NIA 한국지능정보사회진흥원

클라우드 네이티브 기반 행정·공공 서비스 확산 지원

MSA 기반의 표준프레임워크 템플릿 소개

12/22. 발표 자료

INDEX

클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

Cloud Native & MSA(Micro Service Architecture)

NIA 한국지능정보사회진흥원

클라우드 네이티브 개념

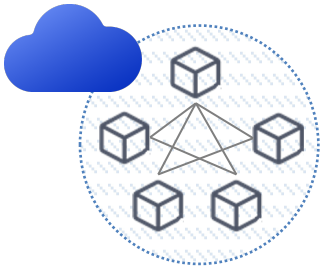
클라우드 이전과 이후

서버마다 정해진 기능

IP, NAME 이 주어지고 웹 서버, 데이타베이스 등
기능을 수동으로 관리
➔ 서버 확장/이관이 복잡

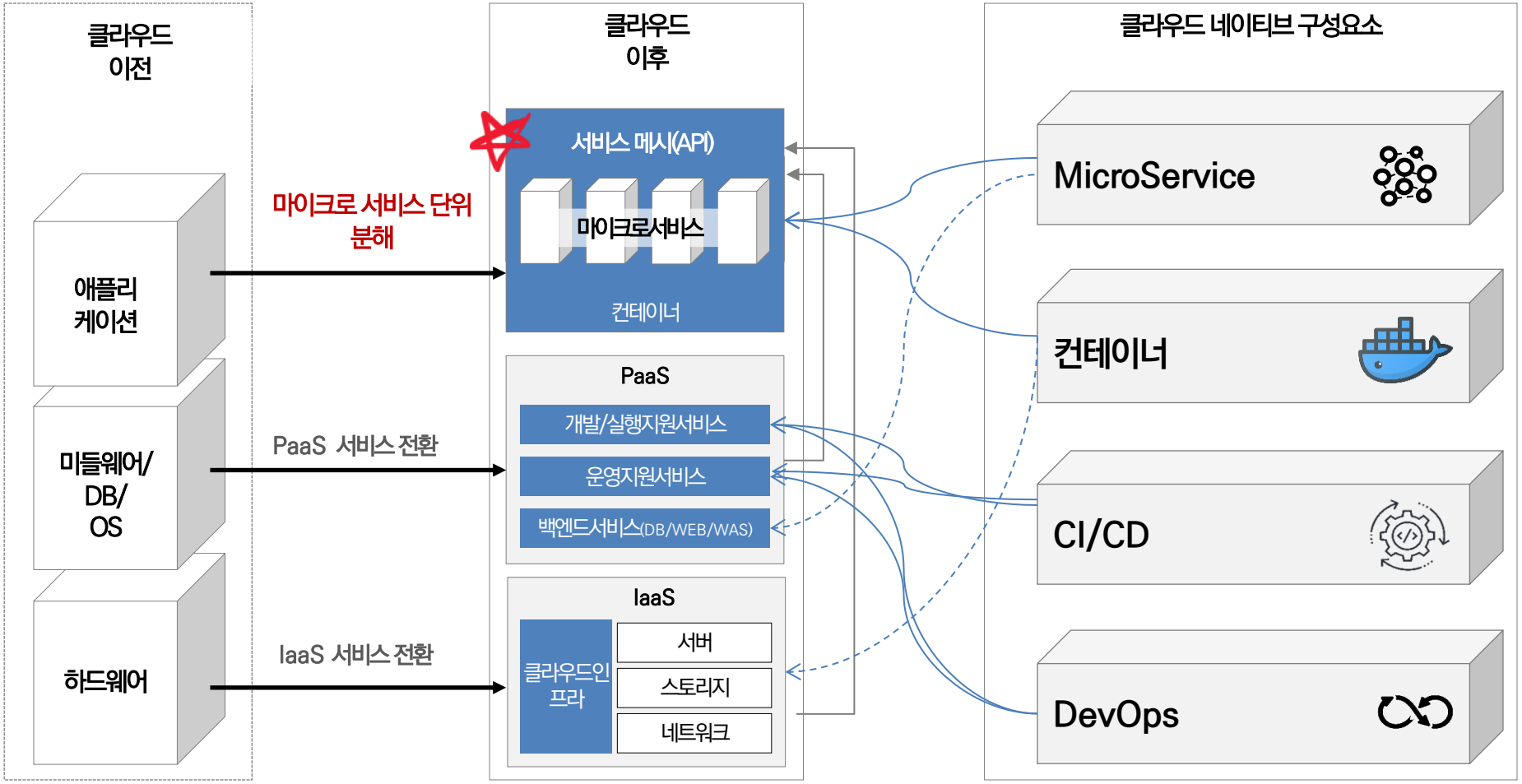
자유롭게 사용

IP, NAME 이 중요하지 않고
어딘가에 떠 있기만 하면 되는 환경
수 많은 리소스를 추상적으로 관리
➔ 서버 확장/이관이 용이



클라우드 네이티브 애플리케이션

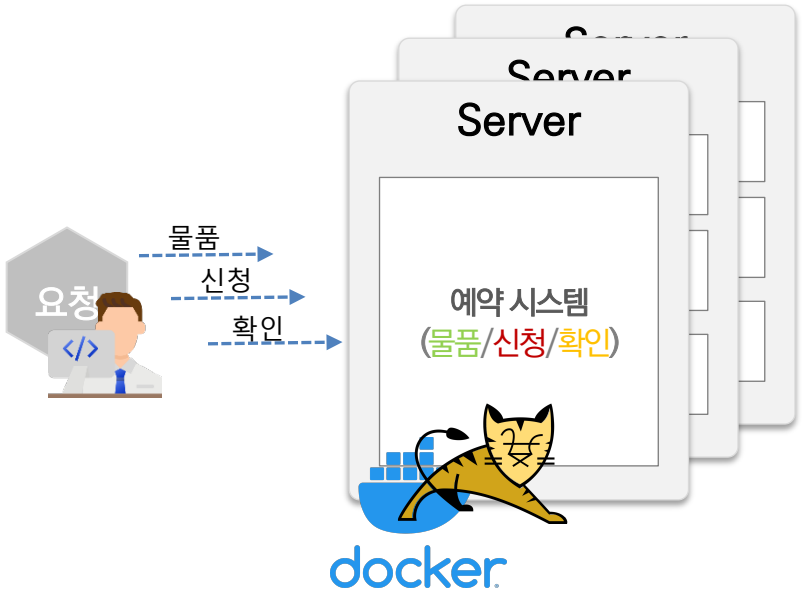
클라우드 환경에서 어떻게 애플리케이션을 만들고 배포하는게 효율적일까?



마이크로 서비스 아키텍처 (1/3)

탄력적, 선택적인 서비스 확장을 제공합니다.

