

Spring Cloud로 개발하는 マイクロ서비스 アプリケイ션



Microservices

+



Spring
Cloud

```
class Book {
    def self, title, price, author;
    self.title = title
    self.price = price
    self.author = author
}

public static void main(String[] args)
{
    var fs = require('fs');
    fs.readFile('/JONE.txt' /* 1 */,
        function (err, data) {
            console.log(data); // 3
        });
}

<@interface NextInnovationDelegate : NSObject <UIApplicationDelegate> >

<@implementation NextInnovationDelegate >
<@end>
```



목차



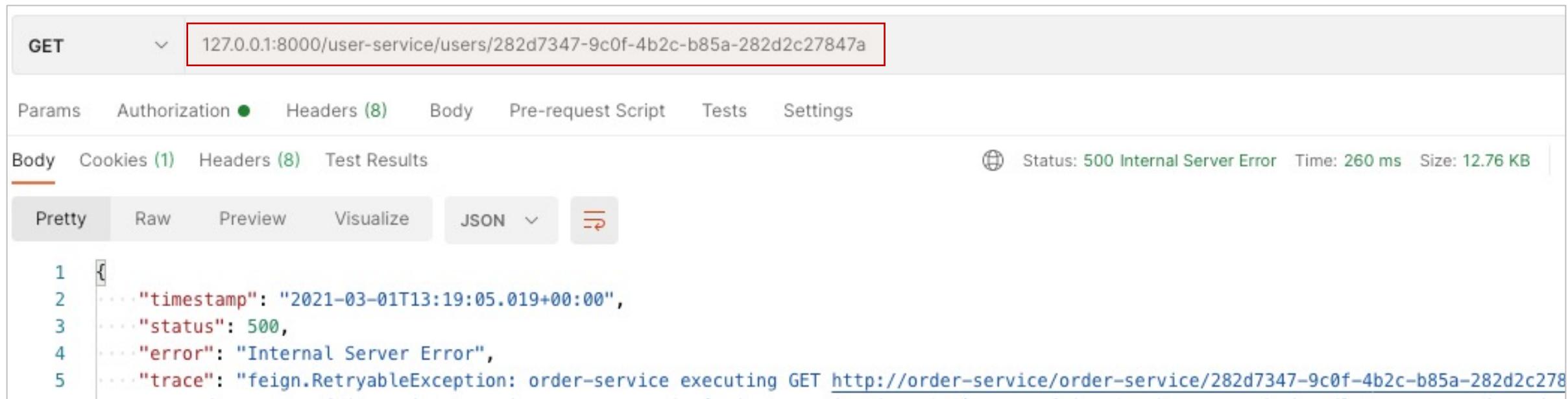
- Section 9: 암호화 처리를 위한 Encryption과 Decryption
- Section 10: 마이크로서비스간 통신
- Section 11: 데이터 동기화를 위한 Kafka 활용 ①
- Section 12: 데이터 동기화를 위한 Kafka 활용 ②
- **Section 13: 장애 처리와 Microservice 분산 추적**
- Section 14: Microservice Architecture 패턴
- Section 15: 애플리케이션 배포를 위한 컨테이너 가상화
- Section 16: 애플리케이션 배포 – Docker Container
- Appendix: Microservice 패턴

Section 13.

장애 처리와 Microservice 분산 추적

- CircuitBreaker
- Resilience4j
- Distributed Tracing
- Trace ID and Span ID
- Zipkin server 활용

Microservice 통신 시 연쇄 오류



The screenshot shows a POSTMAN interface with the following details:

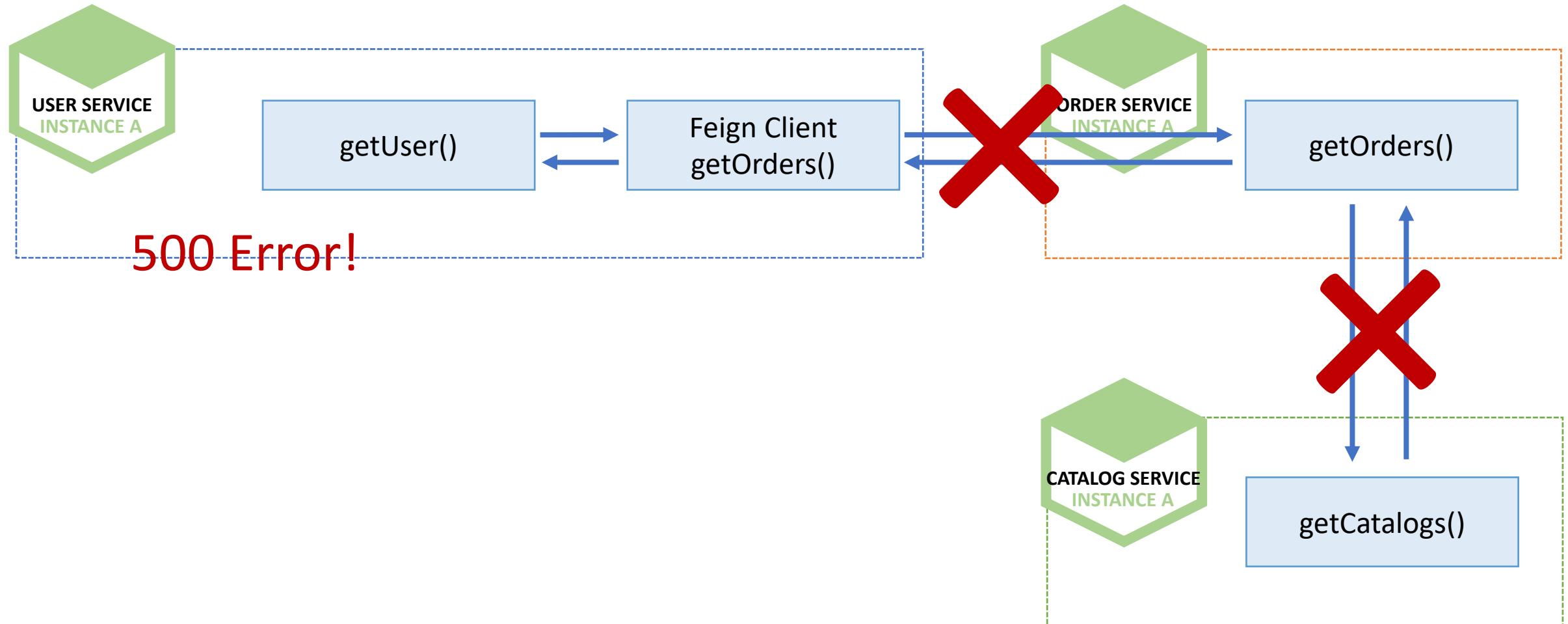
- Method: GET
- URL: 127.0.0.1:8000/user-service/users/282d7347-9c0f-4b2c-b85a-282d2c27847a
- Headers (8) selected in the top navigation.
- Body tab selected in the bottom navigation.
- Status: 500 Internal Server Error
- Time: 260 ms
- Size: 12.76 KB

