Preparation keywords:

- Independent
- XML
- JSON
- Service definition
  - Request / response format
  - Endpoint
- Terminology
  - Transport (Http / Mq)
- Client (consumer)
- Server (provider)
- Groups
  - SOAP based: poses restriction between client / server.
  - REST styled: Architecture Approach
- SOAP (XML)
- Simple Object Access Protocol
- Service Definition: WSDL
  - Endpoint
  - Request / response structure
- REST
- RESTful web services
  - Define services with different concepts that are already present in HTTP
  - Resources
  - WADL: To specify your RESTful web services
  - Swagger: To documentary Microservices (More popularity than WADL)
- Build RESTful webservices: build your microservices
- I18in: internationalization

Microservices

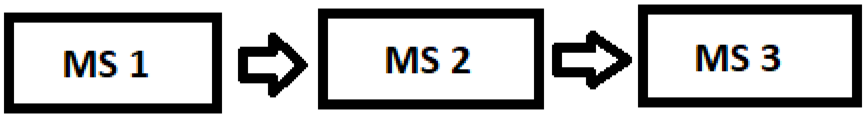
- Actuator (dependency): Monitoring API services and show the APIs.
- Hal-browser (dependency): It's use to monitoring services too and make APIs as links easy to consume.
- @ApiModal: To description an object.
- @Component: Dao Services determine with this annotation.
- Static filtering: Ex. @JsonIgnore from Jackson.
- Dynamic filtering: Mapping Jackson value class in Request method With @JsonFilter on bean class.
- Versioning RESTful services: when two different method with same mapping URI. (Type: Media, URI, Request parameter).
- Basic authentication: Spring security dependency works with hashed API pass and username.
- JPAREpository:
- Monitoring web App: Actuator (it's recommend) or hal-browser
- ResponseEntity (Return type): Best practice to return from methods insert or update records.
- Richardson Maturity modal: Important best practice model of RESTful webservices.
- Richardson model:
  - Level 0: Expose SOAP in REST style.
  - Level 1: Expose resources with proper URI.

    - Level 2: lvl 1 + proper Http method.
    - Level 3: lvl 2 + Hateoas
- Hateoas

Richardson model:

1- Consumer first: Most important best practice in RESTful services design. Think about consumers that they understand naming of resources?
2- Make best use of Http: Best practice to use of whatever Http provide. RESTful services are based on Http. It's about type of request, correct one and response status rightly.
3- No secure info in URI.
4- Use URI resources name in plural. Ex. /users/
5- Use nouns for resources.

Microservices (MS): It's on REST defined. Small autonomous services that work together. Small well chosen deployment unit.

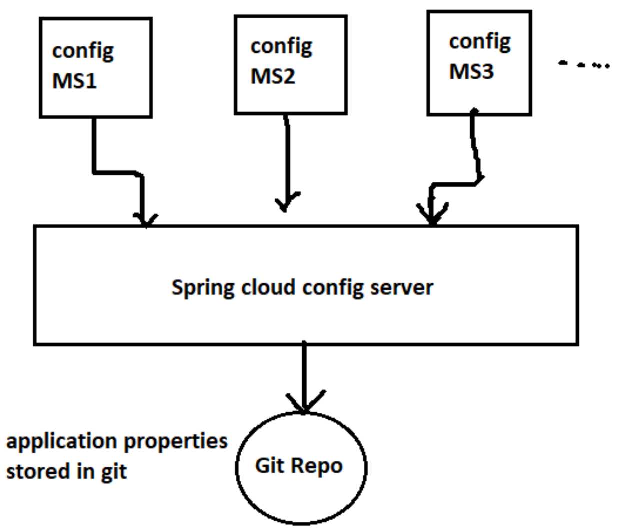


Microservices challenges:

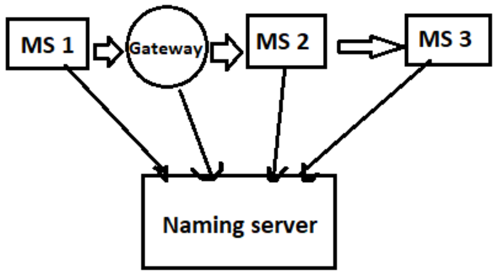
1- Bounded context: how do you identify each of MSs. What to do in each of MS. How to decide what to do what not to do.
2- Configuration management
3- Dynamic scale up / down, about load of MS instances.
4- Visibility : How to find out which MS caused the problem. So need great visibility to what is happening in MSs automatically.
5- Pack of cards: How do you prevent one MS being down taking down the entire application.

- Spring cloud: This tool answered to the MS challenges.
    - Spring cloud config

            - Config: To solve configuration management. Identify name and port with definition of client config and server config. Git repo as config management. Git contains the whole of MSs configs.

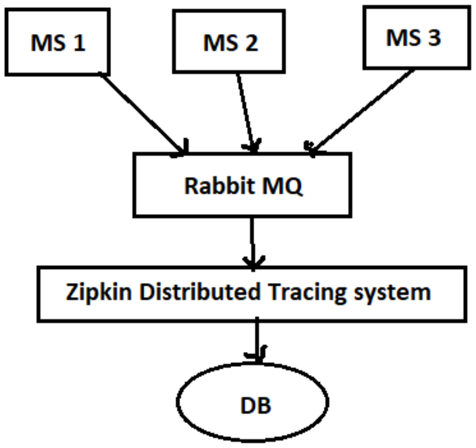

config MS1
config MS2
config MS3
Spring cloud config server
application properties stored in git
Git Repo

            - Bus: Enables the MSs, API gateway to talk to each other.
    - Dynamic scale up / down challenge solutions
        - Eureka: Naming server or Service registration. Service discovery functionality.
        - Ribbon: Load balancing BETWEEN ms instances. This dependency already integrated in Feign.
        - Feign: solutions for visibility . Mechanism to write simple RESTful client. REST service client to call RESTful services. To invoke the other MSs. Feign already uses Ribbon.
    - Visibility / monitoring solution
        - Zipkin: Distributing tracing server. To trace a request across multiple component.
        - API gateway: Handle a lot of common features. Ex. Zuul, Spring gateway.
            - Features like Authentication , Authorization, security, Rate limits, fault toleration and service aggregation.