서버 증설이 용이한 클라우드 환경에 적합한 아키텍처

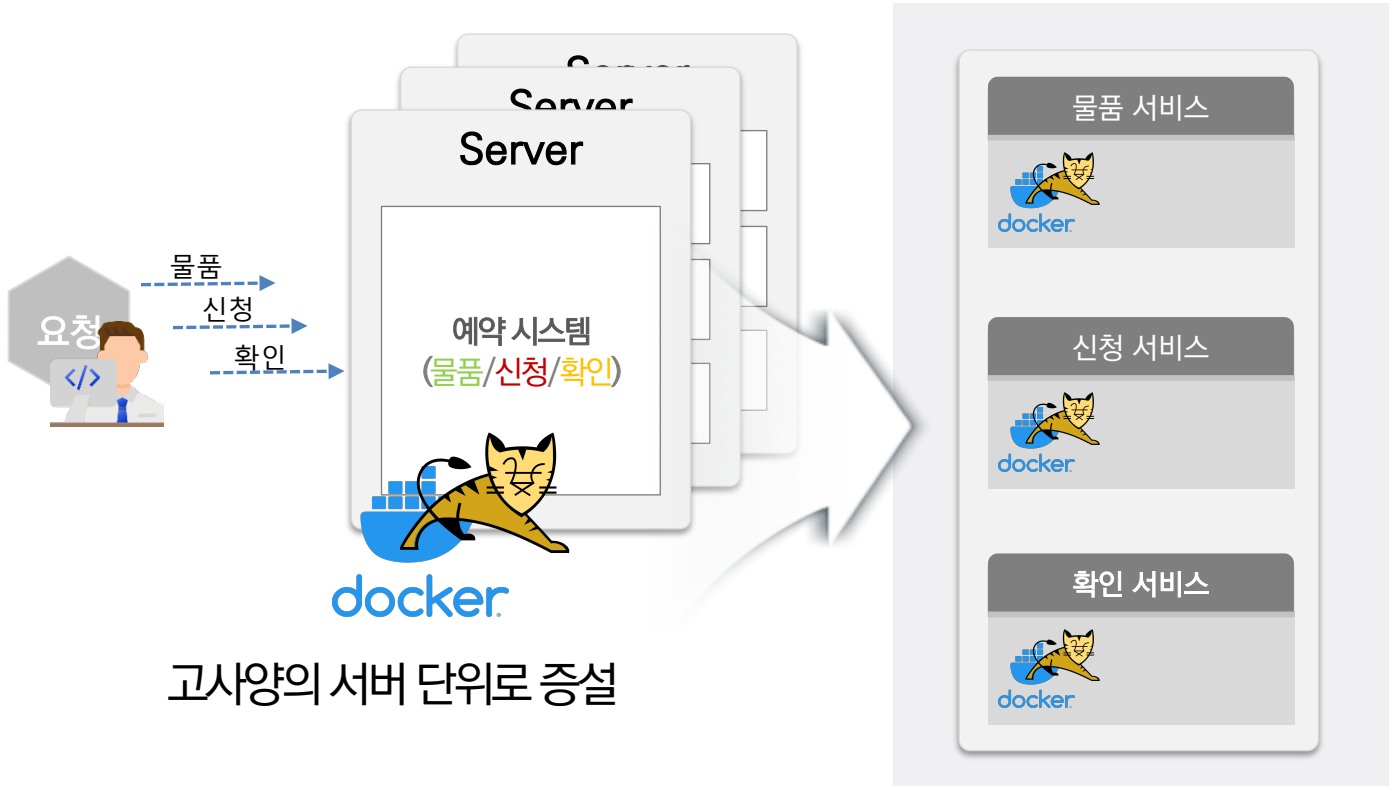


고사양의 서버 단위로 증설

마이크로 서비스 아키텍처 (1/3)

탄력적, 선택적인 서비스 확장을 제공합니다.

서버 증설이 용이한 클라우드 환경에 적합한 아키텍처

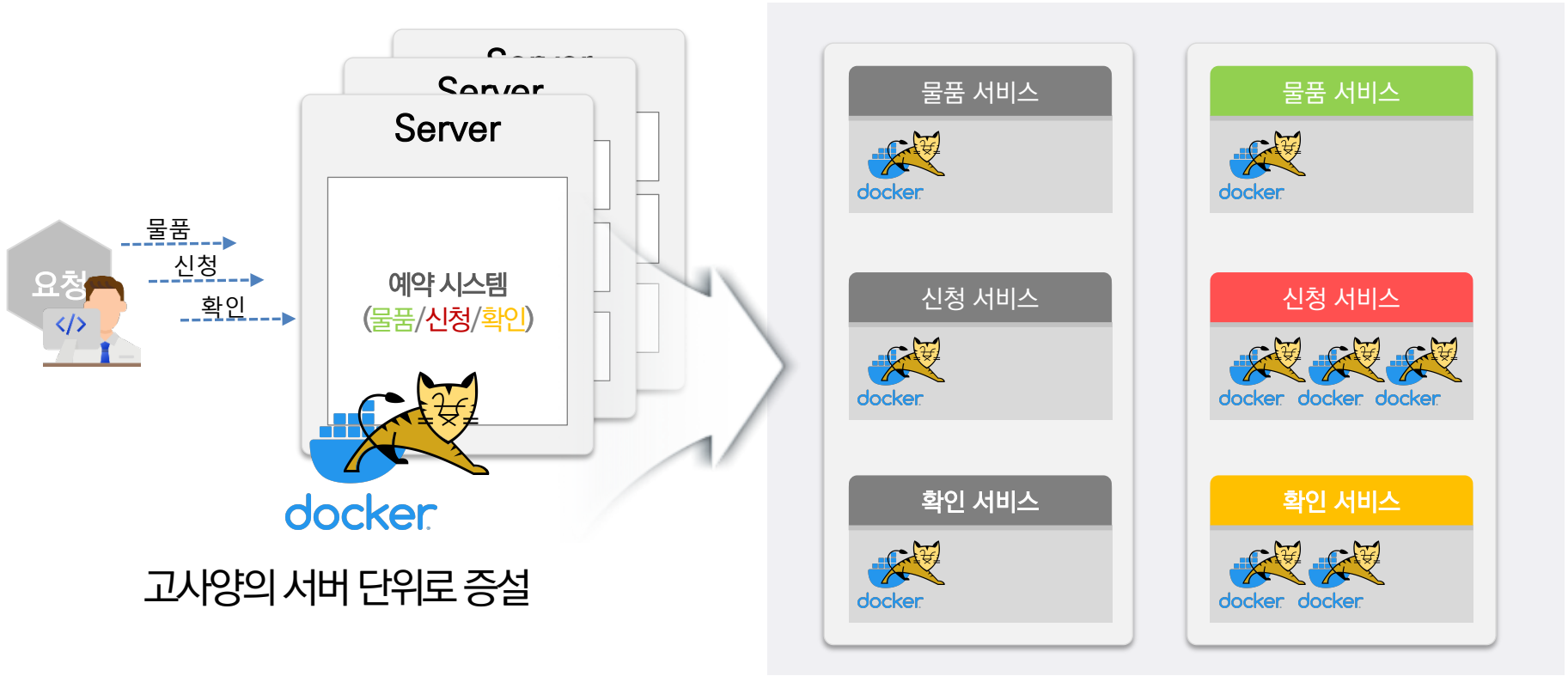


고사양의 서버 단위로 증설

마이크로 서비스 아키텍처 (1/3)

탄력적, 선택적인 서비스 확장을 제공합니다.

서버 증설이 용이한 클라우드 환경에 적합한 아키텍처

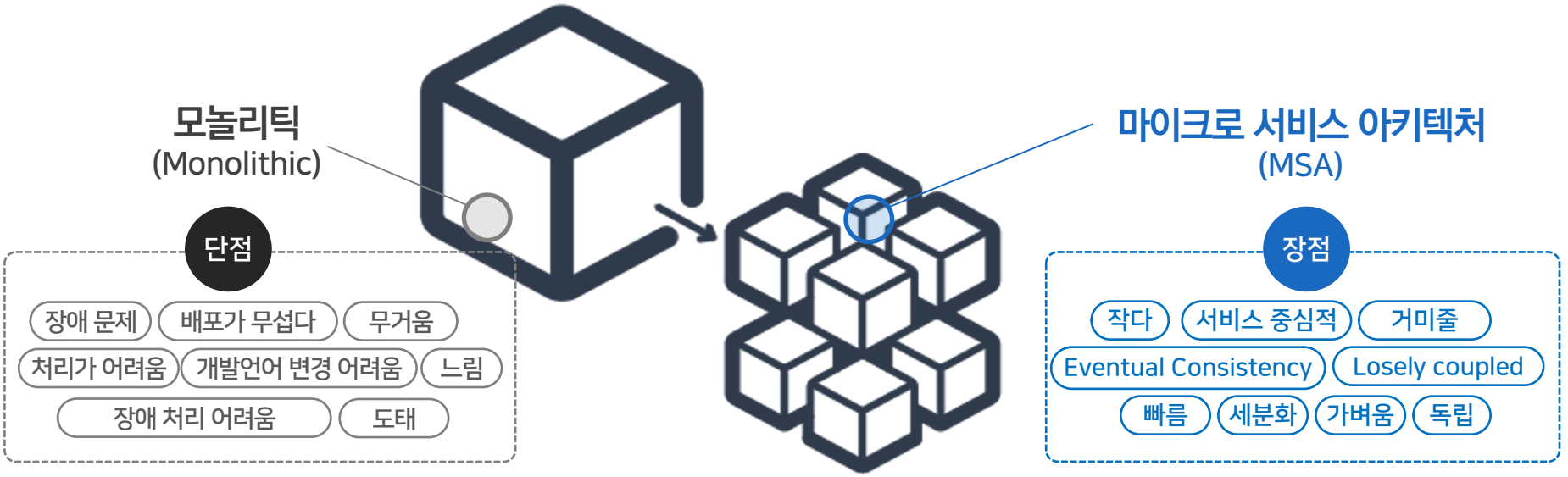


고사양의 서버 단위로 증설

마이크로 서비스 아키텍처 (2/3)

탄력적, 선택적인 서비스 확장을 제공합니다.

