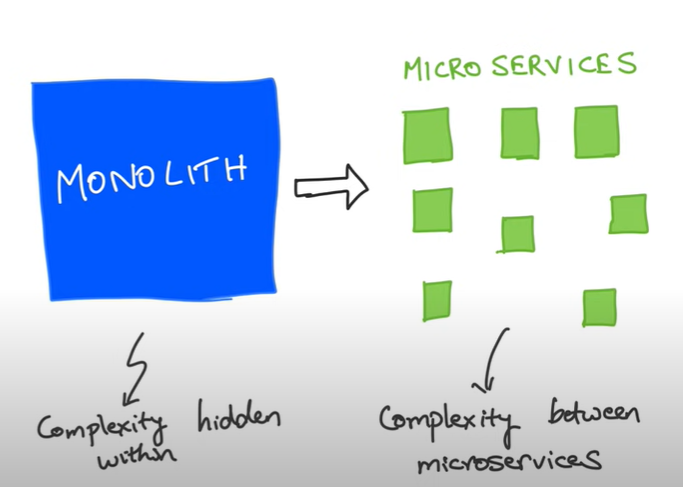
## Agenda

* 3 microservices services
* Make them communicate
* Spring cloud is one way to create microservices



## Urls

<http://localhost:8081/catalog/5>

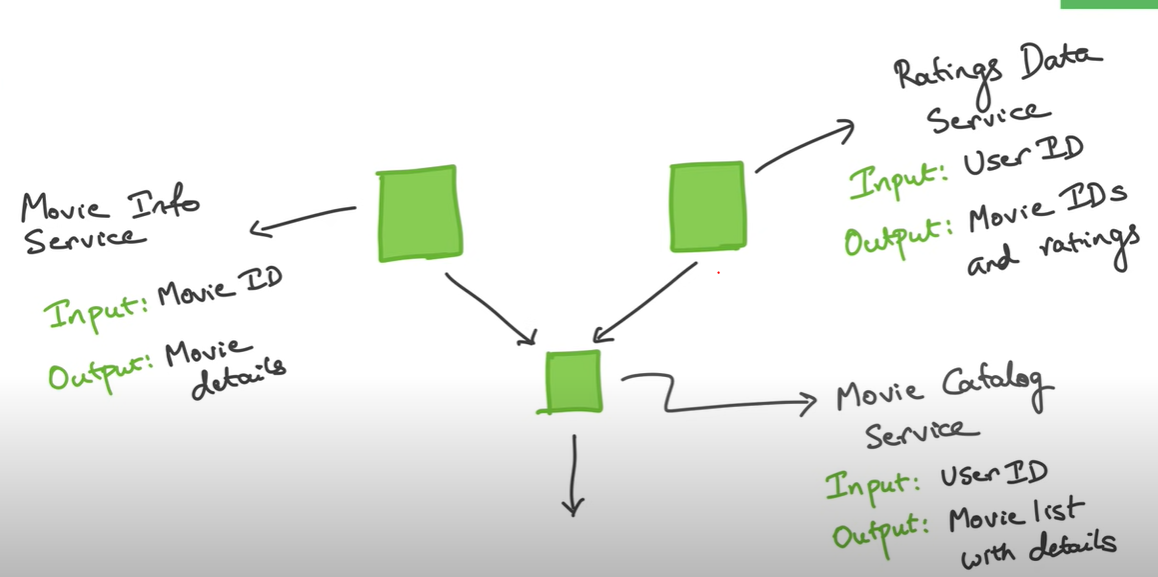
<http://localhost:8082/movies/foo>

http://localhost:8083/ratingsdata/foo

<http://localhost:8083/ratingsdata/users/123>

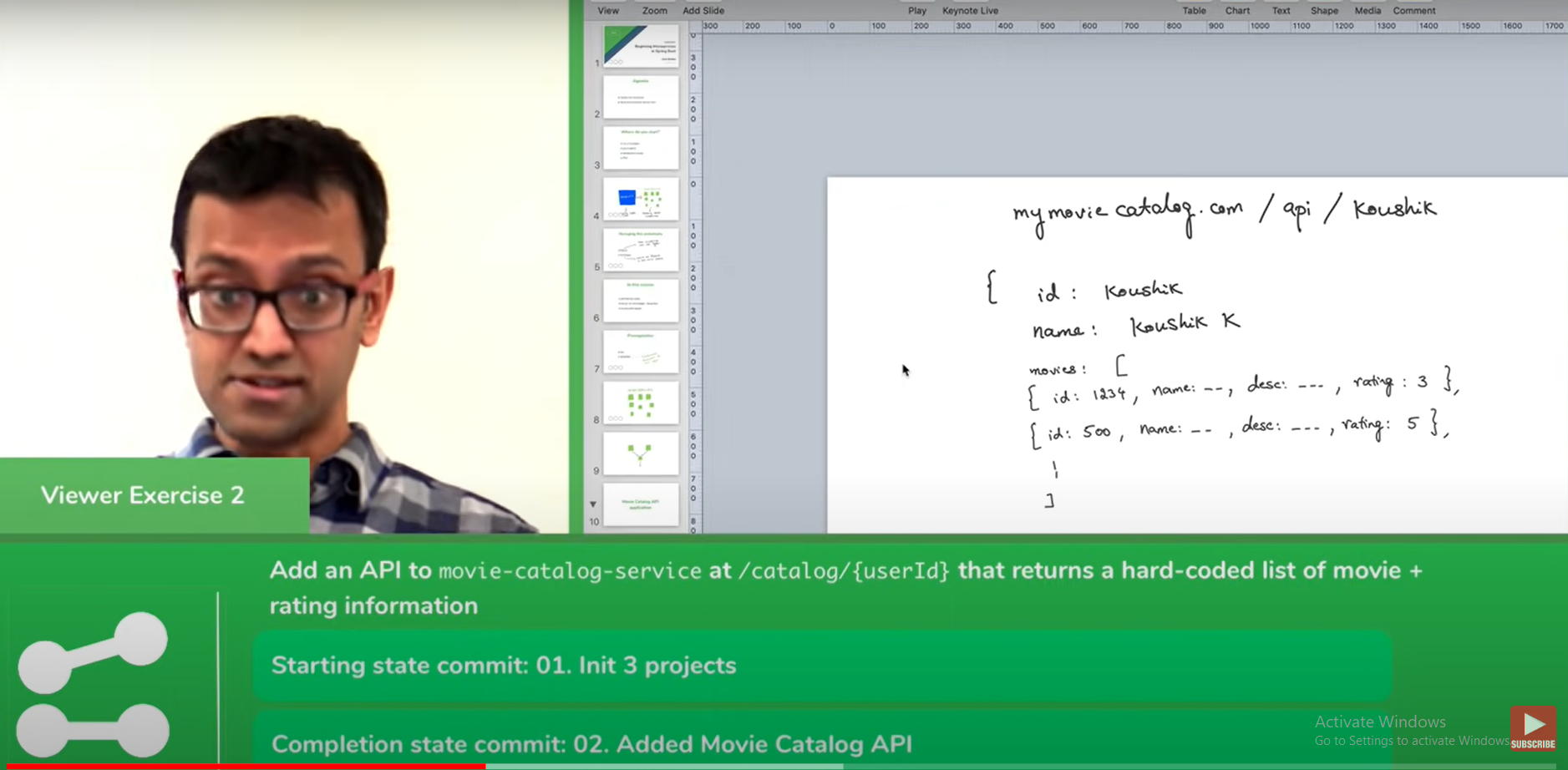
## Microservice vs web services

## Movie catalog application



## Create 3 services with bare minimum dependency. So create 3 services with web dependency

## start with catalog service



## RestTemplate to make API calls

* Can use Feign instead refer udemy 161
* Web client (requires reactive programming) => asynchronous

Note :

**While doing unmarshaling make sure that class have no args constructor**

## how do we create singleton instance of object? Here how do we create single instance of RestTemplate and share it everywhere..

@SpringBootApplication

public class MovieCatalogServiceApplication {

@Bean

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

public static void main(String[] args) {

SpringApplication.run(MovieCatalogServiceApplication.class, args);

}

}

*@Autowired*

private RestTemplate restTemplate;

