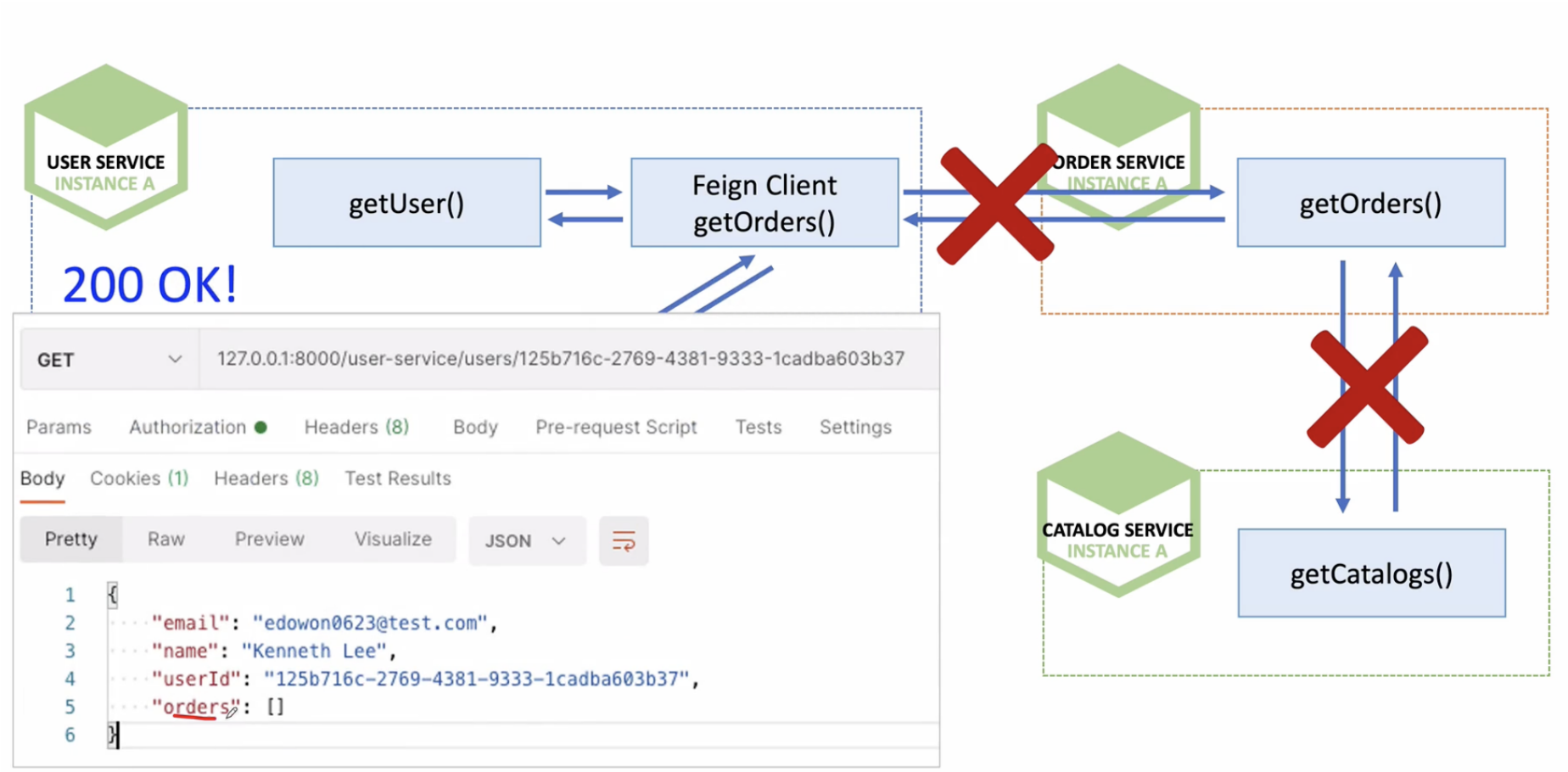
# MSA 16주차

| | |
|---|---|
| 스터디원 | 소현 |
| Status | Done |
| URL | https://github.com/SPRING-STUDY-2023/sohyeon-spring-cloud-msa/pull/10 |
| 비고 | MSA Section13 |
| 제출 마감일 | @December 30, 2023 |

