**Microservice Docs**

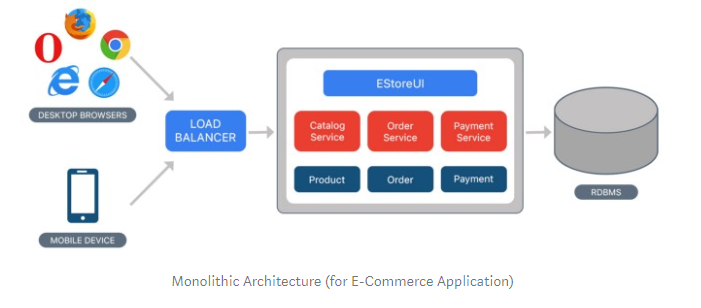
<https://www.dineshonjava.com/microservices-with-spring-boot/>

<https://dzone.com/articles/microservices-part-4-spring-cloud-circuit-breaker>

Monolith means composed all in one piece. The **Monolithic** application describes a single-tiered **software** application in which different components combined into a single program from a single platform. Components can be:

* Authorization — responsible for authorizing a user
* Presentation — responsible for handling HTTP requests and responding with either HTML or JSON/XML (for web services APIs).
* Business logic — the application’s business logic.
* Database layer — data access objects responsible for accessing the database.
* Application integration — integration with other services (e.g. via messaging or REST API). Or integration with any other Data sources.
* Notification module — responsible for sending email notifications whenever needed.

**Example for Monolithic**



**Microservices Architecture**

 It coined in 2005 by Dr Peter Rodgers then called micro web services

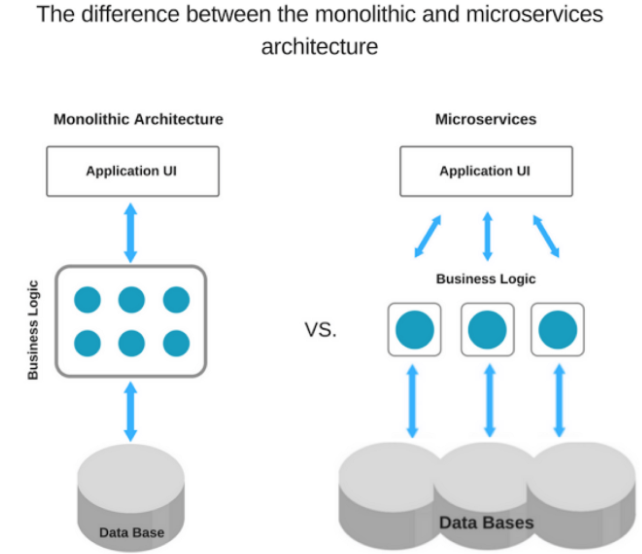
**Microservices** are an approach to application development in which a large application is built as a suite of modular services (i.e. loosely coupled modules/components). Each module supports a specific business goal and uses a simple, well-defined interface to communicate with other sets of services.

Instead of sharing a single database as in Monolithic application, each microservice has its own database. Having a database per service is essential if you want to benefit from microservices, because it ensures **loose coupling**. Each of the services has its own database. Moreover, a service can use a type of database that is best suited to its needs.

Consider the same example of the e-commerce application, which consists of several components/modules. Define each component/module as a separate **loosely coupled** service depending on the requirement, which may collaborate with each other based on the scenario. We can have following services for a complete application:

* Authorization Service — Responsible for authorizing customer.
* Order Service — takes an order and process it.
* Catalog Service — Manage products and check products inventory.
* Cart Service — Manage user cart, this service can utilize Catalog service as a data source.
* Payment Service — Manage and Authorize payments.
* Shipping Service — Ships ordered products.





## 2. Microservices Benefits

* The smaller code base is easy to maintain.
* Easy to scale as an individual component.
* Technology diversity i.e. we can mix libraries, databases, frameworks etc.
* Fault isolation i.e. a process failure should not bring the whole system down.
* Better support for smaller and parallel team.
* Independent deployment
* Deployment time reduce

## 3. Microservices Challenges

* Difficult to achieve strong consistency across services
* ACID transactions do not span multiple processes.
* Distributed System so hard to debug and trace the issues
* Greater need for an end to end testing
* Required cultural changes in across teams like Dev and Ops working together even in the same team.