@Bean create singleton instance it is producer, @Autowired is consumer

## Playing with WebClient instead of RestTemplate

Add dependency

<dependency>

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

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

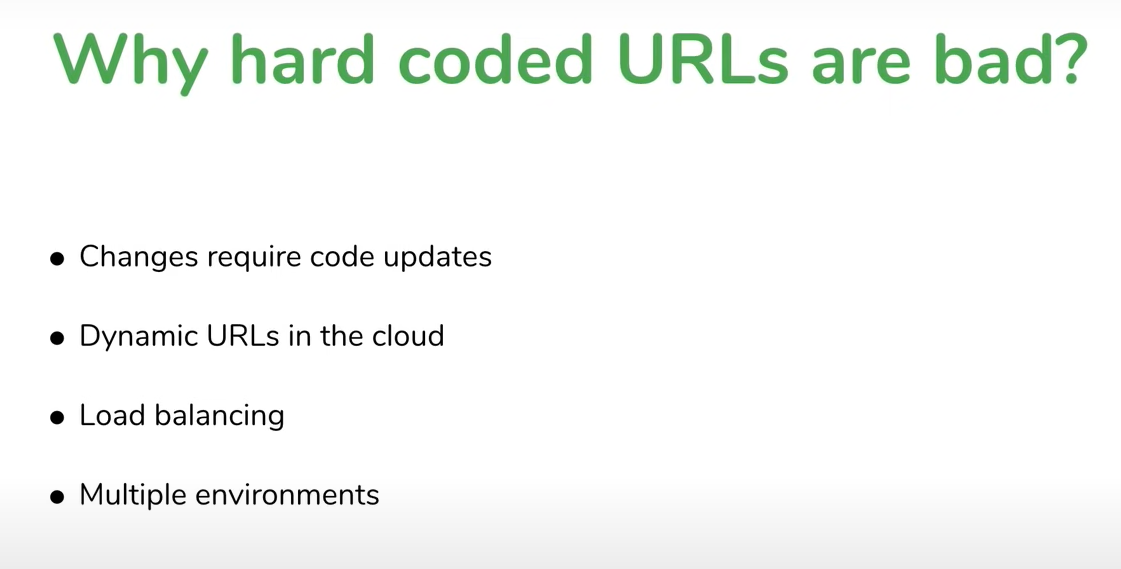
</dependency>

<Refer 13>

## Do not return a list from controller rather wrap it inside object and then return.

<Refer 14,15>

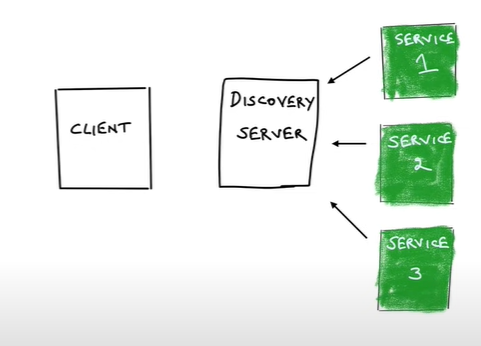
## What wrong are we doing?



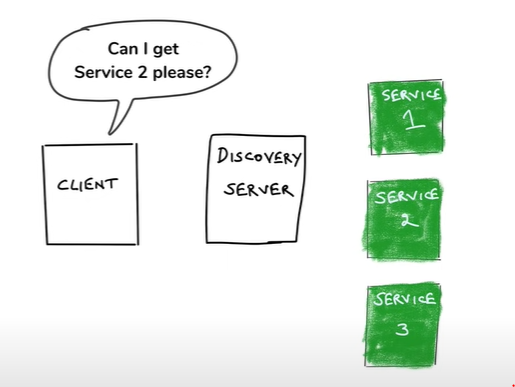
Solution : Discovery service

1. Client side discovery server

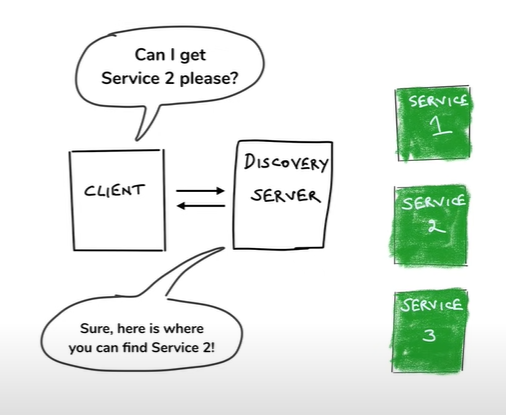
* All the three servers register with discovery server



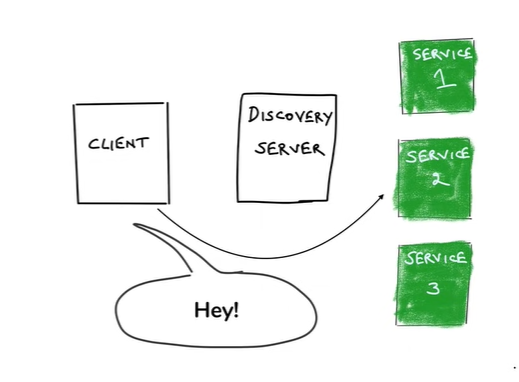
* When client wants service2 it ask to discovery server.



* Discovery server gives address to client back.

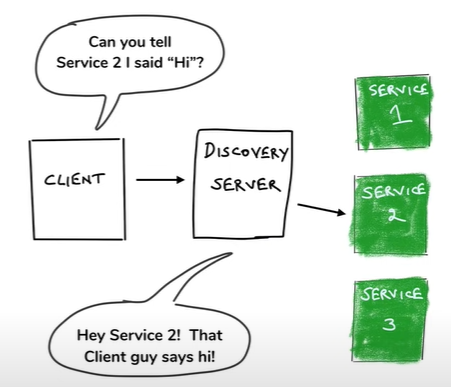


* Then client based on that address accesses service2



Disadvantages: its little more chatty

1. Server side discovery server



Client passes message directly to a medium which it suppose to pass service2. Then that medium itself pass that message to service2. So here unlike client side discovery server there is no extra hop.