## 장애 처리와 Microservice 분산 추적

### CircuitBreaker와 Resilience4J의 사용

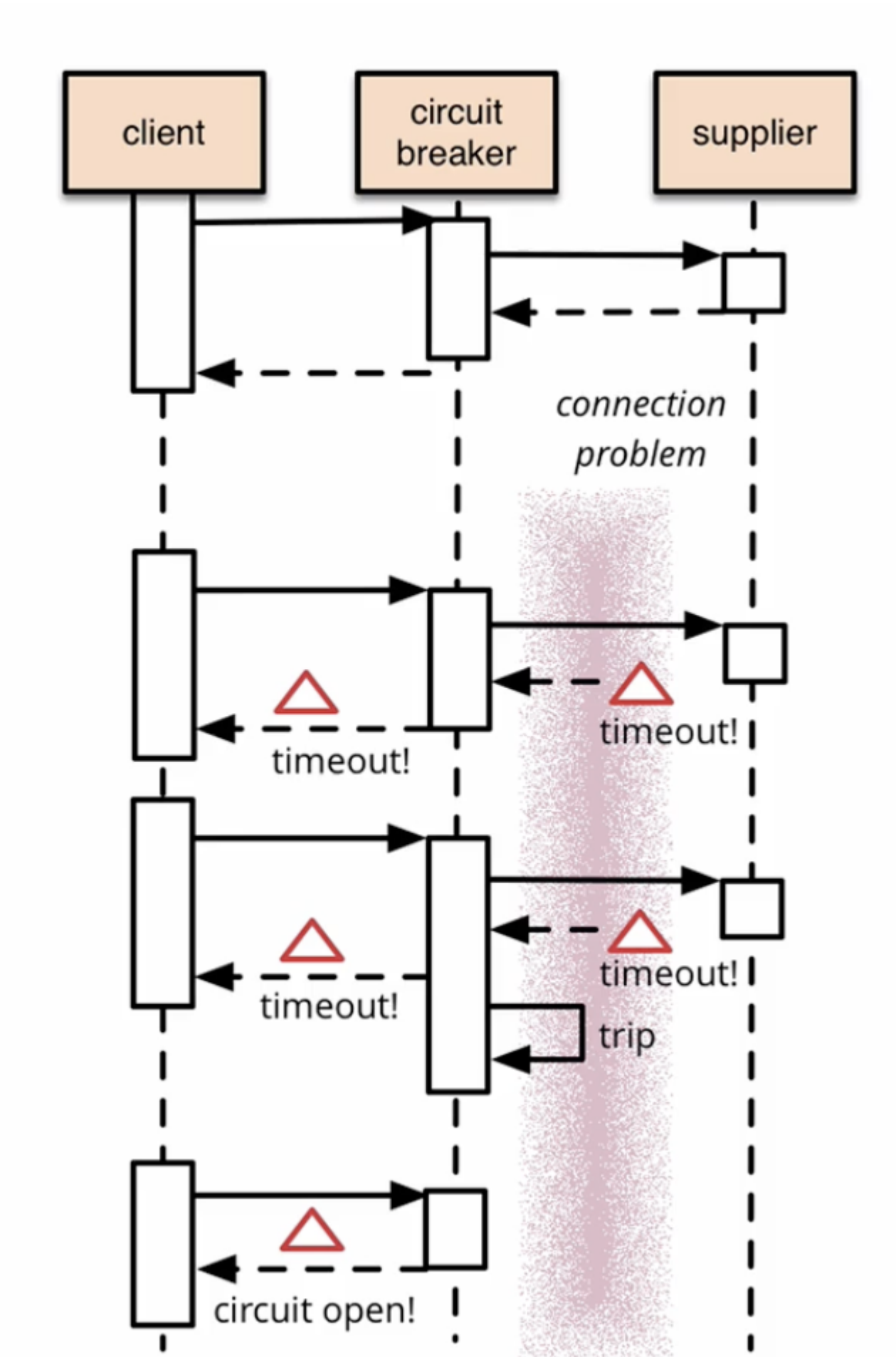
#### Microservice 통신 시 연쇄 오류




#### CircuitBreaker

- 장애가 발생하는 서비스에 반복적인 호출이 되지 못하게 차단
- 특정 서비스가 정상적으로 동작하지 않을 경우 다른 기능으로 대체 수행 → 장애 회피