The response body is displayed in JSON format:

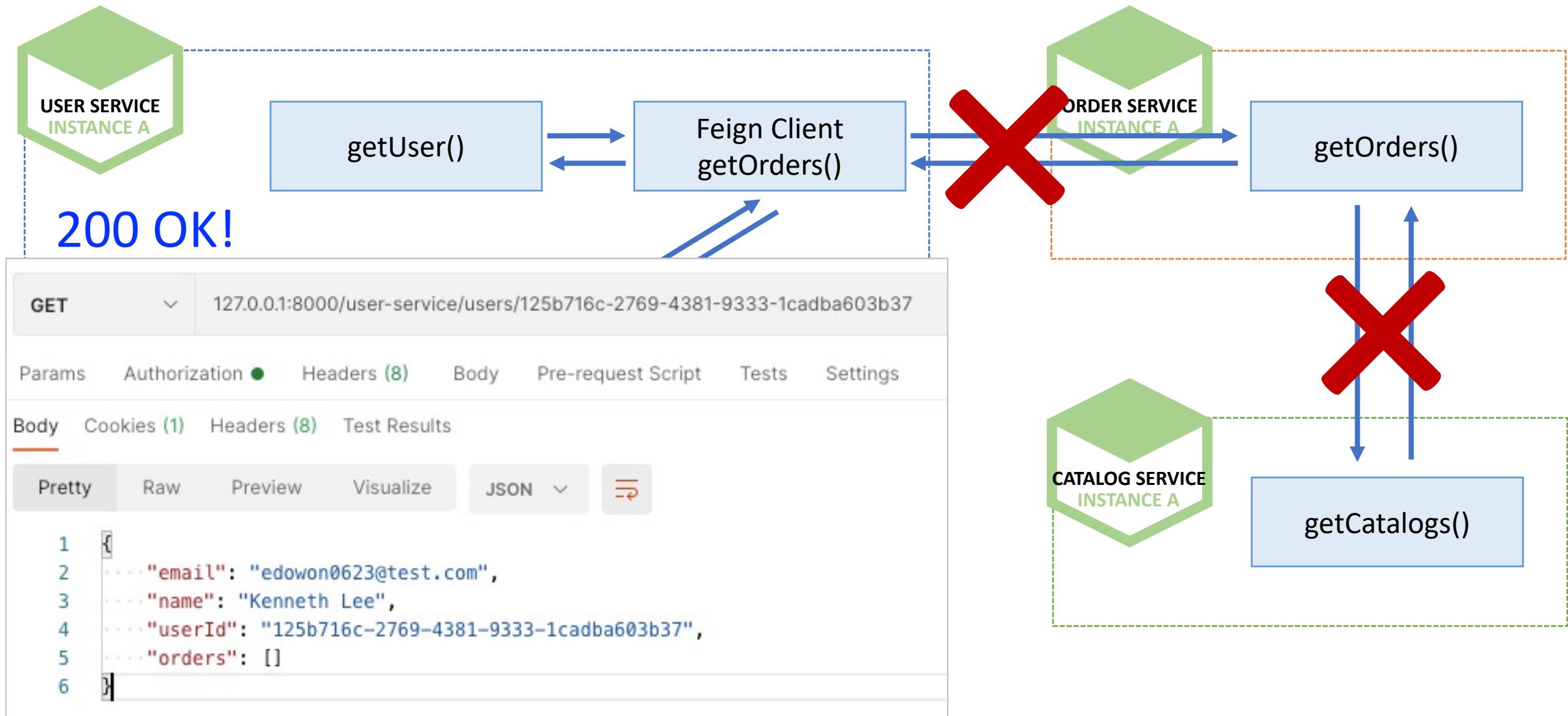
```
1 {
2   "timestamp": "2021-03-01T13:19:05.019+00:00",
3   "status": 500,
4   "error": "Internal Server Error",
5   "trace": "feign.RetryableException: order-service executing GET http://order-service/order-service/282d7347-9c0f-4b2c-b85a-282d2c27847a"
```

```
java.net.UnknownHostException Create breakpoint : order-service
    at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:567) ~[na:na]
    at java.base/java.net.Socket.connect(Socket.java:648) ~[na:na]
    at java.base/sun.net.NetworkClient.doConnect(NetworkClient.java:177) ~[na:na]
    at java.base/sun.net.www.http.HttpClient.openServer(HttpClient.java:474) ~[na:na]
```