**\*\* Spring Cloud uses CLIENT SIDE DISCOVERY SERVICE \*\***

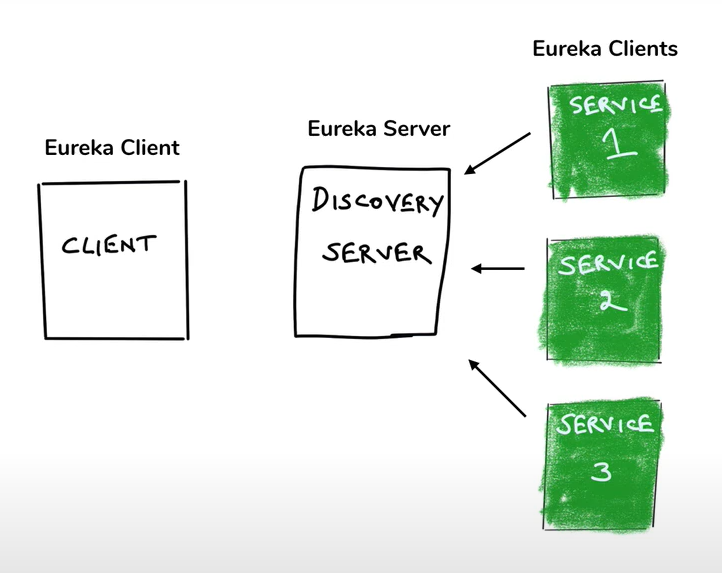
## Technology to implement discovery service

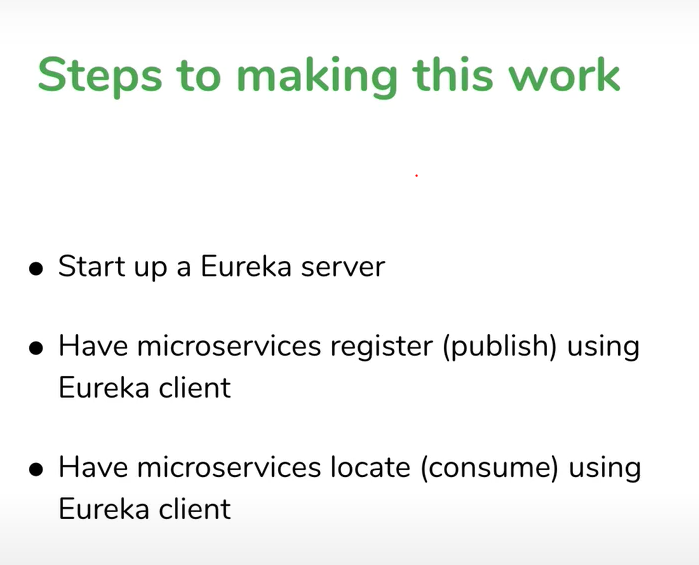
Eureka

Netflix OSS

Eureka is one of projects that made them open source

Like Eureka, Ribbon, Hysterix, Zuul, …….





## Creating discovery server

Create new service with Eureka server as dependency

Add @EnableEurekaServer in main

Registering microservices with discovery server (eureka)

Add dependency

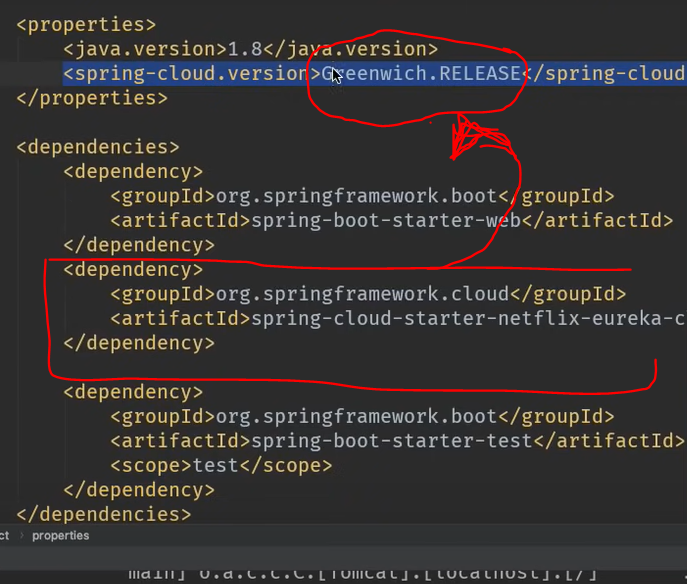
<dependency>

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

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

<version>4.1.1</version>

</dependency>



## To avoid hardcoded url’s

*@SpringBootApplication*

public class MovieCatalogServiceApplication {

*@Bean*

*@LoadBalanced*

public RestTemplate getRestTemplate() {

return new RestTemplate();

