

Service Mesh

- 각 서비스 마이크로서비스 간 통신을 최적화 하고 다운타임을 방지하며 전체 서비스를 관리하는 Outer Architecture 를 Service Mesh 라고 한다.
- Service Mesh 는 복잡한 내부 네트워크를 추상화를 통해 서비스 간의 통신이 빠르고, 안정적이며, 신뢰성을 보장한다.

Service Mesh

- Configuration Management : 설정변경 시 서비스의 재빌드와 재부팅 없이 즉시 반영
- Service Discovery : API Gateway 가 서비스를 검색하는 매커니즘
- Load Balancing : 서비스간 부하 분산
- API Gateway : API 서버 앞단에서 API 엔드포인트 단일화 및 인증, 인가, 라우팅 기능 담당
- Centralized Logging : 서비스별 로그의 중앙집중화
- Centralized Metrics : 서비스별 메트릭 정보의 중앙집중화
- Distributed Tracing : 서비스간 호출 추적과 성능, 분석 관리
- Resilience & Fault Tolerance : 서비스간 장애 전파 차단
- Auto Scaling & Self Healing : 자동 스케일아웃과 복구 자동화
- Packaging, Deployment & Scheduling : 패키징, 빌드 및 배포 자동화
- Test Automation : 서비스 테스트 자동화

Spring Cloud 기반 Service Mesh 구축

- Spring Cloud 는 애플리케이션 스택의 일부로 모든 MSA 관심사를 해결하도록 잘 통합된 다양한 자바 라이브러리들의 묶음
- 구성관리(Spring Cloud Config)
- Service Discovery(Eureka)
- Circuit Breakers(Resilience4j)
- 지능형 라우팅(Spring Cloud Gateway) 등 기능 제공

Spring Cloud

- Spring Cloud 는 개발자가 분산 시스템 구성에 필요한 다양한 기능(설정 관리 및 공유, 서비스 등록 및 관리, 서비스 요청 라우팅 등)을 제공한다.

Spring Cloud

| 서비스 | 설명 | 컴포넌트 |
|-----------------------|--|------------------------------------|
| Config 서비스 | 별도의 통합된 설정 관리 서비스 제공을 통해 환경 독립적 서비스 제공 | Spring Config |
| Service Discovery | 서비스 서비스에 대한 물리적 위치 정보 대신 논리적 서비스 위치 정보 제공 | Eureka (Spring Cloud Netflix) |
| Event Bus 서비스 | 분산 메시징 지원을 위한 서비스 연계 지원 | Spring Cloud Bus (AMQP & RabbitMQ) |
| Circuit Breaker 서비스 | 서비스 간 호출 시, 문제가 있는 서비스에 대한 차단 지원 서비스 | Spring Cloud Circuit Breaker |
| Client Load Balancing | 서비스 호출 시에 분산 형태로 호출 할 수 있는 client 적용 서비스 library | Spring Cloud Load Balancer |
| Service Router 서비스 | 서비스 호출 시, routing 을 통해 실제 서비스에 위치 제공 | Spring Cloud Gateway |
| API Gateway 서비스 | Microservice 에 대한 API 관리 및 모니터링 서비스 | Spring Cloud Gateway |
| Security 서비스 | Load balanced 환경에서의 OAuth2 인증 지원 서비스 | Spring Cloud Security |

OpenFeign

- OpenFeign은 마이크로서비스 아키텍처에서 다른 서비스와의 HTTP 통신을 매우 쉽게 만들 수 있는 도구
- Spring Cloud 환경에서 많이 사용되며, 마이크로서비스 간의 효율적인 통신을 지원
- OpenFeign은 다른 서비스의 API를 호출하는 클라이언트 역할을 수행
- @RestController는 클라이언트의 요청을 처리하는 서버 역할을 수행

OpenFeign

Portal-service - gradle

```
implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'org.springframework.boot:spring-boot-starter-web'
implementation 'org.springframework.boot:spring-boot-starter-actuator'
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
implementation 'org.springframework.cloud:spring-cloud-starter-config' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.springframework.cloud:spring-cloud-starter-sleuth'
implementation 'org.springframework.cloud:spring-cloud-starter-zuul'
implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'
implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-resilience4j'
implementation 'org.apache.tomcat.embed:tomcat-annotations-api:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-core:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-websocket:9.0.73'
implementation 'net.logstash.logback:logstash-logback-encoder:7.4' // logstash logback
implementation 'commons-io:commons-io:2.13.0'
implementation 'commons-net:commons-net:3.9.0' // FTPClient
implementation 'mysql:mysql-connector-java:8.0.33'
implementation 'io.jsonwebtoken:jjwt:0.9.1'
```

@EnableFeignClients

```
@ComponentScan(basePackages={"org.egovframe.cloud.common", "org.egovframe.cloud.serv
@EntityScan({"org.egovframe.cloud.servlet.domain", "org.egovframe.cloud.portalservic
@EnableFeignClients
@EnableDiscoveryClient
@SpringBootApplication
public class PortalServiceApplication {

    Run | Debug
    public static void main(String[] args) {
        SpringApplication.run(primarySource:PortalServiceApplication.class, args);
    }
}
```

@EnableFeignClients : OpenFeign을 활성화하도록 지정

OpenFeign

@FeignClient

```
@FeignClient(value = "board-service", configuration = CustomFeignConfiguration.class)
public interface BoardApiClient {
    /**
     * 게시판 한건 조회
     *
     * @param boardNo
     * @return
     */
    @GetMapping("/api/v1/boards/{boardNo}")
    BoardResponseDto findById(@PathVariable("boardNo") Integer boardNo);
}
```