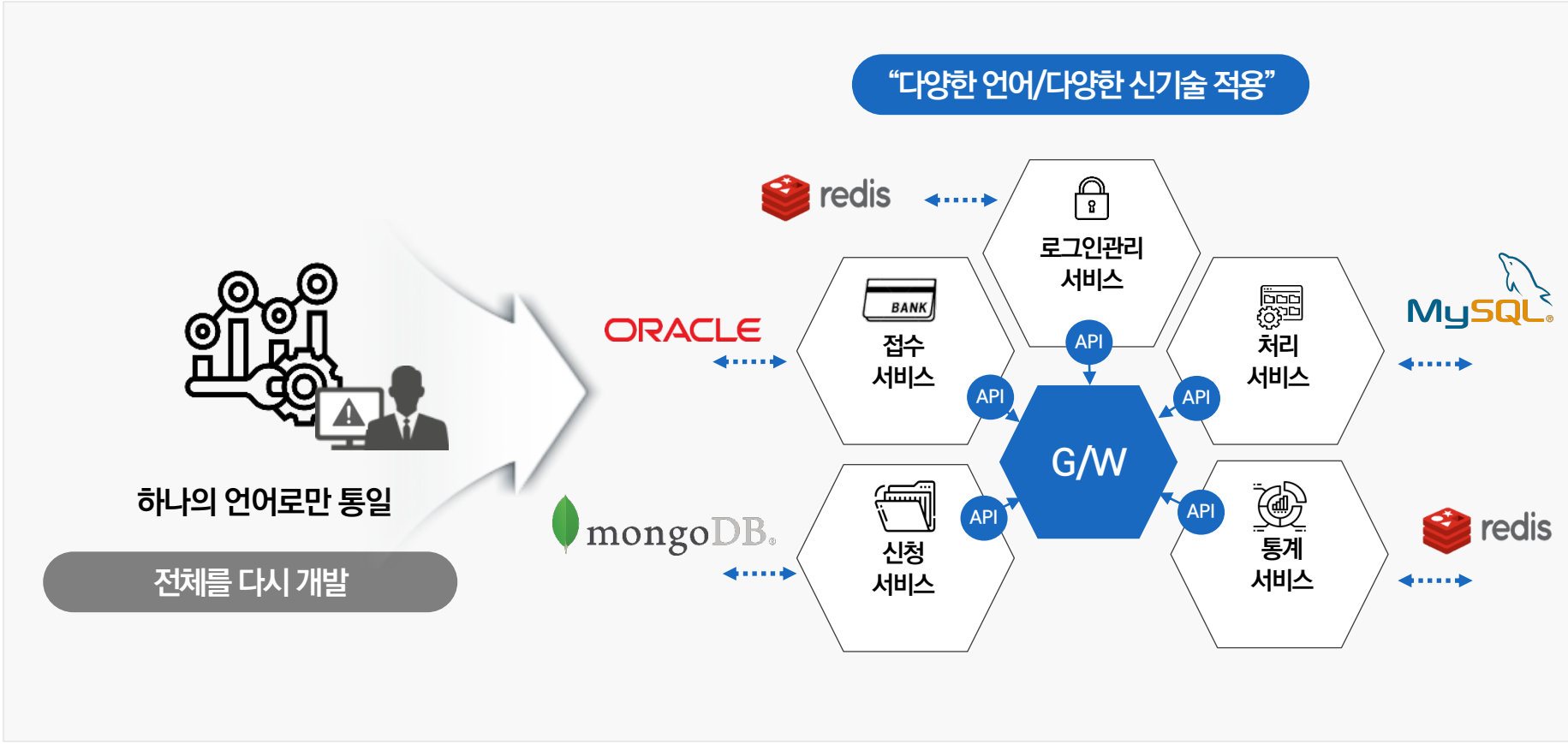
API로 통신하는 소규모의 독립적인 서비스로 구성하여,
클라우드 환경에 최적화된 느슨한 결합 (빠른, 신속한, 편리한 업무)



마이크로 서비스 아키텍처 (3/3)

탄력적, 선택적인 서비스 확장을 제공합니다.

기존에는 요구사항이나 전체를 다시 개발, 빠른 요구에 대응 부합하는 선택적 개발
업무 용도에 적합한 개발언어, DBMS 등의 기술 적용



마이크로 서비스 아키텍처

MSA 많이 사용하나요?

점점 커져가는 서비스를 감당하기 위해 클라우드 환경에 적합한
마이크로 서비스 아키텍처를 도입

구인 공고

자격요건
- 웹 어플리케이션 개발, 운영 경력 5년 이상이신 분(또는 그에 준하는 역량을 갖추신 분)
- JAVA에 익숙하고, Kotlin 또는 그 외 개발언어 사용 경험이 있으신 분
- Spring 프레임웍(Spring Boot)을 이용한 Web Application 개발 경험이 있으신 분
- MVC 또는 Reactive framework 기반의 웹 서비스나 API 개발 경험이 있으신 분
- MySQL(Maria DB) 등의 RDBMS 경험이 있으신 분

우대사항
- 광고시스템 관련 업무 경험이 있으신 분 (디스플레이, 키워드 광고 등)
- Elasticsearch, Solr 사용 경험이 있으신 분
- JPA, Hibernate 등 ORM 사용과 도메인 모델링 경험이 있으신 분
- AWS를 활용한 개발, 운영 경험이 있으신 분
- 시스템 모니터링 및 알람 구성 경험이 있으신 분
- 빌드/테스트/배포 자동화 경험이 있으신 분
- Microservices 아키텍처 기반의 시스템 개발 경험이 있으신 분
- MQ(Kafka, RabbitMQ, ActiveMQ 등) 사용 경험이 있으신 분

- MSA 기반의 신규 플랫폼 프로젝트 참여

자격요건
- Java+Spring 개발 경력 3년 이상
- JPA(Hibernate) 등의 ORM 라이브러리 사용 경험
- MySQL, Oracle 등 RDBMS 경험 (기본적인 SQL 작성 능력을 보유하신 분)
- TDD를 경험해보았거나 JUnit을 활용한 테스트를 학습
- 리팩토링과 코드리뷰를 두려워 하지 않는 분
- 새로 배우는 것을 두려워 하지 않는 분
- 내가 만든 결과물의 완성도에 대한 욕심이 있으신 분

우대사항
- 물류 서비스 개발/운영 경험이 1년 이상
- 레거시 시스템 개선 경험
- AWS(cloud) 기반 서비스 개발에 익숙한 개발자
- SpringBoot 기반 개발 경험이 있으신 분(2.0 이상)
- Vue.js, React 등 프론트엔드 프레임워크 기반 개발 경험이 있으신 분
- 동료에게 지식을 공유하기 좋아하는 분(세미나, Blog 등)
- 문서 작성을 즐기시는 분
- 자기주도적으로 업무를 진행하기 원하시는 분

자격요건
- 여러 조직간의 협업에 뛰어나고, 좋은 의사 소통 능력을 가지신 분
- 동료들과 적극적으로 소통하며 팀 플레이어로써 업무 하실 수 있는 분
- 5년 이상의 backend architecture 설계 및 구현 경험이 있는 분
- 오픈 소스 기술, 소프트웨어 개발 및 시스템 엔지니어링에 대한 광범위한 경험이 있는 분
- Cloud background (docker, K8S and AWS/ Google/ Azure) 이 있는 분

우대사항
- 애자일 소프트웨어 개발 사례에 대한 경험과 개선, 계획, 데모, 회고와 같은 스프린트 행사에 기여할 수있는 능력이 있으신분
- 자신의 팀이나 조직의 다른 곳에서 다른 엔지니어를 멘토링한 경험이 있으신분
- CI / CD (지속적 통합 및 지속적 전달) 사례, 자동화 된 품질 게이트 및 자동화 된 배포 경험이 있으신분.
- API Gateway, Service Mesh 를 company-wide 적용 경험이 있는 분
- Distributed systems, Micro services, Messaging services 을 경험하신 분
- 핀테크 관련 조직에서의 소프트웨어 개발 경력이 있는 분

