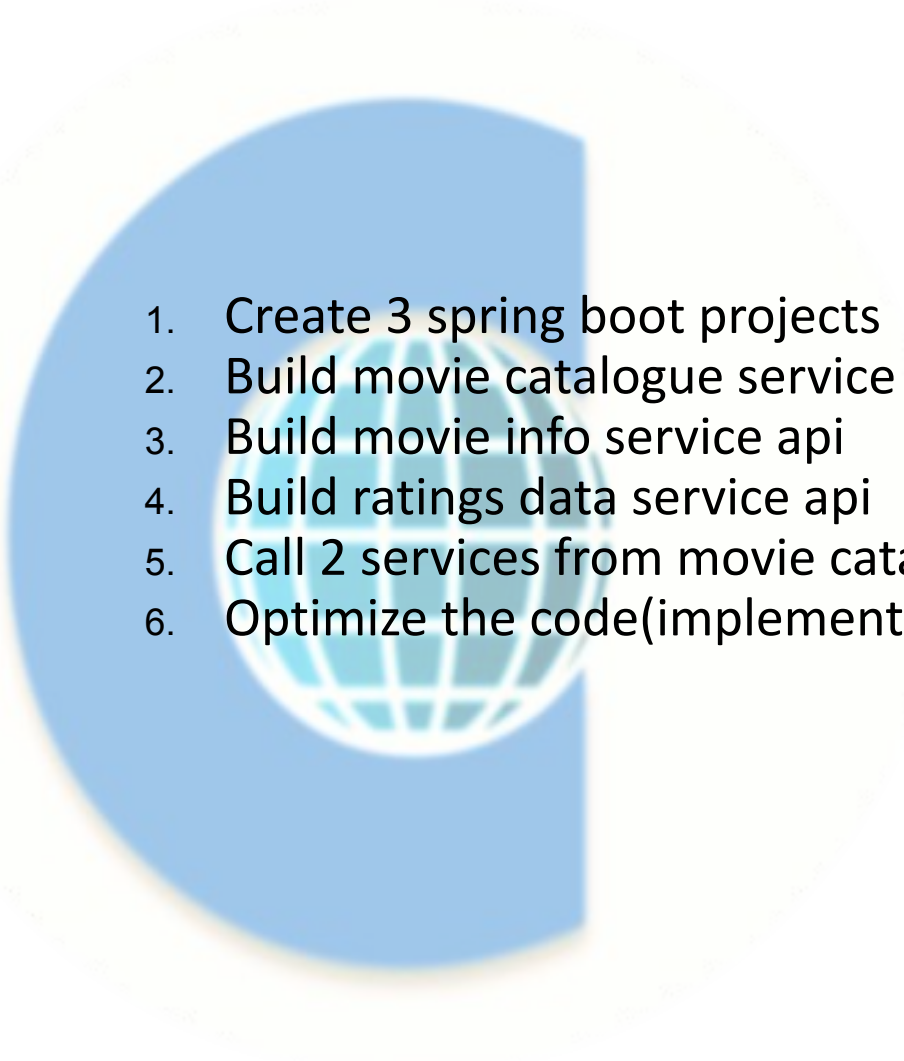




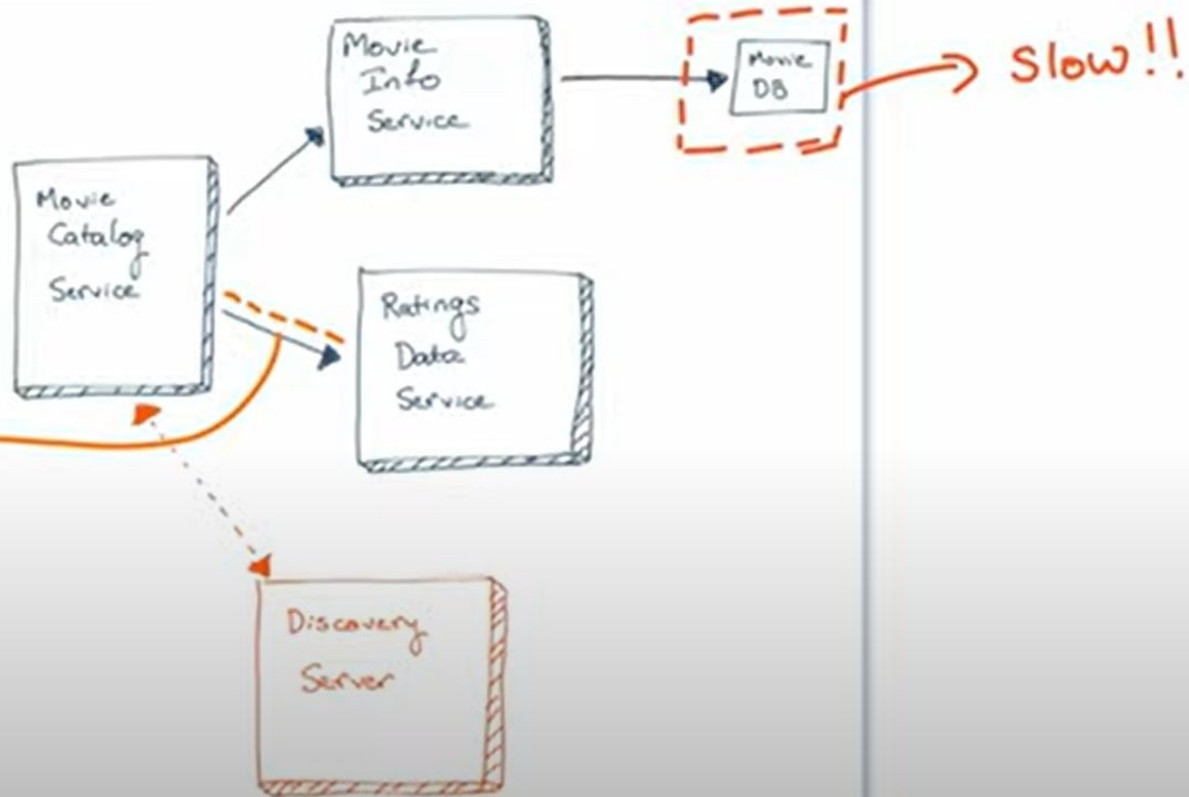
microservices



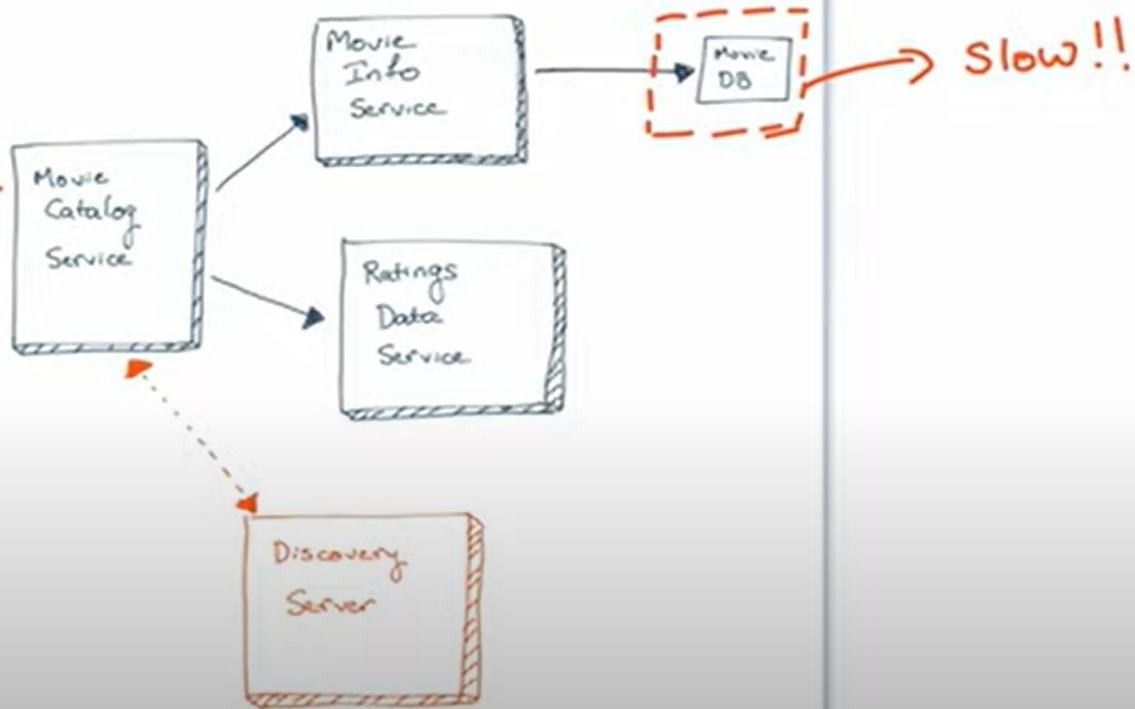
<https://www.baeldung.com/resilience4j>

- 
1. Create 3 spring boot projects
  2. Build movie catalogue service api
  3. Build movie info service api
  4. Build ratings data service api
  5. Call 2 services from movie catalogue service
  6. Optimize the code(implement in better way )

Why is  
this slow?



Let me not  
send requests  
to it for a bit



# CIRCUIT BREAKER

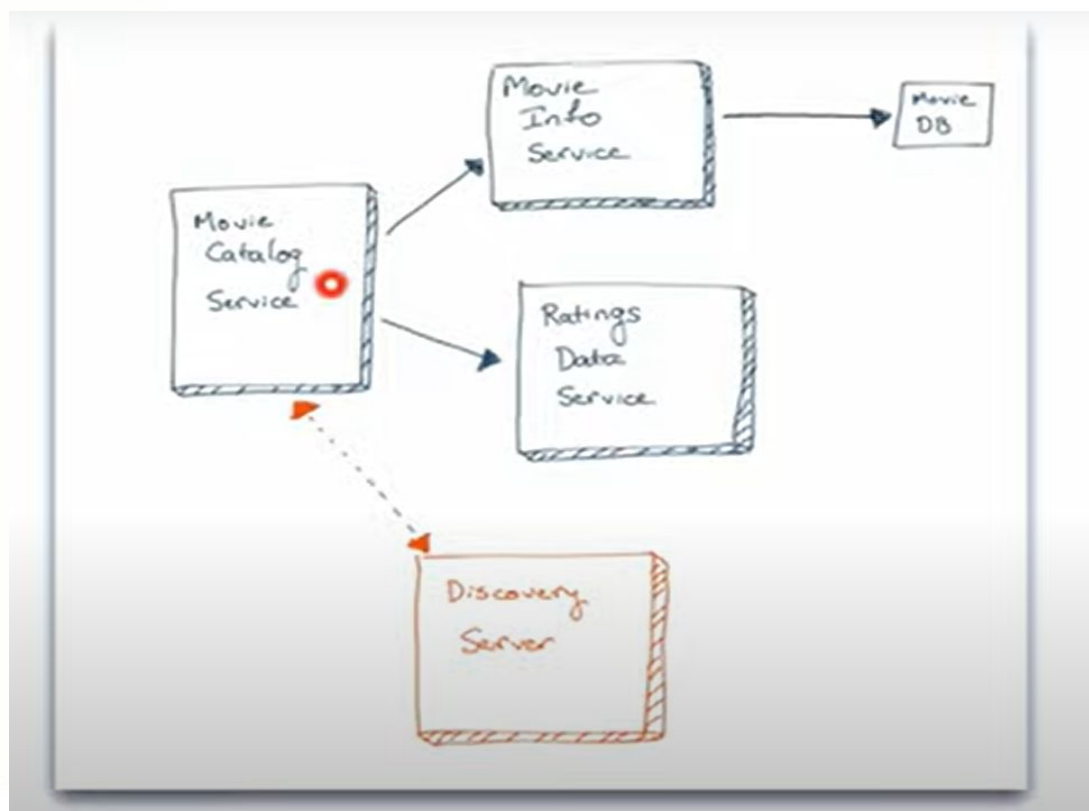


Its basic function is to interrupt current flow after a fault is detected.