Microservice 통신 시 연쇄 오류



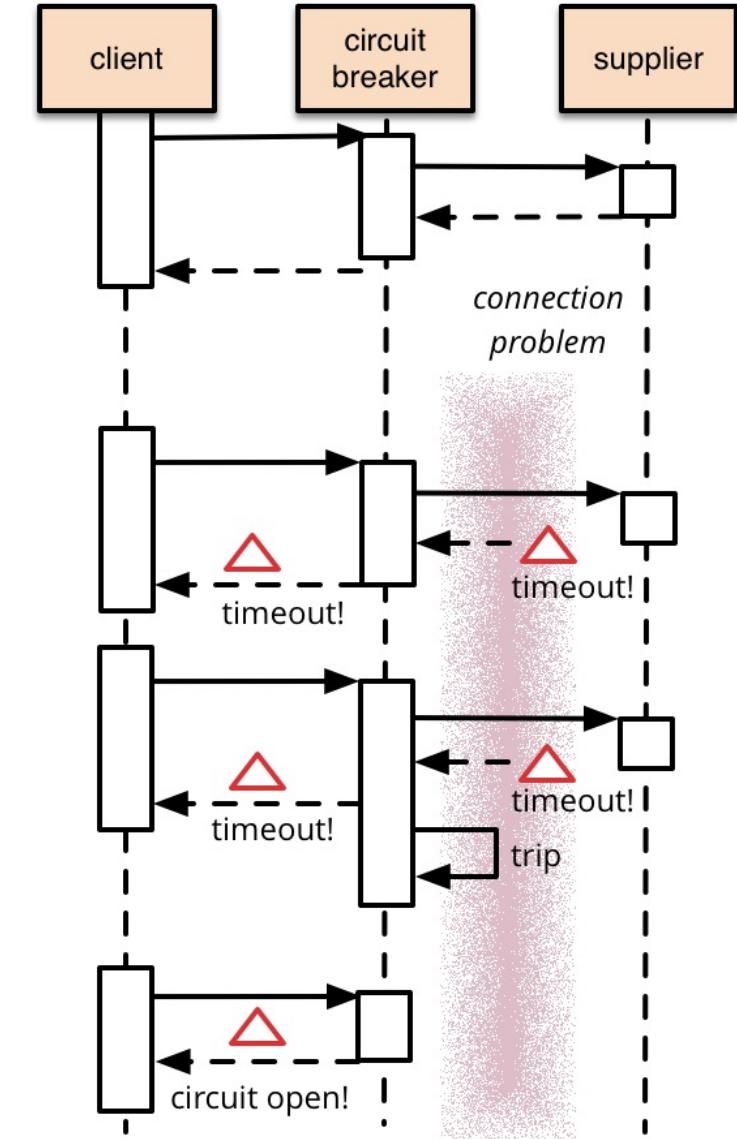
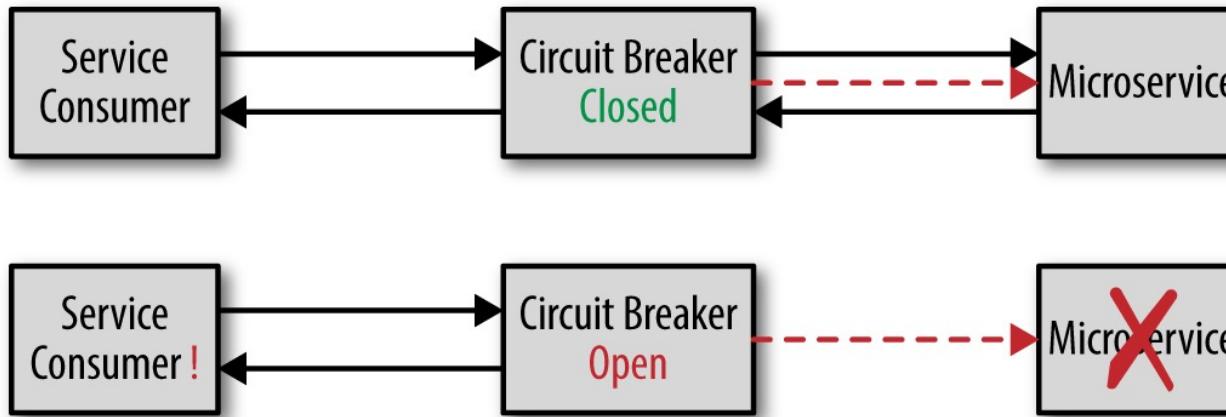
Microservice 통신 시 연쇄 오류



CircuitBreaker

■ CircuitBreaker

- <https://martinfowler.com/bliki/CircuitBreaker.html>
- 장애가 발생하는 서비스에 반복적인 호출이 되지 못하게 차단
- 특정 서비스가 정상적으로 동작하지 않을 경우 다른 기능으로 대체 수행 → 장애 회피



Spring Cloud Betflix Hystrix

■ Spring Cloud Netflix Hystrix

<dependency>		</dependency>	
CURRENT		REPLACEMENT	
	Hystrix		Resilience4j
@SpringCloudConfigurable	Hystrix Dashboard / Turbine		Micrometer + Monitoring System
@EnableDiscoveryClient	Ribbon		Spring Cloud Loadbalancer
@EnableZuulProxy	Zuul 1		Spring Cloud Gateway
public class Application {	Archaius 1		Spring Boot external config + Spring Cloud Config

Replacements technologies. Taken from [1]

er-ws)

r-ws)



Resilience4j

■ Resilience4j

- resilience4j-circuitbreaker: Circuit breaking
- resilience4j-ratelimiter: Rate limiting
- resilience4j-bulkhead: Bulkheading
- resilience4j-retry: Automatic retrying (sync and async)
- resilience4j-timelimiter: Timeout handling
- resilience4j-cache: Result caching
- <https://resilience4j.readme.io/docs/getting-started>
- <https://github.com/resilience4j/resilience4j>

Introduction

Resilience4j is a lightweight, easy-to-use fault tolerance library inspired by [Netflix Hystrix](#), but designed for Java 8 and functional programming. Lightweight, because the library only uses [Vavr](#), which does not have any other external library dependencies. Netflix Hystrix, in contrast, has a compile dependency to Archaius which has many more external library dependencies such as Guava and Apache Commons Configuration.