}

public static void main(String[] args) {

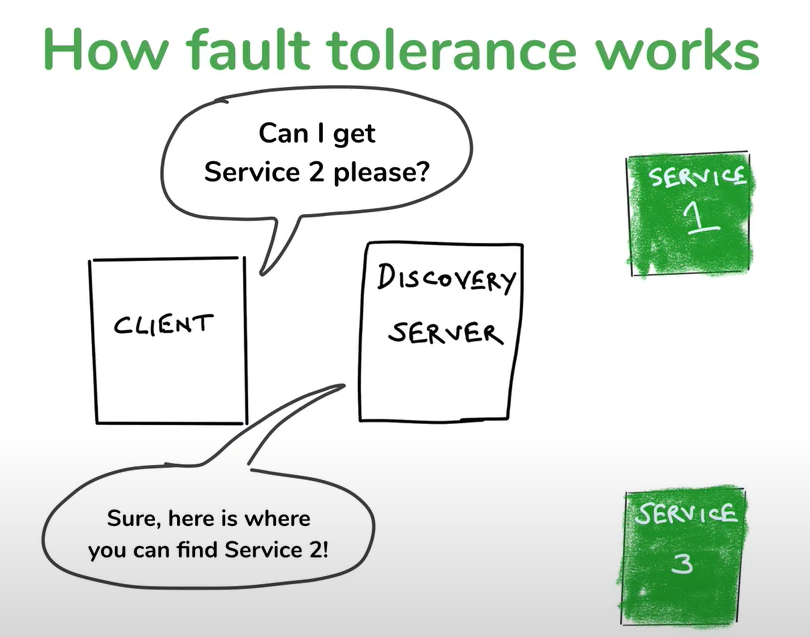
SpringApplication.*run*(MovieCatalogServiceApplication.class, args);

}

}

## How fault tolerance works?

Service 2 is down

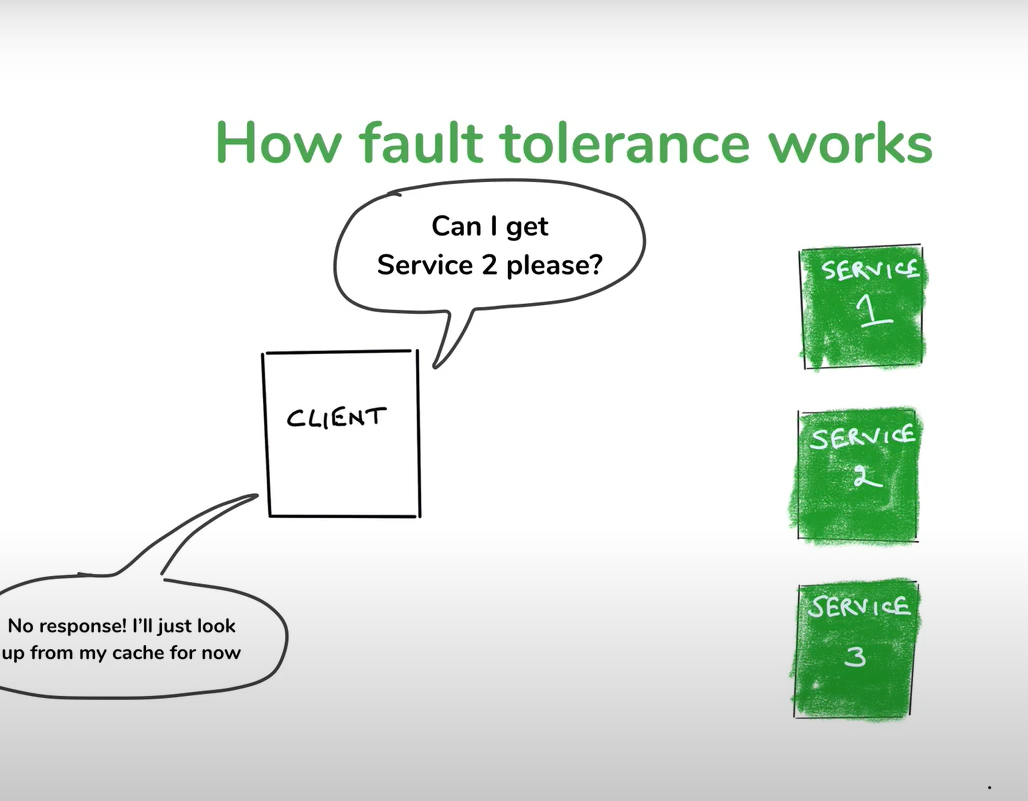


Solution is “Heart Beat” :- Services will ping discovery client after every interval of time that I am alive. So that discovery server knows service 2 is down.