Board-service - BoardApiController

```
/**
 * 게시판 단건 조회
 *
 * @param boardNo 게시판 번호
 * @return BoardResponseDto 게시판 상세 응답 DTO
 */
@GetMapping("/api/v1/boards/{boardNo}")
public BoardResponseDto findById(@PathVariable Integer boardNo) {
    return boardService.findById(boardNo);
}
```


Spring WebFlux

- 비동기 및 논블로킹(non-blocking) 방식으로 웹 애플리케이션을 개발할 수 있게 해주는 리액티브 프로그래밍 모델을 기반으로 하는 Spring Framework의 모듈
- Spring WebFlux는 특히 대규모의 동시성 처리가 필요한 웹 애플리케이션, 예를 들어 실시간 데이터 스트리밍이나 채팅 애플리케이션, 또는 고성능 API 서버 등에 적합
- Mono는 0 또는 1개의 데이터를 비동기적으로 처리하는 데 사용되고, Flux는 0부터 무한대의 데이터를 처리할 수 있는 스트림을 제공

Spring WebFlux

Reserve-check-service – build.gradle

```
implementation 'org.springframework.boot:spring-boot-starter-data-r2dbc'
implementation 'org.springframework.boot:spring-boot-starter-webflux'
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-actuator'
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
implementation 'org.springframework.cloud:spring-cloud-starter-config' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-reactor-resilience4j'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'com.playtika.reactivefeign:feign-reactor-spring-cloud-starter:3.2.11'
```

ReserveService.java

```
/**
 * 예약 물품별 기간안에 있는 예약된 수량 max 조회
 */
@Param(reserveItemId)
@Param(startDate)
@Param(endDate)
@Return
public Mono<Integer> countInventory(Long reserveItemId, LocalDateTime startDate,
LocalDateTime endDate) {
    // ...
    .transform(CircuitBreakerOperator.of(circuitBreakerRegistry.circuitBreaker(RESERVE_ITEM_CIRCUIT_BREAKER_NAME)))
    .onErrorResume(throwable -> Mono.empty())
    .zipWith(validator.getMaxByReserveDate(reserveItemId, startDate, endDate))
    .flatMap(tuple -> Mono.just(tuple.getT1().getTotalQty() - tuple.getT2()));
}
```

Circuit Breaker – Spring Cloud Resilience4j

- Spring Cloud Resilience4j 는 MSA 환경에서 장애 내성(Fault Tolerance)을 구현하기 위한 도구
- 장애 처리, 회로 차단, 재시도 및 폴백 기능을 제공

Circuit Breaker – Spring Cloud Resilience4j

1. 회로 차단 (Circuit Breaker): Resilience4j는 회로 차단 패턴을 구현하여 서비스 간의 통신에서 장애가 발생할 때 회로를 차단하고 장애가 해결되기를 기다린다. 이렇게 함으로써 장애 전파를 방지하고 서비스의 가용성을 높인다.
2. 재시도 (Retry): 장애가 발생했을 때 Resilience4j는 지정된 시간 간격으로 서비스 호출을 다시 시도할 수 있는 재시도 메커니즘을 제공한다. 이를 통해 일시적인 장애로 인해 서비스가 중단되지 않도록 할 수 있다.
3. Fallback: Resilience4j는 실패한 요청에 대한 대체 데이터 또는 동작을 제공하는 폴백 메커니즘을 지원한다. 이로써 사용자에게 오류 대신 안정적인 데이터나 메시지를 제공할 수 있다.
4. 한도 설정 (Rate Limiting): Resilience4j는 특정한 리소스나 서비스 호출에 대한 요청 속도 제한을 설정할 수 있다. 이를 통해 과도한 요청으로 인한 서비스 과부하를 방지할 수 있다.
5. 격리 (Bulkhead): Resilience4j는 서비스 호출을 분리된 스레드 풀에서 처리함으로써 서비스 호출 간의 격리를 유지하고 한 서비스의 장애가 다른 서비스에 영향을 미치지 않도록 처리한다.
6. 지연 지표 (Latency Metrics): Resilience4j는 서비스 호출의 지연과 성능을 모니터링하고 지연 지표를 수집한다. 이를 통해 시스템의 성능을 분석하고 최적화할 수 있다.

Spring Cloud Resilience4j

Reserve-check-service – build-gradle

```
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.springframework.cloud:spring-cloud-starter-sleuth'
implementation 'org.springframework.cloud:spring-cloud-starter-zipkin'
implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'
implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-resilience4j'
implementation 'org.springframework.cloud:spring-cloud-starter-kubernetes'
implementation 'org.apache.tomcat.embed:tomcat-embed-core:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-websocket:9.0.73'
implementation 'net.logstash.logback:logstash-logback-encoder:7.4' // logstash logback
implementation 'commons-io:commons-io:2.13.0'
implementation 'commons-net:commons-net:3.9.0' // FTPClient
implementation 'mysql:mysql-connector-java:8.0.33'
implementation 'io.jsonwebtoken:jjwt:0.9.1'

//messaging
implementation 'org.springframework.cloud:spring-cloud-stream'
implementation 'org.springframework.cloud:spring-cloud-stream-binder-rabbit'
```