**Service Discovery:**

Service Discovery work as a encharge or as a mediator to communicate between Client and the micro services all service have register in the discovery server

Have two types:

**Client side discovery server:**

When client request for any service to discovery server then discovery server searched in registered service and say yes it is there and make request to the correct service.

**Server Side Discovery server:**

When client says to discovery server can you tell service 2 I said hii then discovery server this request too service 2 .

**Note : Spring cloud uses the client side service discovery.**

**Technology to Implement the discovery server is:**

Eureka(its is because of netfix which make it open source it contain so many libraries.

Eureka

Ribbon

Zuul

Hysterixes

**Steps to making this work:**

Start eurka server

Have microservice register(publish) using eurka client

Have microservice locate consume using eurka .

## What Is the Eureka Server?

The Eureka server is nothing but a service discovery pattern implementation, where every microservice is registered and a client microservice looks up the Eureka server to get a dependent microservice to get the job done.

The Eureka Server is a Netflix OSS product, and Spring Cloud offers a declarative way to register and invoke services by Java annotation.

* Service Discovery (How do services find each other?)
* Client-side Load Balancing (How do we decide which service instance to use?)

**Practical how to create eurka server**

1.create springboot app bt starter.io

2.add eureka server dependency

**<dependency>**

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

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

**</dependency>**

**3.in the application directory add follows:**

**#beacause eurka server have multiple insatnces so that we are saying to eurka server stop to work as a client u are only the boss to wrk here as a eureka server dont try to find another eurak server.**

**server.port=5004**

**eureka.client.register-with-eureka=false**

**eureka.client.fetch-registry=false**

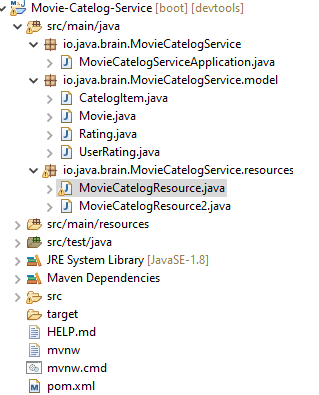
|  |  |
| --- | --- |
| Name | Description |
| spring.application.name | Unique name for a Eureka server service. |
| eureka.client.serviceUrl.defaultZone | It consults with other Eureka servers to sync the service registry. As it is in standalone mode, I am giving the local server address. |
| server.port | In which port the server will be bound. |
| eureka.client.register-with-eureka | This determines if this server registers itself as a client; as I said earlier, the Eureka server is also acting as a client so that it can sync the registry. The value being false means it prevents itself from acting as a client. |
| eureka.client.fetch-registry | Does not register itself in the service registry. |