Unlike a fuse, which operates once and then must be replaced, a circuit breaker can be reset (either manually or automatically) to resume normal operation.

# CIRCUIT BREAKER PATTERN

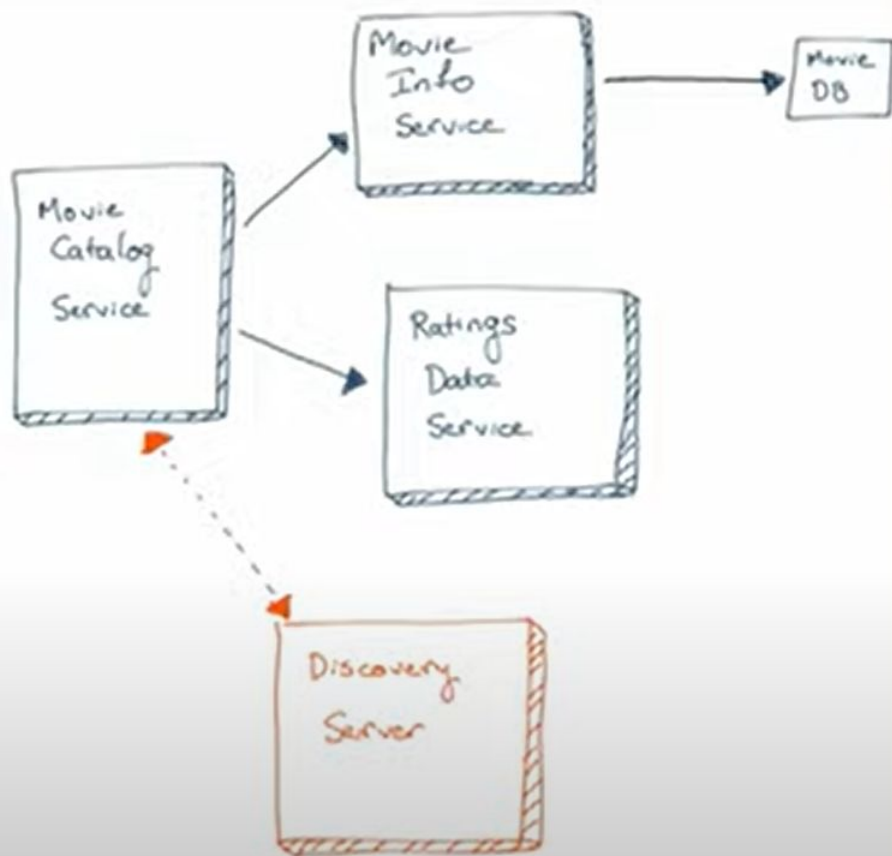
- Detect something is wrong
- Take temporary steps to avoid the situation getting worse
- Deactivate the “problem” component so that it doesn’t affect downstream components





I got  
circuit  
breaker!

Now  
what do I do?



REQUESTS



WEB SERVER



REQUESTS



WEB SERVER

SUCCESS

TIME OUT

SUCCESS

ERROR

TIME OUT

TIME OUT

# CIRCUIT BREAKER PARAMETERS

## When does the circuit trip?

- Last  $n$  requests to consider for the decision
- How many of those should fail?
- Timeout duration

# WHEN DOES CIRCUIT COMES BACK (UNTRIP)

How long after a circuit trip to try again?



example:

Last n requests to consider for the decision: 5

How many of those should fail: 3

Timeout duration: 2s

How long to wait (sleep window): 10s

Requests to a microservice

100ms

3s

300ms

3s

4s

## Requests to a microservice



| - - - - - |  
Sleep Window

<https://github.com/Netflix/Hystrix/wiki/configuration>


# HOW TO DECIDE NUMBER?

IT'S TRICKY

HIT AND TRIAL

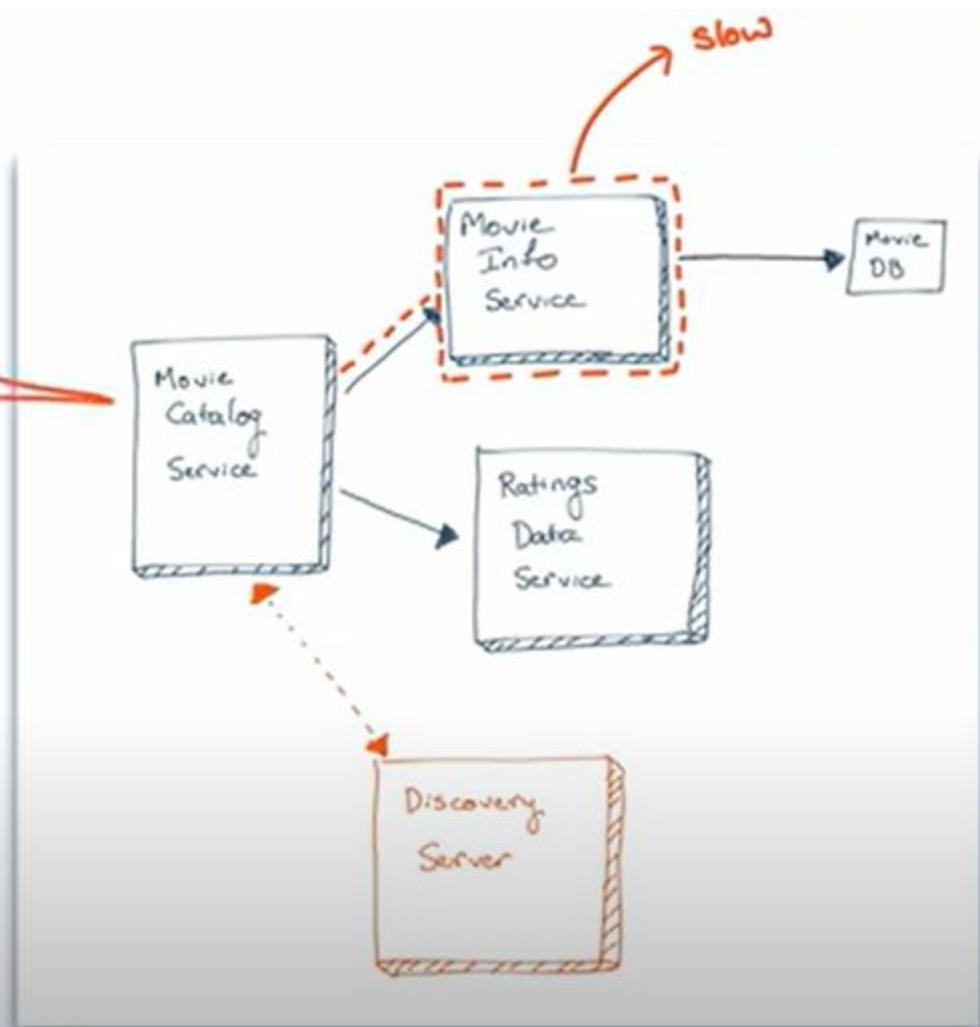
DEPENDS ON MAINLY 2 -> HOW MANY REQUESTS COMING IN  
HOW BIG YOU'RE THREAD POOL IS





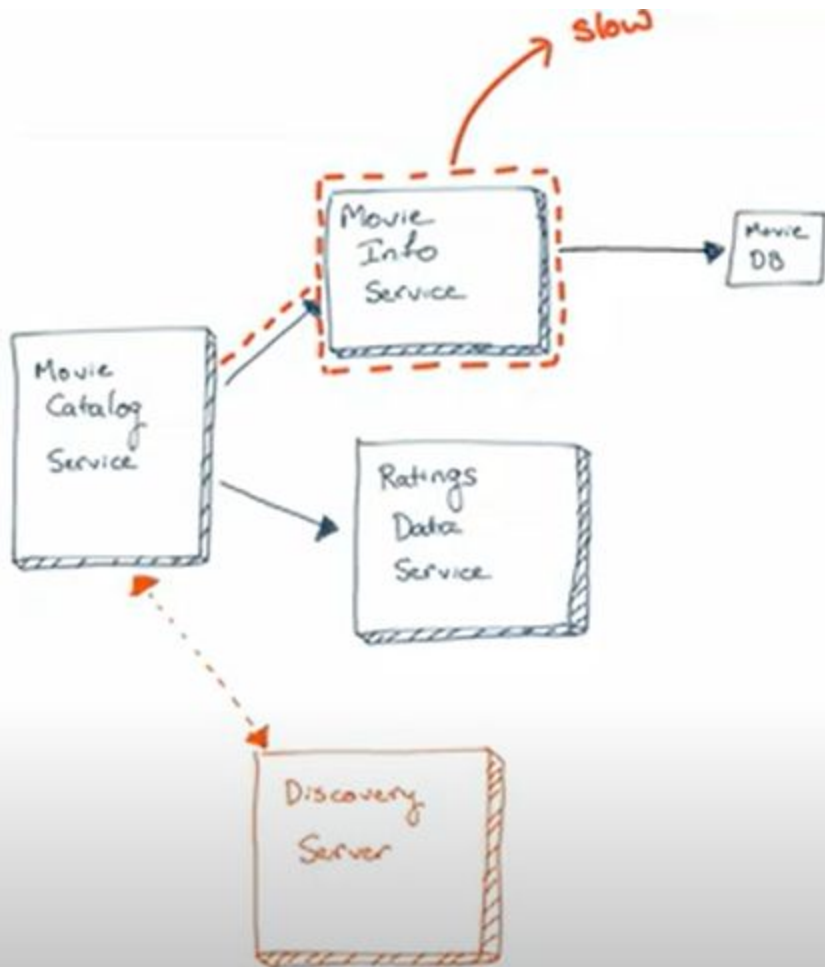
CIRCUIT NEEDS TO BE TRIPPED ..  
NEXT ????????????????????

I need to stop  
calling MovieInfo Svc.



I need to stop  
calling MovieInfo Svc.

But what do I do  
when a request  
comes in?





**WE NEED A FALLBACK  
MECHANISM**

# FALLBACK MECHANISM

- Throw an error
- Return a fallback “default” response
- Save previous responses (cache) and use that when possible



# Why CIRCUIT BREAKERS?

- Failing fast
- Fallback functionality
- Automatic recovery

# CIRCUIT BREAKER PATTERN

When to break  
circuit

What to do when  
circuit breaks

When to resume  
requests





**HYSTERIX**

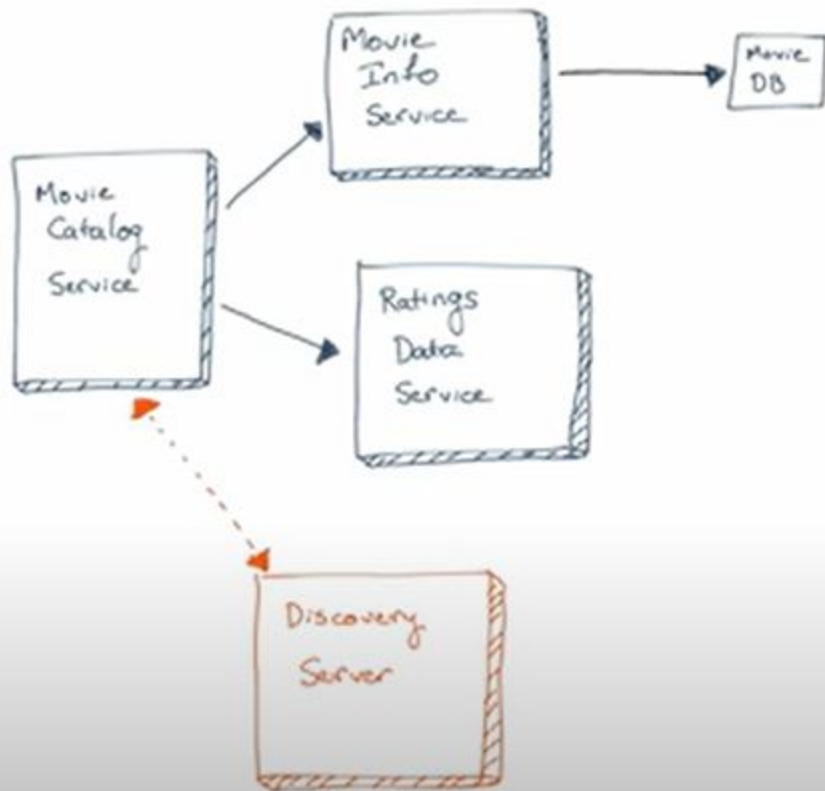
# Hystrix

- Open source library originally created by Netflix
- Implements circuit breaker pattern so you don't have to
- Give it the configuration params and it does the work
- Works well with Spring Boot

# Adding Hystrix to a Spring Boot microservice

- Add the Maven `spring-cloud-starter-netflix-hystrix` dependency
- Add `@EnableCircuitBreaker` to the application class
- Add `@HystrixCommand` to methods that need circuit breakers
- Configure Hystrix behavior

I got  
circuit  
breaker!



