

전자정부 표준프레임워크 MSA 템플릿 개요 및 개발 환경



Contents

1. MSA템플릿 구성 소개
2. MSA템플릿 개발 환경
3. MSA템플릿 Service Mesh



1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

① 소규모(관리자, 사용자)/대규모(관리자, 사용자) 총 4개의 포털

E 기존 표준프레임워크 템플릿(심플/포털) 기능 중심으로 MSA로 구현

The screenshot displays two main sections: the MSA Admin interface on the left and a comparison of two portal templates on the right.

MSA Admin Interface (Left):

- Header: MSA Admin
- Left sidebar:
 - 서비스 관리
 - 컨텐츠 관리
 - 계시판 관리
 - 배너 관리
 - 첨부파일 관리
 - 설정 관리
 - 공통코드 관리
 - 공통코드 상세 관리
 - 메뉴 관리
 - 권한별 메뉴 관리
 - 이용약관 관리
 - 개인정보처리방침 관리
 - 권한 관리
 - 인가 관리
 - 권한인가 관리
- Center content: 메뉴관리 (Menu Management) screen. It shows a list of menu items under '대규모 Admin' and allows for basic configuration like menu number, name, and URL.

eGovFrame Comparison (Right):

- 표준프레임워크 경량화면 eGovFrame:** This section shows a comparison of the standard framework template (eGovFrame) with the MSA portal template. It highlights the following differences:
 - 2021 표준프레임워크 컨트리뷰션** (2021 Standard Framework Review) - A green card.
 - '21년 8차 표준프레임워크 온라인 정기교육** (2021 Q8 Standard Framework Online Regular Education) - A red card.
 - '21년 9차 표준프레임워크 온라인 정기교육** (2021 Q9 Standard Framework Online Regular Education) - A blue card.
- Features:** The red card highlights features such as '안녕하세요' (Hello), '전자정부 표준프레임워크 센터입니다.' (This is the E-Government Standard Framework Center), and '현재 전자정부 표준프레임워크 및' (Current E-Government Standard Framework and).
- Buttons:** Each card has a '+ 더보기' (View more) button at the bottom right.

1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

● 실행 환경

- 표준프레임워크 4.2
- Spring Boot 2.7.12
- Spring Cloud 2021.0.7
- Openjdk 1.8
- Gradle 7.3
- Docker engine 20.10.7

1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

기술 스택

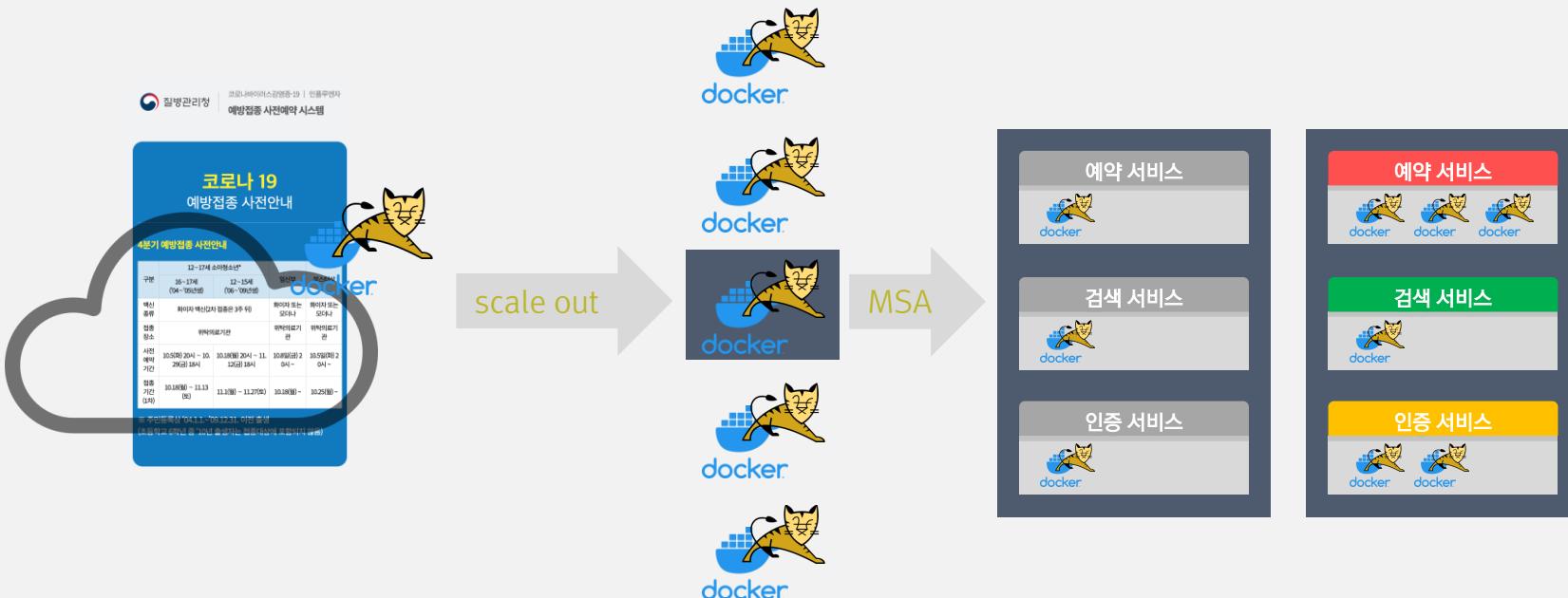
| | | | | | | | |
|---|---|---|---|---|---|---|---|
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| | | | |  |  |  |  |