Reserve-check-service – resilience4JConfig

```
@Configuration
public class Resilience4JConfig {

    public CircuitBreakerRegistry circuitBreakerRegistry() {
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
            .failureRateThreshold(failureRateThreshold:50) // Circuit 열지 말지 결정하는 실패 threshold 퍼센테이지
            .waitDurationInOpenState(Duration.ofSeconds(seconds:5)) // (half closed 전에) circuitBreaker가 open 되기 전에 기다릴 시간
            .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.COUNT_BASED) // circuit breaker count 기반 처리
            .slidingWindowSize(slidingWindowSize:10) // 통계 대상 건수 -> N건의 요청중..
            .build();
        return CircuitBreakerRegistry.of(circuitBreakerConfig);
    }
}
```

- minimumNumberOfCalls : 최소요청수, Circuit Breaker가 실패율과 느린 응답 비율을 계산할 최소 요청 수(실패비율을 계산할 때 필요한 최소 호출수)를 지정
- slidingWindowType : Circuit Breaker가 닫힌 상태에서 호출결과를 기록하기 위한 슬라이딩 윈도우 타입 설정
- slidingWindowSize : 통계건수, Circuit Breaker가 닫힌 상태에서 호출결과를 기록하기 위한 슬라이딩 윈도우 크기 설정
- failureRateThreshold : 실패율, 실패확률이 failureRateThreshold 에 도달하면 서킷 브레이커를 open 상태로 만들고 호출을 차단함
- waitDurationInOpenState : Circuit Breaker 유지시간 설정

Spring Cloud Resilience4j

```
/**
 * urlPath 설정
 *
 * @param responseDto
 * @return
 */
private String getUrlPath(MenuSideResponseDto responseDto) {
    if ("contents".equals(responseDto.getMenuType())) {
        return "/content/" + responseDto.getConnectId();
    }

    CircuitBreaker circuitBreaker = circuitBreakerFactory.create("board");
    BoardResponseDto board = circuitBreaker.run(() ->
        boardServiceClient.findById(responseDto.getConnectId()),
        throwable -> new BoardResponseDto());

    return "/board/" + board.getSkinTypeCode() + "/" + responseDto.getConnectId();
}
```

Circuit Breaker는 circuitBreakerFactory.create("board")를 통해 생성

run() 메서드는 Circuit Breaker로 보호된 코드를 실행

boardServiceClient.findById(responseDto.getConnectId())는 정상적으로 서비스가 동작할 때 호출되는 메서드

throwable -> new BoardResponseDto()는 예외 발생 시 실행될 대체 로직입니다. 여기서는 예외가 발생하면 빈 BoardResponseDto 객체를 반환

Service Discovery – Eureka

- Eureka 는 MSA 의 장점 중 하나인 동적인 서비스 증설 및 축소를 위하여 필수적으로 필요한 서비스의 자가 등록, 탐색 및 부하 분산에 사용될 수 있는 라이브러리
- 마이크로 서비스들의 정보를 레지스트리 서버에 등록할 수 있도록 기능을 제공

Service Discovery – Eureka

Discovery – build.gradle(서버)

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    implementation 'org.apache.tomcat:tomcat-annotations-api:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-core:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-websocket:9.0.73'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

EnableEurekaServer

```
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@EnableEurekaServer  
@SpringBootApplication  
public class DiscoveryApplication {  
  
    Run | Debug  
    public static void main(String[] args) {  
        SpringApplication.run(primarySource:DiscoveryApplication.class, args);  
    }  
}
```


Service Discovery – Eureka

Reserve-check-service –
build.gradle(클라이언트)

@EnableDiscoveryClient

```
implementation 'org.springframework.boot:spring-boot-starter-data-r2dbc'
implementation 'org.springframework.boot:spring-boot-starter-webflux'
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-actuator'
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-reactor-resilience4j'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'com.playtika.reactivefeign:feign-reactor-spring-cloud-starter:3.2.11'
```

```
@ComponentScan({ "org.egovframe.cloud.common", "org.egovframe.cloud.reactive", "org.egovframe.cloud.reservecheck" })
@EnableDiscoveryClient
@EnableReactiveFeignClients
@SpringBootApplication
public class ReserveCheckServiceApplication {

    Run | Debug
    public static void main(String[] args) {
        // TLSv1/v1.1 No longer works after upgrade, "No appropriate protocol" error
        String property = Security.getProperty(key:"jdk.tls.disabledAlgorithms").replace(target:", TLSv1", ", TLSv1.2");
        Security.setProperty(key:"jdk.tls.disabledAlgorithms", property);

        SpringApplication.run(primarySource:ReserveCheckServiceApplication.class, args);
    }
}
```

특정 서비스 디스커버리 기술(Eureka, Consul, Zookeeper 등)에 종속되지 않기 위해
@EnableDiscoveryClient 사용

Service Discovery – Eureka 클라이언트 등록

```
eureka:
  instance:
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}} # random port 사용시 eureka server에 인스턴스가 각각 표시되지 않는다
    preferIpAddress: true # 서버-간 통신 시 hostname 보다는 ip를 우선
  client:
    register-with-eureka: true # eureka 서버에 등록
    fetch-registry: true # 외부 검색 가능
    service-url:
      defaultZone: http://admin:admin@${eureka.instance.hostname:localhost}:8761/eureka
```

Gateway – Spring Cloud Gateway

- API Gateway 란 MSA 에서 언급되는 주요 컴포넌트 중 하나이며, 모든 클라이언트 요청에 대한 end-point 를 통합하는 서비스
- 프록시 서버처럼 동작하며, 인증 및 권한, 모니터링, logging 등의 추가적인 기능도 지원
- Spring Cloud Gateway는 클러스터 내부에서 서비스 간의 트래픽을 관리하고, 라우팅, 필터링, 인증 및 권한 부여 등의 역할을 수행합니다.
- Nginx Ingress Controller는 외부 트래픽을 받아 Kubernetes 클러스터 내부의 서비스로 라우팅해주고, SSL 종료나 로드 밸런싱 같은 기능을 제공합니다.

