1. **Definition**

- Microservices are multiple services that are well managed and deployed, short attributes are that:

+ REST

+ Atonomous services

+ Lightweight communication mechanisms.

+ Deployable, manageable (managing statuses, logs, downtime, …), well designed boundary units.

+ Load balancing between instances of services.

+ Well visibility (Downtime - health, centralized logs, memory, debug-able services, ….)

- With Cloud enabled => instances of each service and be easily managed (scaling up - down) depends on user load. Also, configures can be centrally managed.

-Spring Cloud def: => tools for developers to quickly build some of the common patterns in distributed systems (incld. Configuration management, service discoveries, intelligent routing, …)

- **Challenges and solutions**:

+ Configuration managements

=> Spring could configure server can expose configurations to all services.

=> Used with spring cloud bus to pick up changes quickly when configs in the repo get changed

+ Dynamic scale up / down

=> Naming Ureka Server (keep track registered instances of microservices -> service discovery) => Ribbon (client side load balancing)

=> Feign (Easy invocation to other REST services) – using this with ribbon load balancing

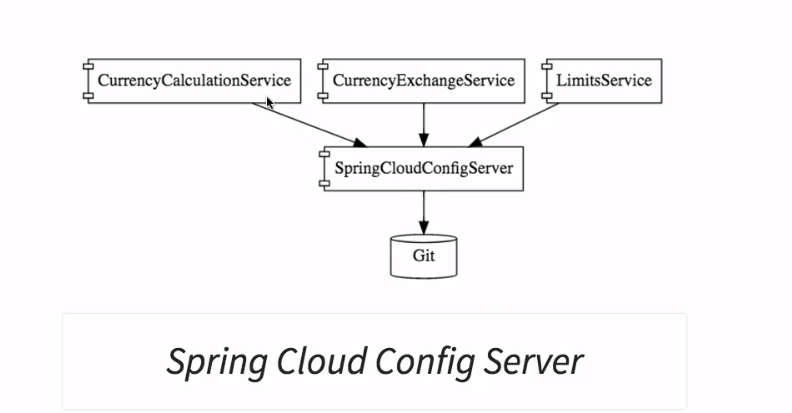
+ Visibility and monitoring

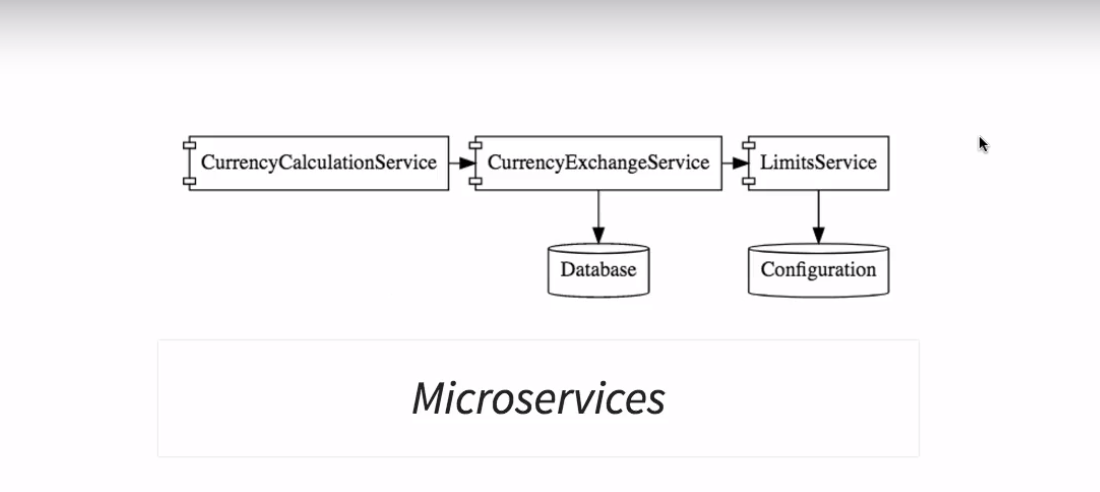
* Zipkin Distributed Tracing ( with spring cloud sleuth -> assigning unique ids to requests and so zipkin can trace request across multiple components).
* Netflix Zuel API gateway (a gateway intercepting all incoming requests, then perform common features such as logging / security check, … and then routing to services).