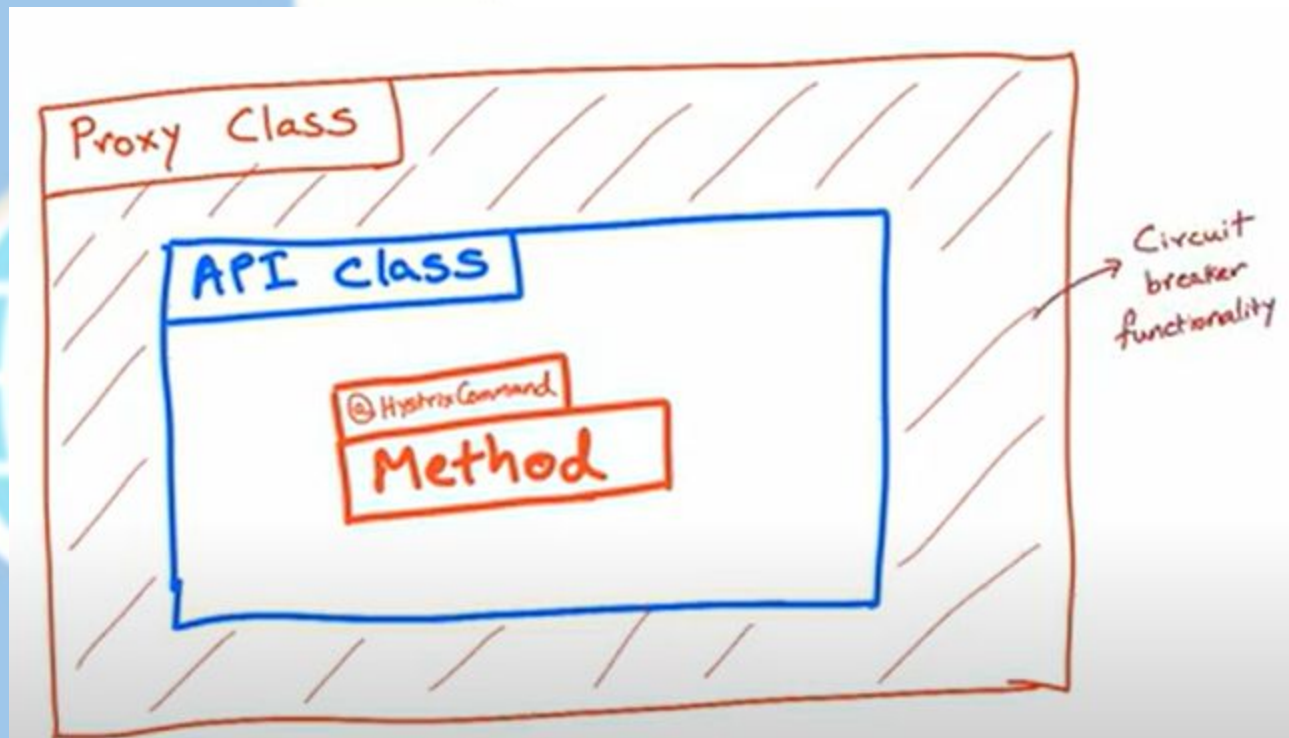


```
@HystrixCommand(fallbackMethod = "getFallbackCatalog")  
public List<CatalogItem> getCatalog(@PathVariable("userId") String userId) {
```

```
    public List<CatalogItem> getFallbackCatalog(@PathVariable("userId") String userId) {  
        return Arrays.asList(new CatalogItem("No movie", "", 0));  
    }  
}
```

# How does Hystrix work





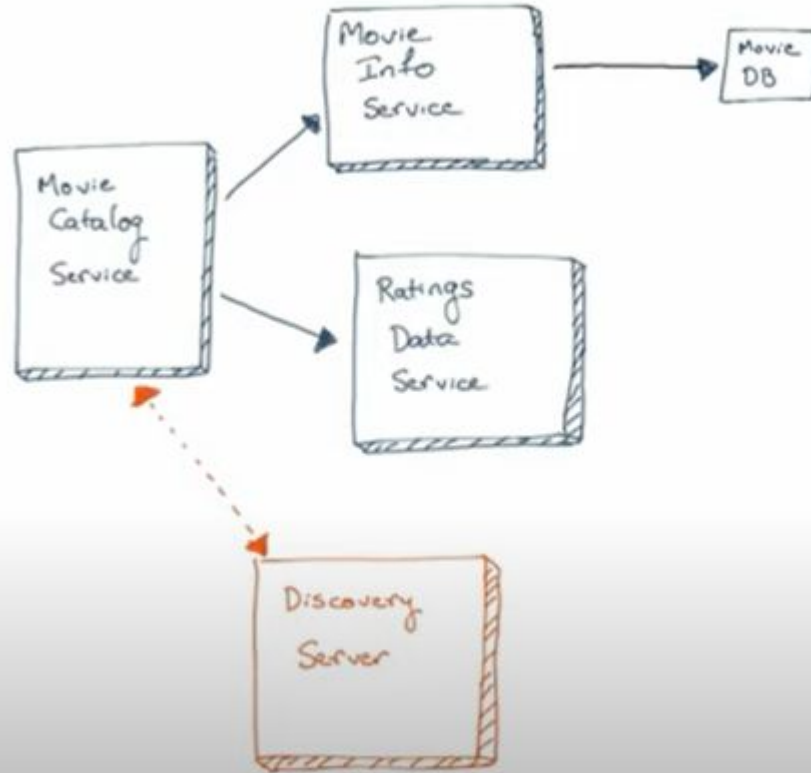
# hysterix

Maintaining state, not actively developing state

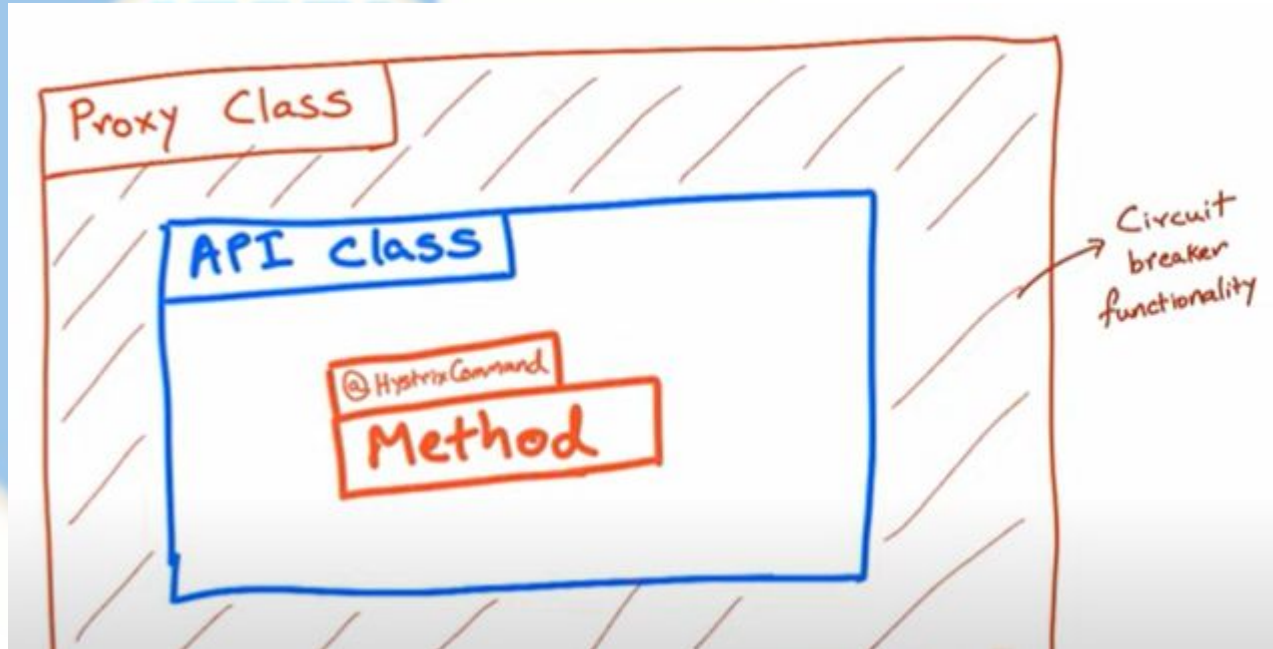




## Problem with Hystrix proxy



Granular, sep methods



Wraps around  
instance

# PARAMETERS

```
@HystrixCommand(fallbackMethod = "getFallbackUserRating",  
  
)  
public UserRating getUserRating(@PathVariable("userId") String userId) {  
    return restTemplate.getForObject("http://ratings-data-service/ratingsdata/user/" + userId, UserRating.class);  
}
```

# PARAMETERS

```
@HystrixCommand(fallbackMethod = "getFallbackUserRating",
    commandProperties = {
        @HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds", value = "2000"),
        @HystrixProperty(name = "circuitBreaker.requestVolumeThreshold", value = "5"),
        @HystrixProperty(name = "circuitBreaker.errorThresholdPercentage", value = "50"),
        @HystrixProperty(name = "circuitBreaker.sleepWindowInMilliseconds", value = "5000")
    }
)
public UserRating getUserRating(@PathVariable("userId") String userId) {
    return restTemplate.getForObject("http://ratings-data-service/ratingsdata/user/" + userId, UserRating.class);
}
```