1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

Cloud Native

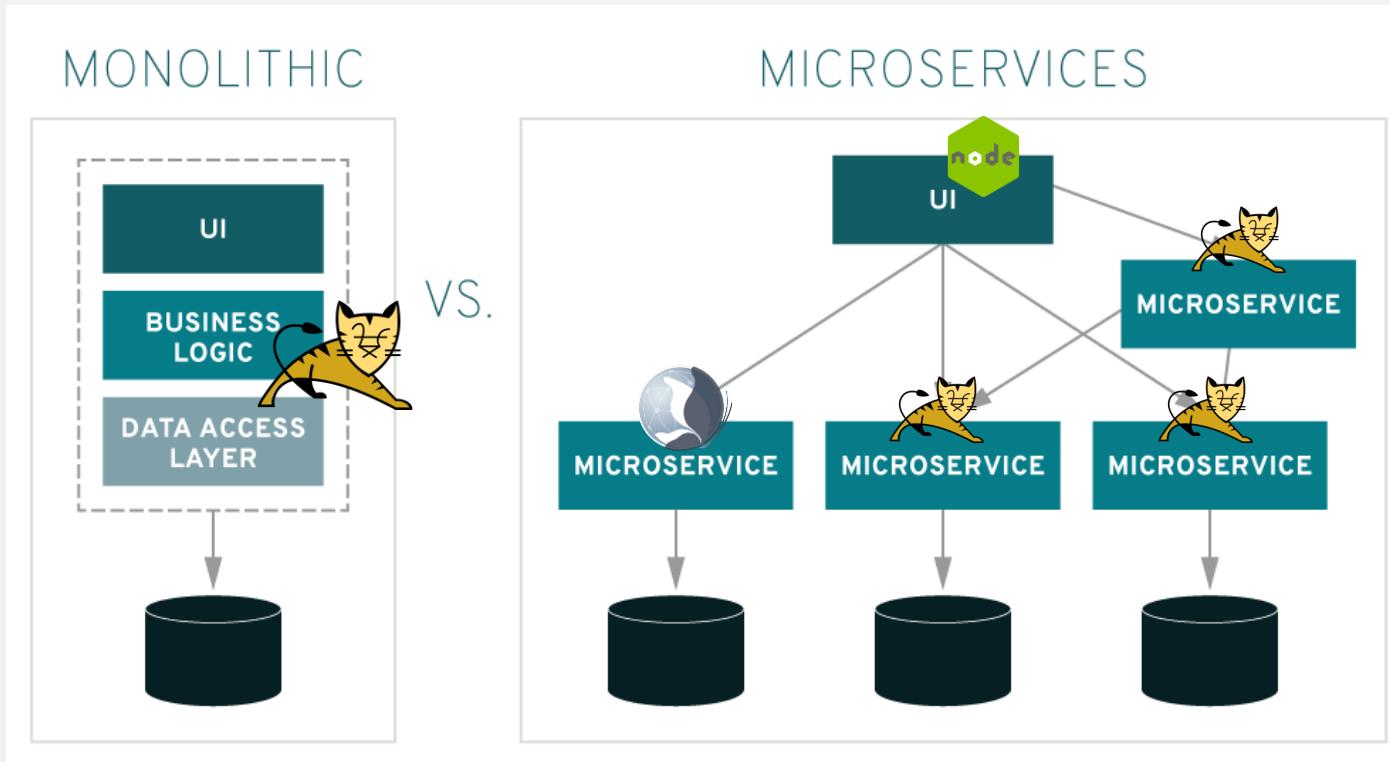
 MSA는 클라우드 환경에 적합한 아키텍처



1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

MicroServices Architecture에 대한 이해

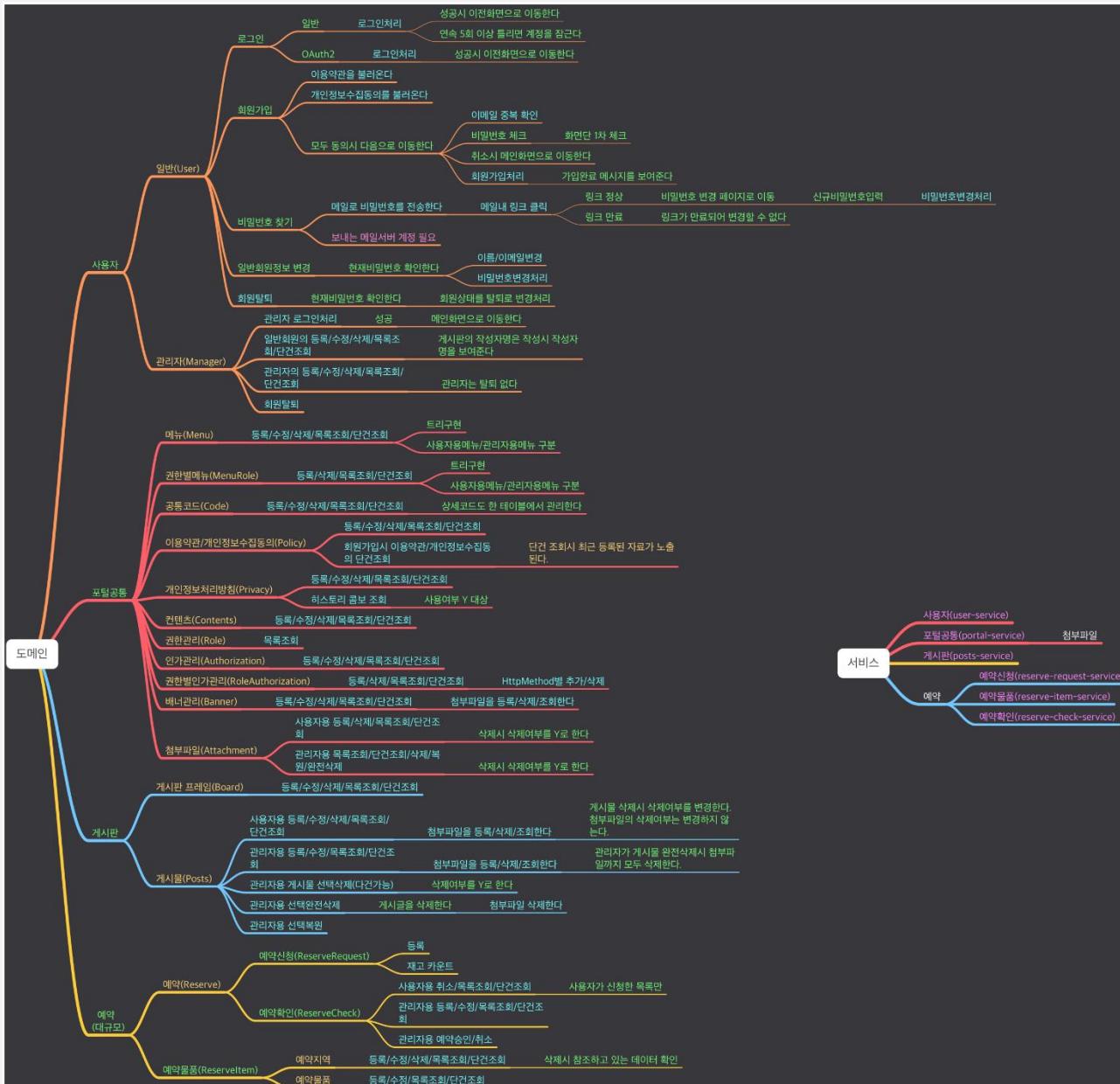


☞ <https://www.redhat.com/ko/topics/microservices/what-are-microservices>

1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

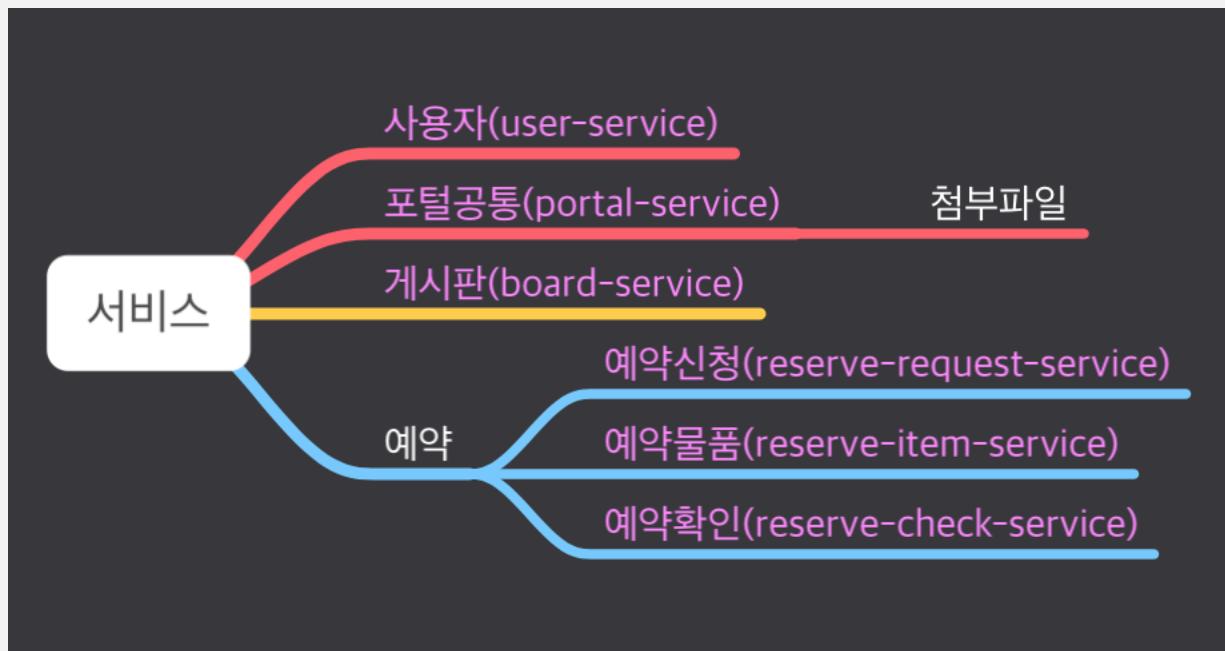
서비스 구성



1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

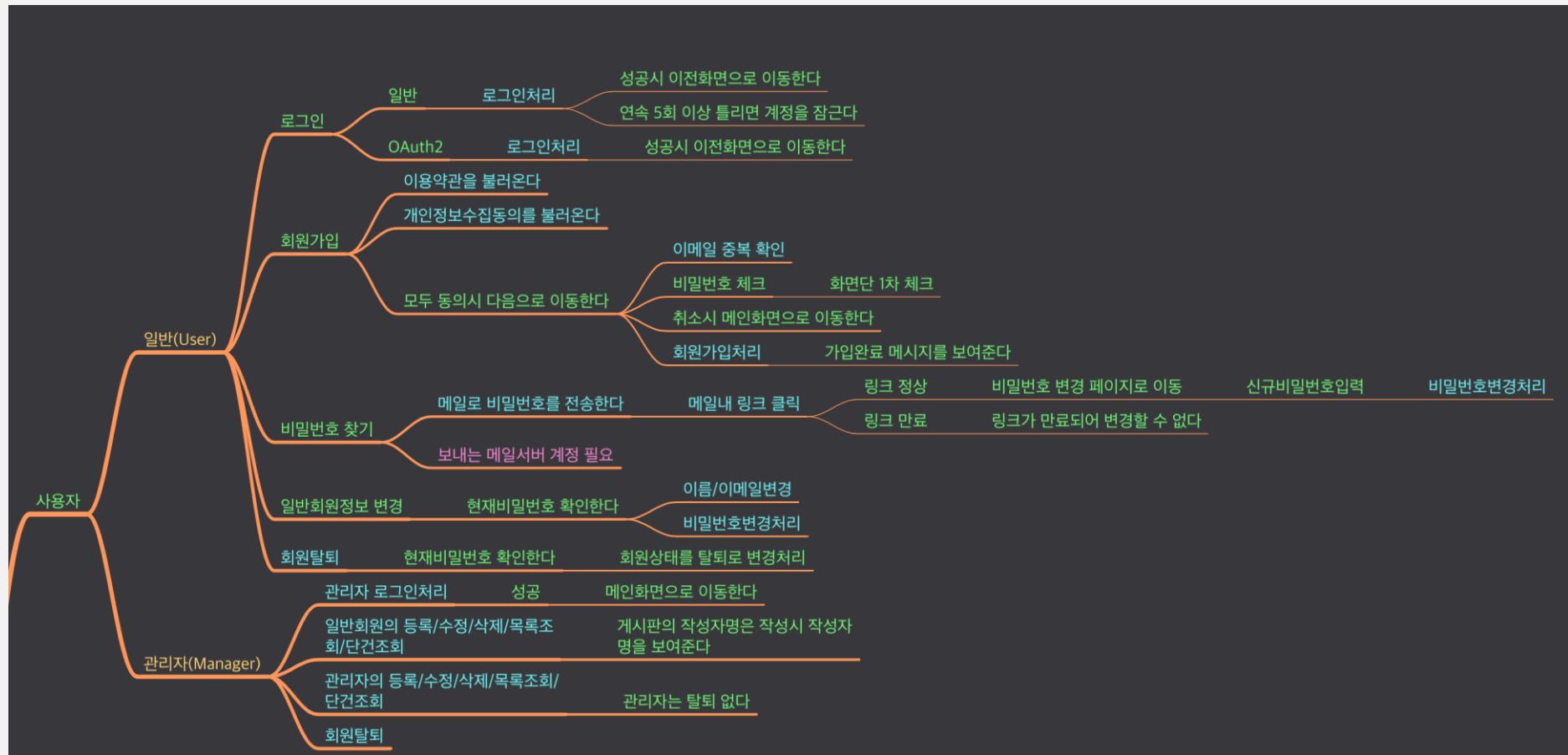
○ 서비스 구성



1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

● 서비스 도메인 구성



1. MSA 템플릿 구성 소개

1. MSA 템플릿 개요

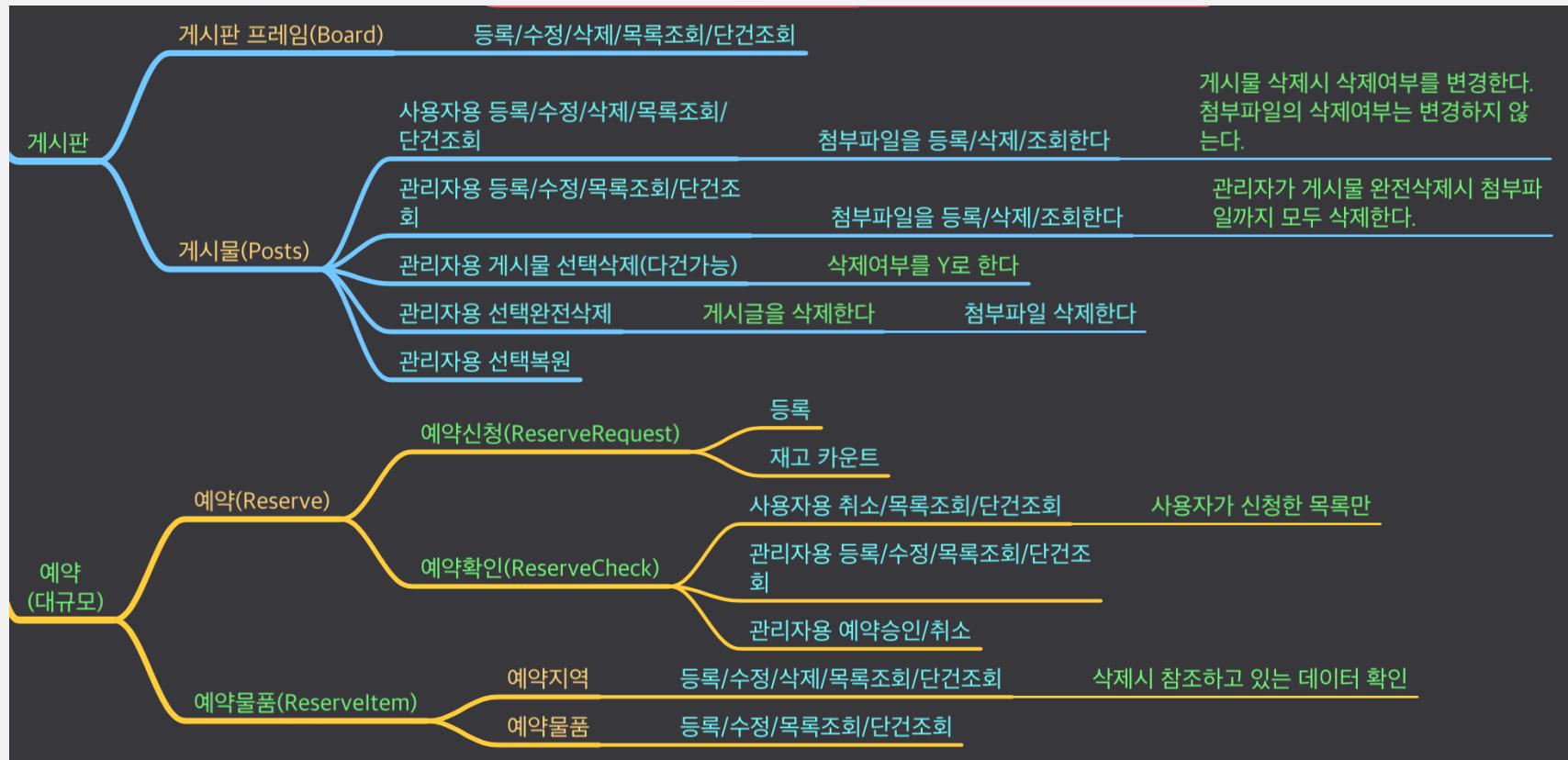
① 서비스 도메인 구성



1. MSA 템플릿 구성 소개

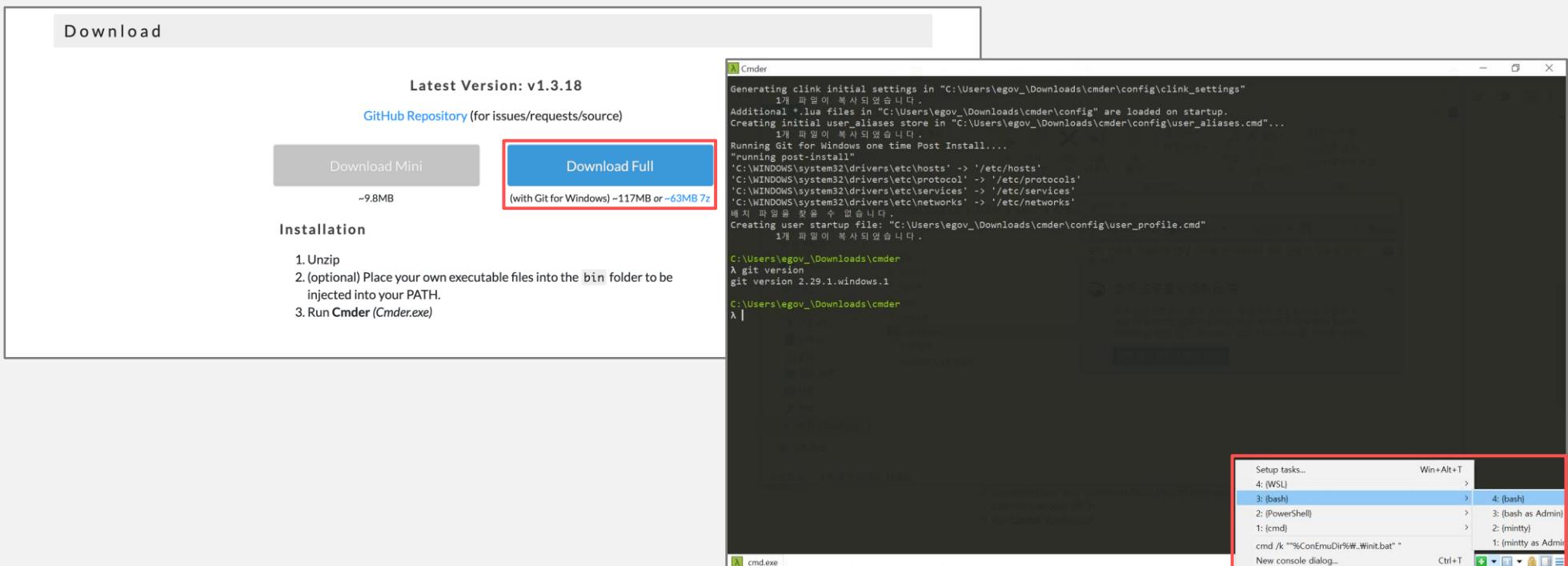
1. MSA 템플릿 개요

① 서비스 도메인 구성



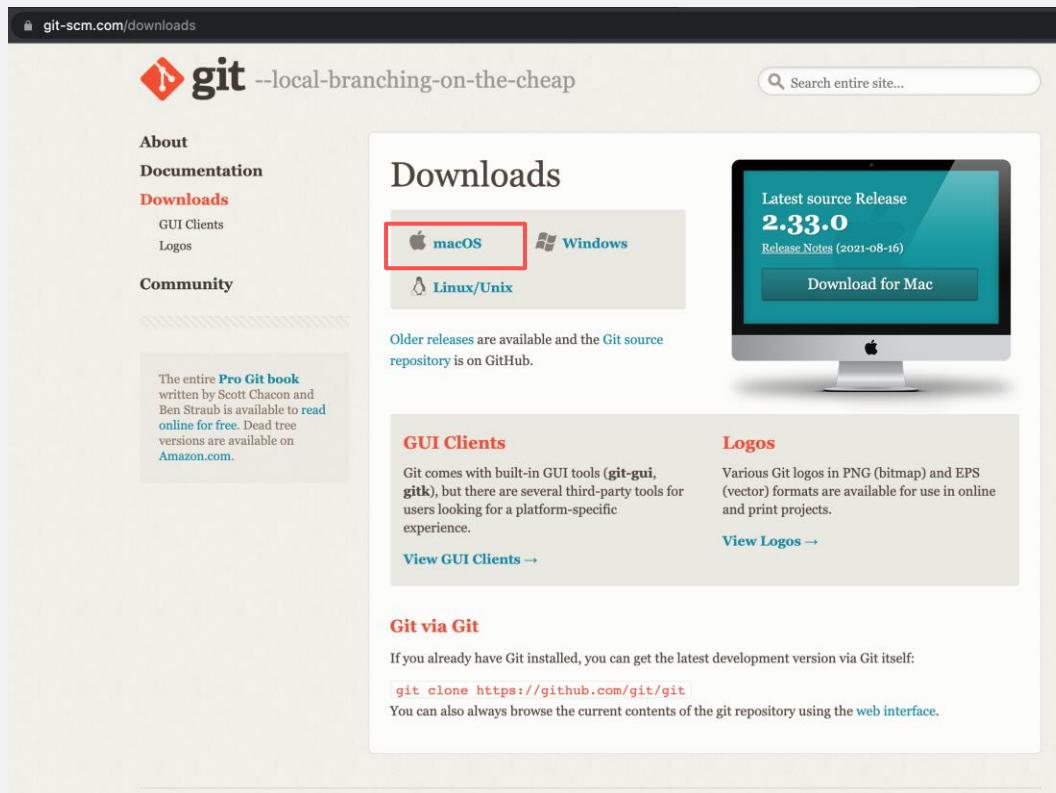
① (Windows) cmder – <https://cmder.app/>

- ① 프로그램 다운로드 – “Download Full” (git 이 함께 설치됨)
- ② 압축을 풀고 “Cmder.exe” 실행
- ③ + 아이콘 클릭 > bash > bash 선택



① (macOS) git – <https://git-scm.com/downloads>

- ① git-scm.com 에서 macOS 용 다운 받아 설치
- ② 또는 brew install git 명령으로 설치



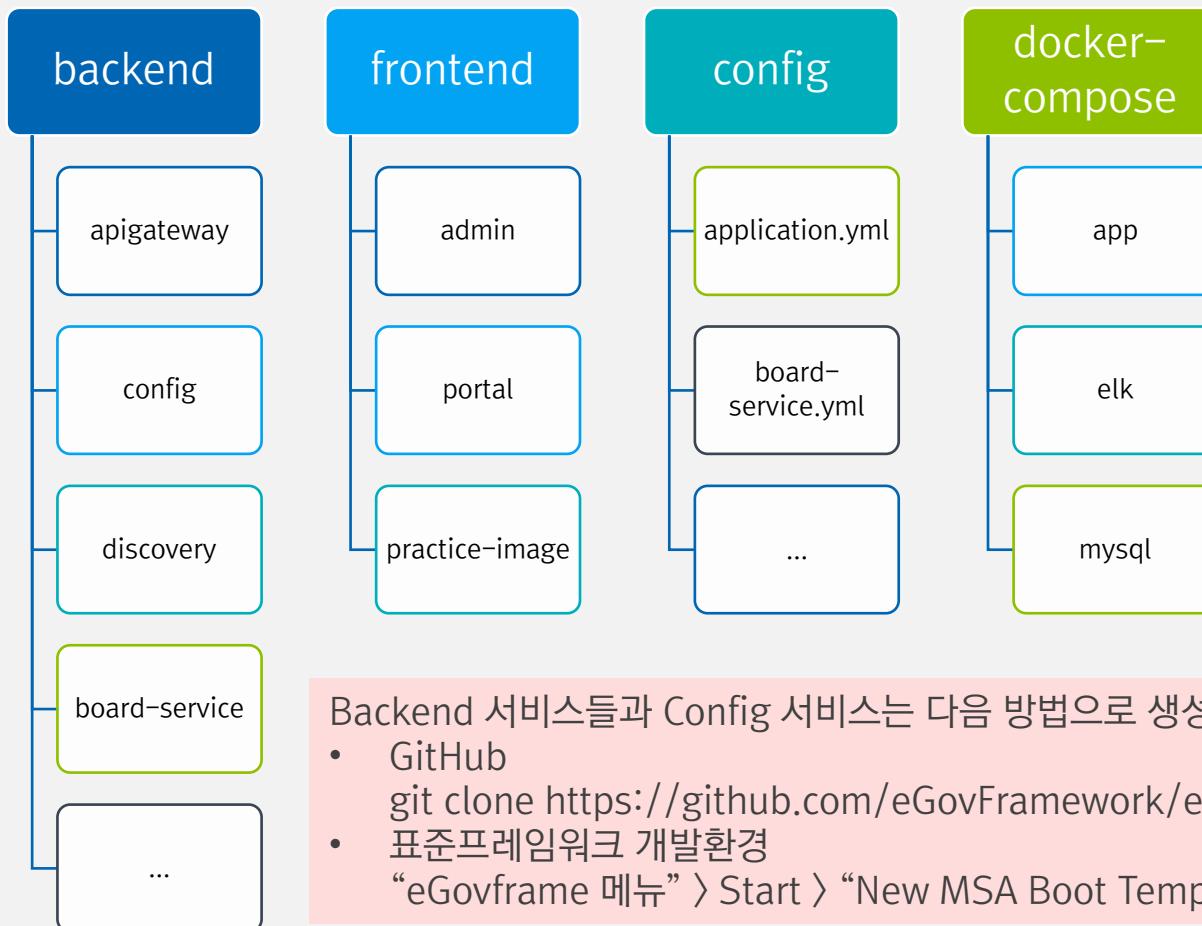
2. MSA 템플릿 개발 환경

1. MSA 템플릿 개요

● 실습 자료 다운로드

- ① cmder bash 창에서
- ② “git version” 명령으로 git 이 설치되어 있는지 확인한다.
- ③ “mkdir \${HOME}/workspace.edu”
- ④ “cd \${HOME}/workspace.edu”
