Microservices:

Def :

* Exposed by rest
* Small Deployable Unit
* Cloud enabled

Challenges:

* Defining Boundaries/Bounded Context
* Configuration Management(multiple microservices multiple env and instance)
* Dynamic Scale Up and scale Down (based on load on each microservices)
* Difficult to identify which micro service is down as one is calling other ,debugging is difficult.

Spring Cloud:

* Configuration Management –Spring Cloud config server used to solve this problem(where we store all the configuration in git under one config server).
* Dynamic Scale Up and Scale Down – Ribbon Client-Side Load balancing

X microservice

X microservice 1

X microservice 2

Eureka (Naming sever

* Visibility and Monitoring:

Ziplin Distributed Tracing

Netflix Zule Api Gateway -handles security and logging .

* Fault Tolerance: Hystrix (if service is down it helps to configure default response)

Advantages:

* Enables us to adapt new technology and processes.
* Dynamic Scale up and Scale Down.
* Faster release cycle.

Spring -cloud-config -server: (Used to manage properties for multiple microservice)

* First while creating project add dependency as config server in pom.xml
* After that before @SpringBootApplication add @EnableConfigServer.
* Connect Config Server to git repo using Spring.cloud.config.server.git.uri
* Now to use the config server in other microservice use spring.cloud.config.uri in bootstrap.prop file.
* Which profile to use in other microservice is identified by Spring.profiles.active in bootstrap.prop file
* If any microservice wants to connect to config sever than in pom.xml add config client.

Note:::: How to know on which port your application is running (Simply autowire Environment calss and eg.. environment.getProperty(“local.server.port”))

**How to dynamic change port in response – by creating new run configuration and in vm argument specify -DSever.port)**