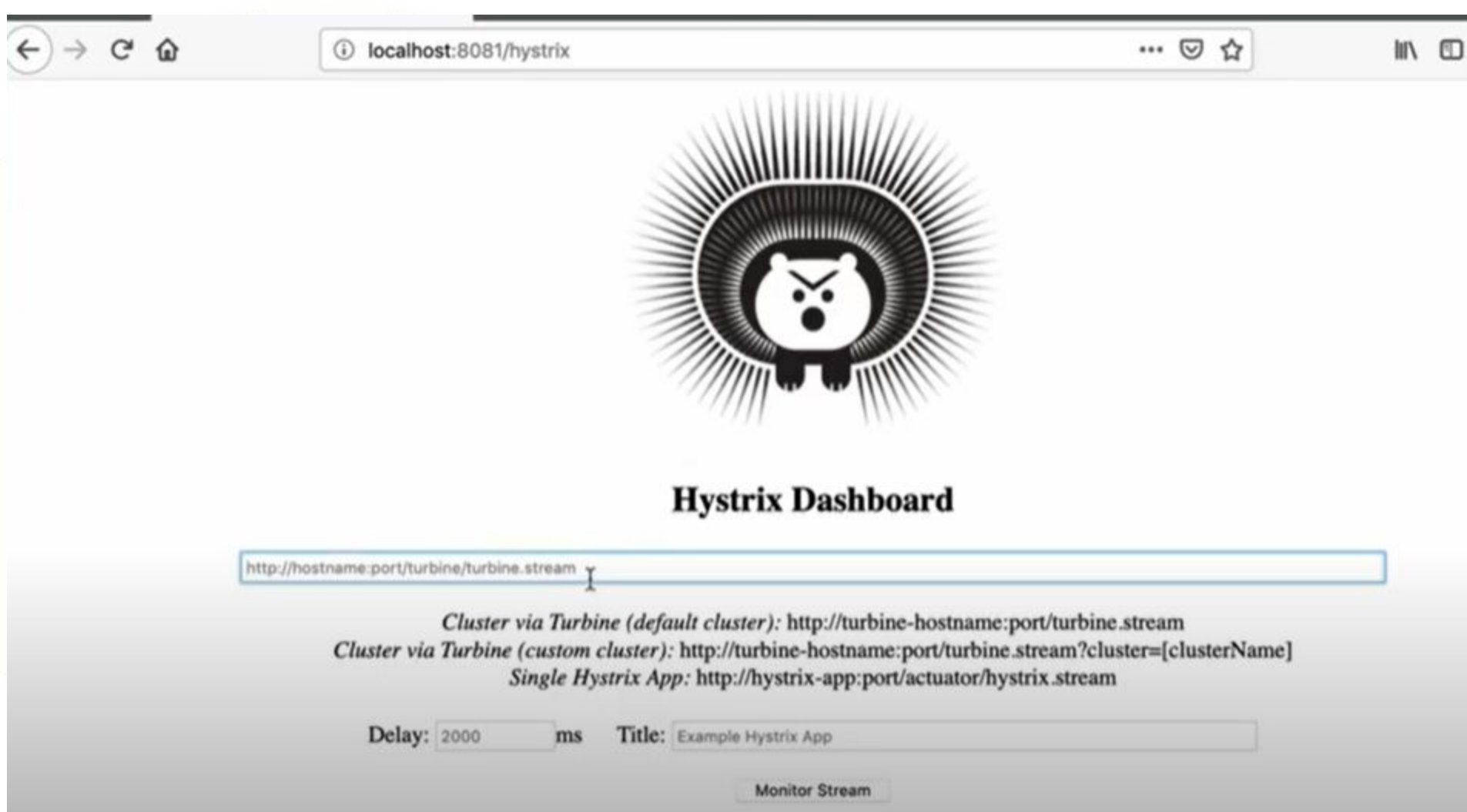
# HYSTRIX DASHBOARD

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-netflix-hystrix-dashboard</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
@EnableCircuitBreaker
@EnableHystrixDashboard
public class MovieCatalogServiceApplication {
```

The Spring logo, consisting of a large blue letter 'S' with a white wireframe globe in the center, is positioned on the left side of the image.

```
1 spring.application.name=movie-catalog-service
2 server.port=8081
3 management.endpoints.web.exposure.include=hystrix.stream
```



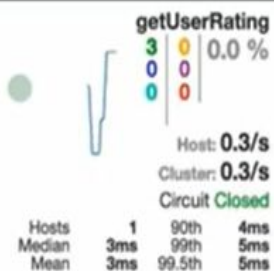
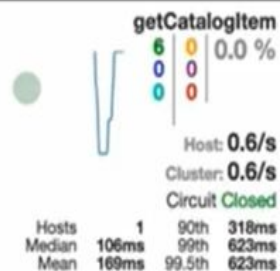
Hystrix Stream: <http://localhost:8081/actuator/hystrix.stream>



**HYSTRIX**  
DEFEND YOUR APP

**Circuit** Sort: [Error then Volume](#) | [Alphabetical](#) | [Volume](#) | [Error](#) | [Mean](#) | [Median](#) | [90](#) | [99](#) | [99.5](#)

[Success](#) | [Short-Circuited](#) | [Bad Request](#) | [Timeout](#) | [Rejected](#) | [Failure](#) | [Error %](#)



**Thread Pools** Sort: [Alphabetical](#) | [Volume](#) |





# Summary

- Understanding possible some causes for failure in microservices
- Threads and pools and impacts of slow microservices
- Timeouts and its limitations
- Circuit breaker pattern
- Hystrix concepts and implementation

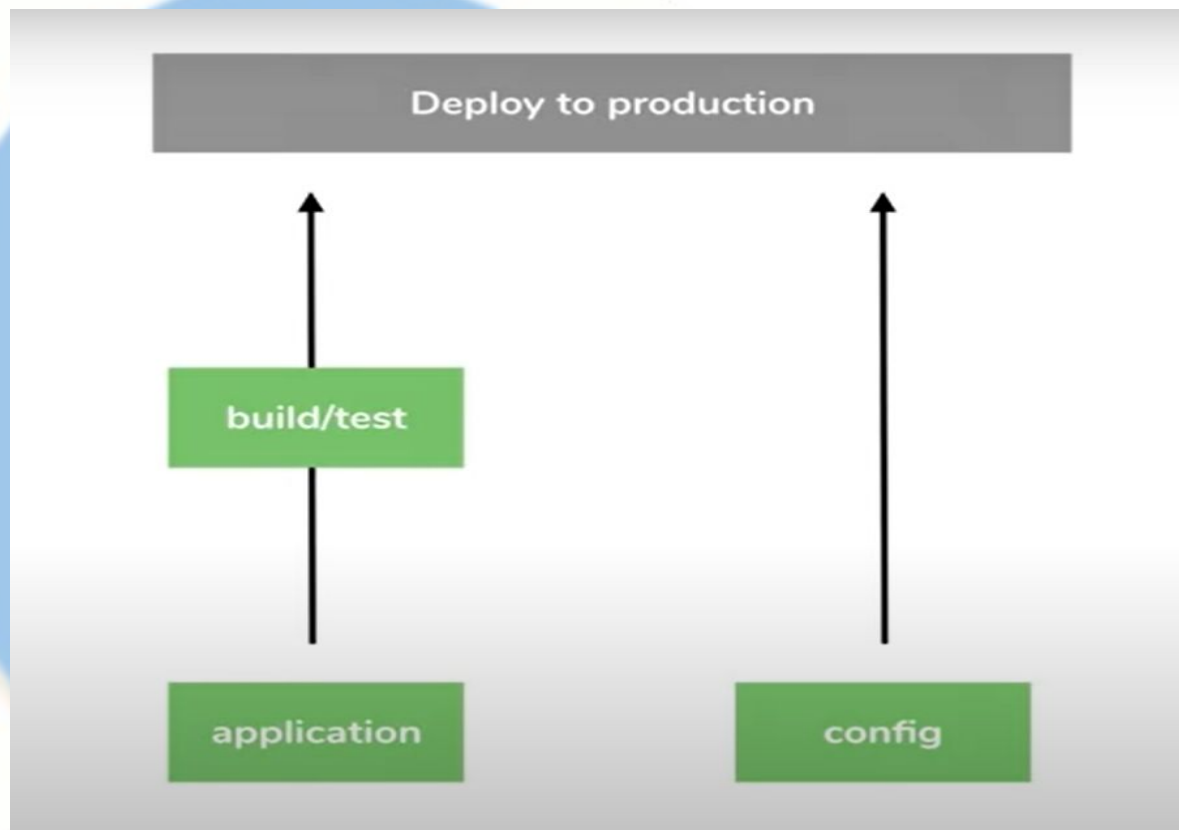
- Understanding microservice configuration goals
- Configuration features in Spring Boot
  - Value, Configprops, Actuator, Spring Profiles, Environment
- Spring Cloud Config server
- Dynamic configuration
- Best practices and patterns

# SPRING BOOT CONFIGURATIONS



# EXAMPLES

- Database connections
- Credentials
- Feature flags
- Business logic configuration parameters
- Scenario testing
- Spring Boot configuration





Xml FILES

XML FILES

YAML, PROPERTIES, JSON

Using property file config

Using property file config



# Using external property sources with Spring Boot





```
my.list.values=One,Two,Three
```

```
@Value("${my.greeting: default value}")  
private String greetingMessage;
```

```
@Value("${my.list.values}")  
private List<String> listValues;
```

Hello Worldsome static message[One, Two, Three]

```
@Value("#{${dbValues}}")  
private Map<String, String> dbValues;
```

```
my.greeting=Hello World  
my.list.values=One,Two,Three  
dbValues={connectionString: 'http://___', userName: 'foo', password: 'bar'}
```

Hello Worldsome static message[One, Two, Three]{connectionString=http://\_\_\_, userName=foo, password=bar}

# ConfigurationProperties

Pull up group of configurations @once  
Create a class with getter, setter  
@ConfigurationProperties("mentionit")  
To create it as bean, @Configuration  
Then, autowire it wherever needed.

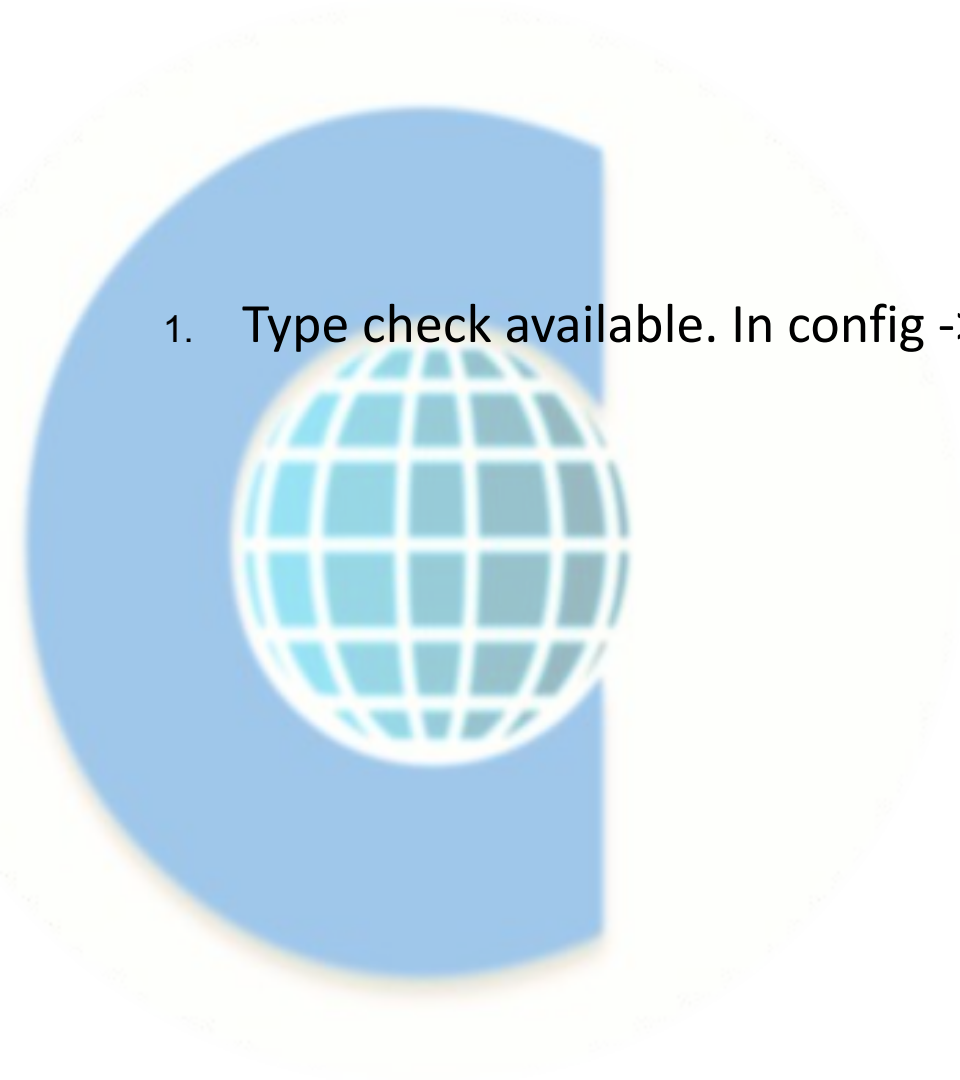
```
@ConfigurationProperties("db")
public class DbSettings {

    private String connection;
    private String host;
    private int port;

    public String getConnection() {
        return connection;
    }

    public void setConnection(String connection) {
        this.connection = connection;
    }

    public String getHost() {
```

- 
1. Type check available. In config -> int, app.prop-> string, then error

# Spring boot actuator

/actuator/configprops

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