- ⑤ “git clone https://github.com/eGovFramework/egovframe-msa-edu.git”

실습 자료 구조



Backend 서비스들과 Config 서비스는 다음 방법으로 생성 가능

- GitHub
git clone <https://github.com/eGovFramework/egovframe-msa-edu>
- 표준프레임워크 개발환경
“eGovframe 메뉴” > Start > “New MSA Boot Template Project” 선택

Docker – <https://docs.docker.com/get-docker/>

💡 Windows Home은 WSL2(Windows Subsystem for Linux 2) 설치하여 ubuntu 환경에서 진행(보장X)

The screenshot shows the Docker documentation website at <https://docs.docker.com/get-docker/>. The page title is "Get Docker". On the left, there's a sidebar with a navigation menu under "Get Docker" containing links like "Get started", "Language-specific guides (New)", "Develop with Docker", "Set up CI/CD", "Deploy your app to the cloud", "Run your app in production", "Educational resources", "Open source at Docker", and "Documentation archive". The main content area has a heading "Get Docker" and a callout box with the text: "Update to the Docker Desktop terms". It explains that professional use of Docker Desktop in large organizations requires a paid subscription, with a grace period until January 31, 2022. Below this, there's a section about Docker being an open platform for developing, shipping, and running applications. At the bottom, there are three cards: "Docker Desktop for Mac" (macOS icon), "Docker Desktop for Windows" (Windows icon), and "Docker for Linux" (Linux Penguin icon).

