Main mtd :
@EnableEurekaServer

3] application.properties

Spring.application.name =
Server.port = 8761 // always 8761 as server will listen to this port as known server else throws TransportException/ConnectionRefused exception

eureka.client.register-with-eureka = false
- 1 - . fetch-registry = false

4] Changes to be done on micros [Component/Listener/Clients]

* dependency
<spring-cloud-starter-netflix-eureka-client>

* Main mtd
@EnableDiscoveryClient // Register with eureka server
* application.properties

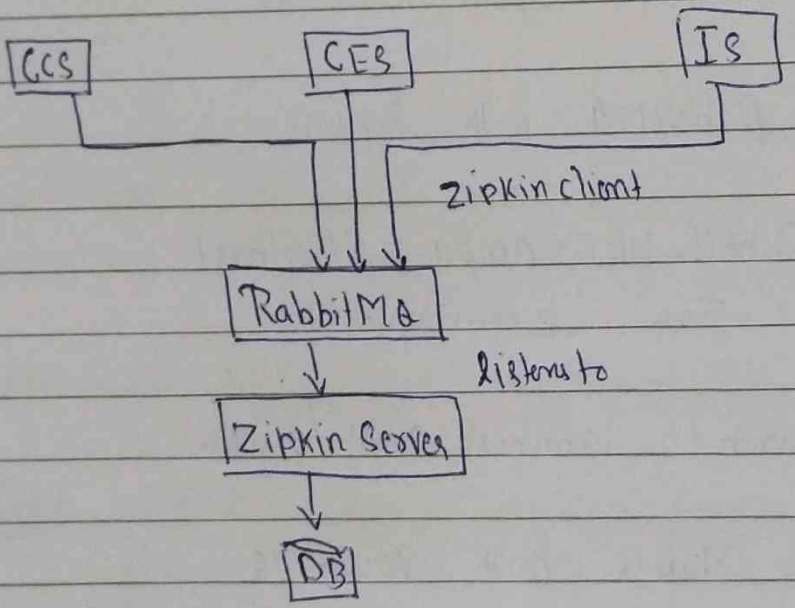
eureka.client.service-url.defaultZone = http://localhost:8761/eureka

Note:

* launch Eureka browser by url
  http://localhost:8761

* mvn dependency: purge-local-repository
  removes project dependencies from local & re-resolve them.

✡ Distributed Tracing System [Used to debug microservices Uses Sleuth, Zipkin, ELK/RabbitMQ



Dependency: for Micros:

* Sleuth: Used to generate unique id for each & every HTTP request

* Zipkin-client: Used to send & convert HTTP Request data to ~~ELK~~ Zipkin understandable format.

✡ Installations Required:

* RabbitMQ: Used as MQ, along with "Erlang" is required as Prerequisite.

* Zipkin Server: Listens to RabbitMQ and can used Centralised UI to see logs.

Main mtd : [for Micro]

```
@Bean
public Sampler getDefaultSampler()
{
    return Sampler.AlwaysSample;
}
```

✡ Run Zipkin Server :

Java -Jar ZipkinJarName.Jar.

✡ Integrate RabbitMQ with Zipkin

> Set RabbiT_URI = amqp://localhost
> Java -Jar ZipkinJarName.Jar

Url to launch : localhost:9411/zipkin

✡ Integrate Micro's with RabbitMQ

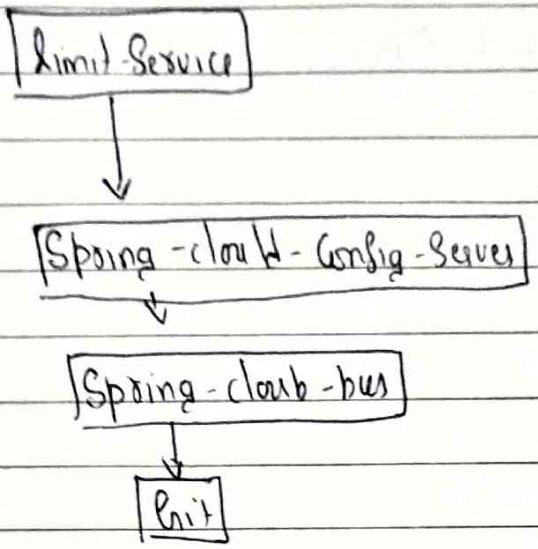
Dependency:
Spring-rabbit

✡ Order of Running Micro's:

1] Naming Server
2] Zipkin Server
3] Micro's

Last] API gateway

Spring-Cloud-bus:



* to make changes made in application properties to reflect in API side we need to call following Url for the instance to refresh & take latest changes:

http://localhost:{limit-service port}/application/refresh
(instance port)

Cons: With this approach we can make only one instance to take latest changes, so if there are 10 instances then we need to call the Above Url 10 time with 10 port no's.

In order to overcome above Cons we use Spring-cloud-bus. where one Url alone can refresh all instances of micro to take latest changes.

*Add dependency in Micro & Spring cloud-Config-Server:

```
< Spring-cloud-started-bus-amqp >
```

* Url to refresh all instances to take latest Git changes:

actuator / bus-refresh

http://localhost:<any Port>/limit-service/bus/refresh

* At application start up all Microservice instances are registered with bus, when an refresh is call an event is called to spring-cloud-bus which will refresh all instances to take latest changes.

* Note:

spring.security.enabled = false [application.properties of Microservices]

This setting is used if we want to skip authorization on call of refresh API's.

✡ Fault Tolerance [Hystrix]:

Dependency: [for Micro's]

```
<spring-cloud-starter-hystrix>
```

Main-mtd:

```
@EnableHystrix
```

Controller level:

```
@HystrixCommand(fallbackMethod="fallbackRetrieveConfig")
@GetMapping(...)
Public ReturnType m1()
{
//throws Exception
}
```

```
Public  LimitConfig  FallBackRetryConfig()
{
    return new Ra(9,99);
}
// Returns some default value if API fails.
```