**Steps to create microservice:**

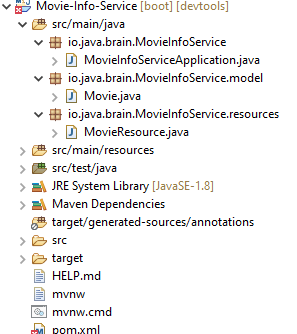
**Movie info project from java brain:**

1. **1.create spring boot application**
2. **Movie-catelog-services**

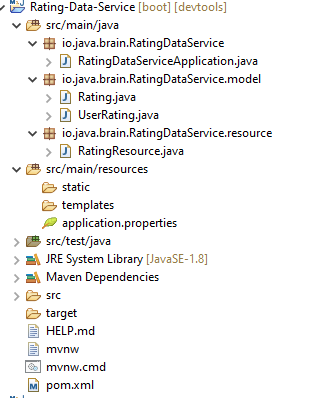
**Directory structure**



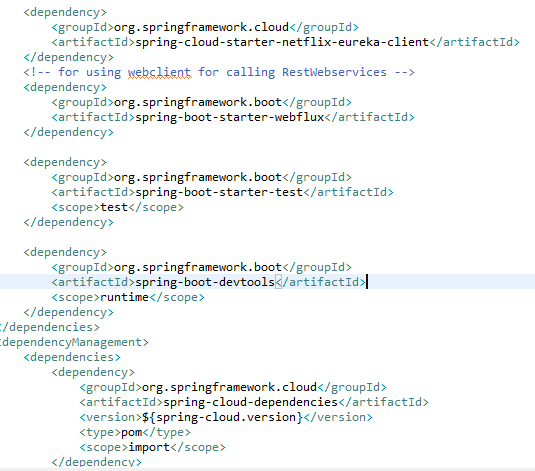
1. **Movie info services**



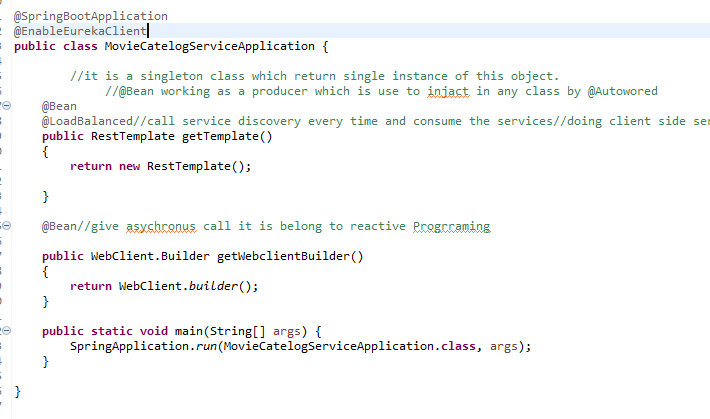
1. **Rating data service**



**In above three services dependency will be**



**In main app class configuration are as follows**

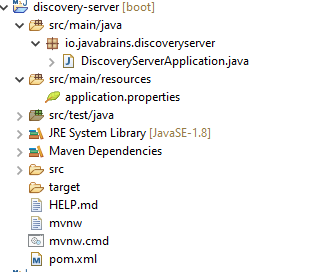


**Resource class is as follows:**

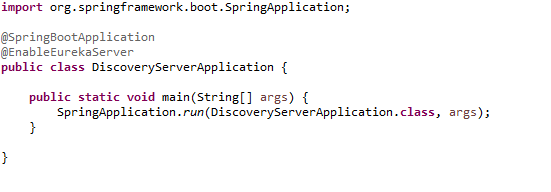


**5.create a discovery server:**

**For integrating client microservise .**

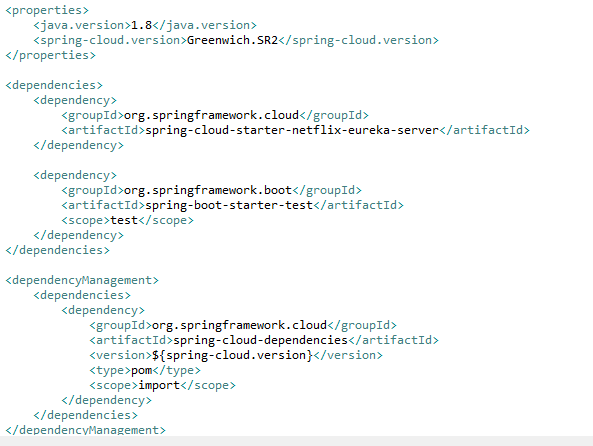


Main App class :



Note : @EnableEurekaServer to enable edureka server

pom.xml



Note :

**Put default port for eureka server: 8761**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Fault tolerance:**

If any microsevices have a fault then what is the impact on the particular application?

Or if any microservice goes down what is the impact of that service on other services.