Resilience4j – CircuitBreaker

■ DefaultConfiguration

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-circuitbreaker-resilience4j</artifactId>
</dependency>
```

pom.xml

```
@Autowired
CircuitBreakerFactory circuitBreakerFactory;
```

```
/*
 * ErrorDecoder */
// List<ResponseOrder> ordersList = orderServiceClient.getOrders(userId);

/*
 * CircuitBreaker */
CircuitBreaker circuitBreaker = circuitBreakerFactory.create("circuitbreaker");
List<ResponseOrder> ordersList = circuitBreaker.run(() -> orderServiceClient.getOrders(userId),
    throwable -> new ArrayList<>());

userDto.setOrders(ordersList);
```

UserServiceImpl.java

Resilience4j – CircuitBreaker

■ DefaultConfiguration

The diagram illustrates the Resilience4j CircuitBreaker pattern. It shows three services: **USER SERVICE INSTANCE A**, **ORDER SERVICE INSTANCE A**, and **CATALOG SERVICE INSTANCE A**. The **USER SERVICE INSTANCE A** calls **getOrders()** via a **Feign Client**. The **ORDER SERVICE INSTANCE A** and **CATALOG SERVICE INSTANCE A** are shown with red X's over their **getOrders()** and **getCatalogs()** methods respectively, indicating they are currently unavailable. A red box highlights the **Fallback method** **getOrders()** in the **Feign Client**, which is returning a **200 OK!** response.

200 OK!

Fallback method
getOrders()

GET 127.0.0.1:8000/user-service/users/125b716c-2769-4381-9333-1cadba603b37

Params Authorization ● Headers (8) Body Pre-request Script Tests Settings

Body Cookies (1) Headers (8) Test Results

Pretty Raw Preview Visualize JSON

```
1 {  
2   ... "email": "edowon0623@test.com",  
3   ... "name": "Kenneth Lee",  
4   ... "userId": "125b716c-2769-4381-9333-1cadba603b37",  
5   ... "orders": []  
6 }
```

Resilience4j – CircuitBreaker

■ Customize CircuitBreakerFactory → *Resilience4JCircuitBreakerFactory*

```
@Configuration  
public class Resilience4JConfiguration {  
    @Bean  
    public Customizer<Resilience4JCircuitBreakerFactory> globalCustomConfiguration() {  
  
        // the circuitBreakerConfig and timeLimiterConfig objects  
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()  
            .failureRateThreshold(4)  
            .waitDurationInOpenState(Duration.ofMillis(1000))  
            .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.COUNT_BASED)  
            .slidingWindowSize(2)  
            .build();  
    }  
}
```

- CircuitBreaker를 열지 결정하는 failure rate threshold percentage
- default: 50

- CircuitBreaker를 open한 상태를 유지하는 지속 기간을 의미
- 이 기간 이후에 half-open 상태
- default: 60seconds

- CircuitBreaker가 닫힐 때 통화 결과를 기록하는 데 사용되는 슬라이딩 창의 유형을 구성
- 카운트 기반 또는 시간 기반

- CircuitBreaker가 닫힐 때 호출 결과를 기록하는 데 사용되는 슬라이딩 창의 크기를 구성
- default: 100

Resilience4j – CircuitBreaker

- Customize CircuitBreakerFactory → **Resilience4JCircuitBreakerFactory**

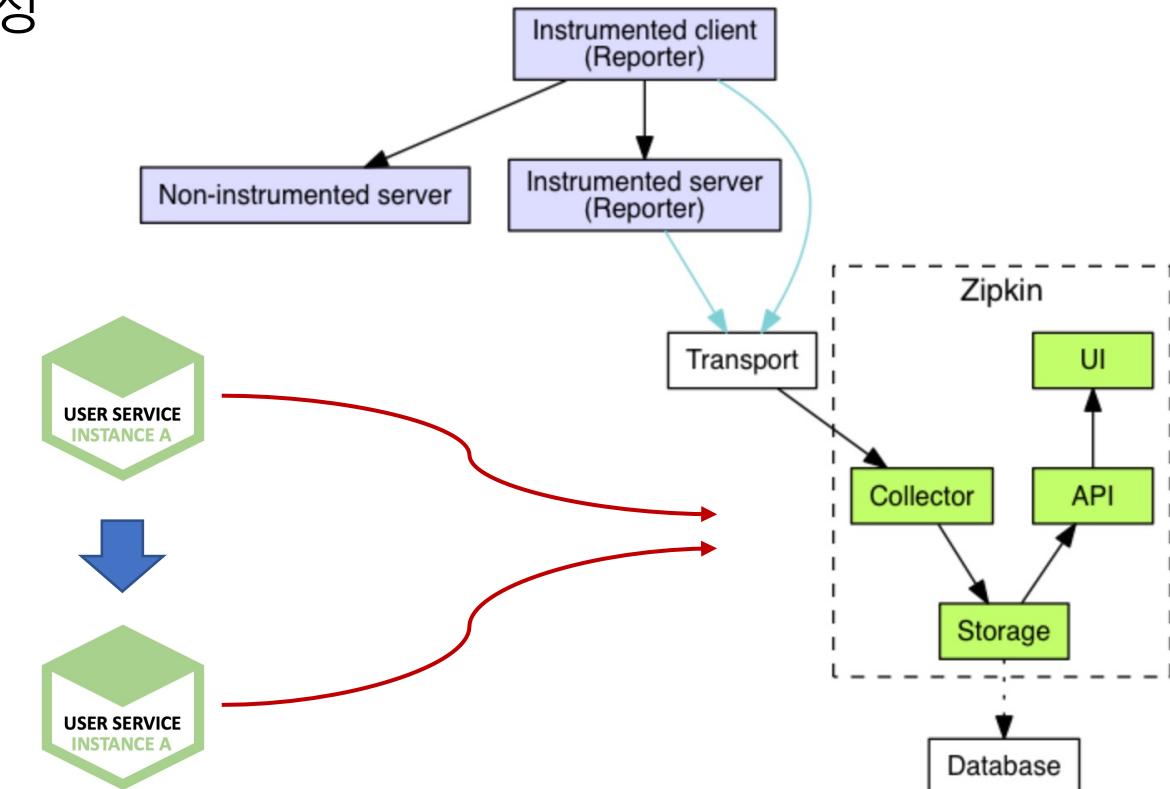
```
TimeLimiterConfig timeLimiterConfig = TimeLimiterConfig.custom()  
    .timeoutDuration(Duration.ofSeconds(4))  
    .build();  
  