GatewayServer

Apigateway – build.gradle

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    implementation 'org.springframework.boot:spring-boot-starter-webflux'  
    implementation 'org.springframework.cloud:spring-cloud-starter-gateway'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka'  
    implementation 'org.springframework.cloud:spring-cloud-starter-config' //  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' //  
    implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' //  
    implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'  
    implementation 'net.logstash.logback:logstash-logback-encoder:7.4' //  
    implementation 'io.jsonwebtoken:jjwt:0.9.1'  
    implementation 'javax.xml.bind:jaxb-api:2.3.1'  
  
    implementation 'org.springdoc:springdoc-openapi-webflux-ui:1.7.0'  
  
    compileOnly 'org.projectlombok:lombok'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}
```

Apigateway - ApigatewayApplication

```
@EnableDiscoveryClient  
@SpringBootApplication  
public class ApigatewayApplication {  
  
    Run | Debug  
    public static void main(String[] args) {  
        SpringApplication.run(primarySource:ApigatewayApplication.class, args);  
    }  
}
```

GatewayServer

| | | |
|---|-------|--|
| <pre>server: port: 8000</pre> | 서버 포트 | <p>라우팅 정보</p> <p>id: user-service: 이 라우트의 ID입니다. 임의로 지정할 수 있습니다.</p> <p>uri: lb://USER-SERVICE: 요청을 라우팅할 대상 서비스의 URI입니다. 여기서 lb://는 로드밸런서(LoadBalancer)를 의미하며, USER-SERVICE라는 이름의 마이크로서비스로 요청을 전달합니다.</p> <p>predicates: 요청의 경로 또는 조건에 따라 라우팅을 결정하는 규칙입니다. 여기서는 /user-service/** 경로로 들어오는 모든 요청을 USER-SERVICE로 전달합니다.</p> <p>filters: 요청이나 응답을 변환하거나 조작할 수 있는 필터</p> |
| <pre>spring: application: name: apigateway</pre> | 서버 명 | |
| <pre>cloud: gateway: routes: - id: user-service uri: lb://USER-SERVICE predicates: - Path=/user-service/** filters: - RemoveRequestHeader=Cookie - RewritePath=/user-service/(?<segment>.*), /\${segment} - id: portal-service uri: lb://PORTAL-SERVICE predicates: - Path=/portal-service/** filters: - RewritePath=/portal-service/(?<segment>.*), /\${segment} - id: board-service uri: lb://BOARD-SERVICE predicates: - Path=/board-service/** filters: - RewritePath=/board-service/(?<segment>.*), /\${segment}</pre> | | |

GatewayServer - OpenApiDocConfig

```
@Configuration
public class OpenApiDocConfig {

    @Bean
    public List<GroupedOpenApi> apis(SwaggerUiConfigParameters swaggerUiConfigParameters, RouteDefinitionLocator locator) {
        List<GroupedOpenApi> groups = new ArrayList<>();

        List<RouteDefinition> definitions = locator.getRouteDefinitions().log(category: "OpenApiDocConfig").collectList().block();

        Optional.ofNullable(definitions)
            .map(collection::stream)
            .orElseGet(Stream::empty)
            .filter(routeDefinition -> routeDefinition.getId().matches(regex: ".*-service"))
            .forEach(routeDefinition -> {
                String name = routeDefinition.getId();
                swaggerUiConfigParameters.addGroup(name);
                GroupedOpenApi.builder().pathsToMatch("/" + name + "/*").group(name).build();
            });
        return groups;
    }
}
```

- RouteDefinitionLocator는 Spring Cloud Gateway에서 정의된 라우트(Route)를 찾아주는 역할
- SwaggerUiConfigParameters Swagger UI를 구성하는 데 사용되는 설정 객체
- Apis 매서드Spring Cloud Gateway의 모든 라우트 정의를 가져와서, 각 라우트에 대해 Swagger 그룹을 생성하고, 이를 기반으로 API 문서를 관리하도록 설정

Spring Cloud Gateway에서 각 마이크로서비스에 대한 OpenAPI 문서를 자동으로 생성하고, Swagger UI에서 이러한 문서들을 그룹화하여 표시할 수 있도록 설정하는 역할

GatewayServer - GlobalFilter

- **pre-filter**: 요청이 처리되기 전에 실행되며, 주로 **요청에 대한 초기 로깅**을 수행

```
@Override
public GlobalFilter apply(Config config) {
    // Pre Filter
    return ((exchange, chain) -> {
        // Netty 비동기 방식 서버 사용시에는 ServerHttpRequest를 사용해야 한다.
        ServerHttpRequest request = exchange.getRequest();
        ServerHttpResponse response = exchange.getResponse();

        if (config.isPreLogger()) {
            log.info(format: "[GlobalFilter Start] request ID: {}, method: {}, path: {}", request.getId(), request.getMethod(), request.getPath());
        }