# Spring Cloud Netflix Hystrix



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

*pom.xml (User-ws)*

```java
@SpringBootApplication
@EnableDiscoveryClient
@EnableFeignClients
@EnableCircuitBreaker
public class PhotoappapiusersApplication {
```

*Application.java (User-ws)*

```yaml
feign:
  hystrix:
    enabled: true
```

*application.yml (User-ws)*

이전 방식

| CURRENT | REPLACEMENT |
| --- | --- |
| Hystrix | Resilience4j |
| Hystrix Dashboard / Turbine | Micrometer + Monitoring System |
| Ribbon | Spring Cloud Loadbalancer |
| ~~Zuul 1~~ | Spring Cloud Gateway |
| Archaius 1 | Spring Boot external config + Spring Cloud Config |

Replacements technologies. Taken from [1]

대체 방식

### Resilience4j

- CircuitBreaker 기능 지원 (장애 회피)

## Users Microservice에 CircuitBreaker 적용

### 사용자 조회 API 호출 (Order-Service 실행되지 않은 상태)

GET | localhost:8000/user-service/users/7a2a95c1-d7db-4c4c-bba4-fdfcd567a1b4 | Send

Params | Authorization | Headers (1) | Body | Pre-request Script | Tests | Settings | Cookies

Body | Cookies | Headers (8) | Test Results | 500 Internal Server Error | 766 ms | 16.73 KB | Save as example

Pretty | Raw | Preview | Visualize | JSON


```
{
    "timestamp": "2023-12-29T07:15:43.875+00:00",
    "status": 500,
    "error": "Internal Server Error",
    "trace": "feign.RetryableException: order-service executing GET http://order-service/order-service/
        7a2a95c1-d7db-4c4c-bba4-fdfcd567a1b4/orders\n\tat feign.FeignException.errorExecuting(FeignException.
        java:277)\n\tat feign.SynchronousMethodHandler.executeAndDecode(SynchronousMethodHandler.java:110)
        \n\tat feign.SynchronousMethodHandler.invoke(SynchronousMethodHandler.java:70)\n\tat feign.
        ReflectiveFeign$FeignInvocationHandler.invoke(ReflectiveFeign.java:96)\n\tat jdk.proxy3/jdk.proxy3.
        $Proxy181.getOrders(Unknown Source)\n\tat com.example.userservice.service.UserServiceImpl.
        getUserByUserId(UserServiceImpl.java:87)\n\tat com.example.userservice.controller.UserController.
        getUser(UserController.java:75)\n\tat java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0
        (Native Method)\n\tat java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke
        (NativeMethodAccessorImpl.java:77)\n\tat java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.
        invoke(DelegatingMethodAccessorImpl.java:43)\n\tat java.base/java.lang.reflect.Method.invoke(Method.
```


## User-Service 수정

- **dependency 추가**

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

- **CircuitBreaker 추가**

```
/* Using a feign client with ErrorDecoder */
// List<ResponseOrder> orders = orderServiceClient.getOrders(userId);
CircuitBreaker circuitbreaker = circuitBreakerFactory.create("circuitbreaker");
List<ResponseOrder> orders = circuitbreaker.run(() -> orderServiceClient.getOrders(userId),
        throwable -> new ArrayList<>());
```