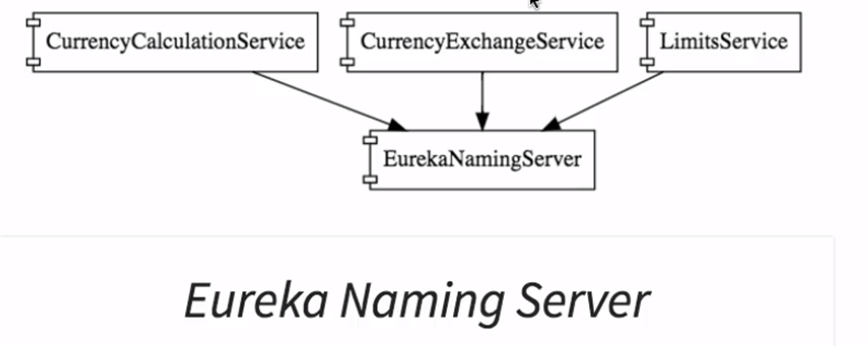
+ Fault tolerance

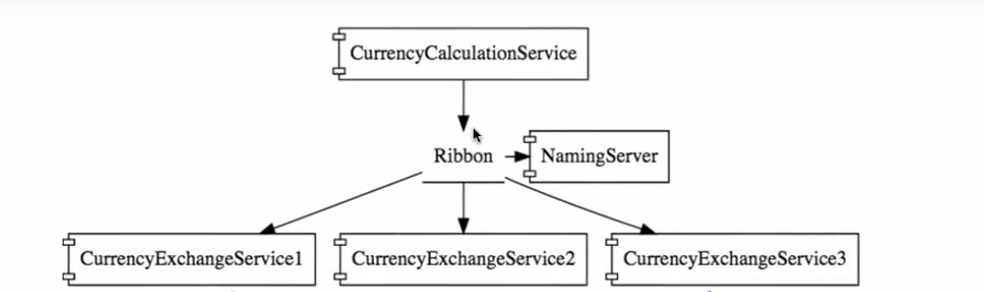
* Hystrix (If a service is down, hystrix helps configure a default response)

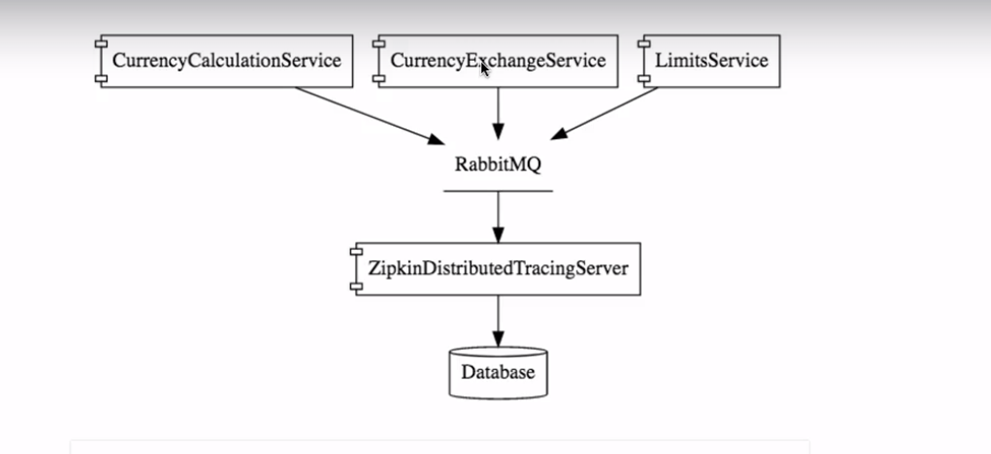
1. **Micro-service architecture – Overall project architecture:**