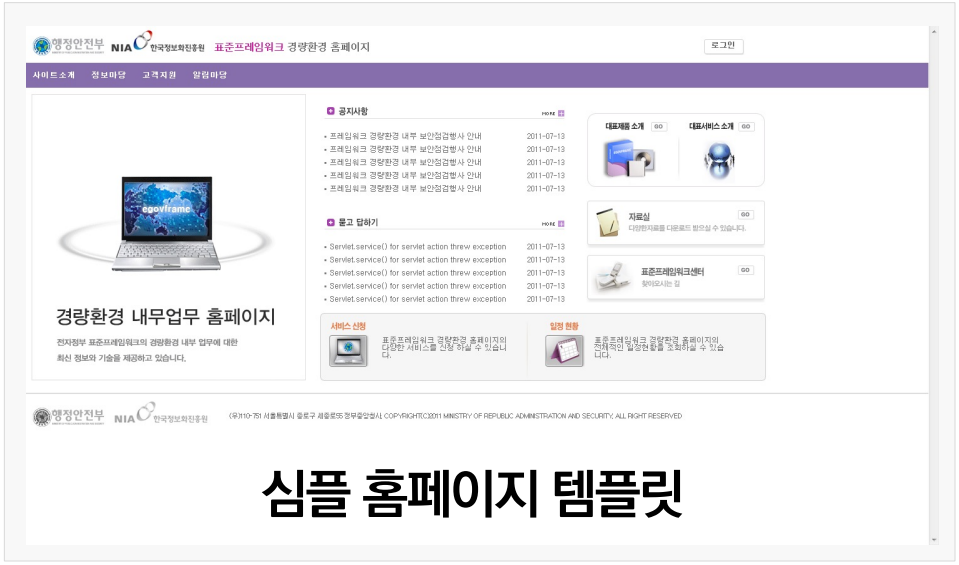
클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

표준프레임워크 MSA 템플릿 소개

NIA 한국지능정보사회진흥원

표준프레임워크 기존 템플릿

표준프레임워크 심플/포털 템플릿 제공



심플 홈페이지 템플릿

포털 홈페이지 템플릿

표준프레임워크 MSA 적용 개발 가이드

표준프레임워크 MSA 개발을 실습 해볼 수 있도록
표준프레임워크 포털에서 개발 가이드 제공

https://www.egovframe.go.kr/home/ntt/nttRead.do?pagerOffset=0&searchKey=&searchValue=&menuNo=76&bbsId=171&nttId=1809

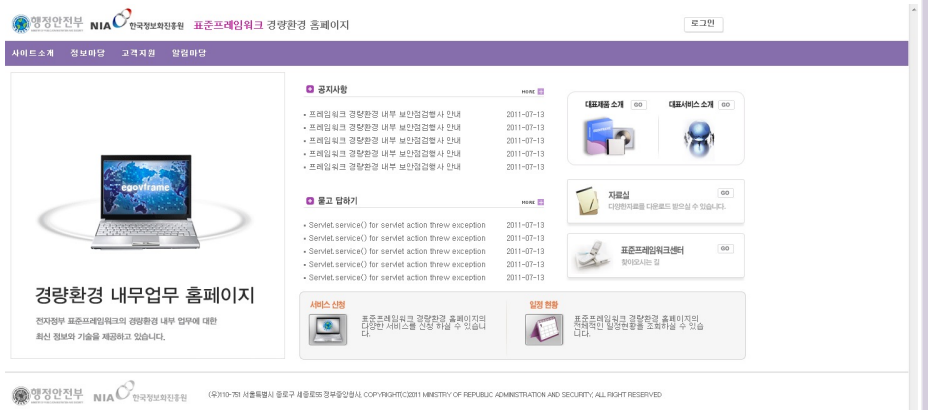
표준프레임워크 MSA 적용 개발 가이드입니다.

<목차구성>

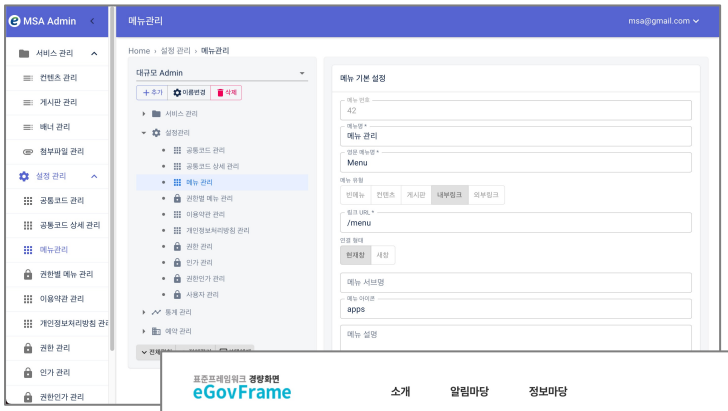
MSA 템플릿 구성 (1/2)

기존 표준프레임워크 템플릿(심플/포털)의 기능 중심으로 MSA 로 구현
관리자와 사용자를 분리

기존 심플 홈페이지 템플릿



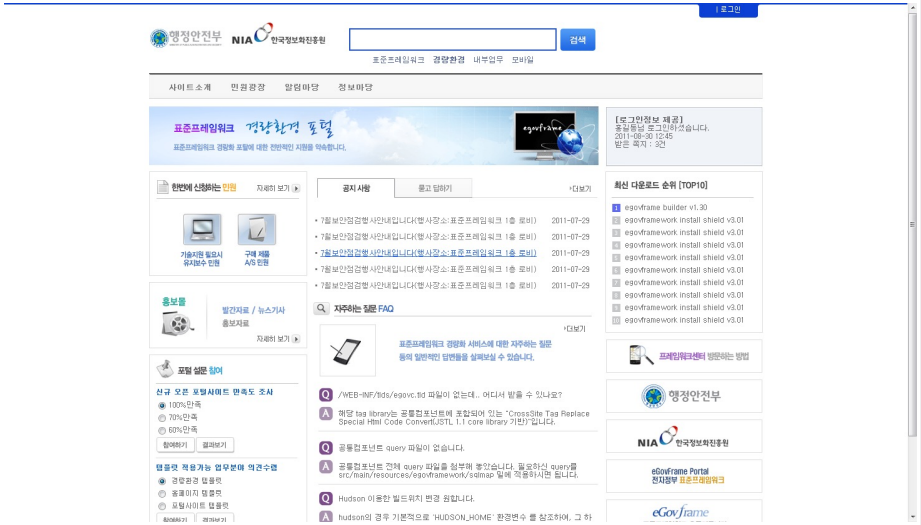
MSA 소규모 템플릿(관리자, 사용자)



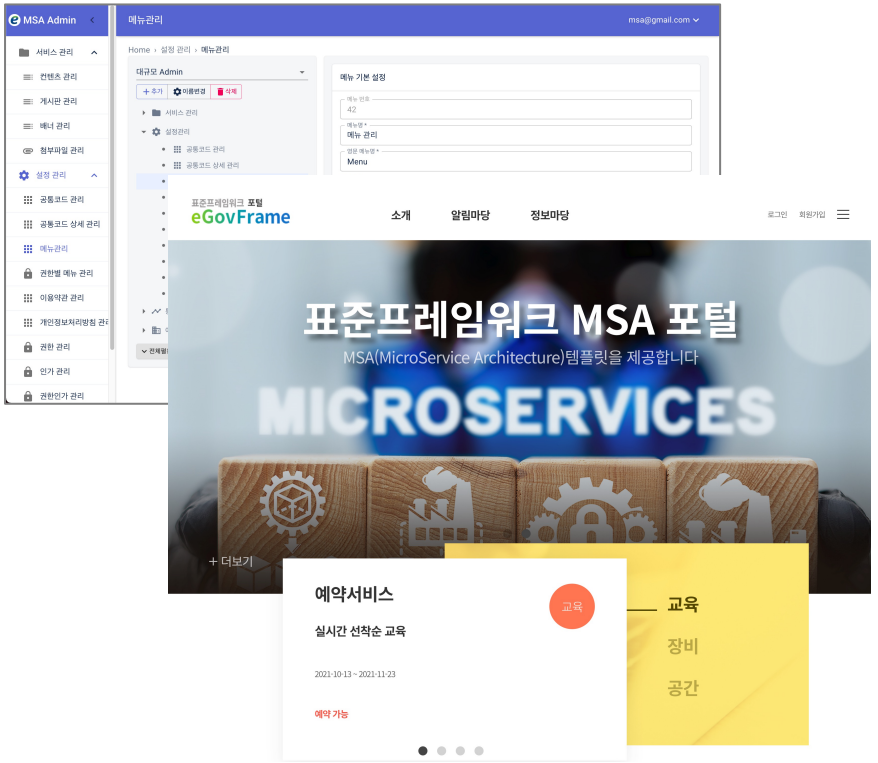
MSA 템플릿 구성 (2/2)