- **config 추가**

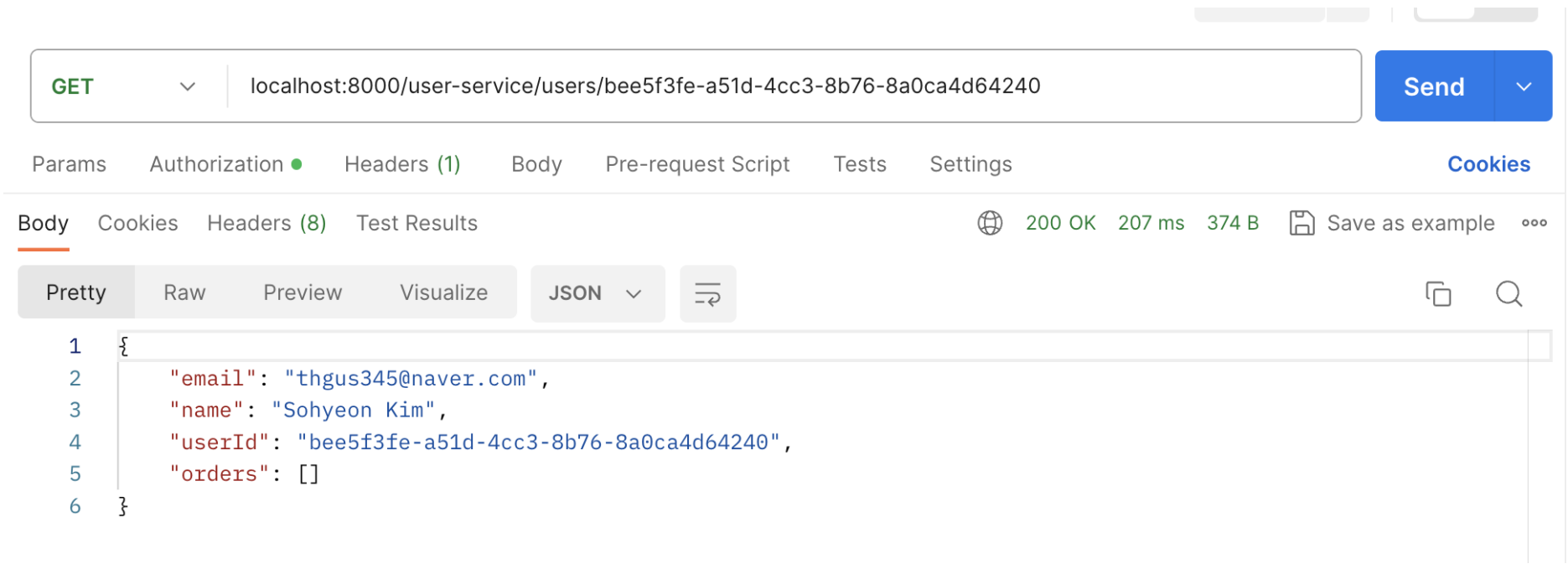
```
@Configuration
public class Resilience4j {
    @Bean
    public Customizer<Resilience4JCircuitBreakerFactory> globalCustomConfiguration() {
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
            .failureRateThreshold(4)
            .waitDurationInOpenState(Duration.ofMillis(1000))
            .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.COUNT_BASED)
            .slidingWindowSize(2)
            .build();

        TimeLimiterConfig timeLimiterConfig = TimeLimiterConfig.custom()
            .timeoutDuration(Duration.ofSeconds(4))
            .build();

        return factory -> factory.configureDefault(id -> new Resilience4JConfigBuilder(id)
            .timeLimiterConfig(timeLimiterConfig)
```

```
                .circuitBreakerConfig(circuitBreakerConfig)
                .build());
    }
}
```