MySQL 설치 및 데이터 생성

mysql/docker-compose



```
# docker-compose.yml 를 사용하여 mysql 을 생성합니다.  
$ docker network create egov-network # 도커 네트워크 생성  
$ cd ${HOME}/workspace.edu/egovframe-msa-edu/docker-compose/mysql  
$ docker-compose up -d
```

MySQL 설치 및 데이터 생성(Docker 미사용시)

- ① 개발환경 다음 경로의 MySql 사용 가능 => C:/eGovFrame-4.2.0/bin/mysql-8.0.36/bin
- ② 또는 다음 URL에서 직접 다운로드 <https://downloads.mysql.com/archives/community/>
‘mysql-5.7.35-winx64.zip’ 파일을 다운로드 한 후 C: 드라이브에 압축을 풁니다.
- ③ cd /c/mysql-5.7.35-winx64/bin
- ④ ./mysqld --initialize-insecure
- ⑤ ./mysqld -install
- ⑥ net start mysql
- ⑦ ./mysql -u root -p
- ⑧ create database msaportal;
- ⑨ create database reservation;
- ⑩ create user 'msaportal' identified by 'msaportal';
- ⑪ grant all privileges on msaportal.* to msaportal;
- ⑫ grant all privileges on reservation.* to msaportal;
- ⑬ exit

MySQL 설치 및 데이터 생성(Docker 미사용시)

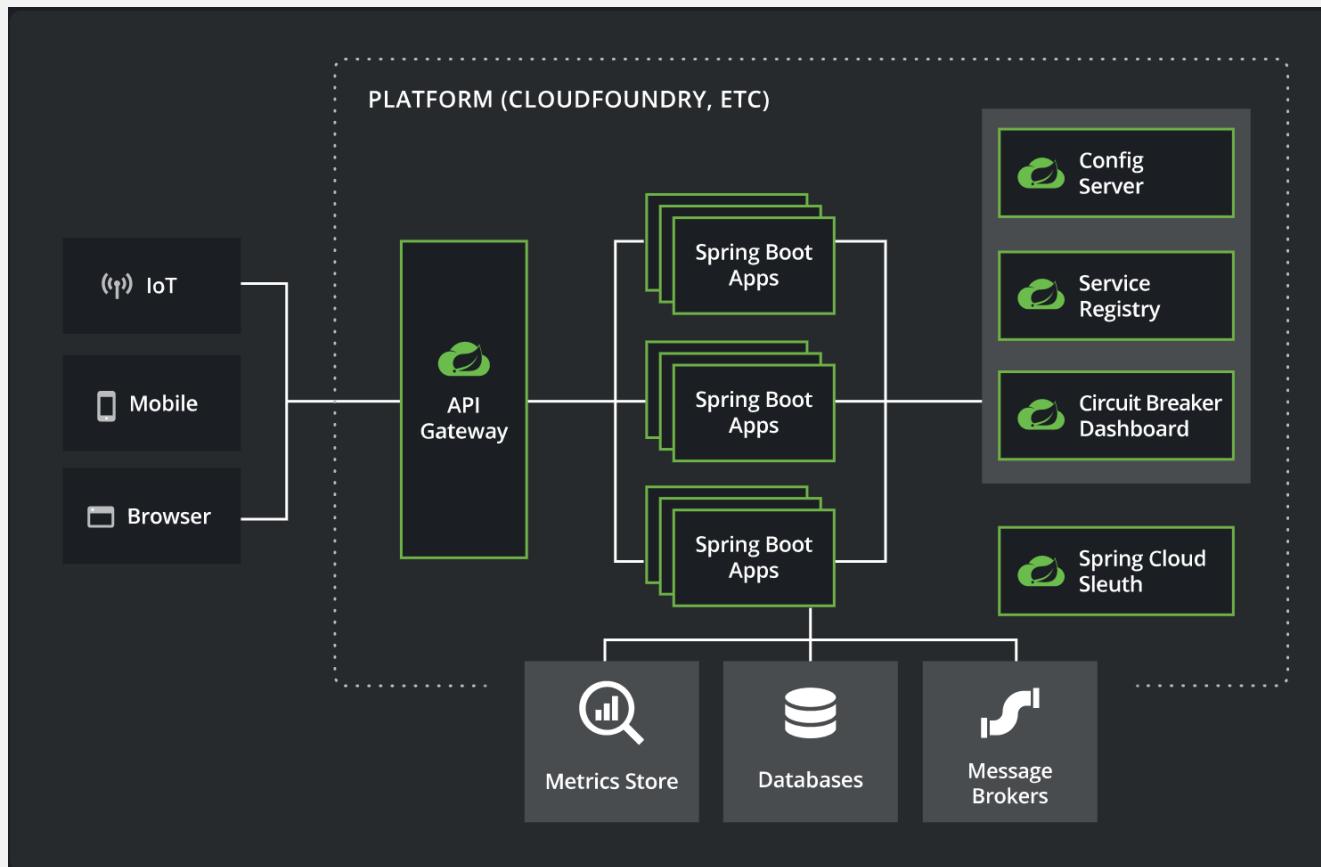
- ① “./mysql -u msaportal -p msaportal < \${HOME}/workspace.edu/egovframe-msa-edu/docker-compose/mysql/msaportal.sql”
- ② “./mysql -u msaportal -p reservation < \${HOME}/workspace.edu/egovframe-msa-edu/docker-compose/mysql/reservation.sql”

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Service Mesh에 대한 이해

💡 하나의 서비스를 여럿으로 나누었지만 서로 데이터를 공유하고 통신할 수 있도록 구성



📌 <https://spring.io/microservice>

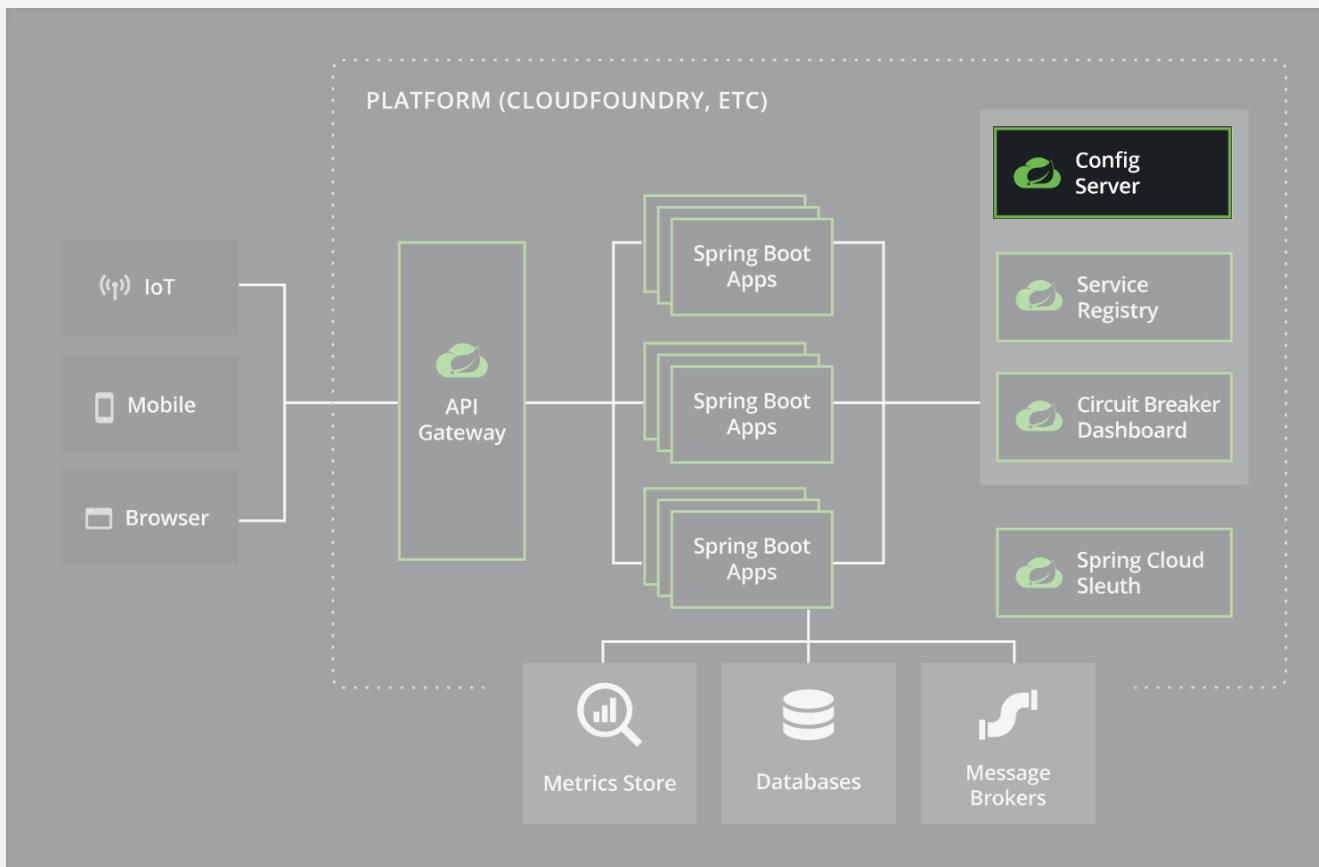
S

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Config Server – Spring Cloud Config

💡 운영 중 설정 정보가 변경된다면? 서비스 인스턴스마다 하나씩 수정할 것인가?
=> 외부에 있는 정보를 읽어 가도록 구성



3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔧 Config Server – Spring Cloud Config

💡 환경설정 정보를 각 인스턴스마다 (중복)관리하지 않고 외부(github, ..)에서 관리

build.gradle

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-config-server'      // spring cloud config server
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'   // spring cloud config - bootstrap.yml
    implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp'    // spring cloud bus
    implementation 'org.springframework.boot:spring-boot-starter-actuator'     // refresh
}
```

ConfigApplication.java

```
@EnableConfigServer // config server 활성화
@SpringBootApplication
public class ConfigApplication {
    public static void main(String[] args) {
        SpringApplication.run(ConfigApplication.class, args);
    }
}
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔧 Config Server – Spring Cloud Config