기존 표준프레임워크 템플릿(심플/포털)의 기능 중심으로 MSA 로 구현
관리자와 사용자를 분리, 예약(Non-blocking)/OAuth2 추가

기존 포털 홈페이지 템플릿



MSA 대규모 템플릿(관리자, 사용자)



MSA 템플릿 실행 환경

전자정부 클라우드 플랫폼에서 실행 확인

표준프레임워크 4.0

- Spring Boot 2.4.5
- Openjdk 1.8+, Gradle 6.8

Spring Cloud 2020.0.3

- Hystrix/Ribbon/Zuul (Maintenance Mode, Replacement)

=〉 Resilience4j/Spring Cloud Loadbalancer/Spring Cloud Gateway

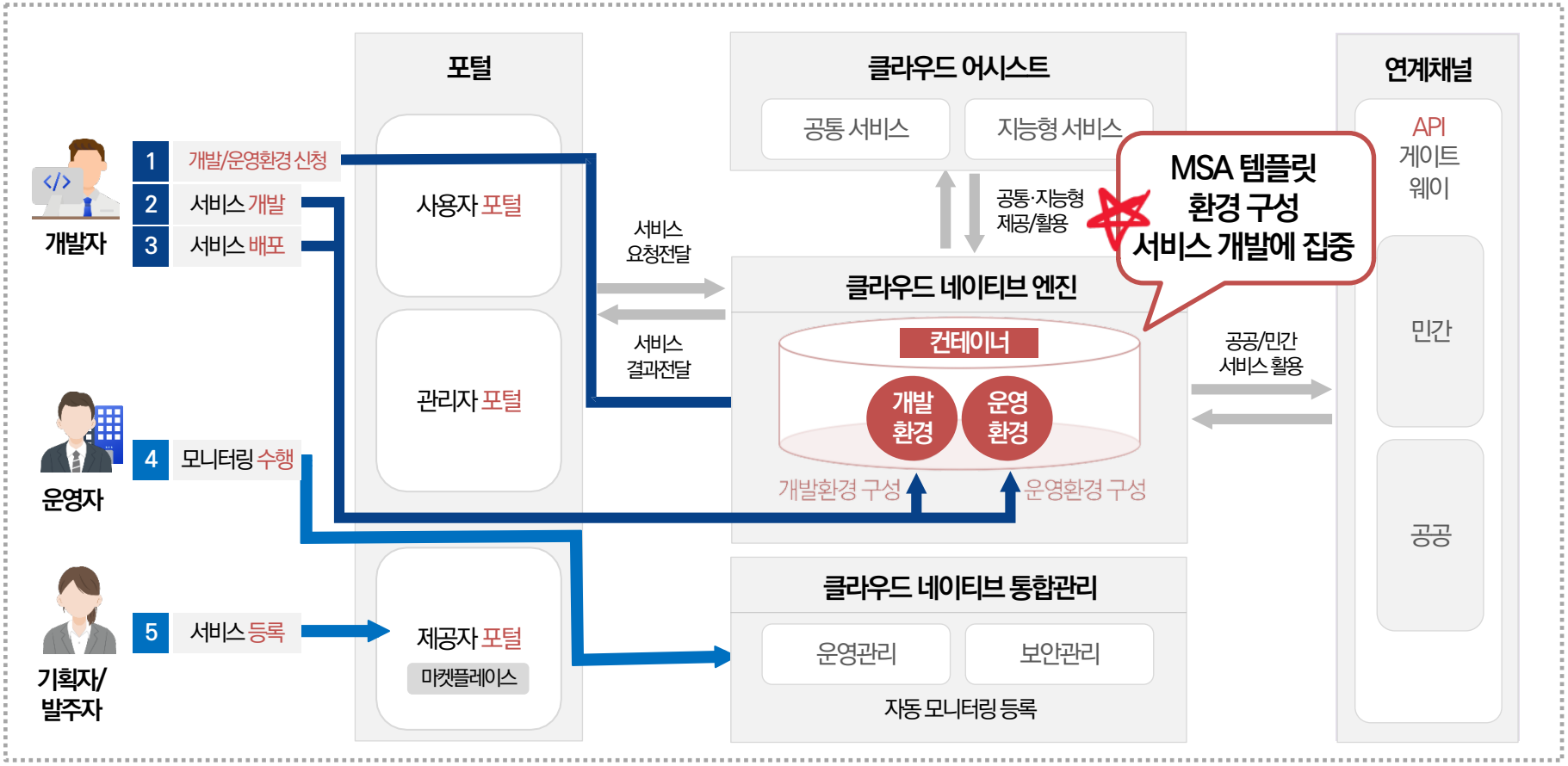
Docker engine 20.10.7

- 로컬/Kubernetes/Cloud Foundry 환경에서 배포

전자정부 클라우드 플랫폼