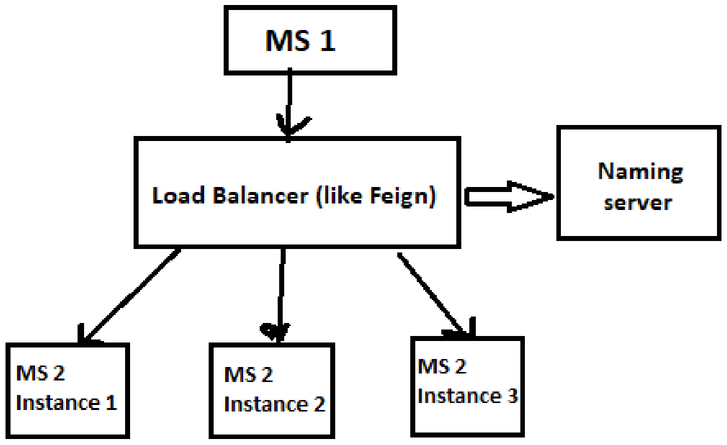


            -
    - Fault tolerance solutions: Ex. Hystrix: When MS is down Hystrix help to configure a default response.
        - Hystrix dependency: A framework help us to build fault tolerance MSs.
        - When MSs are interconnected like chains. Therefore, if an error occurs in one of Microsoft's servers, the other servers will not be affected.
    - Distributing tracing: Centralized kind of information. This able with spring cloud sleuth on zipkin server. Sleuth assign an unique id to a request.
        - Zipkin: it's a distributed tracing system. And centralized all messages (logs) to one place. default port = 9411. This server runs by a specific zipkin.jar file in terminal and it's work with Rabbit MQ.
            - Rabbit MQ: Advanced Message Queuing Protocol (AMQP)
                - Pre installation: Erlang.
                - It use to add functionality to our MS.

- Before stating Zipkin we must to say that Rabbit MQ is running and up to zipkin by this commend: RABBIT_URL=amqp://localhost
- Rabbit MQ dashboard: 127.0.0.1:15672



- Spring cloud bus: We have one URL for instances and once you hit that URL, all the instances would be updated with latest values of Git config. Without Bus we would need to go to every single client and reload config (Ex. by actuator endpoint).



- Spring cloud API gateway
  - To manage common features and it's used instead of zuul. (Zuul not support by Netflix anymore.)
  - Simple effective way to route to APIs.
  - Provide cross cutting concerns (security, monitoring).
  - Build on top of Spring Webflux (Reactive Approach).
  - Features: Match route on any request attribute.
  - Define predicates and filters.
  - Integrates with Spring cloud discovery client (load balancer).
  - Path rewriting.
- Circuit breaker: what if one of MSs is down or is slow? Can we return fallback response? Can we implement a circuit breaker pattern to reduce load? Can we implement rate limiting? Solution: Resilience4j, a framework to fault tolerance that inspired by Netflix Hystrix.
- Resilience4j:
  - @CircuitBrreaker:
    - Close - state: When calling dependent MS continuously.

        - Open – state: We’re not call dependent MS into directly return the fallback response.
        - Half_open - state: Sending a percentage of request to the dependent MS and for rest of request it’ll return the fallback response.

About sample project:

- First of all start Rabbit MQ and run zipkin, then launch naming server , then gateway server, then the other microservices.

Application	Port
Limits Service	8080
exam-micro-service	8200
naming-server	8761
school-micro-service	8000
spring-api-gateway-server	8700
spring-cloud-config-server	8880
student-micro-service	8100, 8101

- Zipkin Installation:
    - Quick Start Page: - https://zipkin.io/pages/quickstart
    - Downloading Zipkin Jar: - https://search.maven.org/remote_content?g=io.zipkin.java&a=zipkin-server&v=LATEST&c=exec
    - Command to run: RABBIT_URI=amqp://localhost java -jar zipkin-server-2.12.9-exec.jar
- Spring cloud config git Commands:
    - mkdir git-configuration-repo
    - cd git-configuration-repo/
    - git init
    - git add -A
    - git commit -m "first commit"
- references:
    - https://github.com/in28minutes/spring-microservices/tree/master/03.microservices
    - https://spring.io/blog/2019/08/16/securing-services-with-spring-cloud-gateway
    - https://spring.io/blog/2019/08/16/securing-services-with-spring-cloud-gateway#:~:text=To%20secure%20our%20services%2C%20we,secured%20resources%20behind%20the%20gateway.
    - https://docs.spring.io/spring-cloud-sleuth/docs/current-SNAPSHOT/reference/html/appendix.html
    - https://dzone.com/articles/spring-cloud-amp-spring-bootimplementing-zipkin-se
    - https://spring.io/projects/spring-cloud-sleuth
    - https://www.baeldung.com/tracing-services-with-zipkin
    - https://www.baeldung.com/jackson-serialize-dates
    - https://www.baeldung.com/zuul-load-balancing
    - https://www.baeldung.com/circular-dependencies-in-spring
- Sample project:
    - https://github.com/sajiniho07/micro-services-practice
- Researcher:
    - https://www.linkedin.com/in/sajad-kamali-0a840594/