## What if discovery server goes down?

* If client don’t get response from server it goes to cache get response

**\*\* All these things happening in background with just one annotation @LoadBalancer \*\***



## What if microservice goes down?

* Solution – Create multiple instances. So that even if one instance goes down then it goes to another service and system will not down.

## What if microservice is slow?

Much much bigger problem.

Why?

A diagram of a service

Description automatically generated

Here in this case the Movie DB is slow, so because of that movie-info-service is also slow and ultimately the catalog-service. **But after some time we find that rating-data-service is slow though it is not slow in actual**. Why?

A diagram of a service

Description automatically generated

Why?

* Answer : Threads

How thread in Web server?

* When request comes to server web server creates thread for that request.

A diagram of a web server

Description automatically generated

It does its work and it removes the thread from thread pool

A white board with blue writing

Description automatically generated

But what happens if thread is taking longer and request keeps coming. So multiple threads keeps on creating. And the time will come when number of threads exceed the limit. And suppose request for another service comes which is faster, then this request will not execute as limit of thread already exceeded. That’s why because one slow service it might feels that faster service is also getting slow.

* Solution 1

We might give timeout to each service. If suppose time exceeds certain limit for a service then that thread will be terminated.

But this solution is not full proof. Suppose if we timeout for thread as 3 milliseconds. And suppose new thread is getting created after every 1 millisecond. So in this case thread blocking situation may occur.

* Solution 2

Circuit breaker pattern

## What if microservice goes down?

* Solution – Create multiple instances. So that even if one instance goes down then it goes to another service and system will not down.

## What if microservice is slow?

Much much bigger problem.

Why?

A diagram of a service

Description automatically generated

Here in this case the Movie DB is slow, so because of that movie-info-service is also slow and ultimately the catalog-service. **But after some time we find that rating-data-service is slow though it is not slow in actual**. Why?

A diagram of a service

Description automatically generated

Why?

* Answer : Threads

How thread in Web server?

* When request comes to server web server creates thread for that request.

A diagram of a web server

Description automatically generated

It does its work and it removes the thread from thread pool

A white board with blue writing

Description automatically generated

But what happens if thread is taking longer and request keeps coming. So multiple threads keeps on creating. And the time will come when number of threads exceed the limit. And suppose request for another service comes which is faster, then this request will not execute as limit of thread already exceeded. That’s why because one slow service it might feels that faster service is also getting slow.

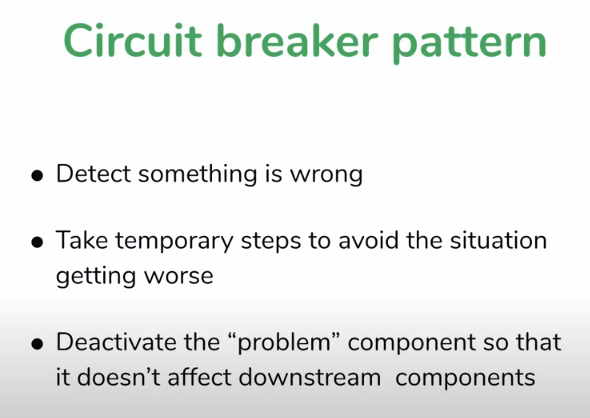
* Solution 1

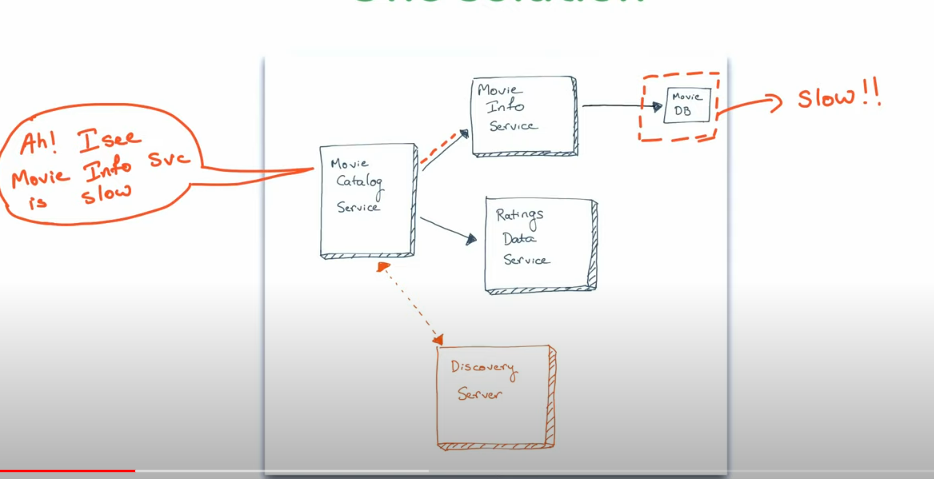
We might give timeout to each service. If suppose time exceeds certain limit for a service then that thread will be terminated.