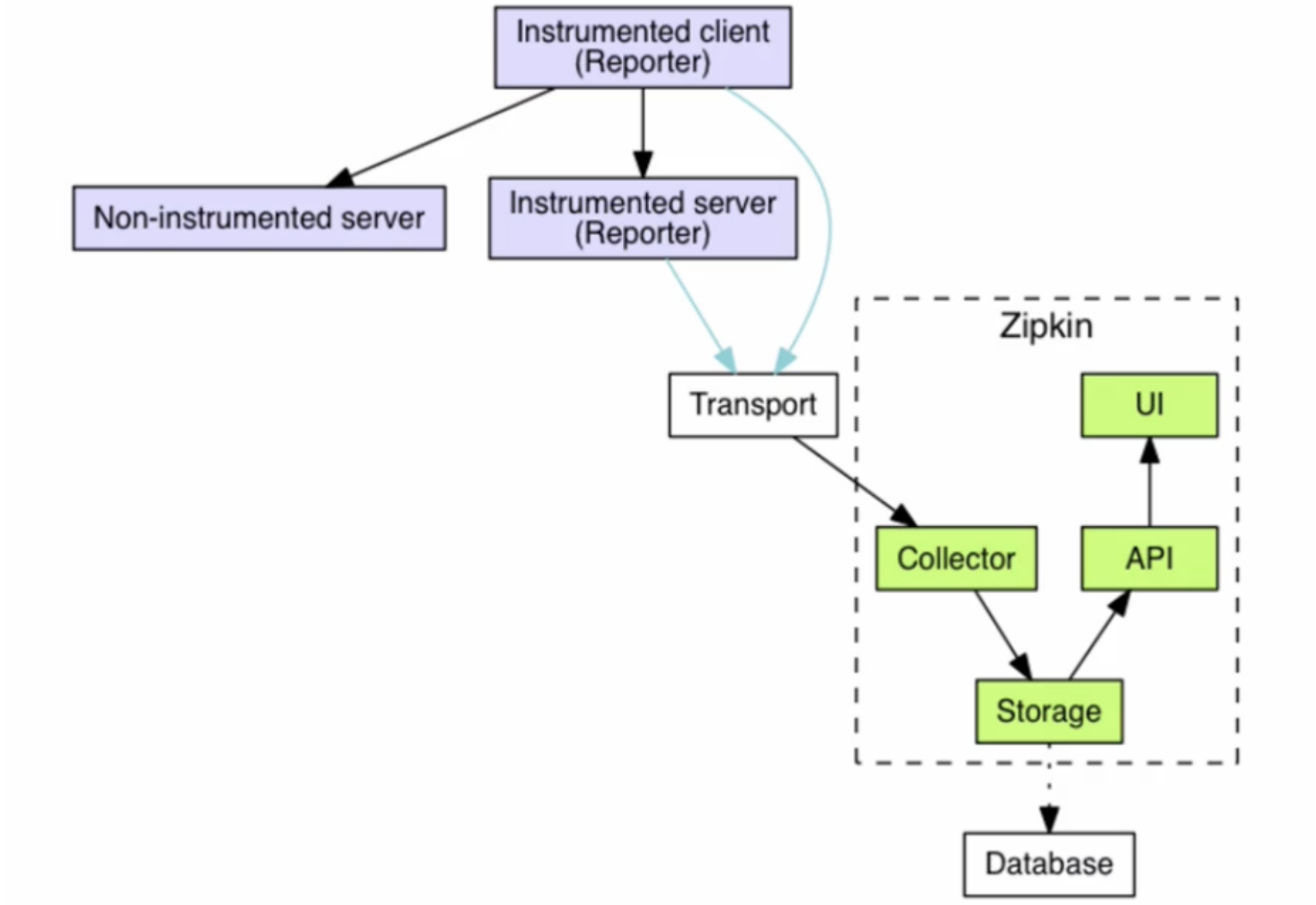
### 사용자 조회 API 호출 (Order-Service 실행 X, CircuitBreaker 추가된 상태)




## 분산 추적의 개요 Zipkin 서버 설치

### Zipkin

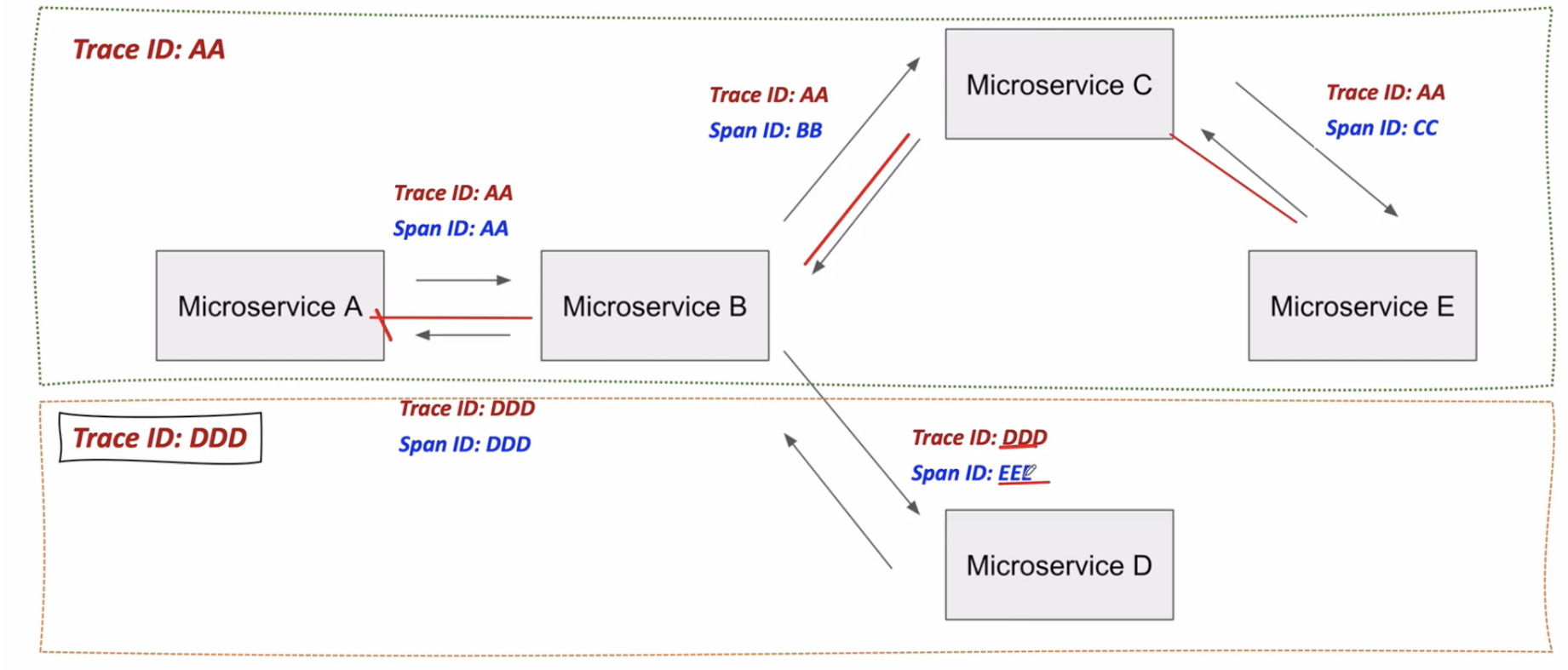
- Twitter에서 사용하는 분산 환경의 Timing 데이터 수집, 추적 시스템 (오픈 소스)
- Google Draper에서 발전, 분산 환경에서의 시스템 병목 현상 파악
- Collector, Query Service, Databasem WebUI로 구성
- Span
  - 하나의 요청에 사용되는 작업의 단위
  - 64 bit unique ID
- Trace
  - 트리 구조로 이루어진 span 셋
  - 하나의 요청에 대한 같은 Trace ID 발급