return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)  
    .timeLimiterConfig(timeLimiterConfig)  
    .circuitBreakerConfig(circuitBreakerConfig)  
    .build());  
}
```

- TimeLimiter는 future supplier의 time limit을 정하는 API
- default: 1초

Microservice 분산 추적

■ Zipkin

- <https://zipkin.io/>
- Twitter에서 사용하는 분산 환경의 Timing 데이터 수집, 추적 시스템 (오픈소스)
- Google Drapper에서 발전하였으며, 분산환경에서의 시스템 병목 현상 파악
- Collector, Query Service, Databasem WebUI로 구성
- **Span**
 - 하나의 요청에 사용되는 작업의 단위
 - 64 bit unique ID
- **Trace**
 - 트리 구조로 이뤄진 Span 셋
 - 하나의 요청에 대한 같은 Trace ID 발급



Microservice 분산 추적

■ Spring Cloud Sleuth

- 스프링 부트 애플리케이션을 Zipkin과 연동
- 요청 값에 따른 Trace ID, Span ID 부여
- Trace와 Span Ids를 로그에 추가 가능
 - servlet filter
 - rest template
 - scheduled actions
 - message channels
 - feign client

Spring Cloud Sleuth 3.0.1



OVERVIEW

LEARN

SAMPLES

Spring Cloud Sleuth provides Spring Boot auto-configuration for distributed tracing.

Features

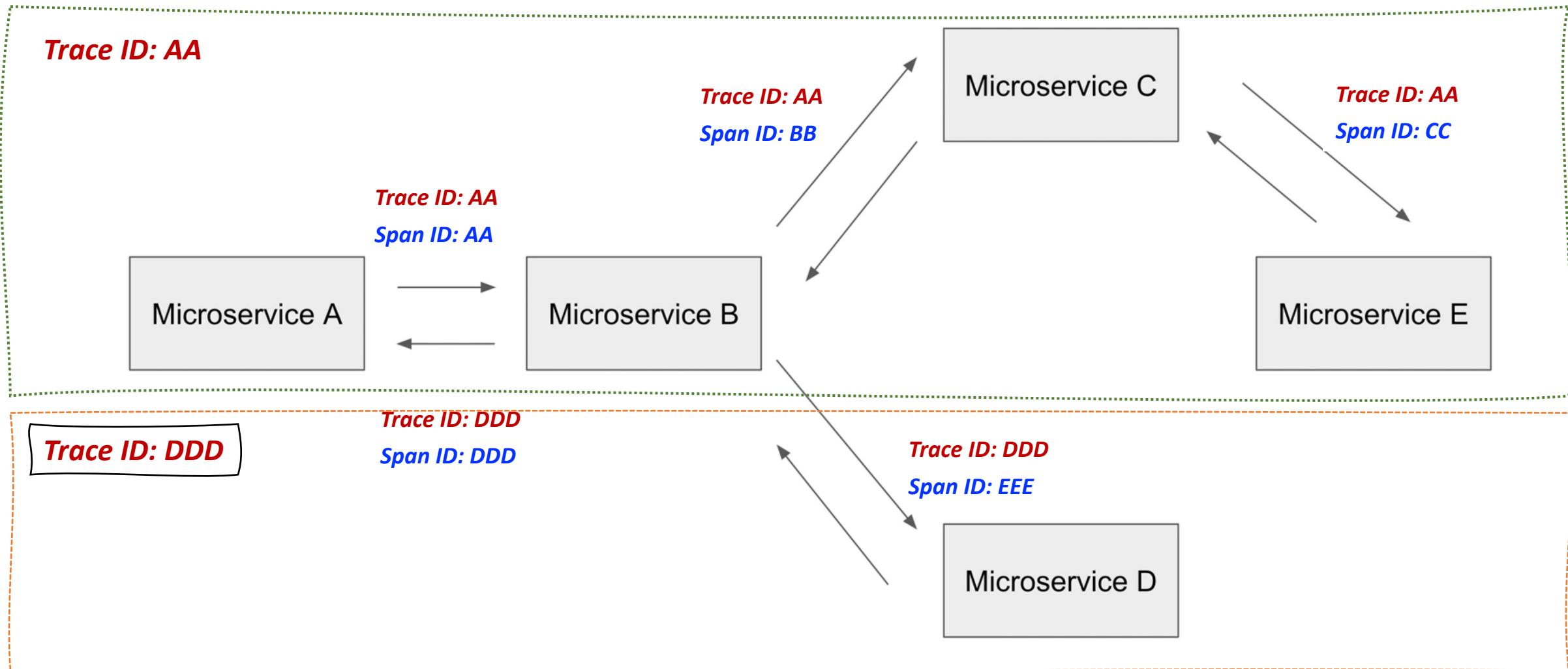
Sleuth configures everything you need to get started. This includes where trace data (spans) are reported to, how many traces to keep (sampling), if remote fields (baggage) are sent, and which libraries are traced.

Specifically, Spring Cloud Sleuth...

- Adds trace and span ids to the Slf4J MDC, so you can extract all the logs from a given trace or span in a log aggregator.
- Instruments common ingress and egress points from Spring applications (servlet filter, rest template, scheduled actions, message channels, feign client).
- If `spring-cloud-sleuth-zipkin` is available then the app will generate and report Zipkin-compatible traces via HTTP. By default it sends them to a Zipkin collector service on localhost (port 9411). Configure the location of the service using `spring.zipkin.baseUrl`.

Microservice 분산 추적

- Spring Cloud Sleuth + Zipkin



Zipkin server 설치



The screenshot shows a web browser displaying the [Zipkin quickstart page](https://zipkin.io/pages/quickstart.html). The page has a dark background with the Zipkin logo (an orange triangle icon) and the word "ZIPKIN" in large white capital letters. Below the logo are social media sharing icons for GitHub, Twitter, and LinkedIn. On the left, there are navigation links for "Home" and "Quickstart". The main content area is titled "Docker" and provides instructions for running a Docker image. It includes a command-line snippet:

```
docker run -d -p 9411:9411 openzipkin/zipkin
```

Below this, another section titled "Java" provides instructions for Java 8+ users, mentioning the "latest release" as a self-contained executable jar. It includes a command-line snippet:

```
curl -sSL https://zipkin.io/quickstart.sh | bash -s  
java -jar zipkin.jar
```

```
▶ curl -sSL https://zipkin.io/quickstart.sh | bash -s  
Thank you for trying Zipkin!  
This installer is provided as a quick-start helper, so you can try Zipkin out  
without a lengthy installation process.  
  