But this solution is not full proof. Suppose if we timeout for thread as 3 milliseconds. And suppose new thread is getting created after every 1 millisecond. So in this case thread blocking situation may occur.

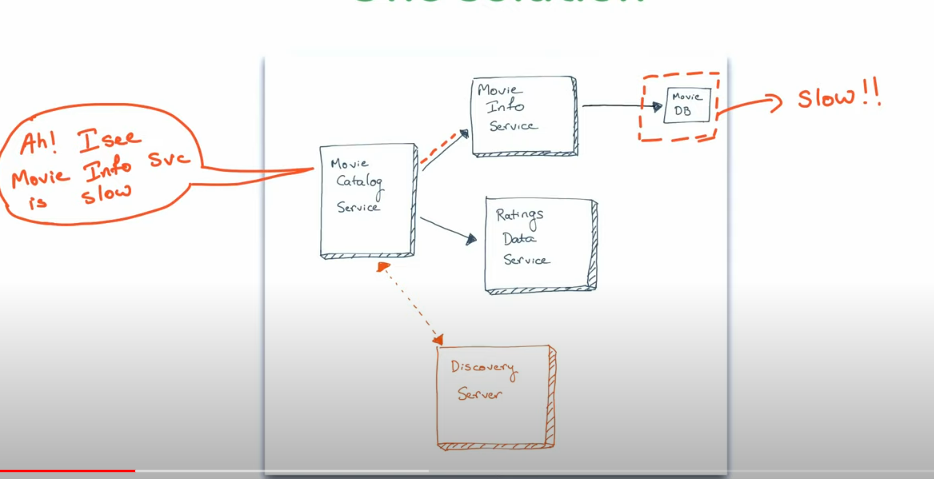
* Solution 2

Circuit breaker pattern



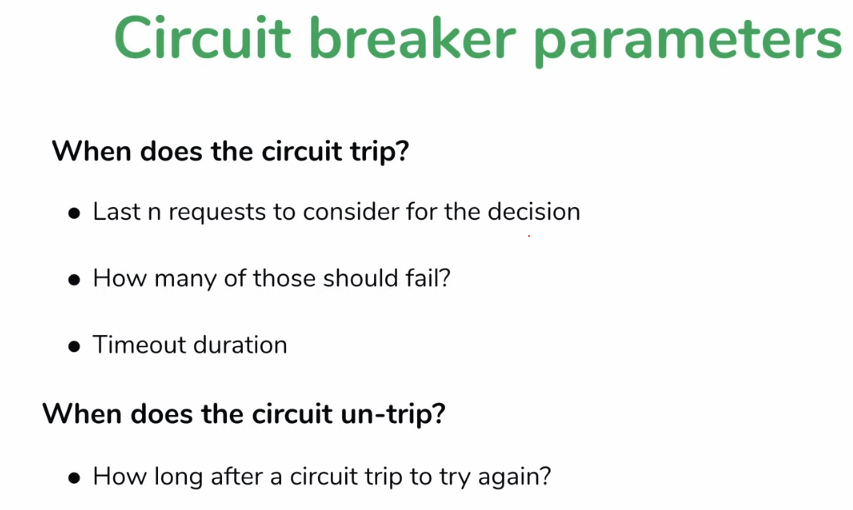


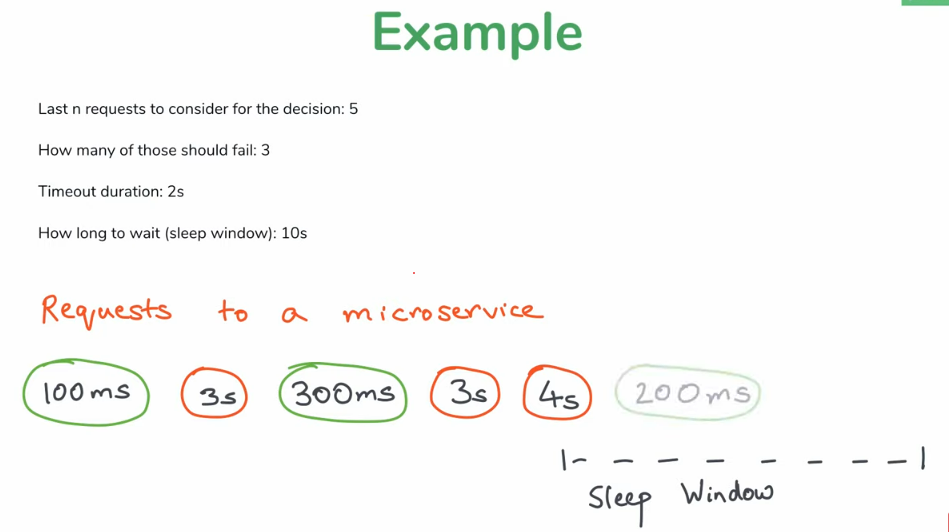
* Here movie-info-service is slow. Which affect the performance.
* As movie-catalog-service calling movie-info-service and rating-service, so this time catalog-service is going to be smarter. It sends request to info-service but if it don’t get response it will sending request to info-service for a while. So that number threads should not exceeds its limit. And whenever catalog-service feels that info-service gives response back and earlier thread get vanished from thread pool it again resumes sending request to info-service.
* So this like breaking circuit if something goes wrong so that whole system wont collapse.



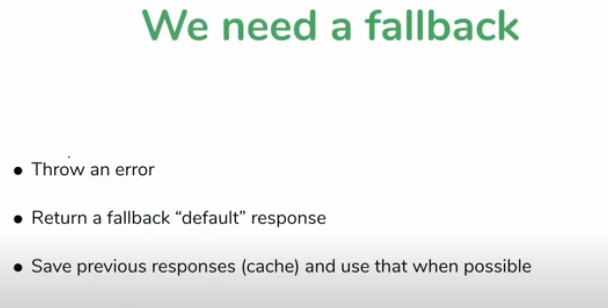
* Where could it be used?