```
management.endpoints.web.exposure.include=*
```

```
<dependency>
  <groupId>org.springframework.boot<
  </groupId>
  <artifactId>spring-boot-starter-actuat
  </artifactId>
</dependency>
```

```
management.endpoints.web.expo
sure.include=*
```

# YAML

? Yet Another Markup Language

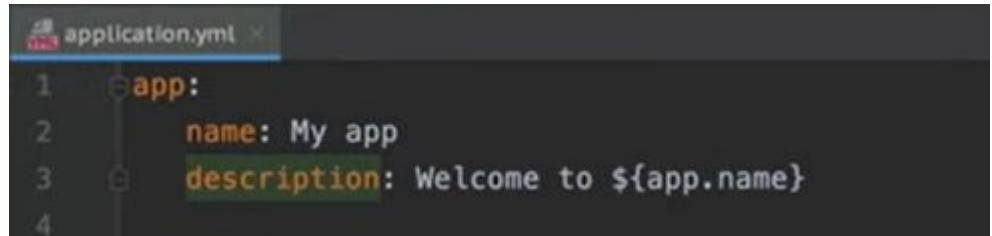
YAML ain't markup language

Key value pair with : instead of =

Double quotes not required, it recognizes text as string, number

Nesting possible

Instead of . use :



```
application.yml
1  app:
2    name: My app
3    description: Welcome to ${app.name}
4
```

# ENVIRONMENT SPECIFIC - spring profile

Different value based configurations, based on different environments

Profile as a set of configuration values, go together as group, & form values

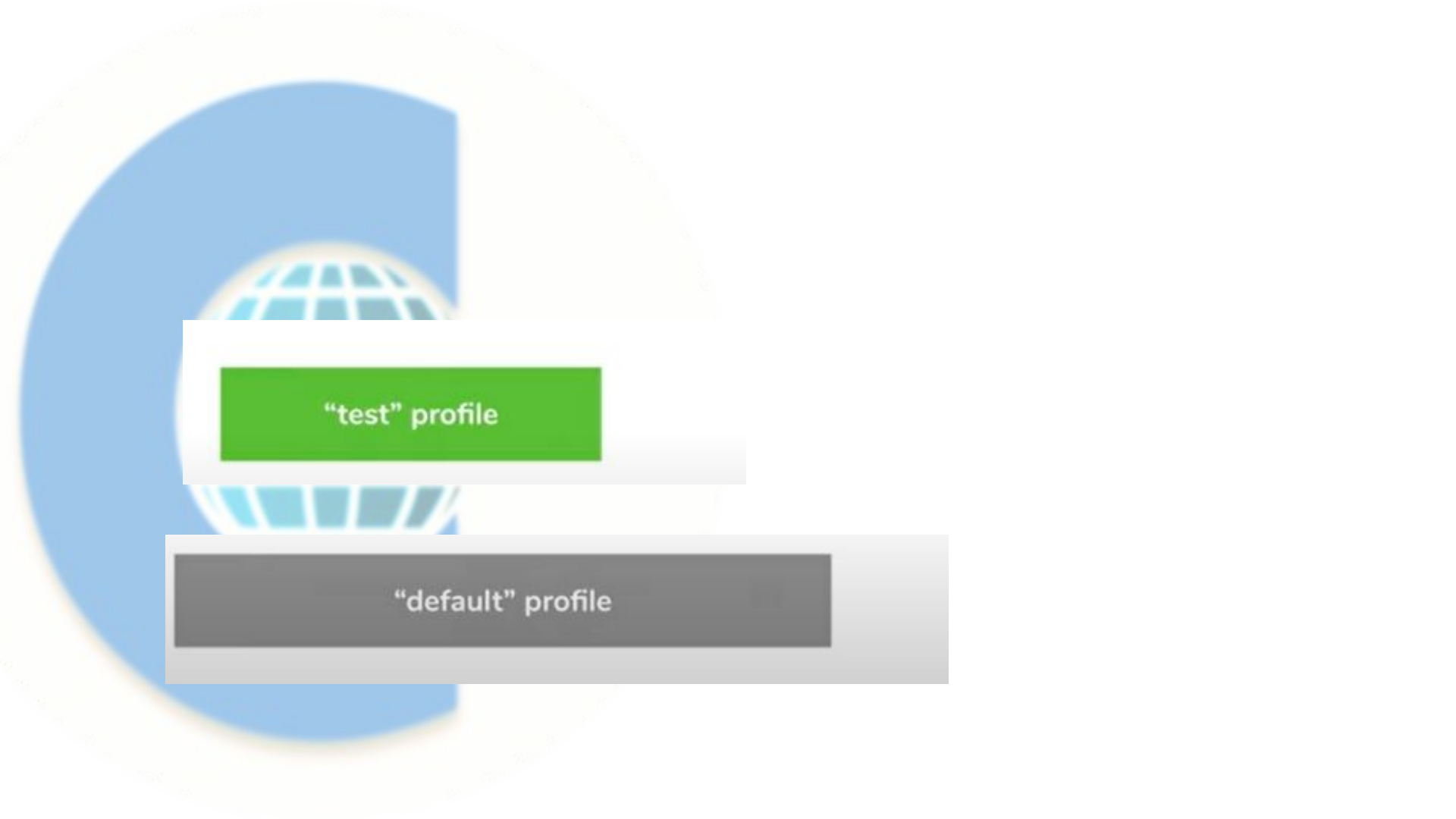
<https://www.baeldung.com/spring-profiles>

**application-`<profileName>`.extn**

STE

P: create a new profile and tell spring to use that

```
spring.profiles.active: test|
```



"test" profile


"default" profile

# Selecting beans by profile

```
@Repository  
public class DataSourceBean {  
}
```

```
@Repository  
public class LocalDataSourceBean {  
}
```



The background of the slide features a large, faint Spring logo, which is a blue stylized 'S' with a green leaf-like shape at the bottom right.

```
@Repository
@Profile("production")
public class DataSourceBean {

}
```

```
@Repository
@Profile("dev")
public class LocalDataSourceBean {

}
```

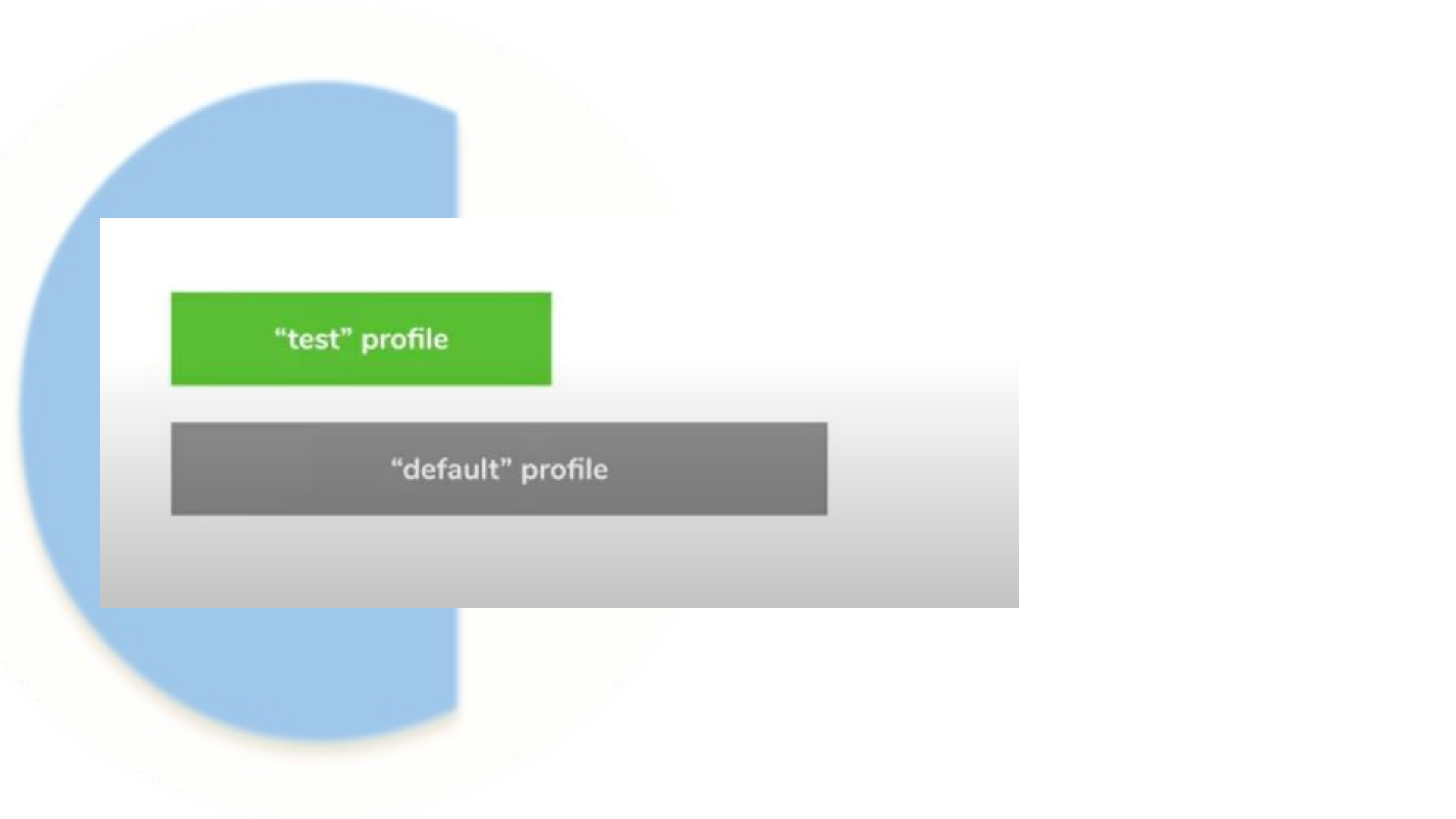
The Spring logo, which consists of a large blue letter 'C' with a white wireframe globe in the center.

All beans instantiated

**Default**

`@Profile("default")`

(or without any `@Profile` )

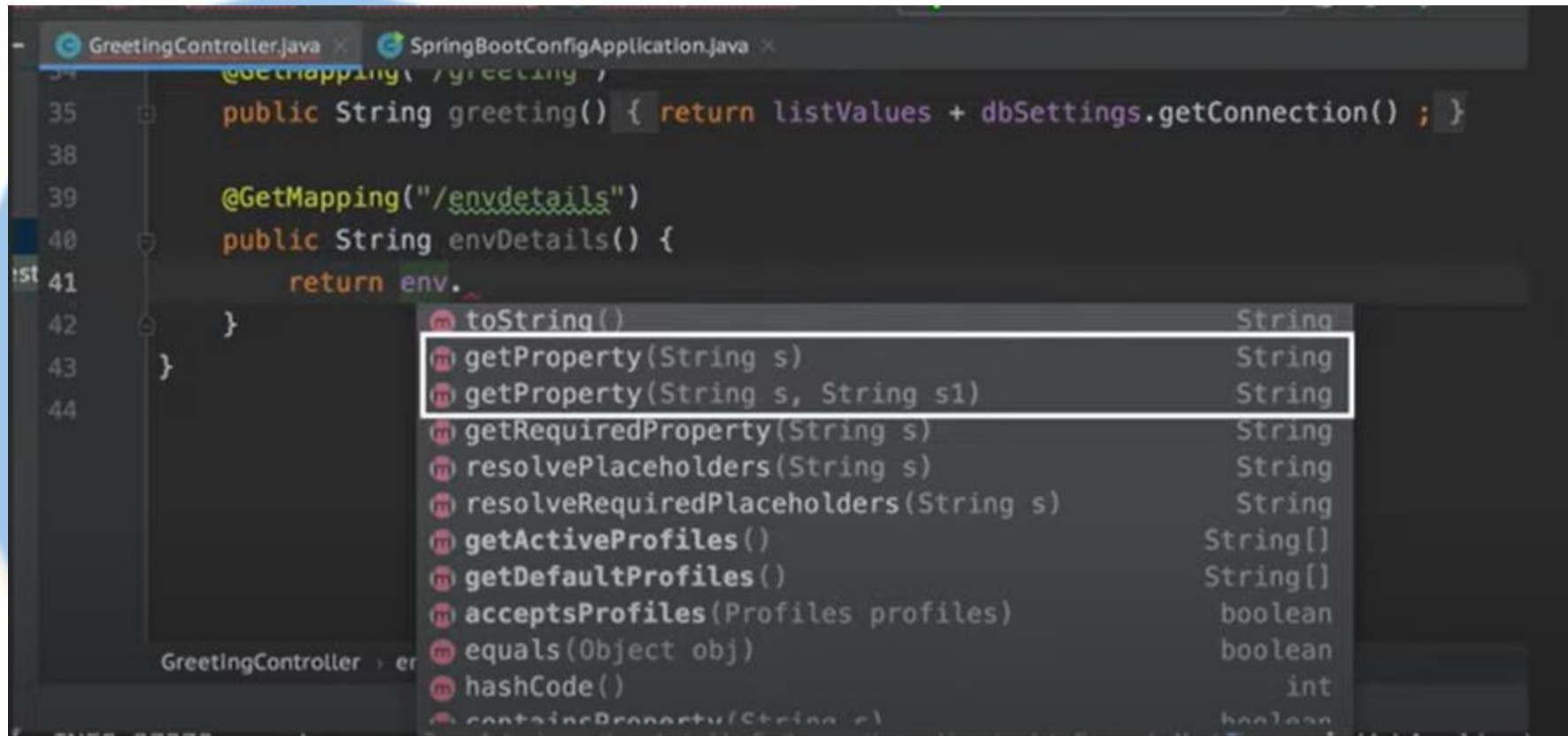


**"test" profile**

**"default" profile**

# Using Environment object - Microservice configuration with Spring Boot