2. **Zuul api gateway**

* Gateway in micro service architecture receive all incoming requests, performs security checks, do filtering and logging and finally routing to services.
* Registered in Eureka -> intercept requests and distribute to services through instance names
  + eureka.client.service-url.default-zone=http://localhost:8761/eureka
* Enable Zuul gateway by annotation @EnableZuulProxy, @EnableDiscoveryClient
* Gateway signature: {gateway-url}/{application-name}/uri.
* Ex: http://localhost:8765/currency-calculation-service/currency-calculation/from/usd/to/vnd/quantity/1000

3. **Feign + ribbon load balancing**

* Feign helps reduce the work when trying to make a micro-service / API call while ribbon helps balancing the call to the service API (client load balancing)
* Configure feign by providing feign API endpoint address
* Detail can be viewed in the following path:
  + Currency-conversion-service/src/proxy/CurrencyExchangeServiceProxy.java

4. **Eureka naming server**

* Eureka Naming Server will keep track of instances of micro-services, when a eureka client ask server for service’s instances -> address of a managed instances will be returned
* Dependency for eureka server:

<dependency>

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

<artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>

</dependency>

* Dependency for eureka client:

<dependency>

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

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>

</dependency>

* Configure a eureka naming server by providing a port and specify @EnableEurekaServer
* Configure a eureka naming client by providing a port, eureka’s server address

and specify @EnableDiscoveryClient

* + eureka.client.service-url.default-zone=http://localhost:8761/eureka

5. **Centralized and distributed tracing: Spring cloud sleuth with Zipkin**

* **spring cloud sleuth** assign a unique ID to a request across components
* **Zipkin is a distributed tracing system** which will receive request span logs (log coming from services) from MQ (using rabbitMQ) to trace it
* **Config zipkin server to read span logs from rabbitmq by using the command:**

RABBIT\_URI=amqp://localhost java –jar zipkin.jar

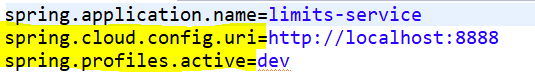
* Remember to add dependencies to services to help request send span logs to zipkin via rabbitmq
* <dependency>
* <groupId>org.springframework.cloud</groupId>
* <artifactId>spring-cloud-starter-sleuth</artifactId>
* </dependency>
* <dependency>
* <groupId>org.springframework.cloud</groupId>
* <artifactId>spring-cloud-starter-zipkin</artifactId>
* </dependency>
* <dependency>
* <groupId>org.springframework.amqp</groupId>
* <artifactId>spring-rabbit</artifactId>
* </dependency>
* Enable sleuth by using

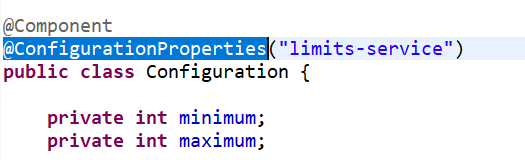
@Bean

Public Sampler defaultSampler() {return Sampler.ALWAYS\_SAMPLER;}

6. **Spring cloud config**

* In the micro service architecture, spring cloud config server is a **central place maintaining configurations (in variety environments)** for **all of the services**.
* Configure **spring cloud server** by **specifying the address to git repo** where config files are managed and add annotation @EnableConfigServer
  + spring.cloud.config.server.git.uri=file:///C:/Users/LVU02/Desktop/Project/in28minsMicroservices/git-localconfig-repo
* Configure **spring cloud client** by specifying the config server address, profile of the configure

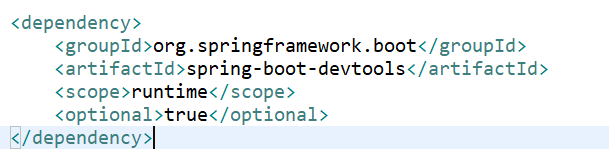




* Remember to use this with spring cloud bus to enable quick refresh of services getting changed configs coming from git repo. Add dependency “spring-cloud-starter-bus-amqp” to services and use link **service\_address /bus/refresh** to acknowledges changed configs. 1 call from any running service will generate an event in spring bus and propagate to all other services.

7. **Devtool live reload**

* When spring boot source code get changed, devtool will trigger a reload



8. **Fault tolerance with hytrix**

* Hytrix is enabled by annotation @EnableHystrix