        // Post Filter
        // 비동기 방식의 단일값 전달시 Mono 사용(WebFlux)
        return chain.filter(exchange).then(Mono.fromRunnable(() -> {
            if (config.isPostLogger()) {
                log.info(format: "[GlobalFilter End ] request ID: {}, method: {}, path: {}, statusCode: {}", request.getId(), request.getMethod(), request.getPath(), response.getStatusCode());
            }
        }));
    });
}
```

- **post-filter**: 요청이 처리된 후 응답이 반환되기 직전에 실행되며, **응답에 대한 로깅**을 수행

인증서비스

- 인증서비스는 사용자 또는 시스템의 신원을 확인하고 인가된 액세스를 제공하는 중요한 보안 구성 요소로 사용자가 자격 증명(아이디, 비밀번호 등)을 제출하면 이를 검증하고, 그 결과로 액세스 권한을 부여하거나 거부하는 역할
- . MSA 환경에서는 서비스간 인증 및 권한 부여를 담당하는 시스템으로 구성하여 보안 및 인증 관리를 효율적으로 하고 서비스간의 통신 및 데이터 보호를 강화

JWT

apigateway – build-gradle

```
implementation 'org.springframework.boot:spring-boot-starter-actuator'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-webflux'
implementation 'org.springframework.cloud:spring-cloud-starter-gateway'
implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
implementation 'org.springframework.cloud:spring-cloud-starter-config' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'net.logstash.logback:logstash-logback-encoder:7.4' // logstash logback
implementation 'io.jsonwebtoken:jjwt:0.9.1'
implementation 'javax.xml.bind:jaxb-api:2.3.1'

implementation 'org.springdoc:springdoc-openapi-webflux-ui:1.7.0'
```

Jwt 추출, 파싱, 검증

Apigateway - reactiveAuthorization

```
try {
    authorizationHeader = authorizations.get(index:0);
    String jwt = authorizationHeader.replace(target:"Bearer", replacement:"");
    String subject = Jwts.parser().setSigningKey(TOKEN_SECRET)
        .parseClaimsJws(jwt)
        .getBody()
        .getSubject();

    // refresh token 요청 시 토큰 검증만 하고 인가 처리 한다.
    if (REFRESH_TOKEN_URI.equals(requestPath + "")) {
        return Mono.just(new AuthorizationDecision(granted:true));
    }
    if (subject == null || subject.isEmpty()) {
        log.error(msg:"토큰 인증 오류");
        throw new AuthorizationServiceException(msg:"토큰 인증 오류");
    }
} catch (IllegalArgumentException e) {
    log.error(format:"토큰 헤더 오류 : {}", e.getMessage());
    throw new AuthorizationServiceException(msg:"토큰 인증 오류");
} catch (ExpiredJwtException e) {
    log.error(format:"토큰 유효기간이 만료되었습니다. : {}", e.getMessage());
    throw new AuthorizationServiceException(msg:"토큰 유효기간 만료");
} catch (Exception e) {
    log.error(format:"토큰 인증 오류 Exception : {}", e.getMessage());
    throw new AuthorizationServiceException(msg:"토큰 인증 오류");
}
```

JWT - AuthenticationFilter

```
public Claims getClaimsFromToken(String token) {  
    return Jwts.parser()  
        .setSigningKey(TOKEN_SECRET)  
        .parseClaimsJws(token)  
        .getBody();  
}  
  
/**  
 * 로그인 요청 뿐만 아니라 모든 요청시마다 호출된다.  
 * 토큰에 담긴 정보로 Authentication 정보를 설정한다.  
 * 이 처리를 하지 않으면 AnonymousAuthenticationToken 으로 처리된다.  
 *  
 * @param request  
 * @param response  
 * @param chain  
 * @throws IOException  
 * @throws ServletException  
 */  
  
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {  
    HttpServletRequest httpRequest = (HttpServletRequest) request;  
    String token = httpRequest.getHeader(HttpHeaders.AUTHORIZATION);  
    if (token == null || "undefined".equals(token) || "".equals(token)) {  
        super.doFilter(request, response, chain);  
    } else {  
        Claims claims = getClaimsFromToken(token);  
        String authorities = claims.get(TOKEN_CLAIM_NAME, requiredType:String.class);  
        List<SimpleGrantedAuthority> roleList = new ArrayList<>();  
        roleList.add(new SimpleGrantedAuthority(authorities));  
  
        String username = claims.getSubject();  
        if (username != null) {  
            SecurityContextHolder.getContext().setAuthentication(new UsernamePasswordAuthenticationToken(username, credentials:null, roleList));  
            chain.doFilter(request, response);  
        } else {  
            SecurityContextHolder.getContext().setAuthentication(authentication:null);  
            HttpServletResponse httpResponse = (HttpServletResponse) response;  
            httpResponse.setStatus(HttpStatus.UNAUTHORIZED.value());  
        }  
    }  
}
```

클라이언트로부터 들어오는 HTTP 요청을 가로채어 JWT를 검증하고, JWT에 포함된 사용자 정보를 기반으로 인증을 처리

getClaimsFromToken(String token)
토큰의 서명을 검증하고, 페이로드에서 사용자 정보와 권한 정보를 추출

doFilter
모든 HTTP 요청에 대해 호출되며, JWT 토큰을 확인하고 인증을 설정하는 역할

Spring Cloud Config 서버

- 분산시스템에서 애플리케이션의 환경설정정보 특히 서비스 비즈니스 로직과 연관성 있는 정보들을 애플리케이션과 분리해 외부에서 관리하도록 한 환경설정 서버
- 환경설정 속성정보를 중앙에서 관리
- 환경설정 속성정보란 DB 접속 정보나 미들웨어(연계서버) 접속정보, 애플리케이션을 구성하는 각종 메타데이터
- 서비스 운영 중에 설정파일을 변경 해야 할 경우에는 Spring Cloud Bus 를 이용하여 모든 마이크로 서비스의 환경설정을 업데이트
- RabbitMQ 또는 Kafka 같은 경량 메시지 브로커들을 사용

Spring Cloud Config 서버

Config – build.gradle

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-config-server'  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'  
    implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus  
    implementation 'org.springframework.boot:spring-boot-starter-actuator' // bus  
    implementation 'org.apache.tomcat:tomcat-annotations-api:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-core:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'  
    implementation 'org.apache.tomcat.embed:tomcat-embed-websocket:9.0.73'  
    implementation 'net.logstash.logback:logstash-logback-encoder:7.4' // logstash logback  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
dependencyManagement {
```