Fetching version number of latest io.zipkin:zipkin-server release...  
Latest release of io.zipkin:zipkin-server seems to be 2.23.2  
  
Downloading io.zipkin:zipkin-server:2.23.2:exec to zipkin.jar...
```

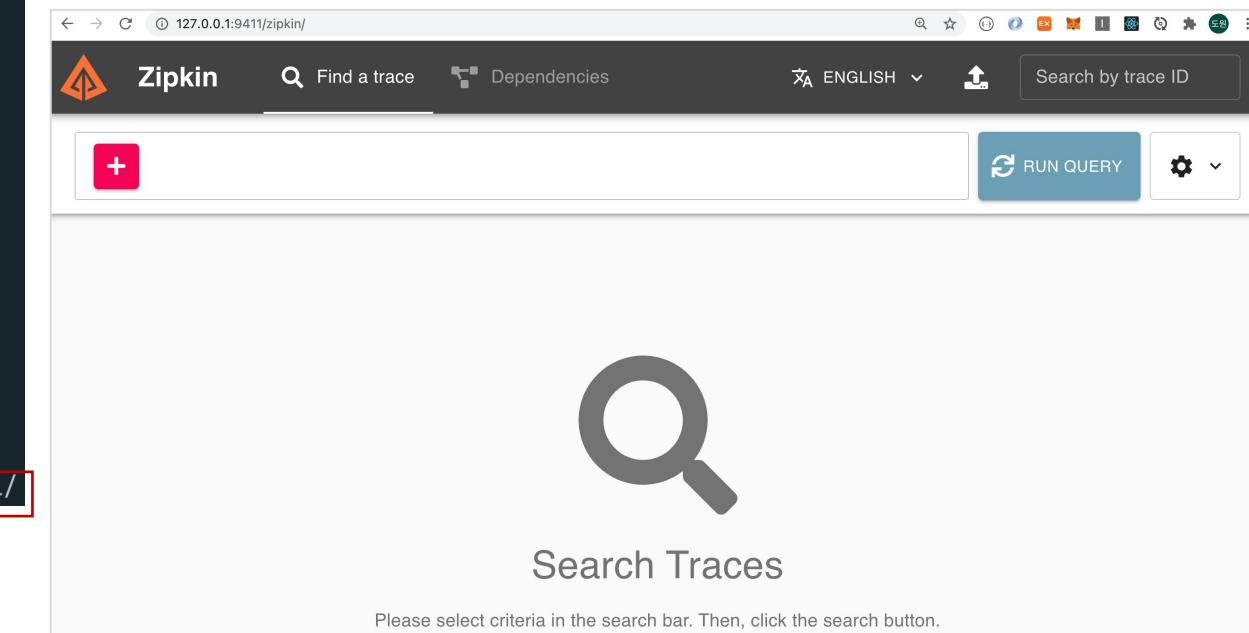
Zipkin server 기동

▶ `java -jar zipkin.jar`

00
0000
000000
00000000
000000000000
00000000000000
00000000 00000000
000000 00000000
000000 0 0 000000
000000 00 00 000000
00000000 0000 0000 00000000
000000 000000 000000 00000000
00000000 000000 000000 00000000
0000000000 00 00 0000000000

:: version 2.23.2 :: commit 7bf3aab ::

```
2021-03-02 01:14:25.648 INFO [/] 87530 --- [oss-http-*:9411] c.l.a.s.Server : Serving HTTP at [/0:0:0:0:0:0:0:0]:9411 - http://127.0.0.1
```



Users Microservice 설정

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
    <version>2.2.3.RELEASE</version>
</dependency>
```

pom.xml

```
spring:
  application:
    name: user-service
  zipkin:
    base-url: http://localhost:9411
    enabled: true
  sleuth:
    sampler:
      probability: 1.0
```

application.yml

```
/* CircuitBreaker */
log.info("Before call orders microservice");
CircuitBreaker circuitBreaker = circuitBreakerFactory.create("my_circuit_breaker");
List<ResponseOrder> ordersList = circuitBreaker.run(() -> orderServiceClient.getOrders(userId),
    throwable -> new ArrayList<>());
log.info("after called orders microservice");

userDto.setOrders(ordersList);
```

UsersServiceImpl.java

Trace ID and Span ID

POST ▼ http://127.0.0.1:8000/order-service/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c/orders

Params Authorization ● Headers (11) **Body** ● Pre-request Script Tests Settings

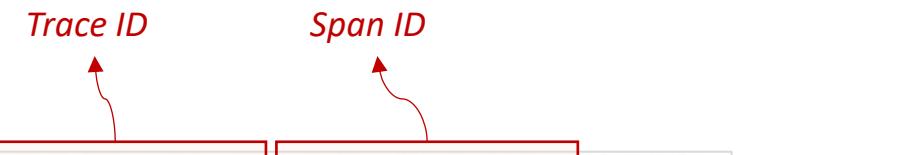
none form-data x-www-form-urlencoded raw binary GraphQL **JSON** ▾

```
1 {  
2   ... "productId": "CATALOG-003",  
3   ... "qty": 5,  
4   ... "unitPrice": 1500  
5 }
```

Body Cookies (1) Headers (3) Test Results

Pretty Raw Preview Visualize JSON ▾

```
2021-03-02 23:47:35.314 INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 -->  
Hibernate: call next value for hibernate_sequence  
Hibernate: insert into orders (order_id, product_id, qty, total_price, unit_price, user_id, id) values (?, ?, ?, ?, ?, ?, ?)  
2021-03-02 23:47:35.413 INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 -->  
[o-auto-1-exec-1] c.e.o.controller.OrdersController : Before add orders data  
[o-auto-1-exec-1] c.e.o.controller.OrdersController : After added orders data
```



Trace ID and Span ID

The screenshot shows the Zipkin UI interface. At the top, there is a navigation bar with the Zipkin logo, a search bar labeled "Find a trace", a "Dependencies" button, a language selector "ENGLISH", and a "Search by trace ID" input field containing the trace ID "a04b6cf3c9338a13". A red box highlights this search field.