전자정부 클라우드 플랫폼 기반 클라우드 네이티브 애플리케이션 개발

개발 초기 단계부터 제공되는 템플릿을 배포해보면 빠르게 환경 구성하여 서비스 개발에 집중

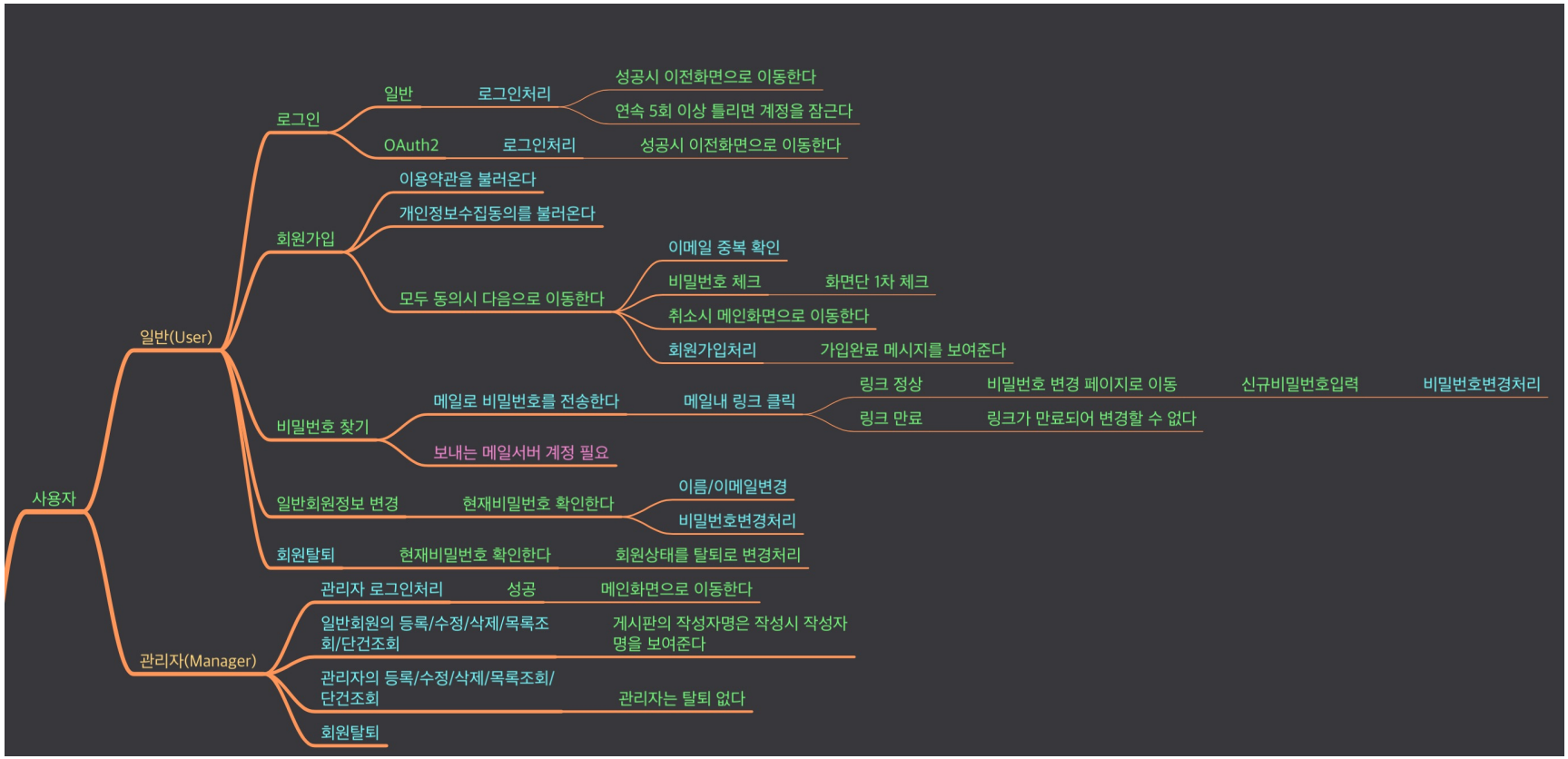


클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

MSA 템플릿
제작 과정

NIA 한국지능정보사회진흥원

MSA 템플릿 회원 서비스 식별 과정 실제 예시



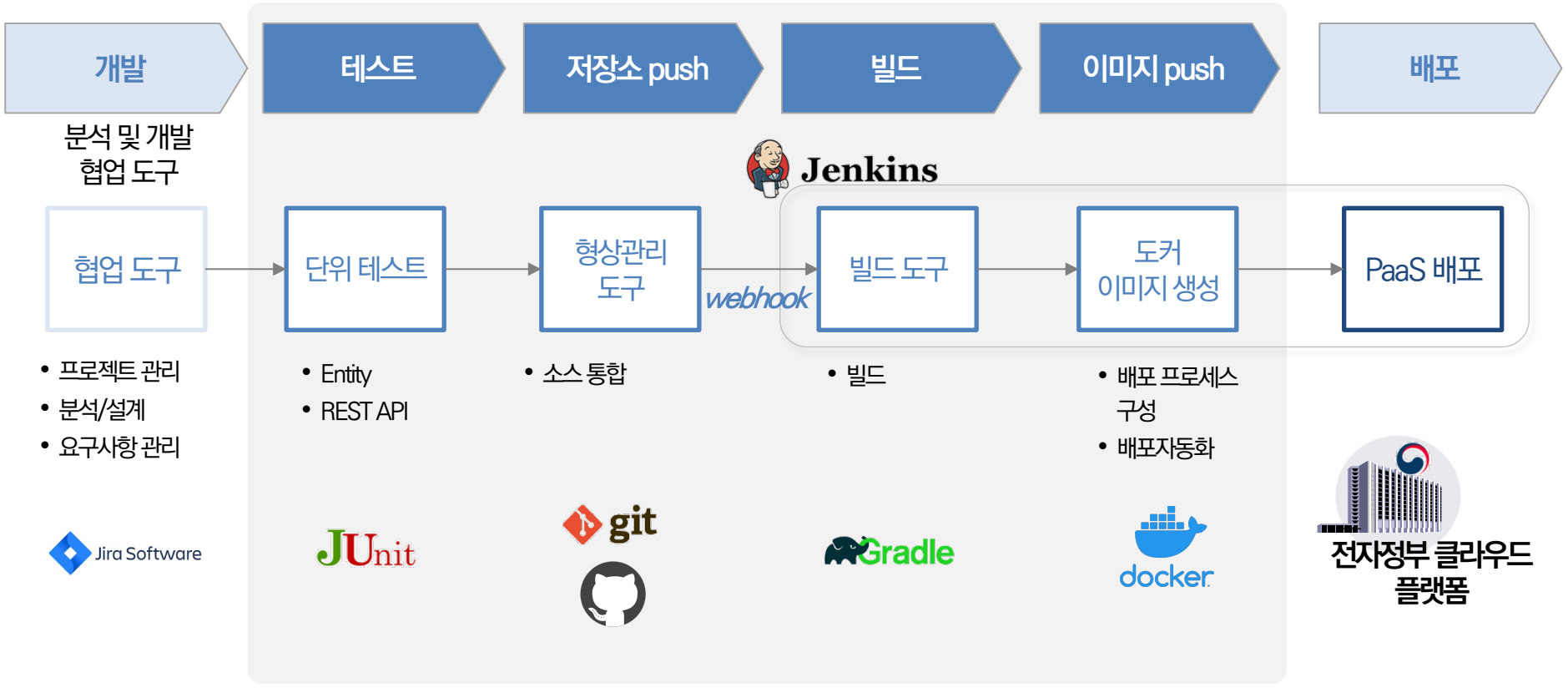
서비스 식별

회원/게시판/포털 서비스 로 식별하였고
대규모 템플릿에 예약(신청/물품/확인) 서비스가 추가

CI/CD 구성

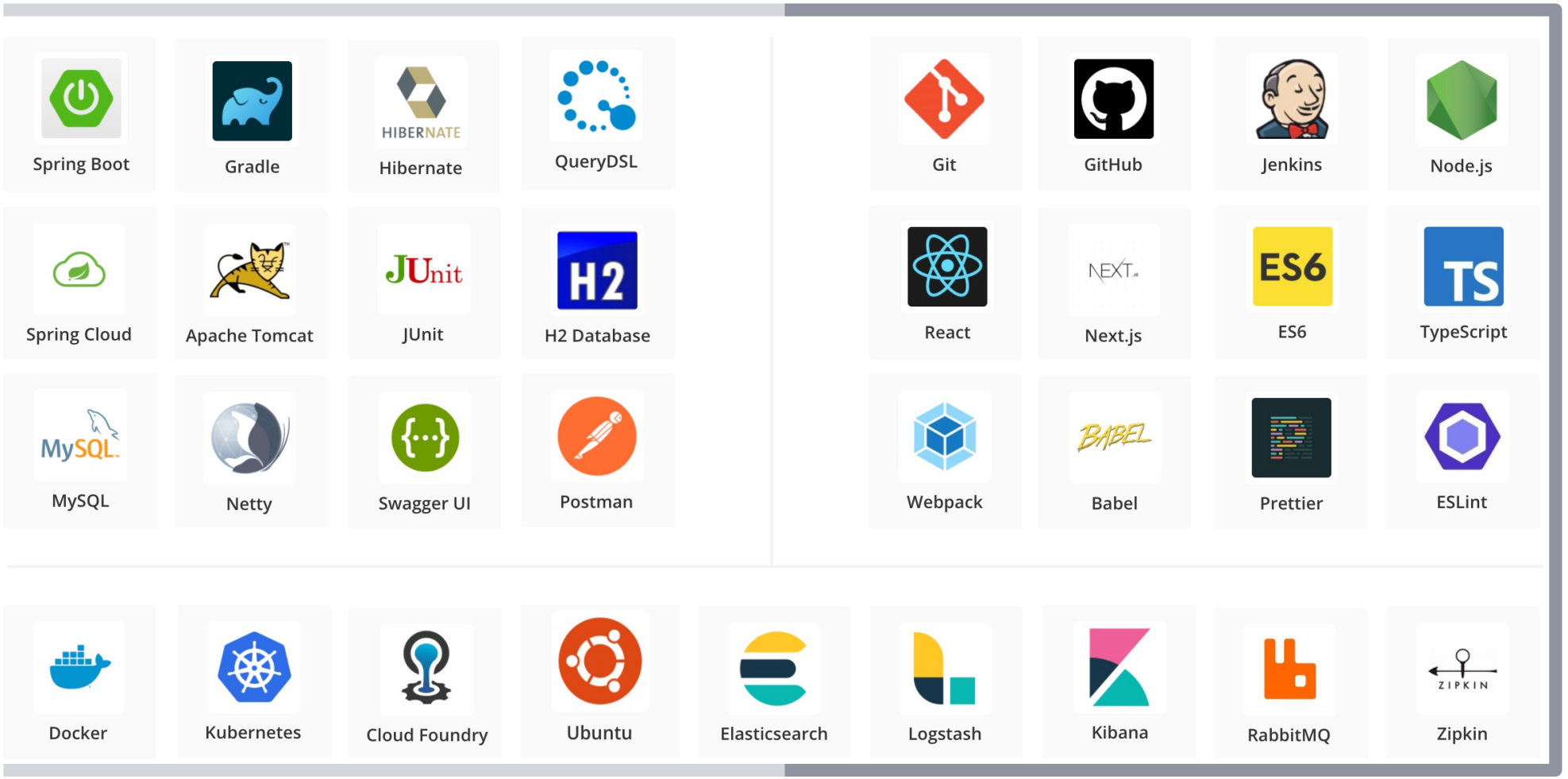
개발된 소스가 저장소에 올라가면 자동으로 빌드하여
도커 이미지를 생성하여 도커 레지스트리에 올린 후 서비스가 배포 된다.