Config - ConfigApplication

```
package org.egovframe.cloud.config;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.config.server.EnableConfigServer;  
  
@EnableConfigServer  
@SpringBootApplication  
public class ConfigApplication {  
  
    Run | Debug  
    public static void main(String[] args) {  
        SpringApplication.run(primarySource:ConfigApplication.class, args);  
    }  
}
```

Spring Cloud Config 서버

```
server:
  port: 8888

spring:
  application:
    name: config-service
  profiles:
    active: native,default # native file repository
  cloud:
    config:
      server:
        native:
          search-locations: ${search.location:file:///${user.home}/workspace.edu/egovframe-msa-edu/config} # Windows
          search-locations: file:///${user.home}/workspace.edu/egovframe-msa-edu/config # MacOS
#
  rabbitmq:
    host: ${rabbitmq.hostname:localhost}
    port: 5672
    username: guest
    password: guest

# config server actuator
management:
  endpoints:
    web:
      exposure:
        include: busrefresh
```

로컬환경 세팅

Spring Cloud Config 클라이언트 설정

Reserve-check-service –
build.gradle(클라이언트)

```
implementation 'org.springframework.boot:spring-boot-starter-data-r2dbc'
implementation 'org.springframework.boot:spring-boot-starter-webflux'
implementation 'org.springframework.boot:spring-boot-starter-validation'
implementation 'org.springframework.boot:spring-boot-starter-security'
implementation 'org.springframework.boot:spring-boot-starter-actuator'
implementation 'org.springframework.cloud:spring-cloud-starter-config' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // config
implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // bus
implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-reactor-resilience4j'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'com.playtika.reactivefeign:feign-reactor-spring-cloud-starter:3.2.11'
```

Reserve-check-service -
bootstrap.yml

```
backend > reserve-check-service > src > main > resources > bootstrap.yml
1  spring:
2    cloud:
3      config:
4        uri: http://localhost:8888
5        name: reserve-check-service
```

Config서버 url

Spring Cloud Config 클라이언트 설정

Portalservice - MessageSourceFiles

```
*/
@RefreshScope
@Component
public class MessageSourceFiles {

    private final MessageRepository messageRepository;
    private final Environment environment;
    private final StorageUtils storageUtils;

    public MessageSourceFiles(MessageRepository messageRepository, Environment environment, StorageUtils storageUtils) {
        this.messageRepository = messageRepository;
        this.environment = environment;
        this.storageUtils = storageUtils;
    }

    @PostConstruct
```

- @RefreshScope 어노테이션을 사용하면, 특정 빈에 대해 설정 값이 변경될 때 /actuator/refresh 엔드포인트를 호출하여 해당 빈을 다시 로드하고 새로운 설정 값을 반영
- 만약 설정 값이 자주 변경되지 않거나, 애플리케이션이 재시작될 때만 변경된 설정이 반영되어도 괜찮다면, @RefreshScope를 사용하지 않을 수 있음

Spring Cloud Sleuth + Zipkin

- Spring Cloud Sleuth는 Spring 애플리케이션에서 분산 트레이싱을 위한 기능을 제공
- Sleuth는 마이크로서비스 간의 요청 흐름을 추적하기 위해 각 요청에 고유한 Trace ID와 Span ID를 할당
- Zipkin은 분산 트레이싱 시스템으로, 마이크로서비스의 트랜잭션 데이터를 수집하고 분석
- Zipkin은 Sleuth와 연동되어 각 마이크로서비스에서 생성된 트레이싱 데이터를 수집하여 시각화

Spring Cloud Sleuth + Zipkin

Board - build.gradle

```
implementation 'org.springframework.cloud:spring-cloud-starter-sleuth'
implementation 'org.springframework.cloud:spring-cloud-sleuth-zipkin'
implementation 'org.springframework.cloud:spring-cloud-starter-zipkin'
implementation 'org.apache.tomcat.embed:tomcat-embed-core:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-el:9.0.73'
implementation 'org.apache.tomcat.embed:tomcat-embed-websocket:9.0.73'
implementation 'net.logstash.logback:logstash-logback-encoder:7.4' // logstash logback
implementation 'mysql:mysql-connector-java:8.0.33'
implementation 'io.jsonwebtoken:jjwt:0.9.1'
```

Config - application.yml

```
# rabbitmq server
spring:
  rabbitmq:
    host: ${rabbitmq.hostname:localhost}
    port: 5672
    username: guest
    password: guest
  zipkin:
    base-url: http://${zipkin.hostname:localhost}:${zipkin.port:9411}
```