```
GreetingController.java
23
24     @Autowired
25     private DbSettings dbSettings;
26
27     @Autowired
28     private Environment env;
29
30     public GreetingController() {
31     }
32
33
34     @GetMapping("/greeting")
35     public String greeting() { return listValues + dbSettings.getConnection() ; }
36
37
38
39     @GetMapping("/envdetails")
40     public String envDetails() {
41         return env.toString();
42     }
43 }
44
```



But, we shouldn't be using it

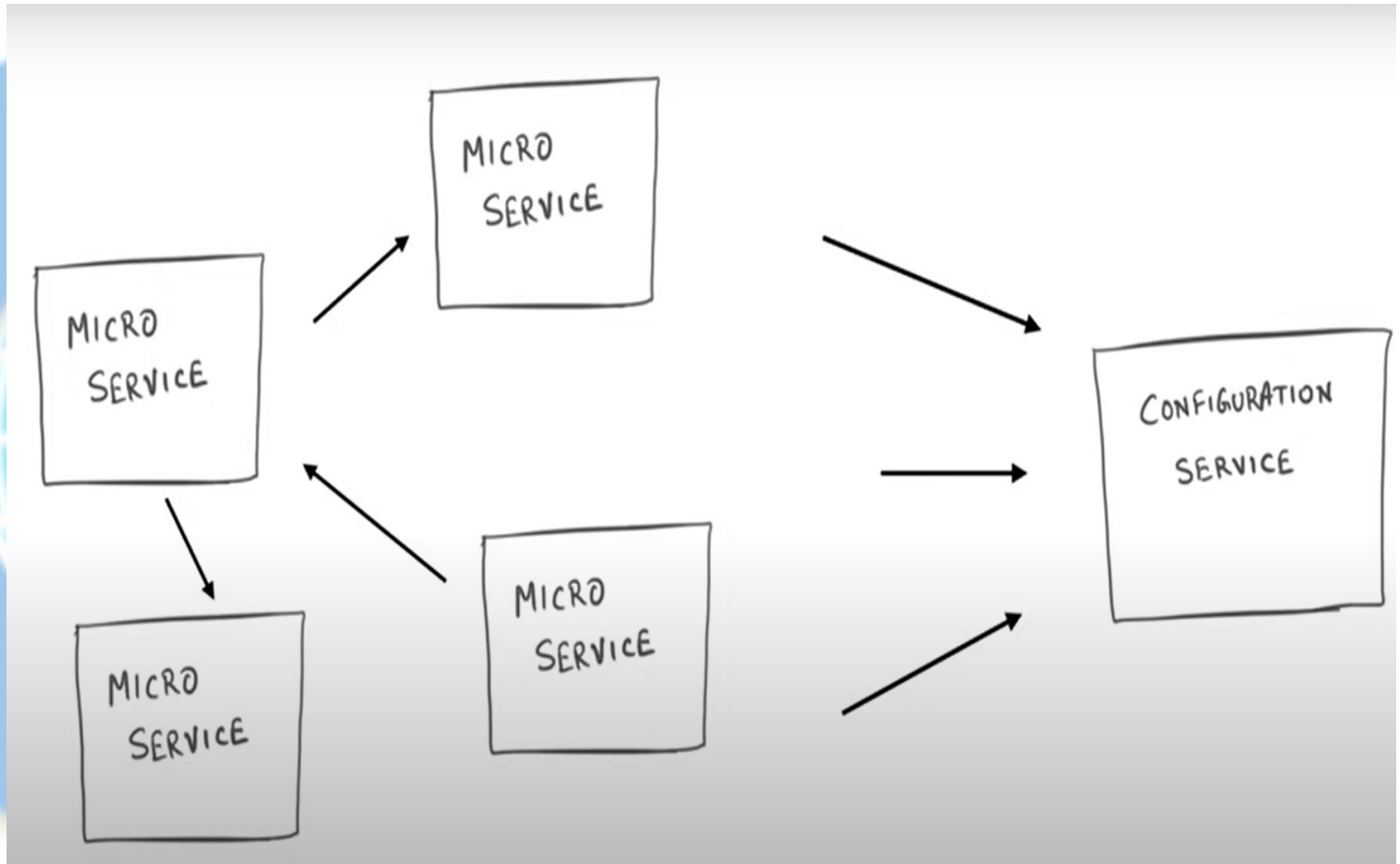


A screenshot of a web browser window displaying the details of a `StandardServletEnvironment`. The browser's address bar shows `localhost:8080/envdetails`. The page content is a JSON-like representation of the environment's configuration, including active profiles, default profiles, and various property sources.