💡 github에 설정 파일들을 올려두고 github 접속 정보를 설정한다.

application.yml

```
server:  
  port: 8888  
  
spring:  
  application:  
    name: config-service  
  cloud:  
    config:  
      server:  
        git:  
          uri: https://github.com/eGovFramework/egovframe-msa-template-config  
          username: '{cipher}7ba3d655c…'  
          password: '{cipher}0526a7e79…' # personal access token  
          search-paths: config # repository 폴더 경로  
          default-label: main # main branch  
          ignore-local-ssh-settings: true  
          skip-ssl-validation: true  
  
      bus:  
        enabled: true # webhook 활성화: /monitor 엔드포인트 호출 가능해진다
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔧 Config Server – Spring Cloud Config

💡 로컬 개발 환경에서는 설정 파일 경로를 지정

application.yml

```
server:  
  port: 8888  
  
spring:  
  application:  
    name: config-service  
  profiles:  
    active: native # native file repository  
  cloud:  
    config:  
      server:  
        native:  
          searchLocations: file://${user.home}/workspace/egovframe-msa-template-config/config  
  
# config server actuator  
management:  
  endpoints:  
    web:  
      exposure:  
        include: busrefresh
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔧 Config Client

💡 각 서비스마다 config 의존성 추가

build.gradle

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-config'      // spring cloud config client  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap'    // spring cloud config - bootstrap.yml  
    implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp'    // spring cloud bus  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'       // refresh  
}
```

bootstrap.yml

```
spring:  
  cloud:  
    config:  
      uri: http://localhost:8888  
      name: portal-service # portal-service.yml이 있으면 불러오게 된다  
    # profiles:  
    #  active: prod # portal-service-prod.yml
```

▶ 이제 각 서비스마다 ‘curl -x POST http://[ip]:[port]/actuator/refresh’ 명령을 보내 설정 정보를 무중단으로 새로 반영 할 수 있다. 🎉 하지만 수 많은 서비스의 인스턴스를 모두 호출할 수는 없는 일이다.

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Spring Cloud Bus – Config Refresh

💡 변경된 환경설정 정보가 각 인스턴스에 자동으로 갱신될 수 있도록 하는 역할

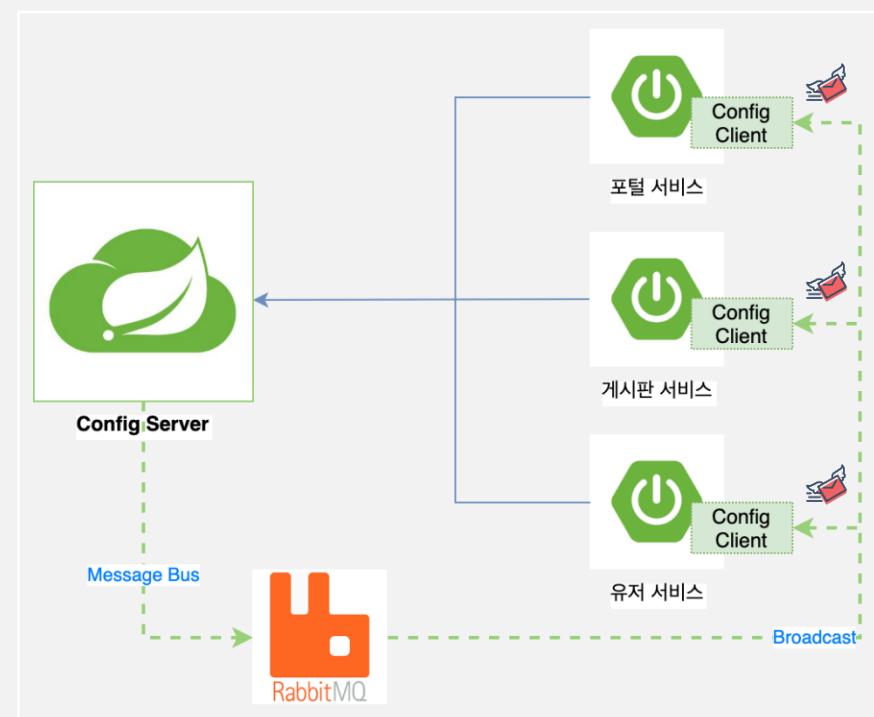
RabbitMQ dockder run 
docker run -d -e TZ=Asia/Seoul --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management

config/application.yml 

```
// RabbitMQ 서버 정보
spring:
  rabbitmq:
    host: localhost
    port: 5672
    username: guest
    password: guest
```

(config server) application.yml

```
spring:
  rabbitmq:
    host: localhost
    port: 5672
    username: guest
    password: guest
```



3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

RabbitMQ(Docker 미 사용시 설치하기)

💡 <https://www.rabbitmq.com/install-windows.html>

Direct Downloads

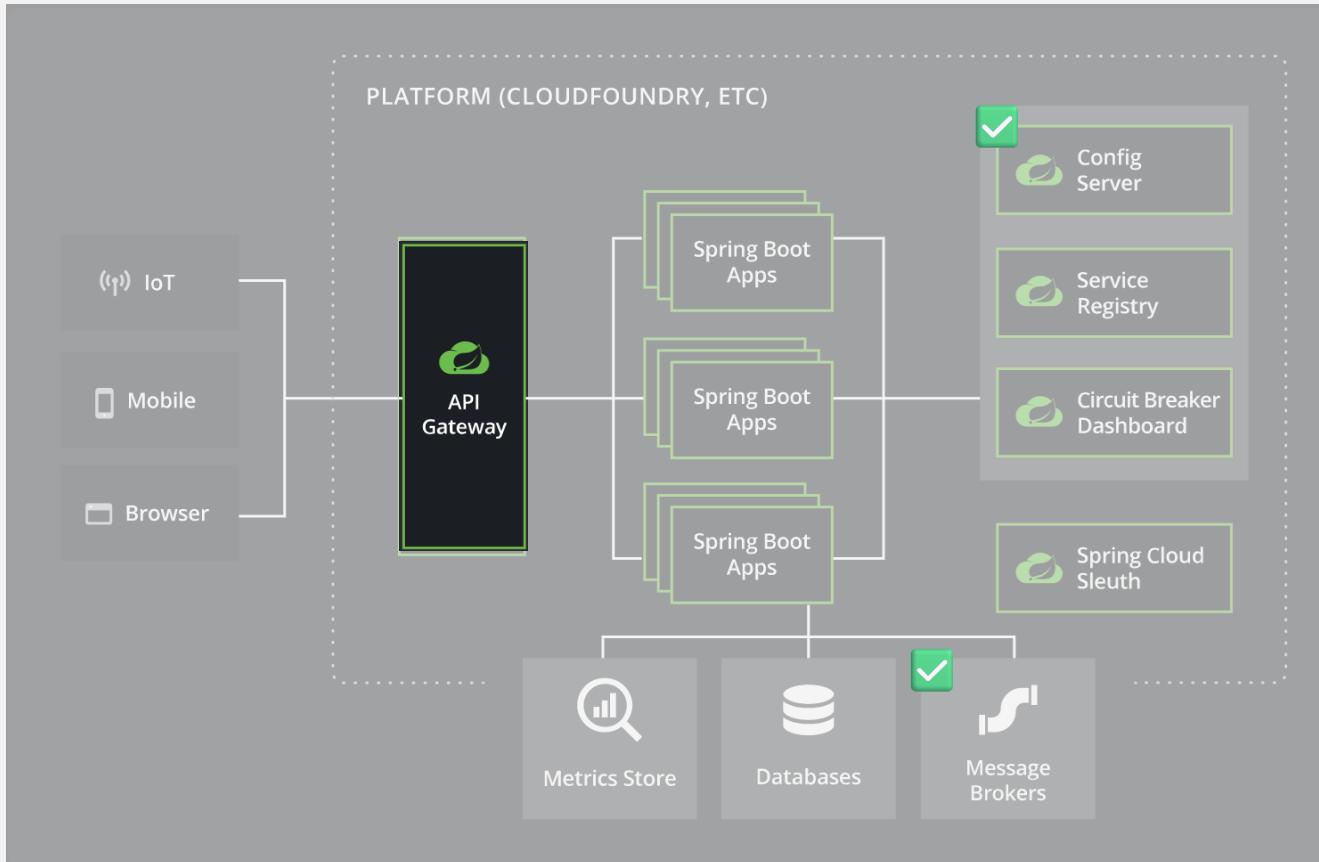
| Description | Download | Signature |
|--|---|---------------------------|
| Installer for Windows systems (from GitHub) | rabbitmq-server-3.9.8.exe | Signature |

- ① RabbitMQ 설치 시 erlang 설치 안내가 나오면 “Windows 64-bit Binary File” 을 설치한다.
- ② RabbitMQ 설치 완료 후 RabbitMQ Command Prompt 창에서
- ③ “rabbitmq-plugins enable rabbitmq_management” 로 관리 기능을 활성화.



API Gateway – Spring Cloud Gateway

💡 외부의 모든 요청을 처리하는 단일 진입점, 인증/인가/로깅등 처리 후 적절한 서비스로 라우팅



3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요



API Gateway – Spring Cloud Gateway

build.gradle

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-gateway'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
    implementation 'org.springframework.boot:spring-boot-starter-webflux'  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'  
    implementation 'org.springframework.cloud:spring-cloud-starter-config'      // spring cloud config client  
    implementation 'org.springframework.cloud:spring-cloud-starter-bootstrap' // spring cloud config - bootstrap.yml  
    implementation 'org.springframework.cloud:spring-cloud-starter-bus-amqp' // spring cloud bus  
    implementation 'org.springframework.boot:spring-boot-starter-actuator'    // refresh  
}
```

ApigatewayApplication.java

```
@EnableDiscoveryClient // discovery client 활성화  
@SpringBootApplication  
public class ApigatewayApplication {  
  
    public static void main(String[] args) {  
        SpringApplication.run(ApigatewayApplication.class, args);  
    }  
}
```

☞ <https://spring.io/blog/2018/12/12/spring-cloud-greenwich-rc1-available-now> (Zuul X)