Zipkin 서버 정보

Spring Cloud Bus 환경 구성 - RabbitMQ 설치


Config-application.yml

```
# rabbitmq server
spring:
  rabbitmq:
    host: ${rabbitmq.hostname:localhost}
    port: 5672
    username: guest
    password: guest
  zipkin:
    base-url: http://${zipkin.hostname:localhost}:${zipkin.port:9411}
egov:
  message: hello
```

- Spring Cloud Config Client를 통해 Config 서버에서 해당 설정을 받아옵니다.
- 클라이언트 애플리케이션에서는 Config 서버와 연결된 상태에서 spring.rabbitmq 설정을 자동으로 받아올 수 있음

Spring Eureka

← → ↺ localhost:8761

 HOME LAST 1000 SINCE STARTUP

System Status

| | | | |
|-------------|---------|--------------------------|---------------------------|
| Environment | test | Current time | 2024-08-16T13:12:23 +0900 |
| Data center | default | Uptime | 00:21 |
| | | Lease expiration enabled | true |
| | | Renews threshold | 13 |
| | | Renews (last min) | 28 |

DS Replicas

localhost

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|-------------------------|---------|--------------------|---|
| APIGATEWAY | n/a (1) | (1) | UP (1) - apigateway-eabc4fd83c0434bca1220dcb1cc8988d |
| BOARD-SERVICE | n/a (1) | (1) | UP (1) - board-service-8ce8f250d1d53385b597c5a4e40f725b |
| PORTAL-SERVICE | n/a (1) | (1) | UP (1) - portal-service-54d06bf673c428f53ecf89b9fdd7e847 |
| RESERVE-CHECK-SERVICE | n/a (1) | (1) | UP (1) - reserve-check-service-dd3912176f8ab77852621fba6f6ad449 |
| RESERVE-ITEM-SERVICE | n/a (1) | (1) | UP (1) - reserve-item-service-99212db9336bff9f567d247d9ffa0b5a |
| RESERVE-REQUEST-SERVICE | n/a (1) | (1) | UP (1) - reserve-request-service-cba8056844e488e4ebc6397d867ca8f9 |
| USER-SERVICE | n/a (1) | (1) | UP (1) - user-service-d4ae97251b38efa0b3f58293c960b7db |

System Metrics

| Name | Value |
|----------------------|---|
| total-avail-memory | 108mb |
| num-of-cpus | 16 |
| current-memory-usage | 50mb (46%) |
| server-uptime | 00:21 |
| registered-replicas | http://localhost:8761/eureka/ |

Potalservice

```
2024-08-16 14:13:00.756 INFO 1288 - [background-preinit, , ] o.h.v.i.util.Version: HV000001: Hibernate Validator 6.2.5.Final

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  _/
      |_|_|_|

:: Spring Boot ::
(v2.7.12)

2024-08-16 14:13:01.628 INFO 2140 - [main, , ] o.s.c.c.c.ConfigServicePropertySourceLocator: Fetching config from server at : http://localhost:8888
2024-08-16 14:13:01.628 INFO 2140 - [main, , ] o.s.c.c.c.ConfigServicePropertySourceLocator: Fetching config from server at : http://localhost:8888
2024-08-16 14:13:01.967 INFO 2479 - [main, , ] o.s.c.b.c.PropertySourceBootstrapConfiguration: Located property source: [BootstrapPropertySource {name='bootstrapProperties-configClient'}, BootstrapPropertySource {name='bootstrapProperties-file:/home/infadm/workspace.edu/egovframe-msa-edu/config/portal-service.yml'}, BootstrapPropertySource {name='bootstrapProperties-file:/home/infadm/workspace.edu/egovframe-msa-edu/config/application.yml'}]
2024-08-16 14:13:02.055 INFO 2567 - [main, , ] o.e.c.p.PortalServiceApplication: No active profile set, falling back to 1 default profile: "default"
2024-08-16 14:13:03.500 INFO 4012 - [main, , ] o.s.d.r.c.RepositoryConfigurationDelegate: Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-08-16 14:13:03.942 INFO 4454 - [main, , ] o.s.d.r.c.RepositoryConfigurationDelegate: Finished Spring Data repository scanning in 427 ms. Found 12 JPA repository interfaces.
2024-08-16 14:13:05.115 INFO 5627 - [main, , ] o.s.i.c.DefaultConfiguringBeanFactoryPostProcessor: No bean named 'errorChannel' has been explicitly defined. Therefore, a default PublishSubscribeChannel will be created.
2024-08-16 14:13:05.141 INFO 5653 - [main, , ] o.s.i.c.DefaultConfiguringBeanFactoryPostProcessor: No bean named 'integrationHeaderChannelRegistry' has been explicitly defined. Therefore, a default DefaultHeaderChannelRegistry will be created.
2024-08-16 14:13:05.391 INFO 5903 - [main, , ] o.s.c.c.s.GenericScope: BeanFactory id=f582de85-cdf9-395f-bddc-a9d4633c9269
2024-08-16 14:13:06.273 INFO 6785 - [main, , ] o.e.c.c.c.MessageSourceConfig: messageSource getBasenameSet=file:///home/infadm/msa-attach-volume/messages/messages
2024-08-16 14:13:06.836 INFO 7348 - [main, , ] o.s.b.w.e.t.TomcatWebServer: Tomcat initialized with port(s): 0 (http)
```

포스트맨

http://223.130.132.226:8000/portal-service/api/v1/contents/1

GET http://223.130.132.226:8000/portal-service/api/v1/contents/1 8000번 포트 -> apigateway Send