The main content area displays a trace for the service "ORDER-SERVICE" with the operation "post /order-service/{userid}/orders". The total duration is 153.720ms, and there is one span. The span details are shown in a box with a red border:

- Duration: 153.720ms
- Services: 1
- Depth: 1
- Total Spans: 1
- Trace ID: **a04b6cf3c9338a13**
- Span ID: a04b6cf3c9338a13 Parent ID: None

Below this, the span timeline shows four segments with the following durations:

- 0ms
- 51.240ms
- 102.480ms
- 153.720ms

The first segment is highlighted with a green background, and the entire span is labeled "ORDER-SERVICE post /order-service/{userid}/orders [153.720ms]".

On the right side, there is a detailed view of the span:

- Annotations:** A green horizontal bar representing the span's duration.
- Tags:** A list of key-value pairs:
 - http.method: POST
 - http.path: /order-service/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c/orders
 - mvc.controller.class: OrdersController
 - mvc.controller.method: createOrder

At the bottom left, a red arrow points from the text "Trace ID" to the green-highlighted part of the log output, which contains the trace ID "a04b6cf3c9338a13".

```
2021-03-02 23:47:35.314 INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
Hibernate: call next value for hibernate_sequence
Hibernate: insert into orders (order_id, product_id, qty, total_price, unit_price, user_id) values (?, ?, ?, ?, ?, ?)
2021-03-02 23:47:35.413 INFO [order-service,a04b6cf3c9338a13,a04b6cf3c9338a13] 96789 --
```

Trace ID and Span ID

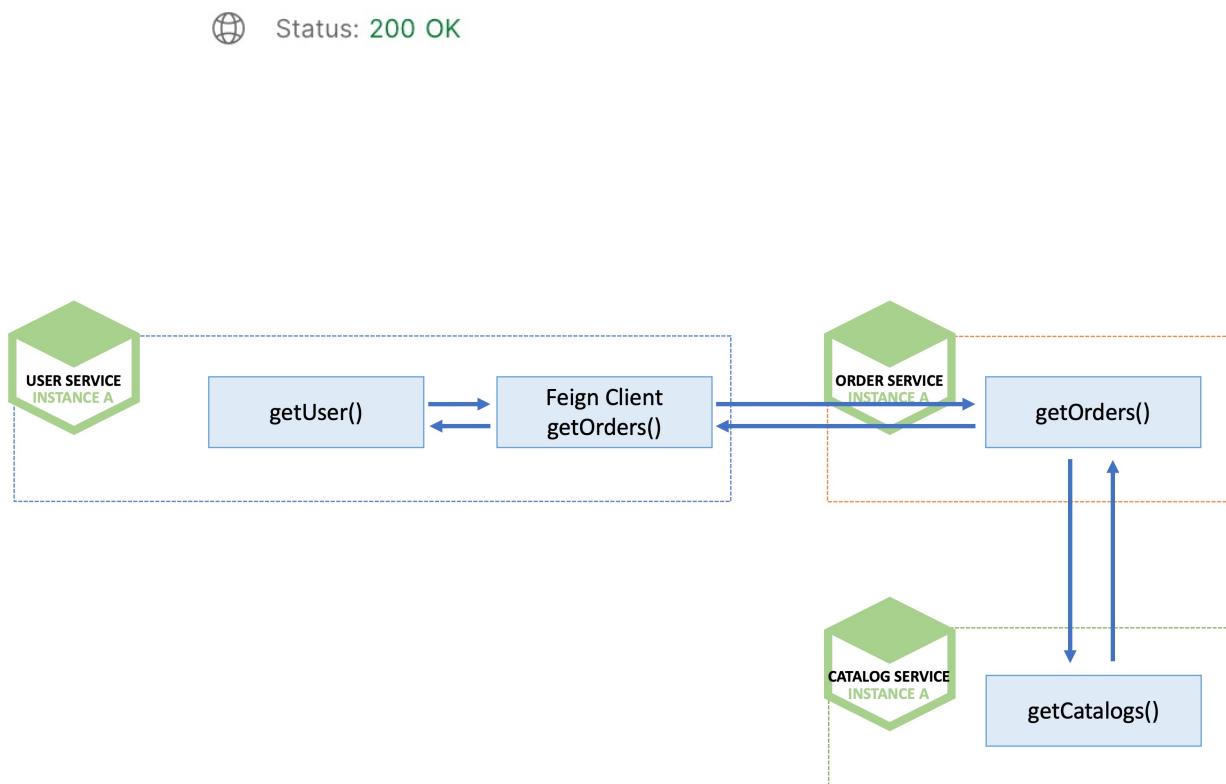
GET 127.0.0.1:8000/user-service/users/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c