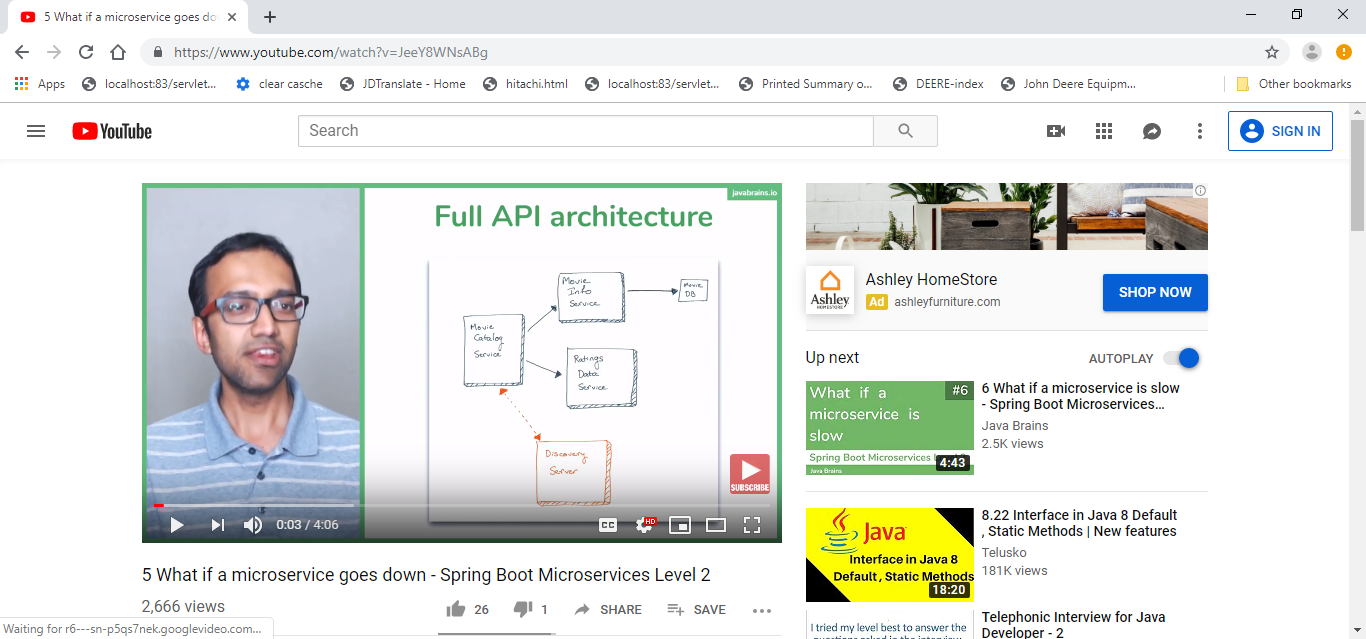
Because of this full system goes down.

**Resilence:**

Means how many faults a system can tolerate that is call resillence capacity.

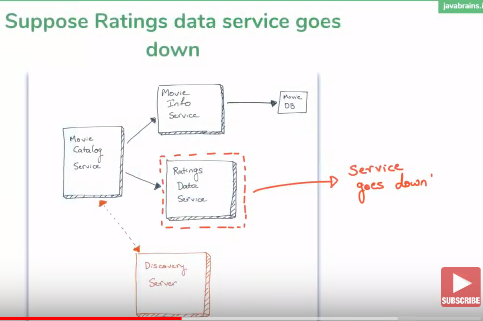
It is recoverable by itself.

To get dynamic data for movie information we have to make movie db account api and access token from this account



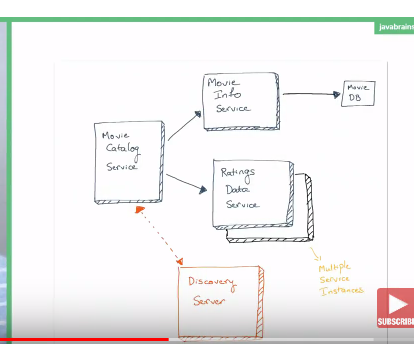
Note :How do make resilient means if any service goes down how to make the other service not to be affect

1. If one microservice goes down



Solution:

To run multiple instances for the same instance.