3. MSA 템플릿 Service Mesh

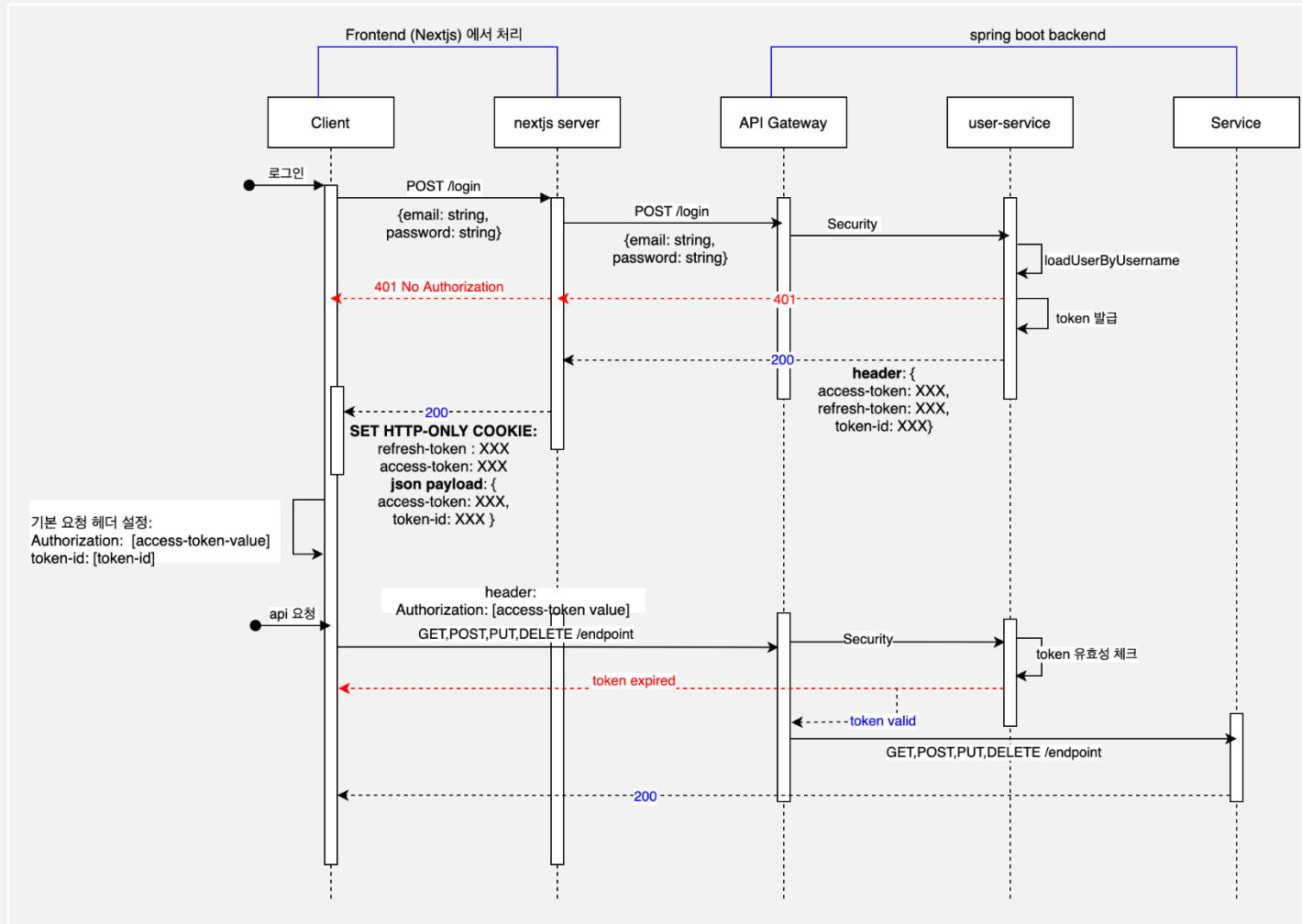
1. MSA 템플릿 개요



API Gateway – Spring Cloud Gateway

application.yml

```
server:  
  port: 8000  
  
spring:  
  application:  
    name: apigateway  
  cloud:  
    gateway:  
      routes:  
        - id: user-service  
          uri: lb://USER-SERVICE  
          predicates:  
            - Path=/user-service/**  
          filters:  
            - RemoveRequestHeader=Cookie  
            - RewritePath=/user-service/(?<segment>.*), /${segment}  
            - SwaggerHeaderFilter  
        - id: portal-service  
          uri: lb://PORTAL-SERVICE  
          predicates:  
            - Path=/portal-service/**  
          filters:  
            - RewritePath=/portal-service/(?<segment>.*), /${segment}  
            - SwaggerHeaderFilter
```

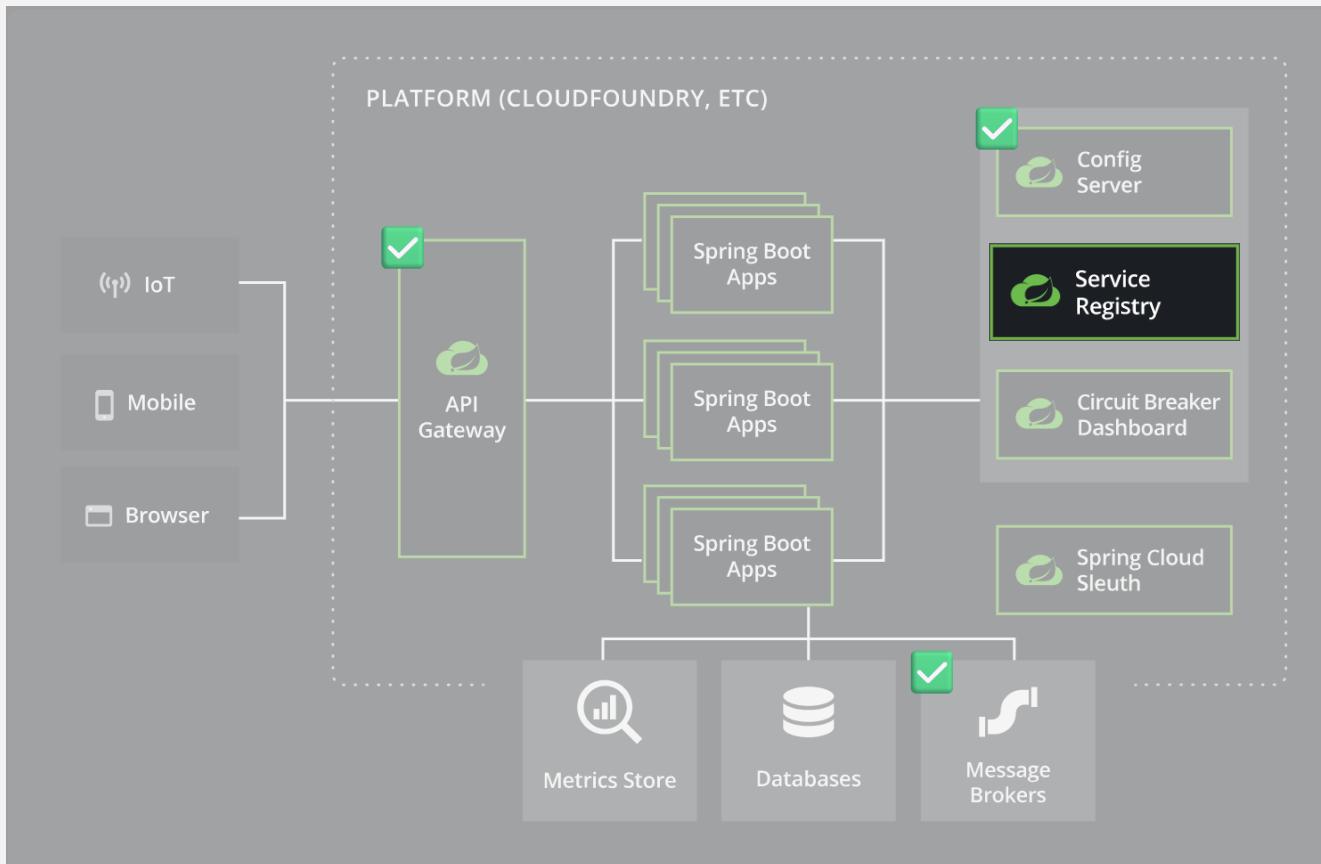
 Authentication/Authorization – JWT


3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔍 Service Discovery – Eureka Server

💡 여러 마이크로 서비스들을 등록하여 관리하고 요청 시 해당 서비스를 찾아 호출한다.



🔍 Service Discovery – Eureka Server

The screenshot shows the Spring Eureka Server interface at `localhost:8761`. The top navigation bar includes links for `HOME` and `LAST 1000 SINCE STARTUP`.

System Status

| | | | |
|-------------|-----|--------------------------|---------------------------|
| Environment | N/A | Current time | 2021-10-12T09:37:20 +0900 |
| Data center | N/A | Uptime | 1 day 15:46 |
| | | Lease expiration enabled | true |
| | | Renews threshold | 15 |
| | | Renews (last min) | 24 |

DS Replicas

localhost

Instances currently registered with Eureka

| Application | AMIs | Availability Zones | Status |
|-------------------------|---------|--------------------|--|
| APIGATEWAY | n/a (1) | (1) | UP (1) - apigateway:e17a90b464fef02b44e0df2db9360e74 |
| BOARD-SERVICE | n/a (2) | (2) | UP (2) - board-service:6d5cc520272834c567dece96d0c01a00 , board-service:03fdc1bcbaad96b472d70a659c3e6e06 |
| PORTAL-SERVICE | n/a (1) | (1) | UP (1) - portal-service:d8144fee720bc9298890ed2a5f0d0daf |
| RESERVE-CHECK-SERVICE | n/a (1) | (1) | UP (1) - reserve-check-service:3b44fb12758277ece1faf7de3971f7fb |
| RESERVE-ITEM-SERVICE | n/a (1) | (1) | UP (1) - reserve-item-service:56bf615b9b65314aac75e6e5ebbdedcc |
| RESERVE-REQUEST-SERVICE | n/a (1) | (1) | UP (1) - reserve-request-service:040f5570cef69bca1579fa04f2e9ed39 |
| USER-SERVICE | n/a (1) | (1) | UP (1) - user-service:52602a6b76498aded793c939737e1a52 |

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔍 Service Discovery – Eureka Server

build.gradle

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-server'  
    implementation 'org.springframework.boot:spring-boot-starter-security'  
}
```

DiscoveryApplication.java

```
@EnableEurekaServer // eureka server 활성화  
@SpringBootApplication  
public class DiscoveryApplication {  
  
    public static void main(String[] args){  
        SpringApplication.run(DiscoveryApplication.class, args);  
    }  
  
}
```

application.yml

```
...  
spring:  
  security:  
    user:  
      name: admin  
      password: admin
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🔍 Service Registry – Eureka Client

build.gradle

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-eureka-client'
}
```

config/application.yml