```
StandardServletEnvironment {activeProfiles=[test], defaultProfiles=[default], propertySources=[MapPropertySource {name='server.ports'}, ConfigurationPropertySourcesPropertySource {name='configurationProperties'}, StubPropertySource {name='servletConfigInitParams'}, ServletContextPropertySource {name='servletContextInitParams'}, PropertiesPropertySource {name='systemProperties'}, OriginAwareSystemEnvironmentPropertySource {name='systemEnvironment'}, RandomValuePropertySource {name='random'}, OriginTrackedMapPropertySource {name='applicationConfig: [classpath:/application-test.yml]'}, OriginTrackedMapPropertySource {name='applicationConfig: [classpath:/application.yml]'}]}
```

# Spring Cloud Config Server

**From single service to  
multiple microservices**



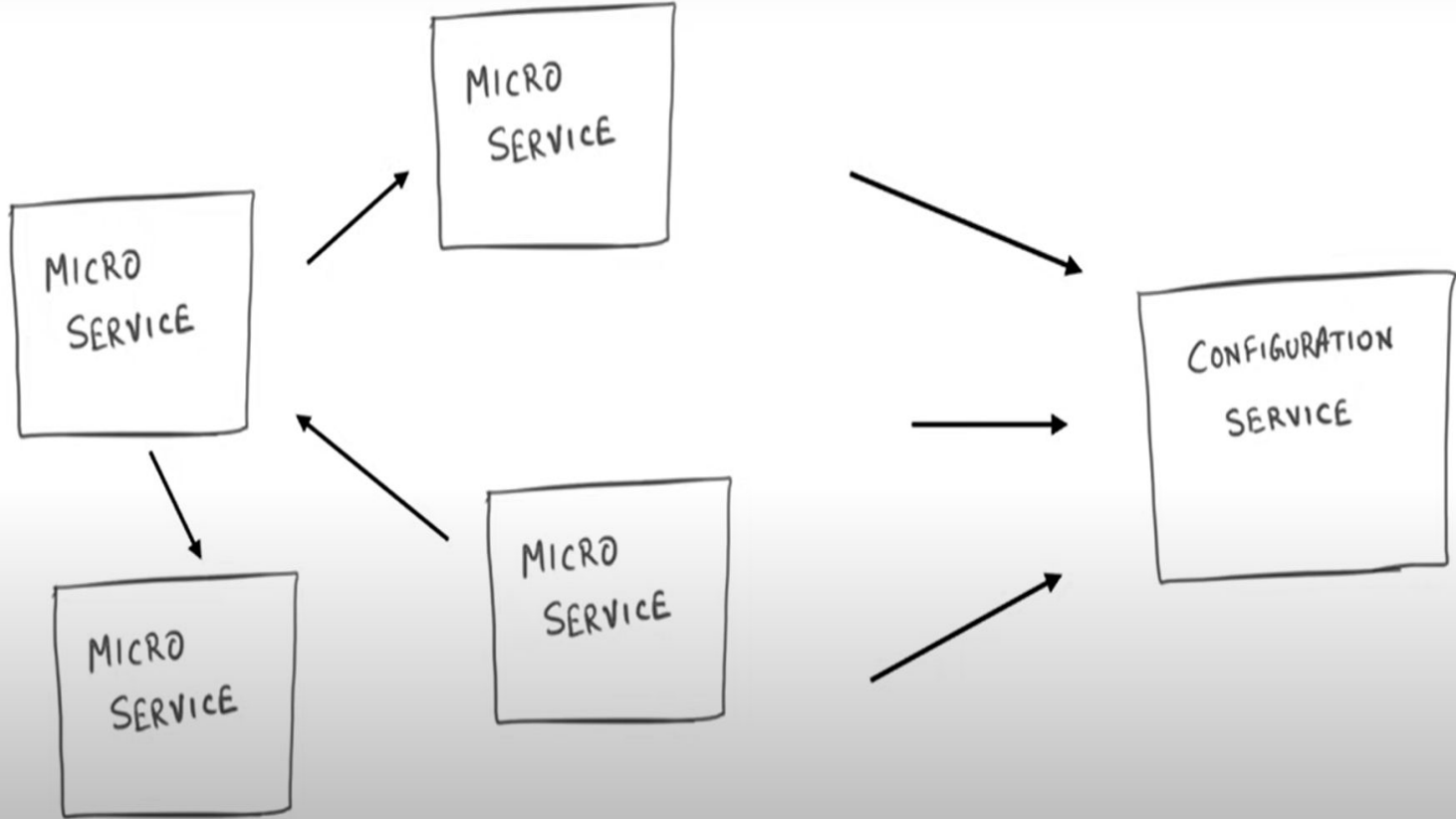


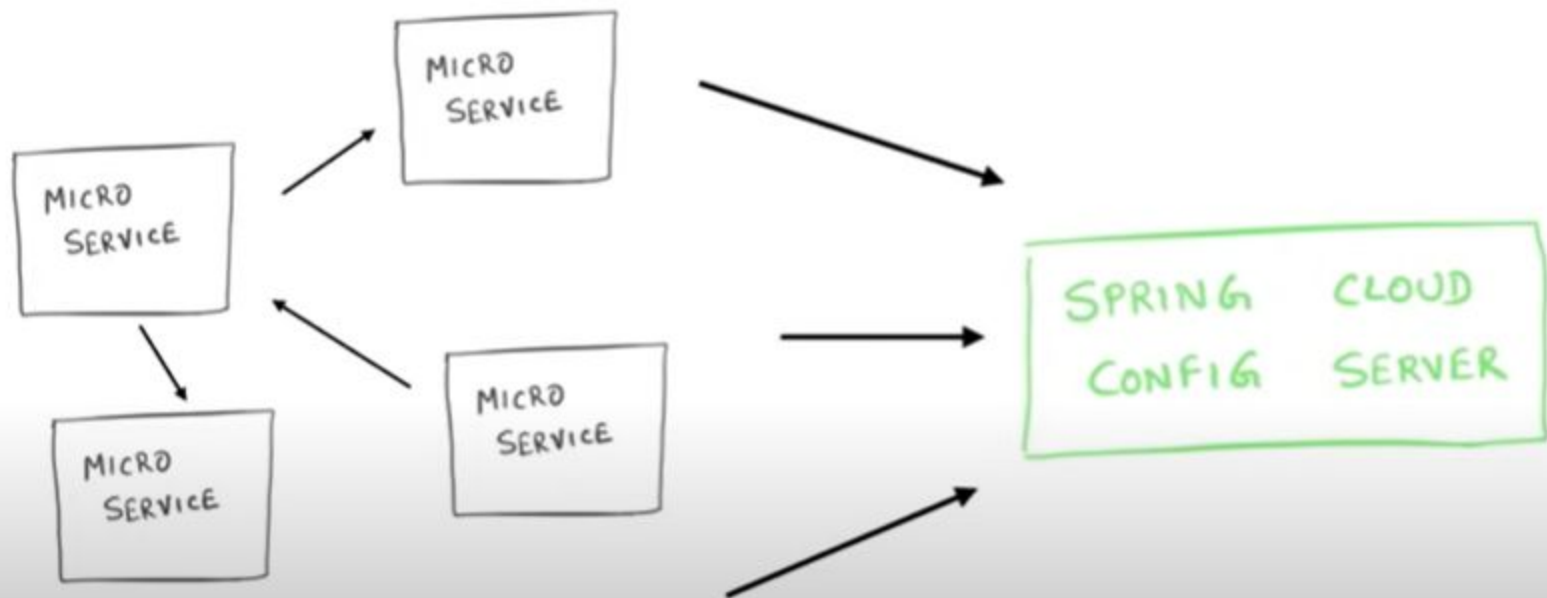


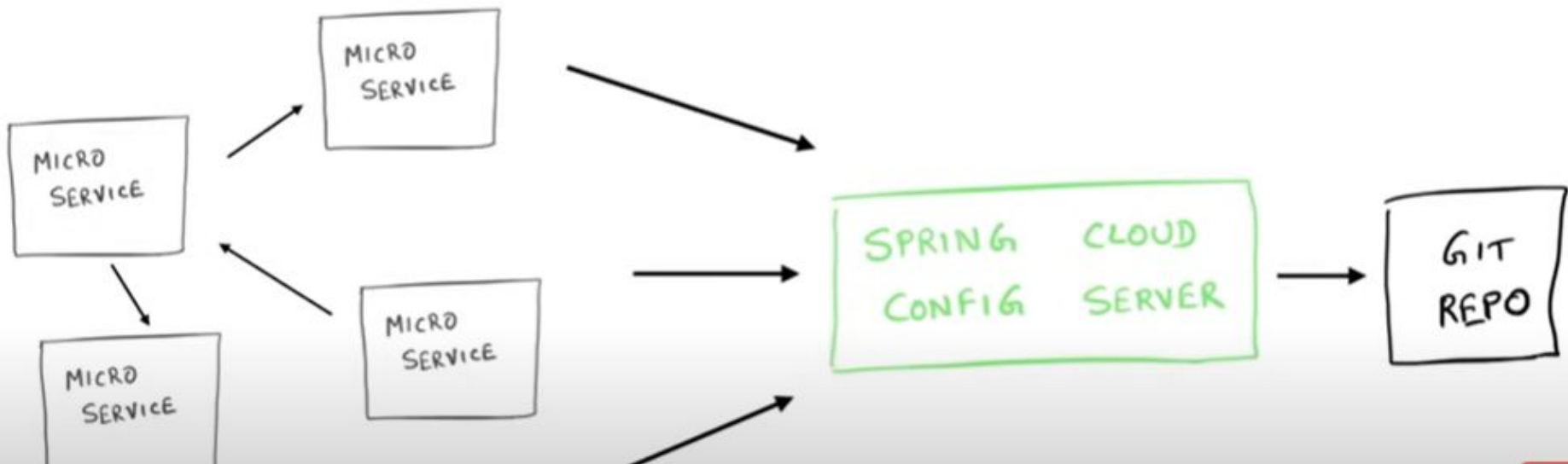
**Config as a  
microservice**

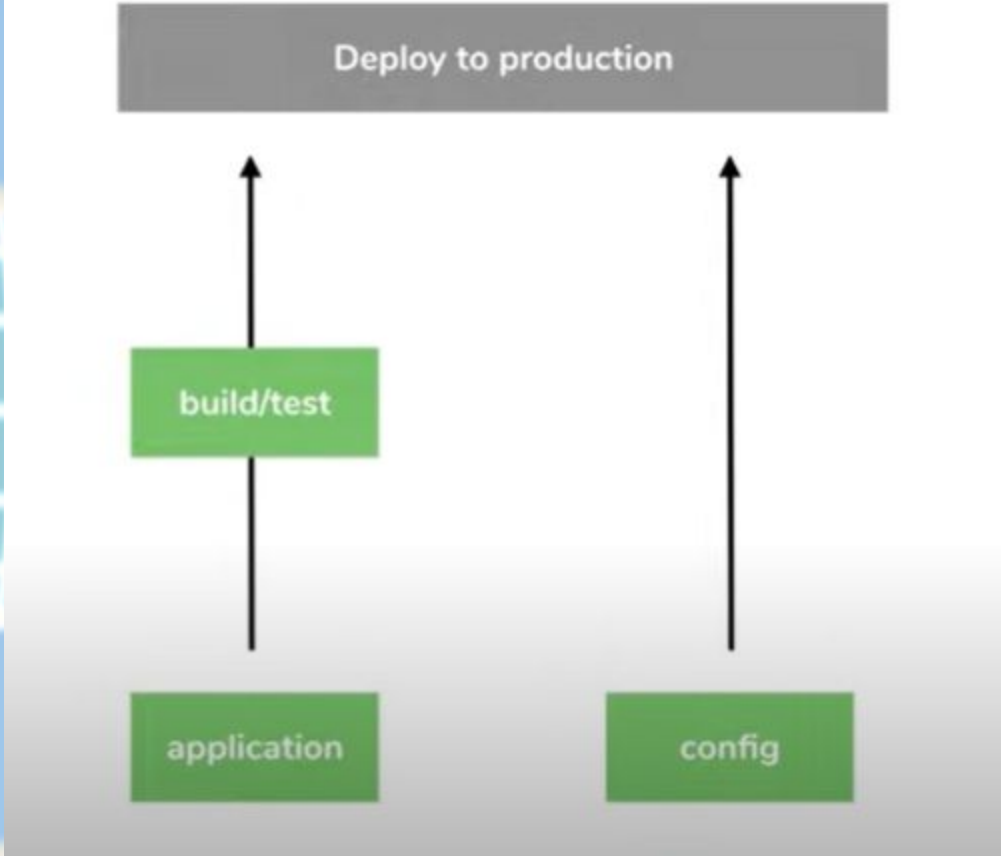
# OPTIONS AVAILABLE

- Apache Zookeeper
- ETCD - distributed key-value store
- Hashicorp Consul
- Spring Cloud Configuration Server









# Set up spring cloud config server from scratch

The screenshot displays the 'Dependencies' section of a Spring Cloud Config Server setup. On the left, a sidebar labeled 'Dependencies' is visible. The main area shows the 'Artifact' as 'spring-cloud-config-server'. Below this, there is a section for 'Options'. The 'Search dependencies to add' section contains the text 'Web, Security, JPA, Actuator, Devtools...'. The 'Selected dependencies' section shows a single dependency: 'Config Server', described as 'Central management for configuration via Git, SVN, or HashiCorp Vault.' A mouse cursor is visible at the bottom center of the screen.

Artifact  
spring-cloud-config-server

> Options

Dependencies

Search dependencies to add  
Web, Security, JPA, Actuator, Devtools...

Selected dependencies

**Config Server**  
Central management for configuration via Git, SVN, or HashiCorp Vault.

# AFTER GENERATING,ADD ANNOTATION

```
@SpringBootApplication
@EnableConfigServer
public class SpringCloudConfigServerApplication {

    public static void main(String[] args) { SpringApplication.run(SpringCloudConfigServer.
    }
```



m pom.xml x

application.properties x

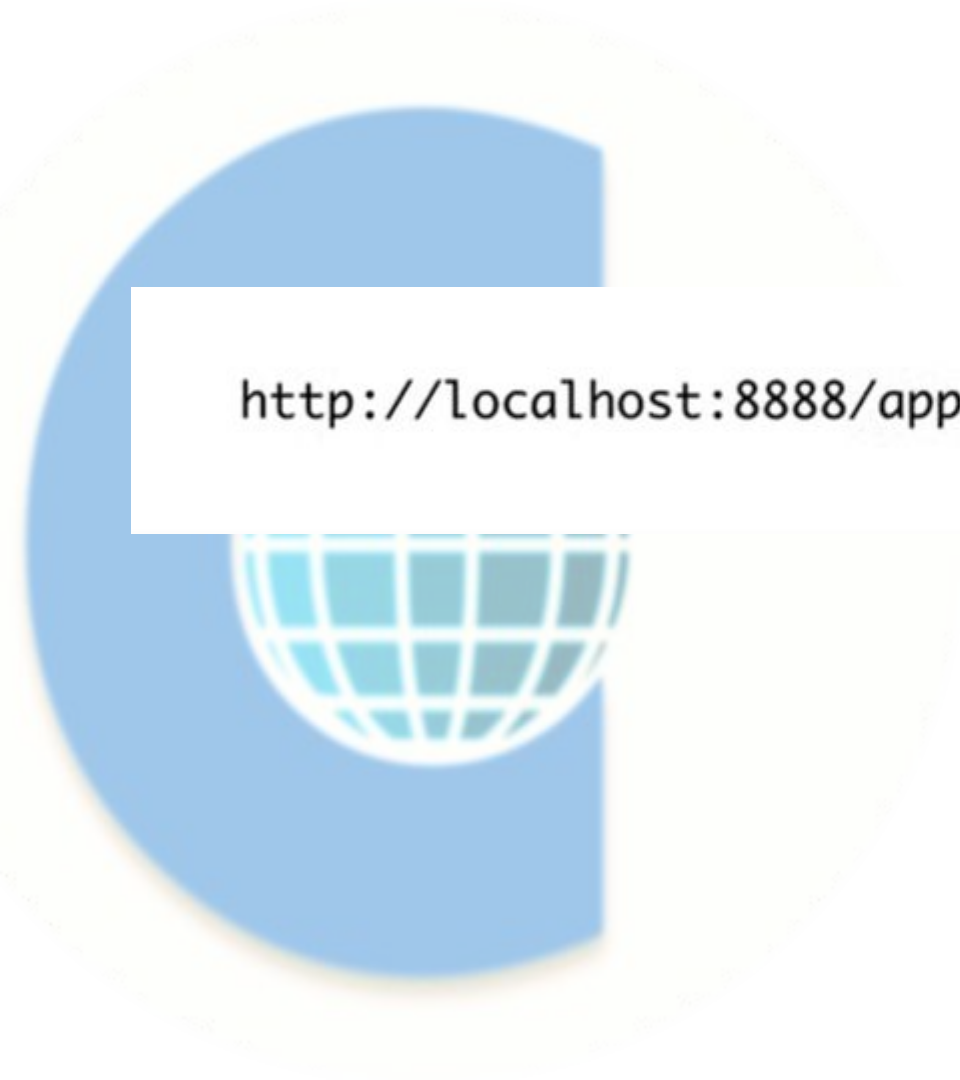
```
1 spring.cloud.config.server.git.uri=
```