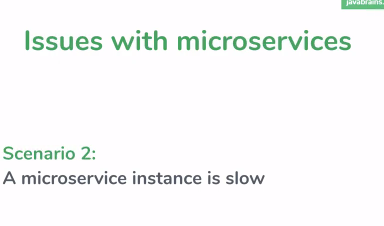


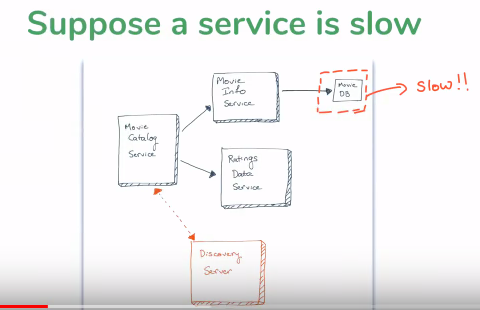
Client side load balancing is performed by :

Ribbon technology

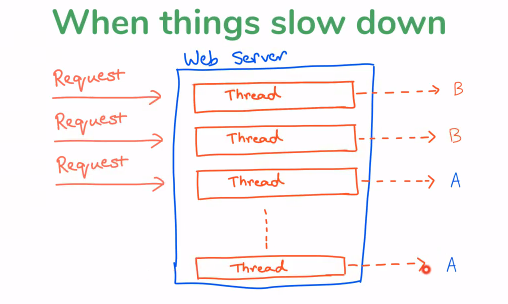
If one microservice have more than one instances that is call Roundrobbin

2.





Because one request goes on the server and make thread it is not responding for A thread so that’s why service B thread is also goes slow down



**Solution :**

1.timeout partal solution for solving this problem.for that we have to set time as follows with calling external api by resttemplate.

