**Microservices**

**Normal API Communication**

1. Movie Catalog Service



1. Movie Info Service
2. Ratings Data Service

Reference:

<https://www.youtube.com/watch?v=bTFIduBgXuo&list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas&index=8>

Youtube Tutorial:

<https://www.youtube.com/watch?v=y8IQb4ofjDo&list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas>

**Service Discovery**

<https://www.youtube.com/watch?v=e09P-CkCvvs&list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas&index=17>

Client Side Service Discovery (**SPRING CLOUD USE THIS MODEL**)





Server Side Service Discovery



**Spring Cloud -> Eureka from Netflix**

https://www.youtube.com/watch?v=GTM2J0nYmbs&list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas&index=19



Note:

Eureka Server will error on Java 11

Fix it by adding jaxb dependency on pom file: <https://github.com/norricorp/sb_eureka_java11/blob/master/pom.xml>

Make sure server port is not used:

<https://stackoverflow.com/questions/49789925/com-sun-jersey-api-client-clienthandlerexception-java-net-connectexception-con>



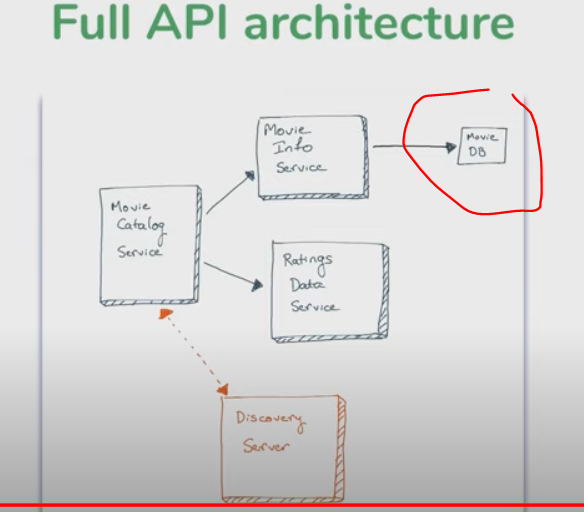
Eureka Client dependency error:

Specify version and add dependency Management

<https://stackoverflow.com/questions/63693570/cannot-resolve-org-springframework-cloudspring-cloud-starter-netflix-eureka-cli>

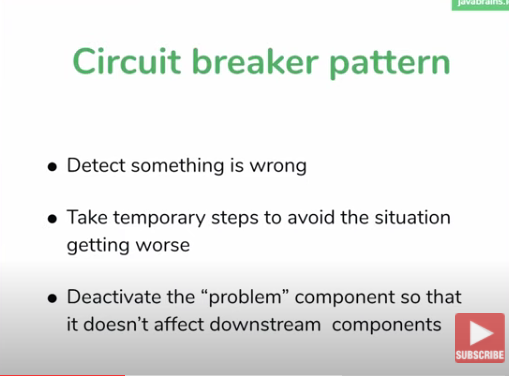
**Calling External API (movieDB)**

<https://www.youtube.com/watch?v=7nKKD2rKpUk&list=PLqq-6Pq4lTTbXZY_elyGv7IkKrfkSrX5e&index=4>



**Making microservice tolerant and resistant**

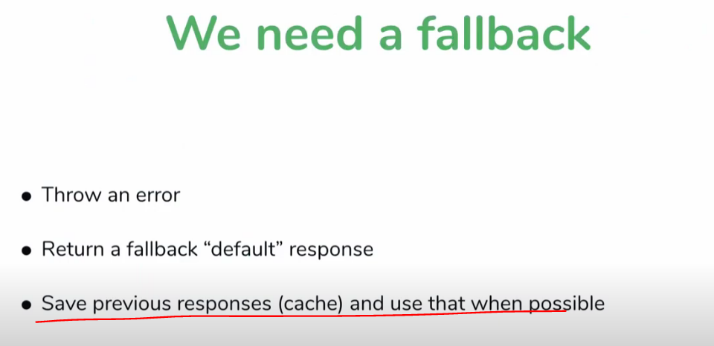
Common Solutions when one of the microservice requests is slow