CI/CD 파이프라인



기술 스택

MSA 템플릿에 적용된 기술



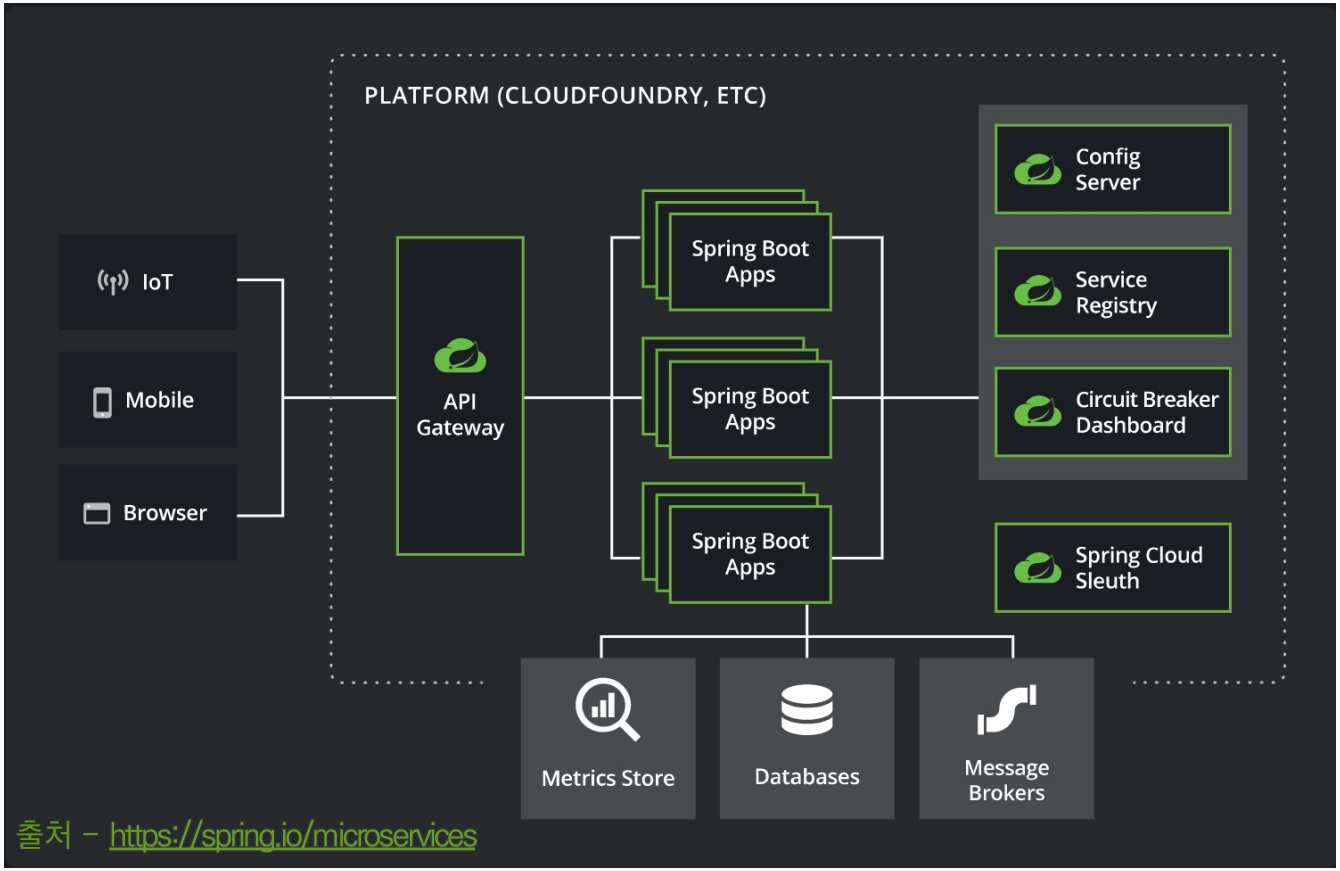
클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

Spring Cloud 기반의 Service Mesh 구현

NIA 한국지능정보사회진흥원

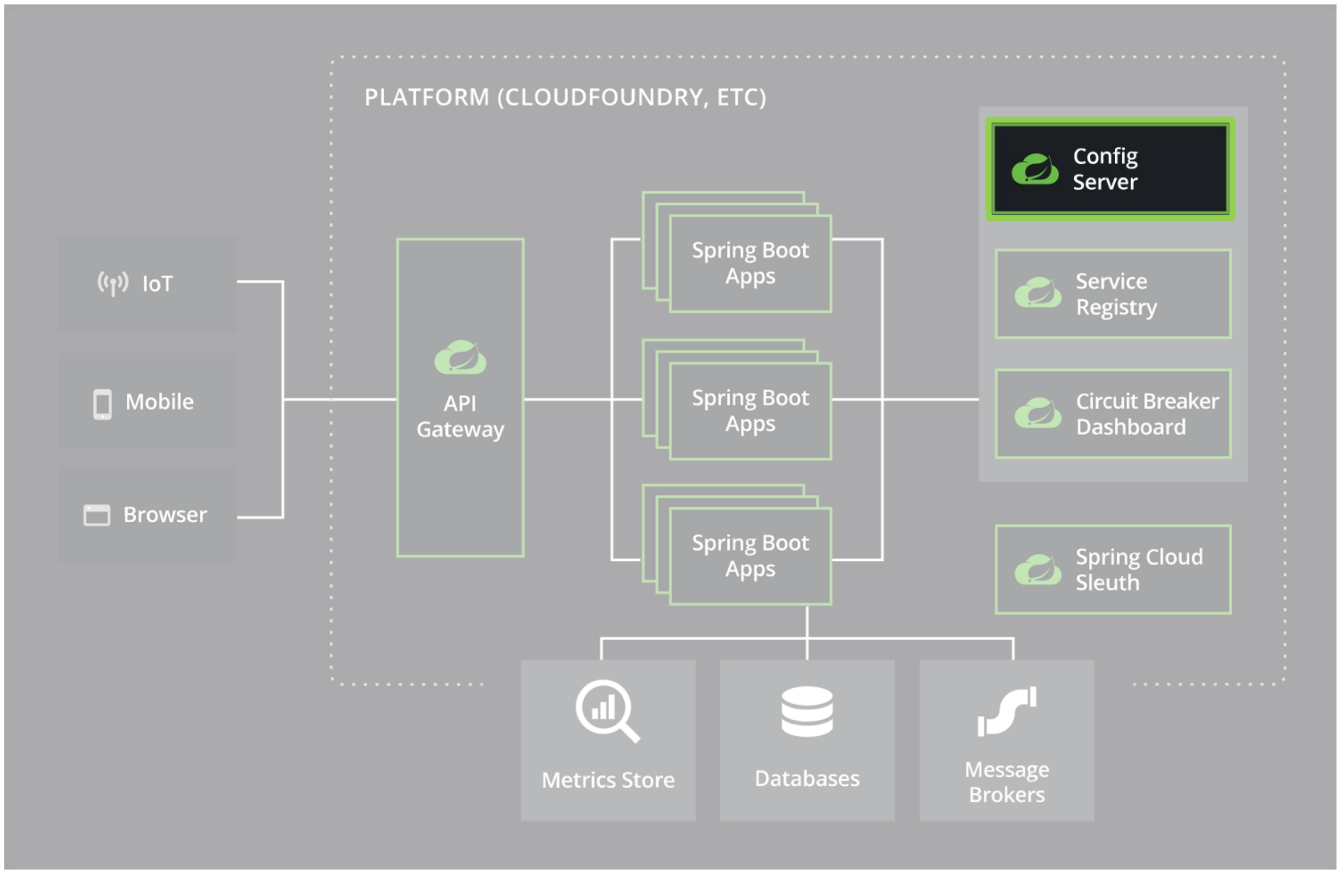
Service Mesh 에 대한 이해

하나의 시스템을 여러 서비스로 나누었지만
서로 데이터를 공유하고 통신할 수 있도록 구성. 템플릿에 포함



Config Server

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



Config Server

Spring Boot Application 생성 ➔ build.gradle 에 config server 의존성 추가 ➔ @EnableConfigServer 선언