**public RestTemplate getTemplate()**

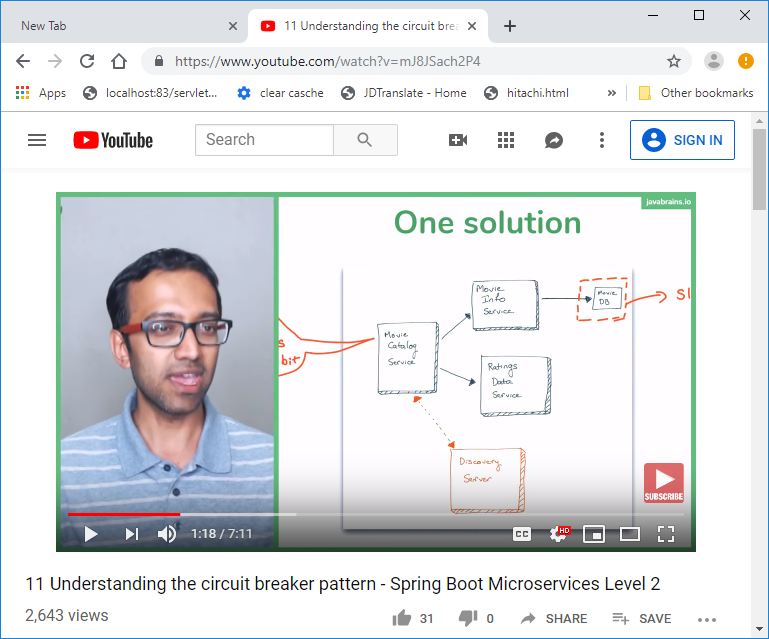
**{**

**HttpComponentsClientHttpRequestFactory clienthttpfactory = new HttpComponentsClientHttpRequestFactory();**

**clienthttpfactory.setConnectTimeout(3000);**