  + If microservice is calling another microservice or multiple microservices, there we can use it.

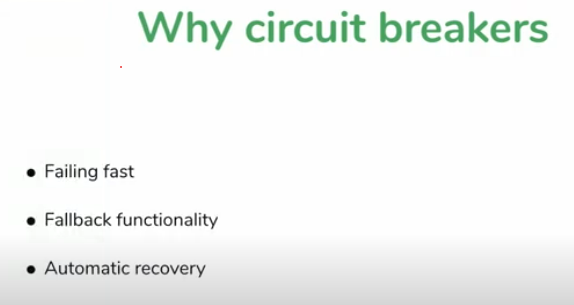
Circuit breaker parameters:





* What if circuit breaks and request keep coming?
  + Solution : Fallback



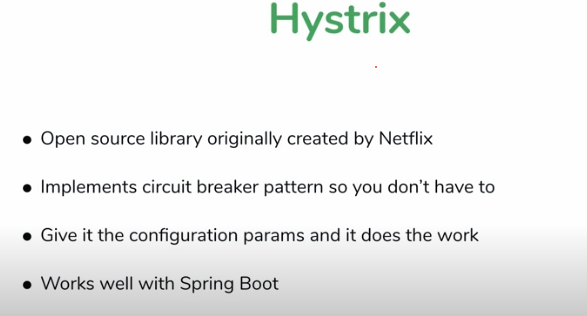


Failing fast : we fail system early if know that it is going to fail anyway

Ex: if already know that request is taking longer to complete then thread pool is going get full. We fail it early.

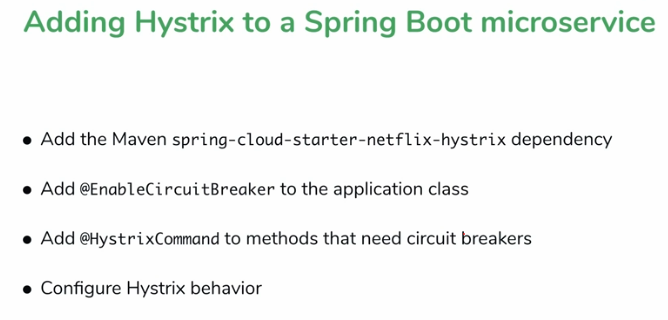
## Hysterix:

Use to implement circuit breaking.



## Note : Hystrix is not in development now, so it will get updated.

Setting up hystrix



* **Resilience4j**
  + <https://resilience4j.readme.io/docs/getting-started>
  + Dependencies
    - Actuator

<dependency>

<groupId>io.github.resilience4j</groupId>

<artifactId>resilience4j-spring-boot2</artifactId>

<version>2.1.0</version>

</dependency>

<dependency>

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

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

</dependency>