```
eureka:
  instance:
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true # eureka 서버에 등록
    fetch-registry: true # 외부 검색 가능
    service-url:
      defaultZone: http://admin:admin@localhost:8761/eureka
```

XXXServiceApplication.java

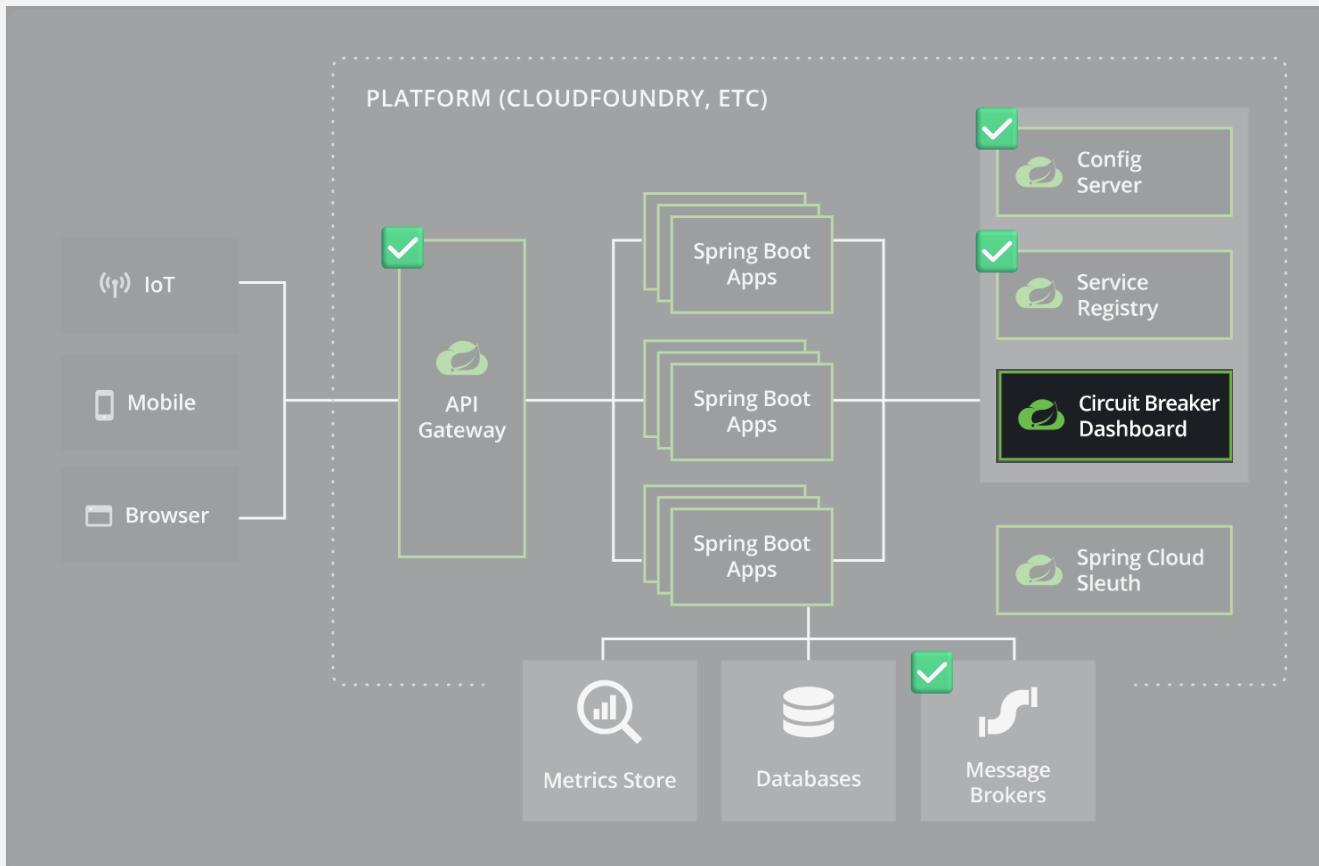
```
@EnableDiscoveryClient // discovery client 활성화
@SpringBootApplication
public class XXXServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(ApiGatewayApplication.class, args);
    }
}
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🚧 Circuit Breaking – Resilience4j

💡 서비스의 장애가 다른 서비스에 전파되지 못하도록 해야 한다.



↗ <https://spring.io/blog/2018/12/12/spring-cloud-greenwich-rc1-available-now> (Hystrix X)

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🚧 Circuit Breaking – Resilience4j

build.gradle

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-circuitbreaker-resilience4j'
    implementation 'org.springframework.cloud:spring-cloud-starter-openfeign'
}
```

Resilience4JConfig.java

```
@Configuration
public class Resilience4JConfig {

    @Bean
    public Customizer<Resilience4JCircuitBreakerFactory> resilience4JCircuitBreakerFactoryCustomizer() {
        CircuitBreakerConfig circuitBreakerConfig = CircuitBreakerConfig.custom()
            .slidingWindowType(CircuitBreakerConfig.SlidingWindowType.TIME_BASED) // circuit breaker time 기반 처리
            .slowCallDurationThreshold(Duration.ofSeconds(10)) // 요청 지연으로 간주하는 시간
            .minimumNumberOfCalls(10) // 통계 최소 요청 건
            .build();

        return circuitBreakerFactory -> circuitBreakerFactory.configureDefault(
            id -> new Resilience4JConfigBuilder(id)
                .circuitBreakerConfig(circuitBreakerConfig)
                .build()
        );
    }
}
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

🚧 Circuit Breaking – Resilience4j

💡 portal-service에서 board-service를 호출하는 경우의 예

BoardServiceClient.java

```
// 호출하는 서비스의 Application에 @EnableFeignClients 선언
@FeignClient(value = "board-service", configuration = CustomFeignConfiguration.class)
public interface BoardServiceClient {

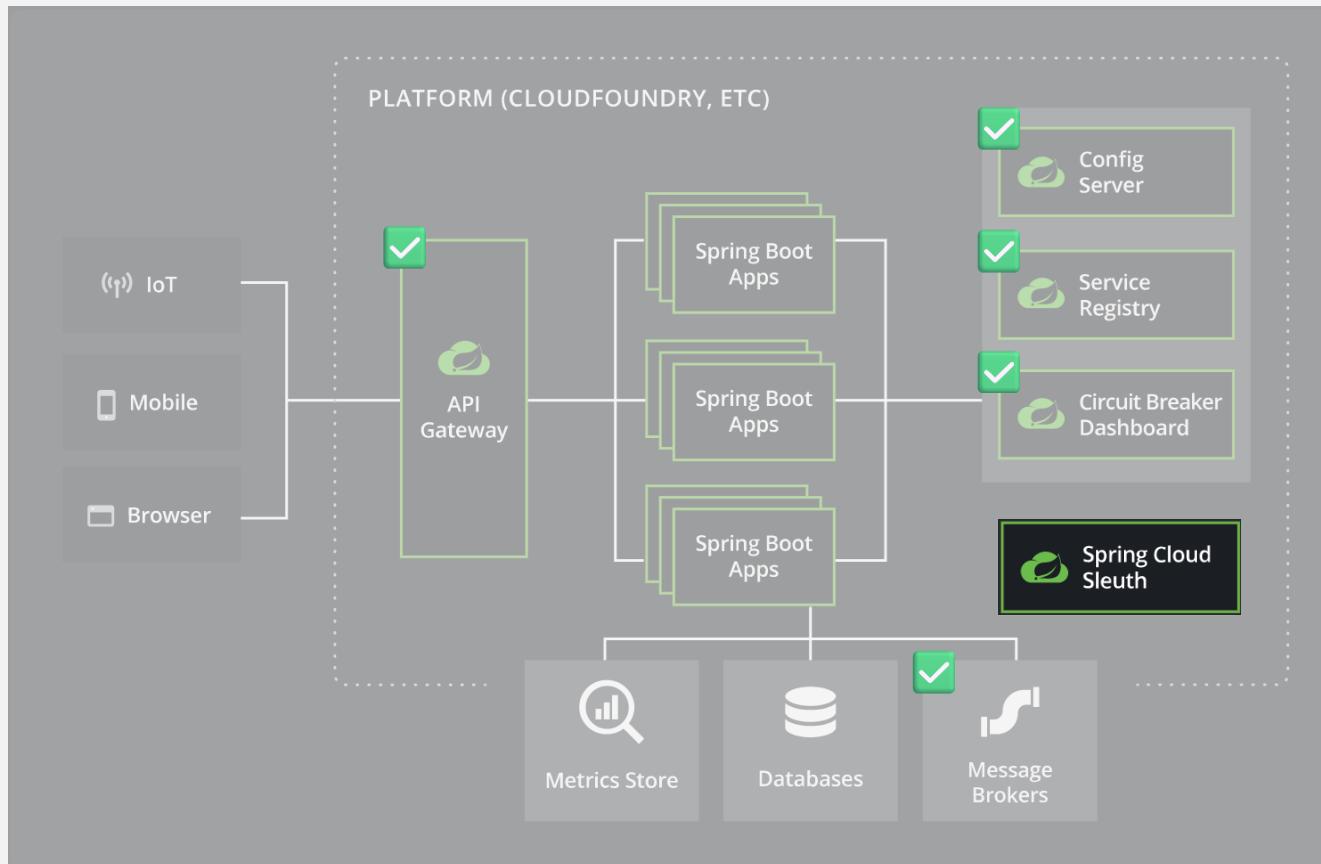
    @GetMapping("/api/v1/boards/{boardNo}")
    BoardResponseDto findById(@PathVariable("boardNo") Integer boardNo);
}
```

MenuRoleService.java

```
// FeignClient 사용하여 서비스간 호출 시 장애가 발생하면 빈 객체를 리턴
CircuitBreaker circuitBreaker = circuitBreakerFactory.create("board");
BoardResponseDto board = circuitBreaker.run(() ->
    boardServiceClient.findById(menuSideResponseDto.getConnectId()),
    throwable -> new BoardResponseDto());
```

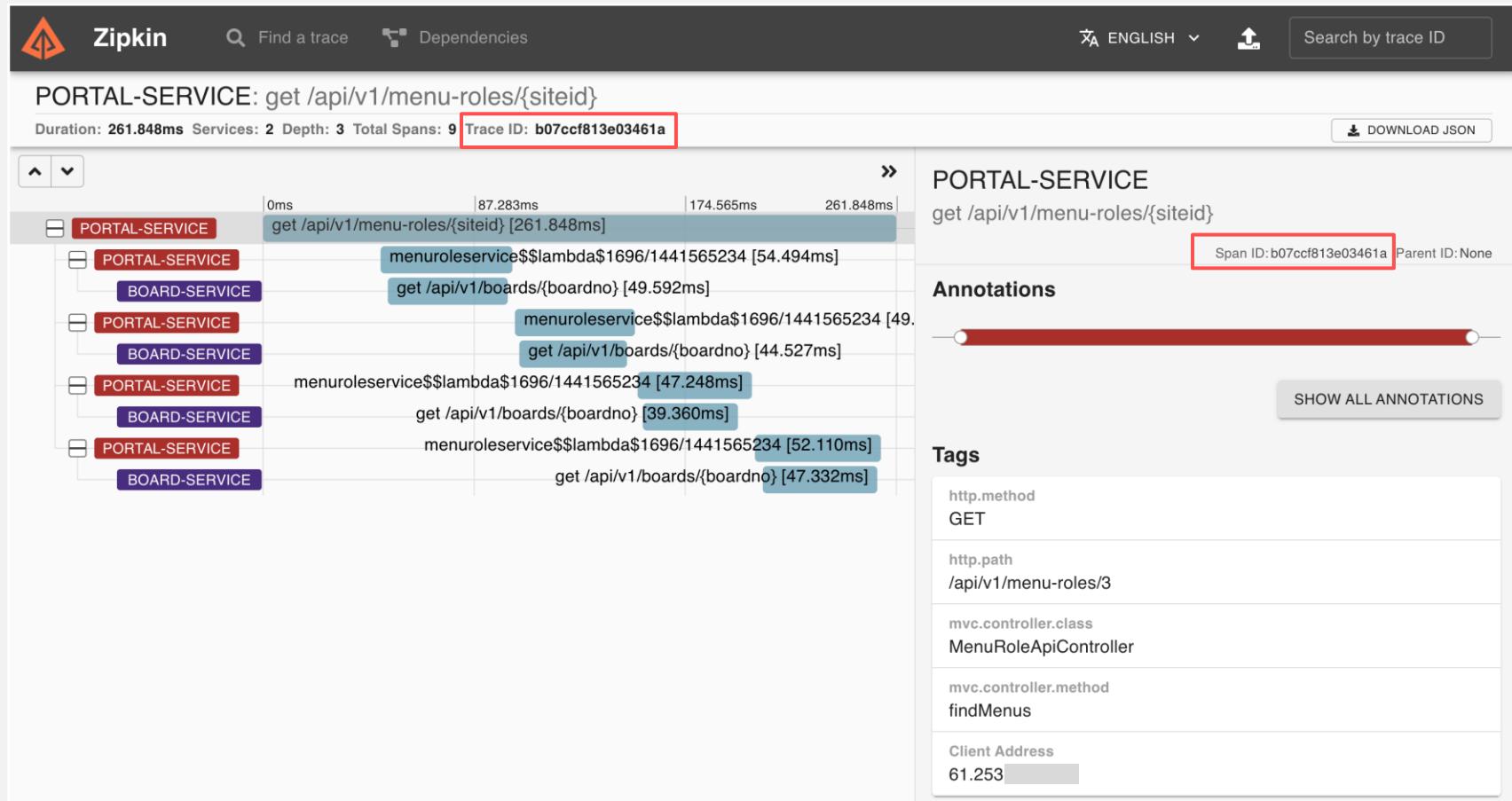
东方财富 Distributed Tracing – Spring Cloud Sleuth + Zipkin