**return new RestTemplate(clienthttpfactory);**

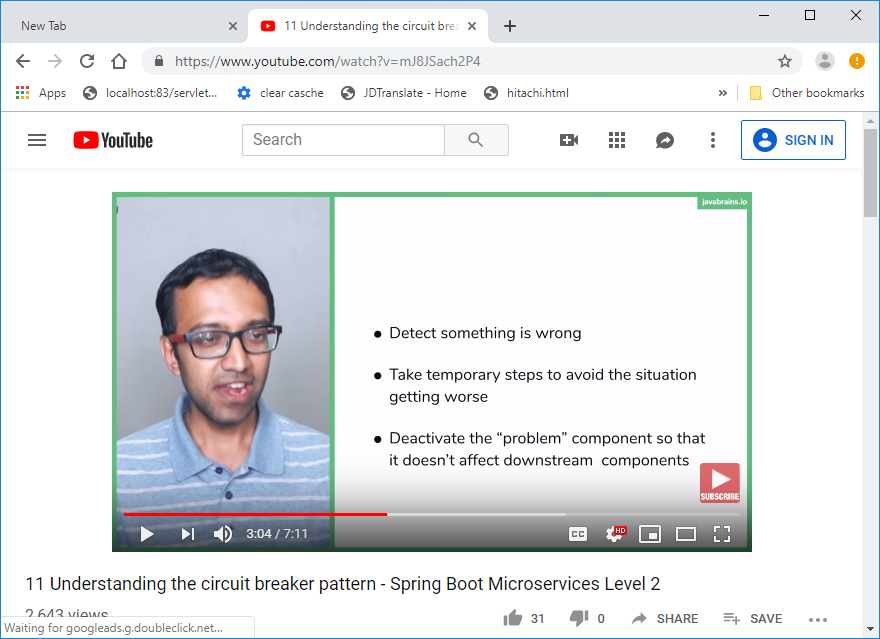
}

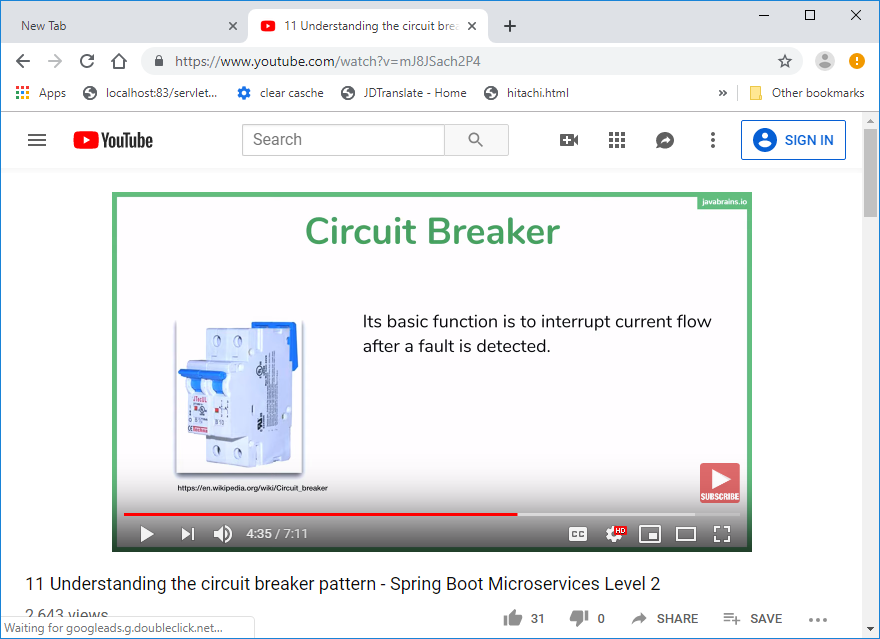


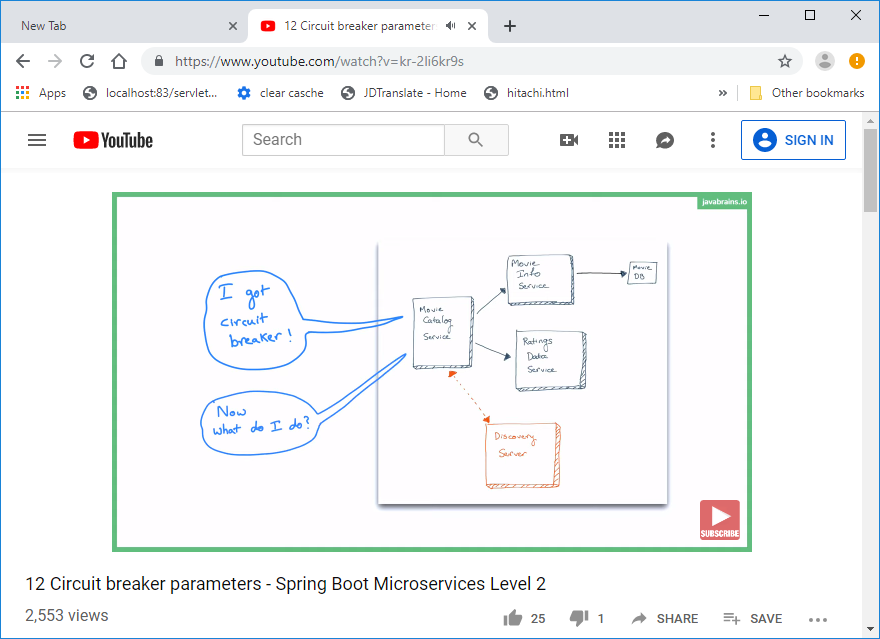
Absolute solution for this problem:

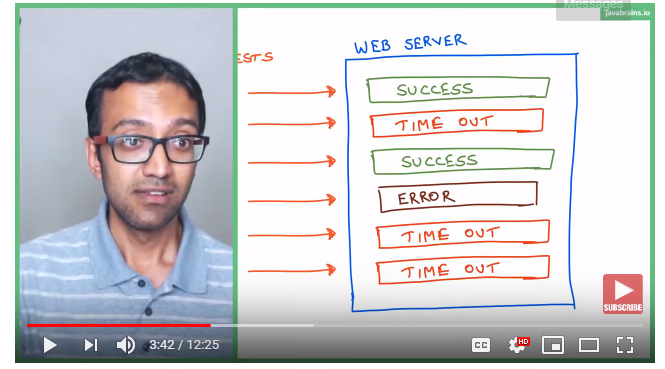
1.Detect the something is wrong

2.get temporary solution to syetem getting worse.









**Circuit breaker parameter:**

What does the circit trip?

1.last n requests to consider for the desion.

2.How many of those should fail.

3.Timeout duration.

Whats does the circuit are untrip?

**We need a fallback:**

1.throw an error//not recommended

2.Return a fall back default response.

2.save previous response in (cache)and use that when possible.

**Why circuit breaker?**

**Falling fast**

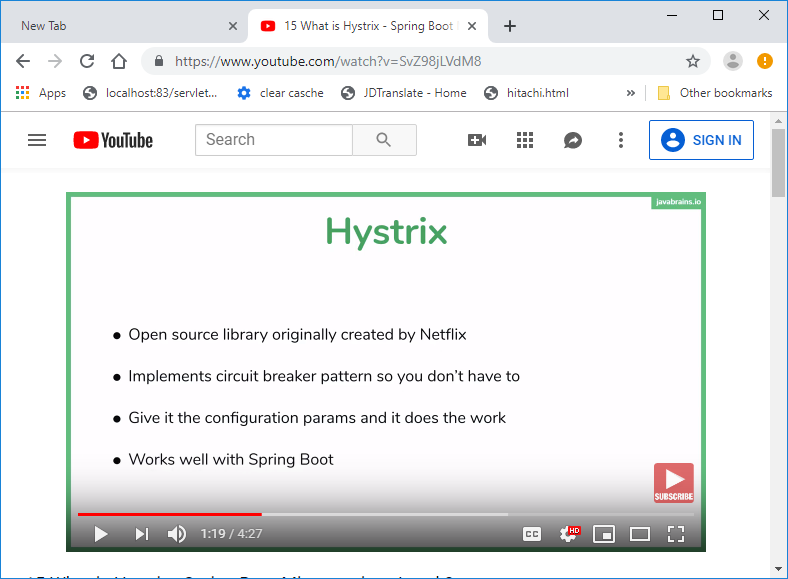
**Falling back functionality**

**Automatic recovery**

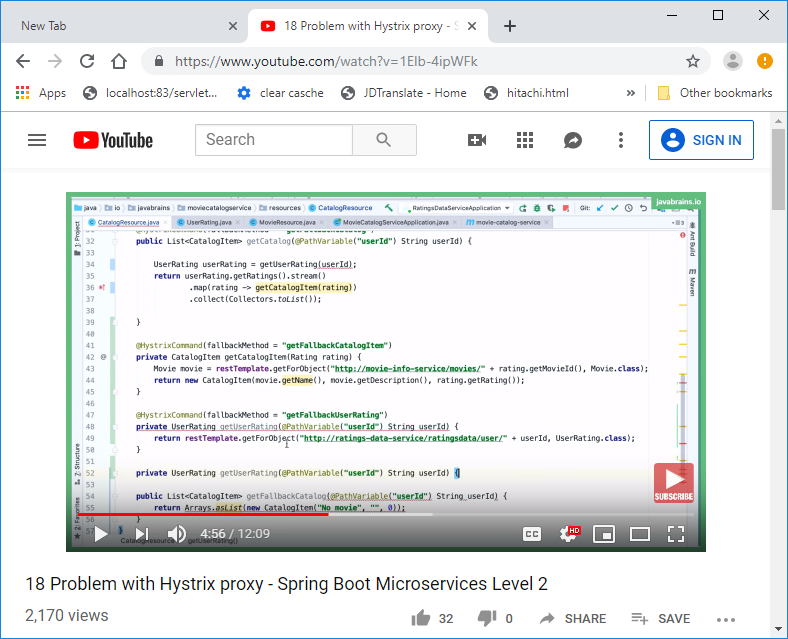
**This is a lot of work**

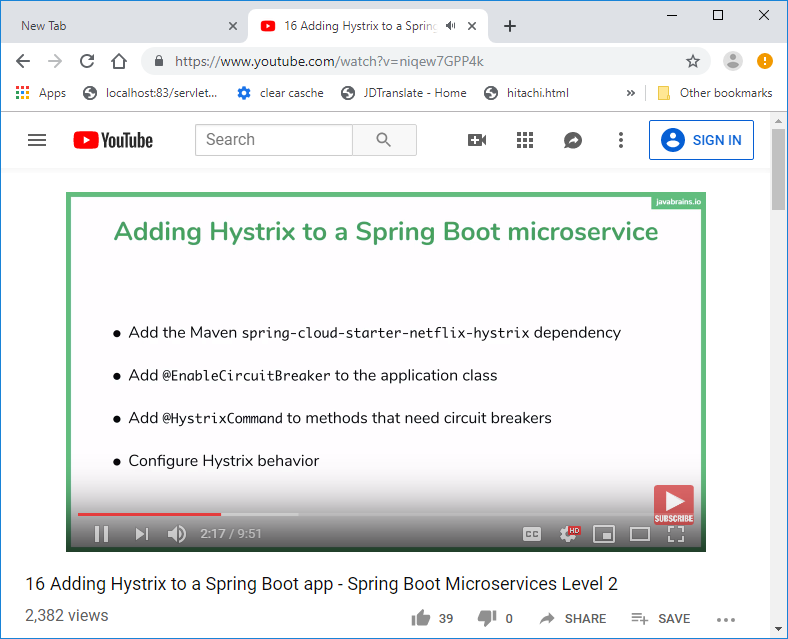
**Hystrix**

**Hystrix**



Adding Hystrix:





In the microservices world, to fulfill a client request, one microservice may need to talk to other microservices. We should minimize this kind of direct dependencies on other microservices, but in some cases, it is unavoidable. If a microservice is down or not functioning properly then the issue may cascade up to the upstream services. Netflix created Hystrix library implementing the [Circuit Breaker pattern](https://martinfowler.com/bliki/CircuitBreaker.html) to address these kinds of issues. We can use **Spring Cloud Netflix Hystrix Circuit Breaker** to protect microservices from cascading failures.

**In this post, we are going to learn:**

* Implementing Circuit Breaker pattern using **@HystrixCommand**
* How to propagate **ThreadLocal** variables
* Monitoring Circuit Breakers using **Hystrix Dashboard**

From **catalog-service,** we are invoking a REST endpoint on **inventory-service** to get the inventory level of a product. What if inventory-service is down? What if inventory-service is taking too long to respond, thereby slowing down all the services depending on it? We would like to have some **timeout**s and implement a **fallback** mechanism.

Add the **Hystrix** starter to catalog-service.

<dependency>

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

<artifactId>spring-cloud-starter-netflix-hystrix</artifactId>

</dependency>