```
pom.xml x application.properties x
1 spring.cloud.config.server.git.uri=${HOME}/code/configrepo
2 |
```

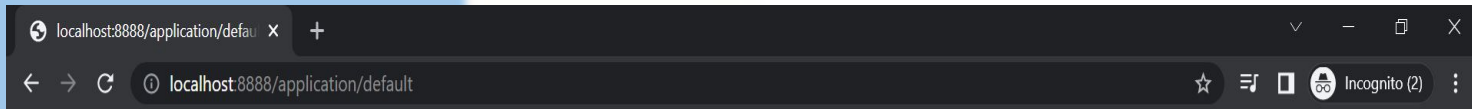
C:\\Users\\user234\\Desktop\\config-git

```
server.port=8888
```

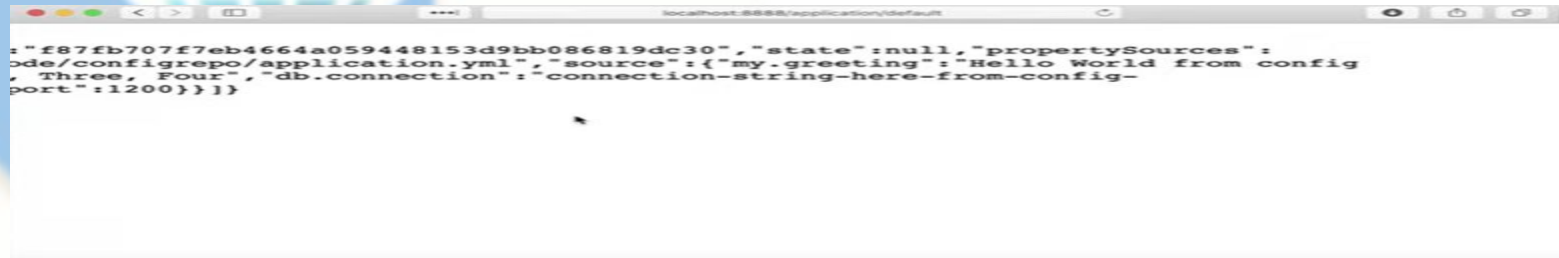
`http://localhost:8888/<file-name>/<profile>`



`http://localhost:8888/application/default`

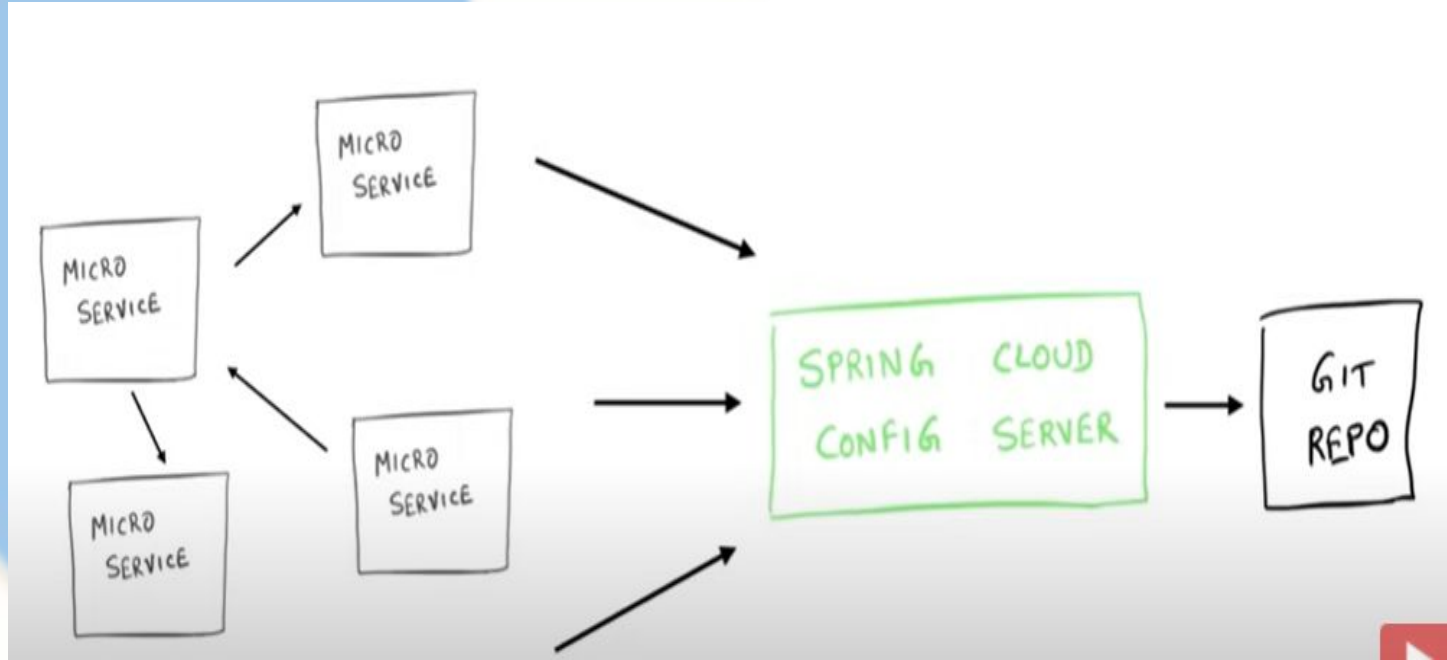


```
{ "name": "application", "profiles": [ "default" ], "label": null, "version": "2b35dc469e5e554449aca783779070ad5adc73a6", "state": null, "propertySources":  
[ { "name": "C:\\Users\\arpriya1\\config\\file:C:\\Users\\arpriya1\\AppData\\Local\\Temp\\config-repo-782838089272389949\\application.yml", "source":  
{ "server.port": 4000, "my.greeting": "hello configserverteam", "my.list.values": "hello, hi, goodmorning", "app.name": "sage it", "app.description": "welcome to config server  
${app.name}", "db.details": "{ connectionString: 'https://..... ', username: 'user203', pwd: 'password is  
empty'}", "user.name": "user202", "user.password": "@#$$^", "user.role": "manager", "management.endpoints.web.exposure.include": "*" } } ] }
```



```
: "f87fb707f7eb4664a059448153d9bb086819dc30", "state": null, "propertySources":  
ode/configrepo/application.yml", "source": { "my.greeting": "Hello World from config  
, Three, Four", "db.connection": "connection-string-here-from-config-  
port": 1200 } } ] }
```

# Centralised, externalised config server



# Setting up spring cloud config client

```
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>

<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-dependencies</artifactId>
      <version>${spring-cloud.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

# OR

Dependencies

Artifact  
demo

> Options

Q

☰

1 selected

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

**Config Client**

Client that connects to a Spring Cloud Config Server to fetch the application's configuration.

✓

```
spring.cloud.config.uri: http://localhost:8888|
```



my.greeting: Hello World from config server db.connection: connection-string-here-from-config-server



# To give yml specific to microservice

In your config git repo,

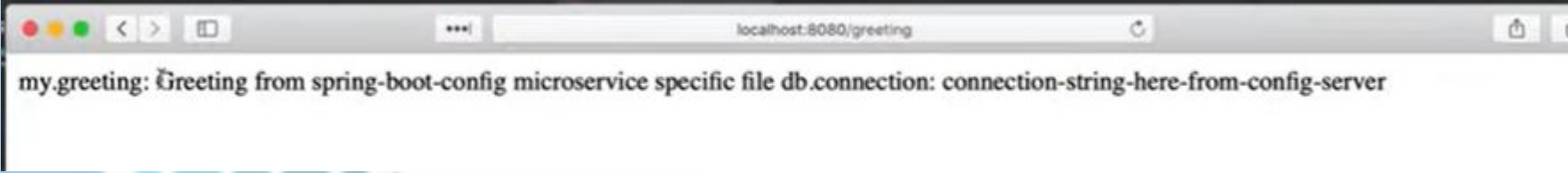
```
$ microservice-name.yml
```

In your config git repo, add this yml and commit it

```
$ vi spring-boot-config.yml
$ git add .
$ git commit -m "Adding spring-boot-config config file"
[master 51e9e58] Adding spring-boot-config config file
1 file changed, 1 insertion(+)
create mode 100644 spring-boot-config.yml
$
```

Go to microservice app.yml

```
spring.application.name: ispring-boot-config
```



my.greeting: Greeting from spring-boot-config microservice specific file db.connection: connection-string-here-from-config-server



Refreshing  
properties at  
runtime

1. Don't have to start cloud config server, everytime
2. In your microservice, have actuator dependency

3.

```
</dependency>  
<dependency>  
  <groupId>org.springframework.boot</groupId>  
  <artifactId>spring-boot-starter-actuator</artifactId>  
</dependency>    I
```

```
@RestController  
@RefreshScope  
public class GreetingController {    I
```

GET localhost:8080/actuator/info

POST localhost:8080/actuator/health

+

...

No Environment

👁

⚙

Untitled Request

Comments (0)

POST

localhost:8080/actuator/refresh

Send

Save

Params

Authorization

Headers (8)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body

Cookies

Headers (3)

Test Results

Status: 200 OK

Time: 6ms

Size: 131 B

Save Response

Pretty

Raw

Preview

Visualize BETA

JSON

≡

🔍

1

{

2

"status": "UP"

3

}

GET localhost:8080/actuator/info POST localhost:8080/actuator/health + ... No Environment

Untitled Request Comments (0)

POST localhost:8080/actuator/refresh Send Save

Params Authorization Headers (9) Body Pre-request Script Tests Settings Cookies Code


none form-data x-www-form-urlencoded raw binary GraphQL BETA

This request does not have a body

Body Cookies Headers (3) Test Results Status: 200 OK Time: 223ms Size: 155 B Save Response

Pretty Raw Preview Visualize BETA JSON

```
1 {
2   "config.client.version",
3   "my.greeting"
4 }
```



# Configuration Strategies - when to use what?

# Spring boot project

<https://www.java67.com/2022/12/10-projects-ideas-to-learn-spring-boot.html>

- 1.1. Online Ticket Booking System using Spring Boot**
- 3. Build An E-Commerce Website And App Using Spring Boot**
- Student management**

Read more: <https://www.java67.com/2022/12/10-projects-ideas-to-learn-spring-boot.html#ixzz7tbSJKL35>

Read more: <https://www.java67.com/2022/12/10-projects-ideas-to-learn-spring-boot.html#ixzz7tbSFtfl>



# Sb project

<https://www.javaguides.net/2018/10/free-open-source-projects-using-spring-boot.html>

Petclinic

Usermanagement

Registration