💡 분산 환경에서 서비스 간 트래픽을 추적하고 분석하여 오류 지점 파악





Distributed Tracing – Spring Cloud Sleuth + Zipkin



3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

东方财富 Distributed Tracing – Spring Cloud Sleuth + Zipkin

build.gradle

```
dependencies {  
    implementation 'org.springframework.cloud:spring-cloud-starter-sleuth'  
    implementation 'org.springframework.cloud:spring-cloud-sleuth-zipkin'  
    implementation 'org.springframework.cloud:spring-cloud-starter-zipkin' // zipkin 3.0 부터 sleuth-zipkin 으로 변경  
}
```

Zipkin server



```
docker run --name zipkin -d -p 9411:9411 -e TZ=Asia/Seoul openzipkin/zipkin
```

config/application.yml



```
// Zipkin 서버 정보  
spring:  
zipkin:  
base-url: http://localhost:9411
```

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Zipkin(Docker 미 사용시 설치하기)

💡 <https://zipkin.io/pages/quickstart>

Java

If you have Java 8 or higher installed, the quickest way to get started is to fetch the [latest release](#) as a self-contained executable jar:

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

- ① cmder bash 창에서 다운 받은 위치로 이동 후
- ② “java -jar zipkin-server-2.23.4-exec.jar” 명령 실행

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Centralized Logging – ELK

💡 흩어져 있는 로그들을 한 곳에 모아 보기 위한 구성

모든 것이 Elasticsearch에서 시작되었죠...

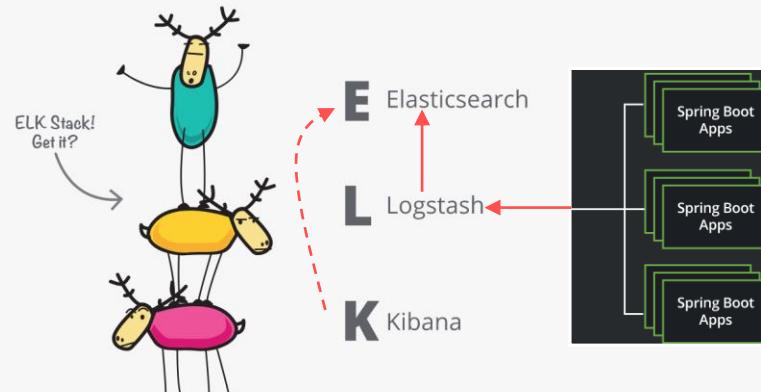
JSON 기반의 분산형 오픈 소스 RESTful 검색 엔진으로, 사용하기 쉽고, 확장 가능하며, 유연하여 검색 분야에서는 사용자와 회사의 팬덤과 높은 인기를 누렸습니다.



Elasticsearch 7.10.2 까지 Apache License 2.0 유지

<https://www.elastic.co/kr/downloads/past-releases#elasticsearch-oss>

유료화의 대안으로 Amazon OpenSearch 를 권장한다.



지금은 발전하여 Logstash, Kibana와 함께 제공됩니다

중심에는 검색 엔진이 있는데, 사용자가 로그를 위해 Elasticsearch를 사용하기 시작했고 이것을 손쉽게 수집해서 시각화하고 싶어했습니다. 그래서 강력한 수집 파이프라인인 Logstash와 유연한 시각화 도구인 Kibana가 도입되었습니다.

📌 <https://www.elastic.co/kr/what-is/elk-stack>

3. MSA 템플릿 Service Mesh

1. MSA 템플릿 개요

Centralized Logging – ELK

ELK docker-compose up



```
$ cd ${HOME}/workspace.edu/egovframe-msa-edu/docker-compose/elk  
$ mkdir -p ${HOME}/workspace.edu/data && chmod -R 770 ${HOME}/workspace.edu/data  
$ docker-compose up -d
```

build.gradle

```
dependencies {  
    implementation 'net.logstash.logback:logstash-logback-encoder:6.6' // logstash logback encoder  
}
```

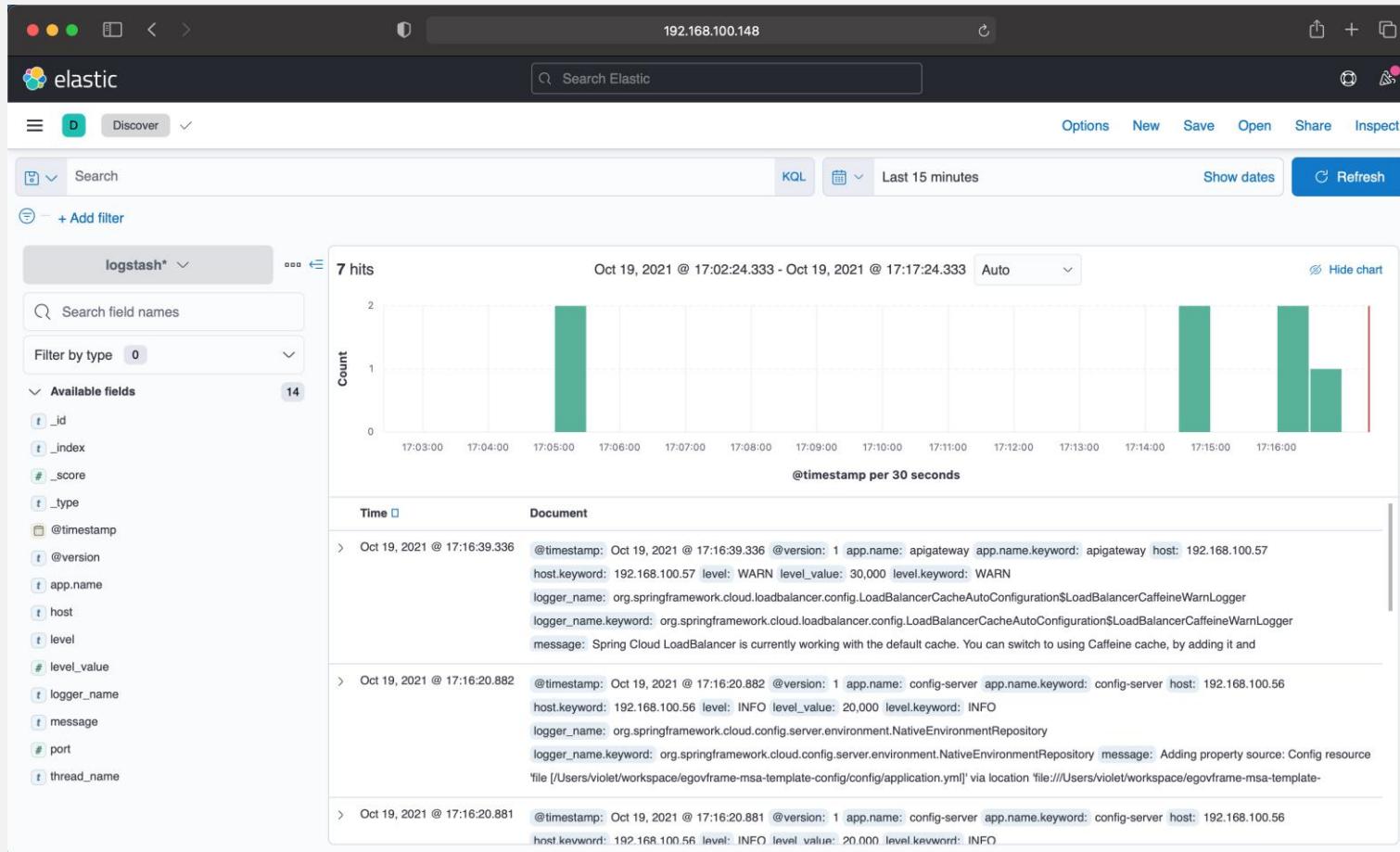
spring-logback.xml(spring boot)

```
...  
<!-- ELK – Logstash 로 로그를 전송하기 위한 appender -->  
<appender name="LOGSTASH" class="net.logstash.logback.appenders.LogstashTcpSocketAppender">  
    <destination>localhost:5001</destination> <!-- Logstash tcp port -->  
    <encoder class="net.logstash.logback.encoder.LogstashEncoder">  
        <customFields>{"app.name":"egov-apigateway"}</customFields>  
    </encoder>  
</appender>  
  
<root level="WARN">  
    <appender-ref ref="LOGSTASH" />  
    <appender-ref ref="STDOUT" />  
</root>
```



Centralized Logging – ELK

💡 모아진 로그들을 Kibana로 볼 수 있다.



MySQL(docker-compose)

- ① cmder bash 창에서
- ② 도커 네트워크 생성 후 docker-compose 파일이 있는 위치로 이동
- ③ “docker network create egov-network”
- ④ “cd \${HOME}/workspace.edu/egovframe-msa-edu/docker-compose/mysql”
- ⑤ mysql 컨테이너 생성 및 확인(데이터베이스 생성 및 데이터는 자동 입력됨)
- ⑥ “docker-compose up -d”
- ⑦ “docker exec -it mysql bash” 컨테이너 내부에 접속
- ⑧ “mysql -u msaportal -p” 패스워드는 msaportal
- ⑨ “show databases;” 로 msaportal, reservation 데이터베이스 생성 확인
- ⑩ “exit;” 두 번 반복하여 컨테이너에서 나옴

① RabbitMQ(docker run)

- ① cmder bash 창에서 RabbitMQ 컨테이너 실행
- ② “docker run -d -e TZ=Asia/Seoul --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:management”

● Docker 실행

💡 Docker를 활용하여 mysql, rabbitmq, zipkin 을 실행합니다.

- ① cmder bash 창에서
- ② “docker network create egov–network” 로 네트워크 생성
- ③ “cd \${HOME}/workspace.edu/egovframe-msa-edu/docker–compose/mysql” 경로로 이동
- ④ “docker–compose up –d”

```
# mysql
$ docker network create egov–network
$ cd ${HOME}/workspace.edu/egovframe-msa-edu/docker–compose/mysql
$ docker–compose up –d

# rabbitmq
$ docker run –d –e TZ=Asia/Seoul --name rabbitmq –p 5672:5672 –p 15672:15672 rabbitmq:management

# zipkin
docker run --name zipkin –d –p 9411:9411 –e TZ=Asia/Seoul openzipkin/zipkin
```