Params Authorization Headers (8) Body Scripts Settings Cookies

Query Params

| Key | Value | Description |
|-----|-------|-------------|
| Key | Value | Description |

Status: 200 OK Time: 103 ms Size: 4.79 KB Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "contentNo": 1,
3   "contentName": "소개",
4   "contentRemark": "소개 설명",
5   "contentValue": "<h3><strong> 등장배경 및 목적</strong></h3><p>개발프레임워크는 정보시스템 개발을 위해 필요한 기능 및 아키텍처를 미리 만들어 제공함으로써 효율적인 어플리케이션 구축을 지원합니다. "전자정부
표준프레임워크"는 공공사업에 적용되는 개발프레임워크의 표준 정립으로 응용 SW 표준화, 품질 및 재 사용성 향상을 목표로 합니다. 이를 통해 "전자정부 서비스의 품질향상" 및 "정보화 투자 효율성 향상"을 달성하고,
대·중소기업이 동일한 개발기반 위에서 공정 경쟁이 가능하게 됩니다.<br><br>※ 표준프레임워크는 기존 다양한 플랫폼(.NET, php 등) 환경을 대체하기 위한 표준은 아니며, java 기반의 정보시스템 구축에 활용하실 수 있는
개발·운영 표준 환경을 제공하기 위한 것입니다.</p><h3><img src=\"http://localhost:8080/portal-service/api/v1/images/editor/editor-202110-1ca352474cd84341a1668e09f5aa968a.png\"><strong> 특징</
strong></h3><p><img src=\"https://www.egovframe.go.kr/statics/home/images/img_P0005_character.png\" alt=\"\"></p><h4>eGovFrame</h4><p>상용 솔루션 연계</p><p>민관학계로 구성 된 자문협의회를
통해 국가적 차원의 표준화 수행</p><p>국가적 표준화 지원</p><p>민·관·학계로 구성 된 자문협의회를 통해 국가적 차원의 표준화 수행</p><p>개발형 표준 준수</p><p>오픈소스 기반의 범용화되고 공개된 기술의 활용으로 특정
사업자에 대한 종속성 배제</p><p>변화 유연성</p><p>각 서비스의 모듈화로 교체가 용이하며 인터페이스 기반 연동으로 모듈간 변경영향 최소화</p><p>모바일 환경 지원</p><p>모바일 환경을 위한 모바일 웹(UX/UI) 및
하이브리드 앱 지원</p><p>편리하고 다양한 환경 제공</p><p>Eclipse 기반의 모델링(UML, ERD), 에디팅, 컴파일링, 디버깅 환경 제공</p><h3><strong>적용 가능 시스템 조건</strong></h3><p>아래 세가지 조건을 모두
만족하는 경우 표준프레임워크 적용 가능</p><ul><li><i>1</i></li><i>자바 기반의 웹 응용 시스템(WAS가 존재하는 경우)</li><i>2</i></li><i>3</i></li><i>2.7 기준) JavaEE(J2EE) <strong>JDK1.5 ~ 1.8</strong>의 환경 (단, 개발환경 2.7
이상에서는 JDK 1.6 필요)<br>(3.0 이상) JavaEE(J2EE) <strong>JDK1.6 ~ 1.8</strong>의 환경<br>(3.5 이상) JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경 (단, 개발환경 3.5.1 부터 JDK 1.8 적용
가능)<br>(3.6 이상) JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경<br>(3.7 이상) JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경 (단, 개발환경 3.7 이상에서는 JDK 1.8 필요)<br>(3.8 이상)
JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경 (단, 개발환경 3.8 이상에서는 JDK 1.8 필요)<br>(3.9 이상) JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경 (단, 개발환경 3.9 이상에서는 JDK 1.8
필요)<br>(3.10 이상) JavaEE(J2EE) <strong>JDK1.7 ~ 1.8</strong>의 환경 (단, 개발환경 3.10 이상에서는 JDK 1.8 필요)</li></ul><p>신규 개발시스템으로써, 기존 시스템과 물리적 혹은 논리적으로 구분되는
경우</li></ul><p>실환경 내 모바일 표준프레임워크의 사용자 경험(UX) 지원 기능은 프레임워크와 개발 언어 종류에 상관없이 활용가능 (javascript 기반)</p><h3><strong>적용 효과</strong></h3><p>정보시스템을
개발하거나 운영할 때 필요한 기본 기능을 미리 구현한 것으로 이를 기반으로 추가 기능을 개발하여 조립함으로써 전체 정보시스템을 완성할 수 있습니다.</p><p><img src=\"https://www.egovframe.go.kr/statics/home/
images/img_P0005_effect.png\" alt=\"\"> 표준프레임워크 적용 전 : 1.정보화사업별 동일한 기능들의 중복 개발, 2.기술 종속으로 인해 선행사업자 의존도 높음, 3.프레임워크 미 보유업체는 경쟁 불리, 4.정보시스템간 상호
연계 시 많은 기간과 인력미 소요, 5.개발표준 미흡으로 유지보수가 어려움, 표준프레임워크 적용 후 : 1.공통컴포넌트 재사용으로 중복 예산 절감, 2.표준화된 개발기반으로 사업자 종속성 해소, 3.프레임워크 무상제공으로
중소기업 경쟁력 향상, 4.표준화된 연계모듈 활용으로 상호연용성 향상, 5.개발표준에 의한 모듈화로 유지보수가 용이\"></p>
6 }
```