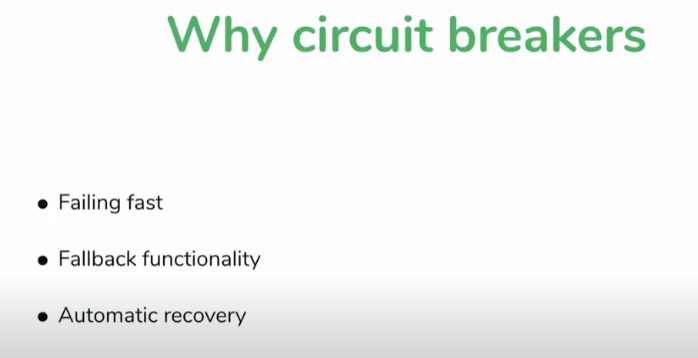
1. Timeout
2. Circuit Breaker Pattern

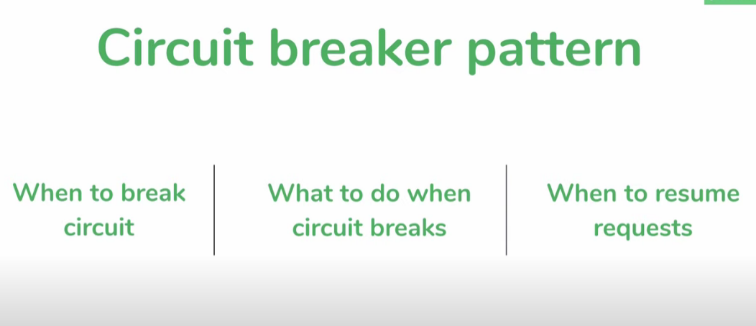
<https://www.youtube.com/watch?v=mJ8JSach2P4&list=PLqq-6Pq4lTTbXZY_elyGv7IkKrfkSrX5e&index=11>

Ideally should be placed on the main caller, not on every microservice.

There should be a fallback mechanism.





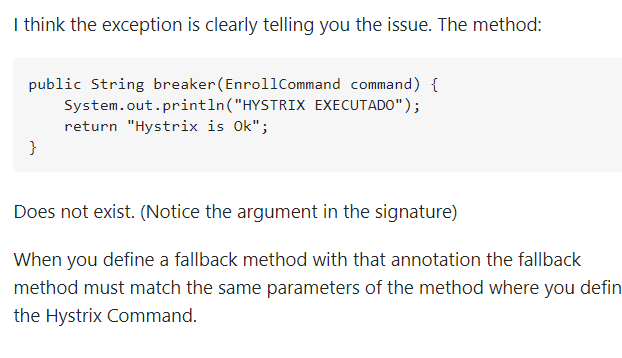
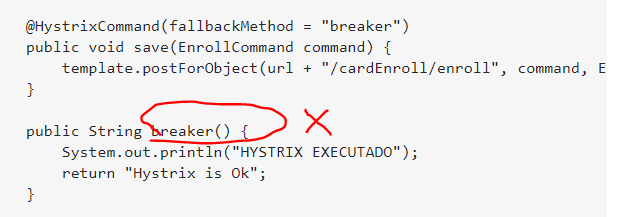


**Hystrix from Netflix**

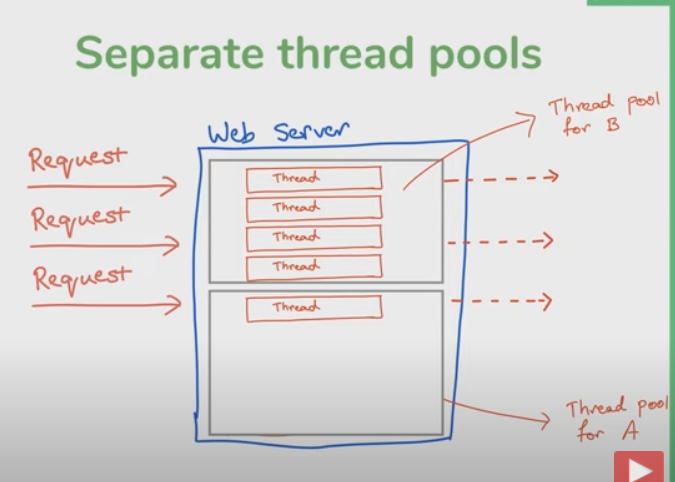
* Implements Circuit Breaker Pattern
* Works well with Spring boot

Hystrix fallback method not found

* When you define a fallback method with that annotation the fallback method must match the same parameters of the method where you define the Hystrix Command.
* <https://stackoverflow.com/questions/41978208/spring-cloud-hystrix-fallback-method-wasnt-found>