## 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 + Zipkin




### Zipkin server 설치

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

### Zipkin server 실행

```
$ java -jar zipkin.jar
```

## Spring Cloud Sleuth + Zipkin을 이용한 Microservice의 분산 추적

### Users Microservice 수정

- **dependency 추가**

```
<!-- zipkin -->
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-sleuth</artifactId>
    <version>2.2.8.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-zipkin</artifactId>
    <version>2.2.8.RELEASE</version>
</dependency>
```

- **application 값 추가**

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

- **log 추가**

```
...
public class UserServiceImpl implements UserService {
    ...

    @Override
    public UserDto getUserByUserId(String userId) {
        ...

        /* Using a feign client with ErrorDecoder */
        // List<ResponseOrder> orders = orderServiceClient.getOrders(userId);
        log.info("Before call orders microservice");
        CircuitBreaker circuitbreaker = circuitBreakerFactory.create("circuitbreaker");
        List<ResponseOrder> orders = circuitbreaker.run(() -> orderServiceClient.getOrders(us
            throwable -> new ArrayList<>());
        log.info("After call orders microservice");

        ...
    }
}
```

## Orders Microservice 수정

- **dependency 추가**

```
// https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-starter-sleuth
implementation group: 'org.springframework.cloud', name: 'spring-cloud-starter-sleuth', versi

// https://mvnrepository.com/artifact/org.springframework.cloud/spring-cloud-sleuth-zipkin
implementation group: 'org.springframework.cloud', name: 'spring-cloud-sleuth-zipkin', versio
```

- **application 값 추가**

  → Users Microservice의 application.yml 추가 사항과 동일

- **log 추가**

```
...
public class OrderController {
    ...

    @PostMapping("/{userId}/orders")
    public ResponseEntity<ResponseOrder> createOrder(@PathVariable String userId, @RequestBod
        log.info("Before add orders data");
        ...
        log.info("After add orders data");
        return ResponseEntity.status(HttpStatus.CREATED).body(responseOrder);
    }

    @GetMapping("/{userId}/orders")
    public ResponseEntity<List<ResponseOrder>> getOrders(@PathVariable String userId) {
        log.info("Before retrieve orders data");
        ...
        log.info("Add retrieved orders data");
        return ResponseEntity.ok(result);
    }
}
```

## 출력된 Log 확인


- Span ID, Trade ID를 Zipkin 대시보드(localhost:9411)에 입력하여 기록 모니터링 확인 가능

## PR

https://github.com/SPRING-STUDY-2023/sohyeon-spring-cloud-msa/pull/10