Params Authorization ● Headers (8) Body Pre-request Script Tests Settings

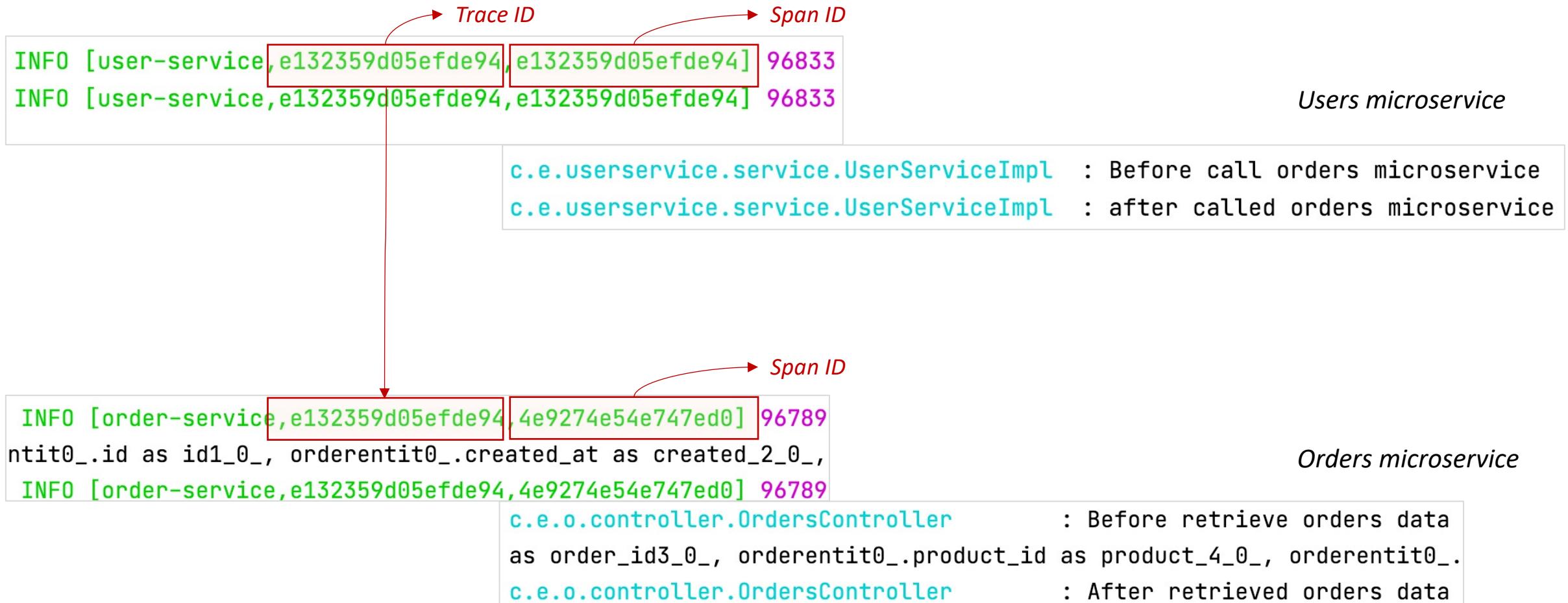
Body Cookies (1) Headers (8) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON ↻

```
1 {
2   "email": "edowon0623@test.com",
3   "name": "Kenneth Lee",
4   "userId": "a09741d4-b5c4-44fb-a9a6-8a6312e9b58c",
5   "orders": [
6     {
7       "productId": "CATALOG-004",
8       "qty": 10,
9       "unitPrice": 1500,
10      "totalPrice": 15000,
11      "createdAt": "2021-03-02T14:47:35.406+00:00",
12      "orderId": "73b93f3d-8dee-4bd0-9a23-05d465be75f9"
13    },
  ]
```



Trace ID and Span ID



Zipkin server 활용

Zipkin Find a trace Dependencies ENGLISH e132359d05efde94

USER-SERVICE: get /users/{userid}

Duration: 3.462s Services: 2 Depth: 4 Total Spans: 4 Trace ID: e132359d05efde94

DOWNLOAD JSON

USER-SERVICE

0ms 1.154s 2.308s 3.462s

get /users/{userid} [3.462s]

userserviceimpl\$\$lambda\$1778/0x00000008016cc040 [3.276s]

get [3.147s]

get /order-service/{userid}/orders [3.139s]

» ORDER-SERVICE

get /order-service/{userid}/orders

Span ID: 4e9274e54e747ed0 Parent ID: 585681926fe50087

Annotations

SHOW ALL ANNOTATIONS

Tags

- http.method GET
- http.path /order-service/a09741d4-b5c4-44fb-a9a6-8a6312e9b58c/orders
- mvc.controller.class OrdersController
- mvc.controller.method getOrder

Zipkin server 활용

The screenshot shows the Zipkin web interface. At the top, there is a navigation bar with the Zipkin logo, a search bar labeled "Find a trace", a "Dependencies" button, a language switcher ("ENGLISH"), and a "Search by trace ID" input field. Below the search bar, there is a filter bar with a "serviceName" dropdown set to "user-service", a red "X" button, and a red "+" button. To the right of the filter bar are "RUN QUERY" and settings buttons.

The main area displays "2 Results". There are two trace entries:

Root	Start Time	Spans	Duration	Action
user-service: get /users/{userid}	13 minutes ago (03/02 23:48:17:274)	4	3.462s	SHOW
user-service: post /users	14 minutes ago (03/02 23:47:24:587)	1	278.307ms	SHOW

Zipkin 에서의 추적

Zipkin

Find a trace

Dependencies

ENGLISH

Search by trace ID

Start Time: 03/02/2021 00:01:41

End Time: 03/03/2021 01:02:41

Select...

The graph displays two service nodes: 'user-service' (blue circle) and 'order-service' (blue circle). A single directed edge connects the 'user-service' node to the 'order-service' node.

user-service

TRACES

Uses (traced requests)

This service uses 1 service

Service Name	Call	Error
order-service	4	0

Orders Microservice 장애 발생

```
@GetMapping(value="/{userId}/orders")
public ResponseEntity<List<ResponseOrder>> getOrder(@PathVariable("userId") String userId) throws Exception {
    log.info("Before retrieve orders data");
    Iterable<OrderEntity> orderList = ordersService.getOrdersByUserId(userId);

    List<ResponseOrder> result = new ArrayList<>();
    orderList.forEach(v -> {
        result.add(new ModelMapper().map(v, ResponseOrder.class));
    });

    try {
        Thread.sleep( millis: 3000);

        throw new Exception("장애 발생!");
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    log.info("After retrieved orders data");

    return ResponseEntity.status(HttpStatus.OK).body(result);
}
```

Orders Microservice 장애 발생



Orders Microservice 장애 발생

USER-SERVICE: get /users/{userid}

Duration: 3.387s Services: 2 Depth: 4 Total Spans: 5 Trace ID: 6c84bb1c6c2a1ed6

DOWNLOAD JSON

Annotations

Tags

error
Request processing failed; nested exception is java.lang.Exception: 장애 발생!

http.method
GET

http.path
/order-service/352e4ef3-40d8-48d3-be57-37a99aaa7e0c/orders

Select...

order-service

Used (traced requests)

TRACES

This service is used by 1 service

Service Name	Call	Error
user-service	1	1