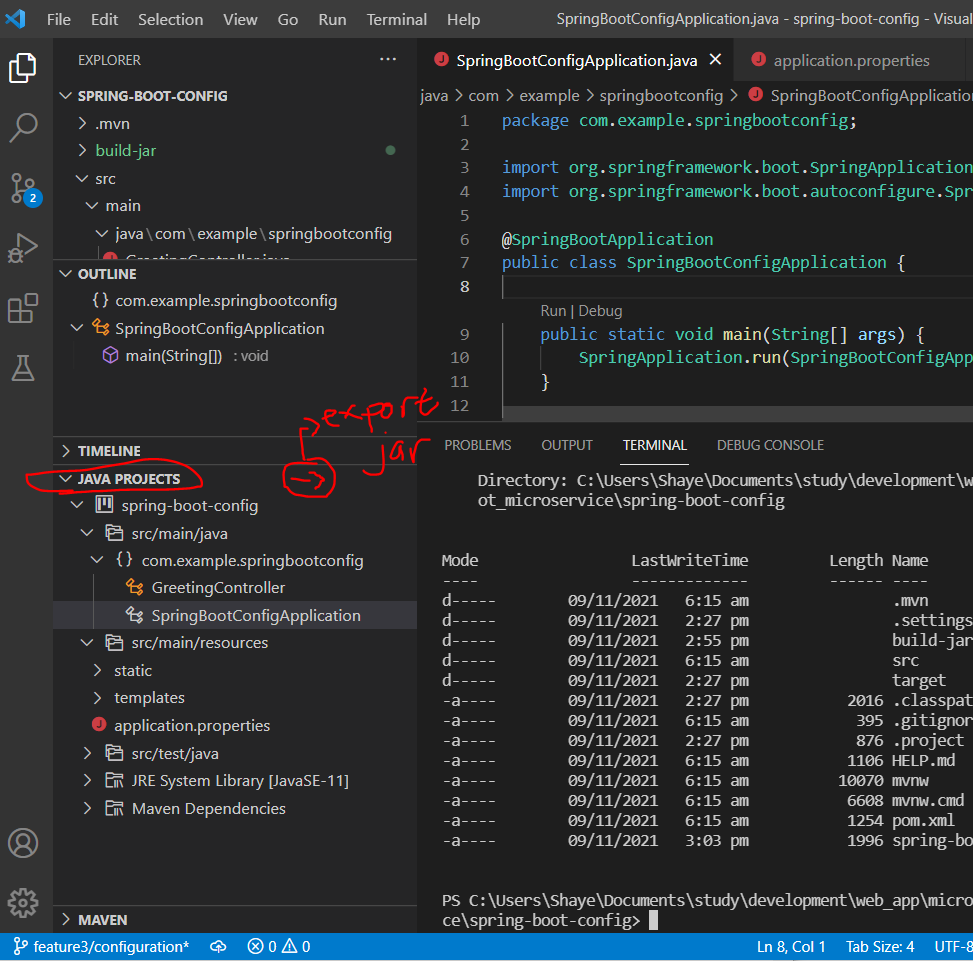
1. Bulkhead pattern



**Microservice Configuration**

\*Export jar file in VS code. (On normal build, jar file is not created.

https://code.visualstudio.com/docs/java/java-project



Manual build jar project:

java -jar .\spring-boot-config.jar

If using spring maven, I think need to use maven tool to build jar package

./mvnw clean package spring-boot:repackage

Output in target/

java -jar .\spring-boot-config.001.SNAPSHOT

Note: To use external properties, it should be same location with jar

\*Important Annotations

@Value

@ConfigurationValue

\*Actuators

* To expose your properties to others
* <http://localhost:8080/actuator/configprops>

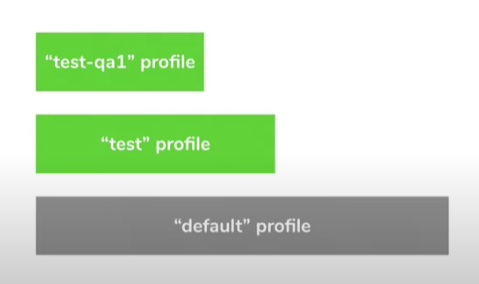
\*YAML File format

- supports iterative pattern

\* Spring profiles

Environment Sensitive Configuration

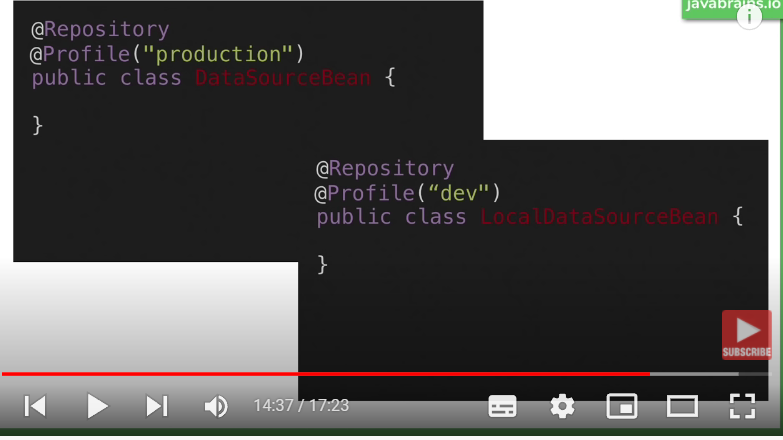
Syntax:



Switching profile command:

java -jar .\spring-boot-config.001.SNAPSHOT --spring.profiles.active=test

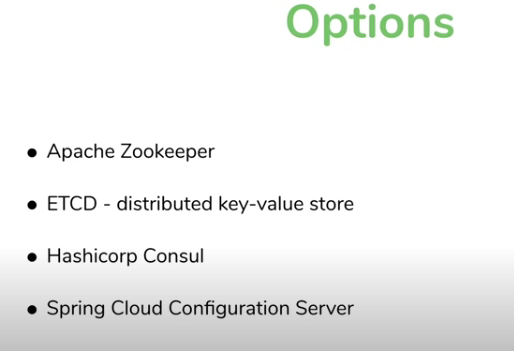
\*Execute bean by profile



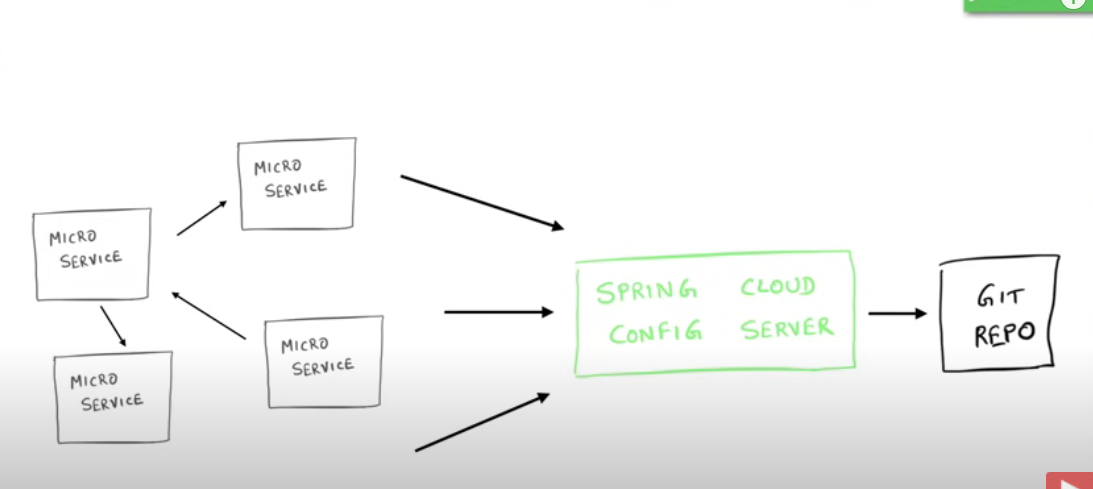
\*Get profile/env information in code using Environment object

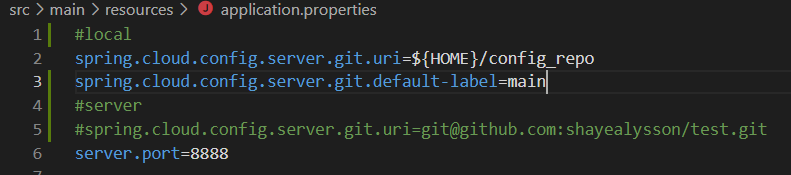
**Config Server**

Config as a microservice



\*Spring Cloud Config Server

<https://www.youtube.com/watch?v=gb1i4WyWNK4&list=PLqq-6Pq4lTTaoaVoQVfRJPqvNTCjcTvJB&index=11>

If git source is local, configure config server:

<https://stackoverflow.com/questions/40024161/spring-config-server-no-such-label-master>

Access:

http://localhost:8888/<file-name>/<profile>

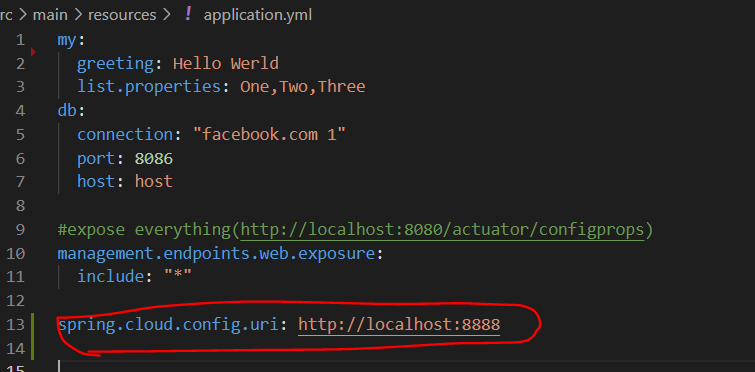
Sample:

<http://localhost:8888/application/default>

Source:

\*Spring Cloud Config Client

<https://www.youtube.com/watch?v=E2HkL766VHs&list=PLqq-6Pq4lTTaoaVoQVfRJPqvNTCjcTvJB&index=12>

Client should know the location of Config server in properties