build.gradle

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

ConfigApplication.java

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

Config Server

배포 시 config server 의 application-prod.yml 파일에 git uri 지정

config-server/application-prod.yml

```yaml
server:
  port: 8888

spring:
  application:
    name: config-service
  cloud:
    config:
      server:
        git:
          uri: https://github.com/eGovFramework/config-server
          username: '{cipher}ba3...'
          password: '{cipher}26a...'
          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
  rabbitmq:
    host: ${rabbitmq.hostname:localhost}
    port: 5672
    username: guest
    password: guest
```

Config Server

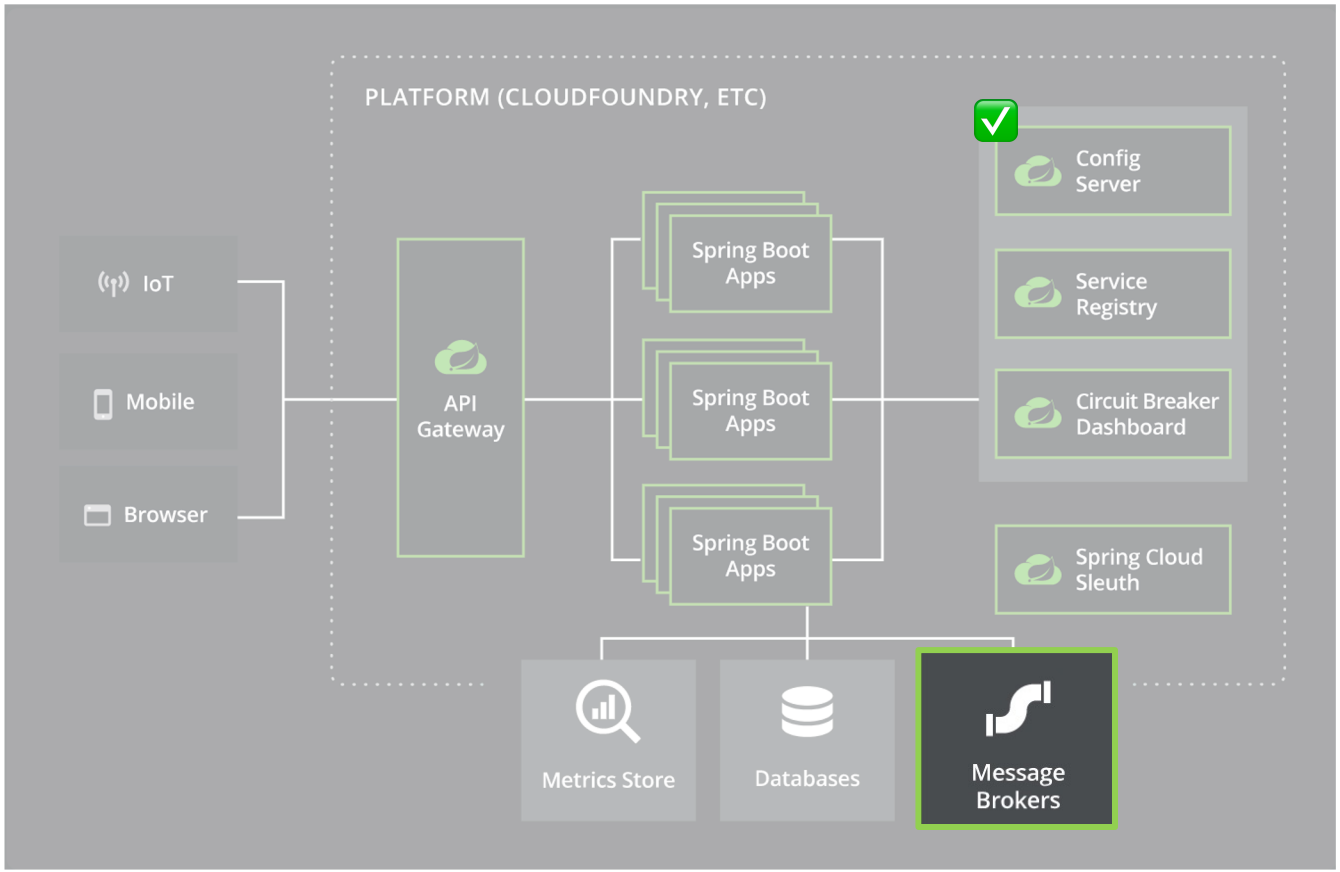
개발 환경 에서는 config server 의 application.yml 파일에 읽어올 설정 파일의 위치 지정

config-server/application.yml

```yaml
server:
  port: 8888

spring:
  application:
    name: config-service
  profiles:
    active: native,default # native file repository
  cloud:
    config:
      server:
        native:
#          search-locations: 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
```

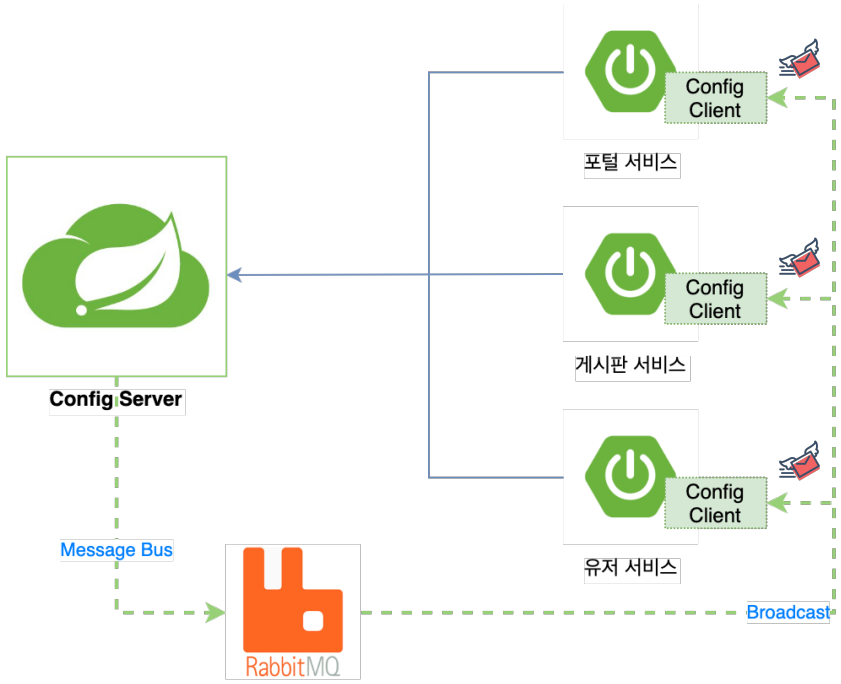
Message Brokers

Message Broker(RabbitMQ) 가 변경된 환경설정 정보를 각 인스턴스에 자동으로 갱신될 수 있도록 하는 역할



Spring Cloud Bus + Message Brokers

Spring Cloud Bus + Message Broker(RabbitMQ) 가 메시지가 발생하면 메시지를 수신 중인 클라이언트에 메시지를 전달



Spring Cloud Bus + Message Brokers

Message Broker(RabbitMQ) 서버 실행
접속 정보 설정

RabbitMQ dockder run

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

RabbitMQ 접속 정보는 ① config server 내부와
② config server 가 관리하는 config 파일에 모두 정의한다.

① config-server/application.yml

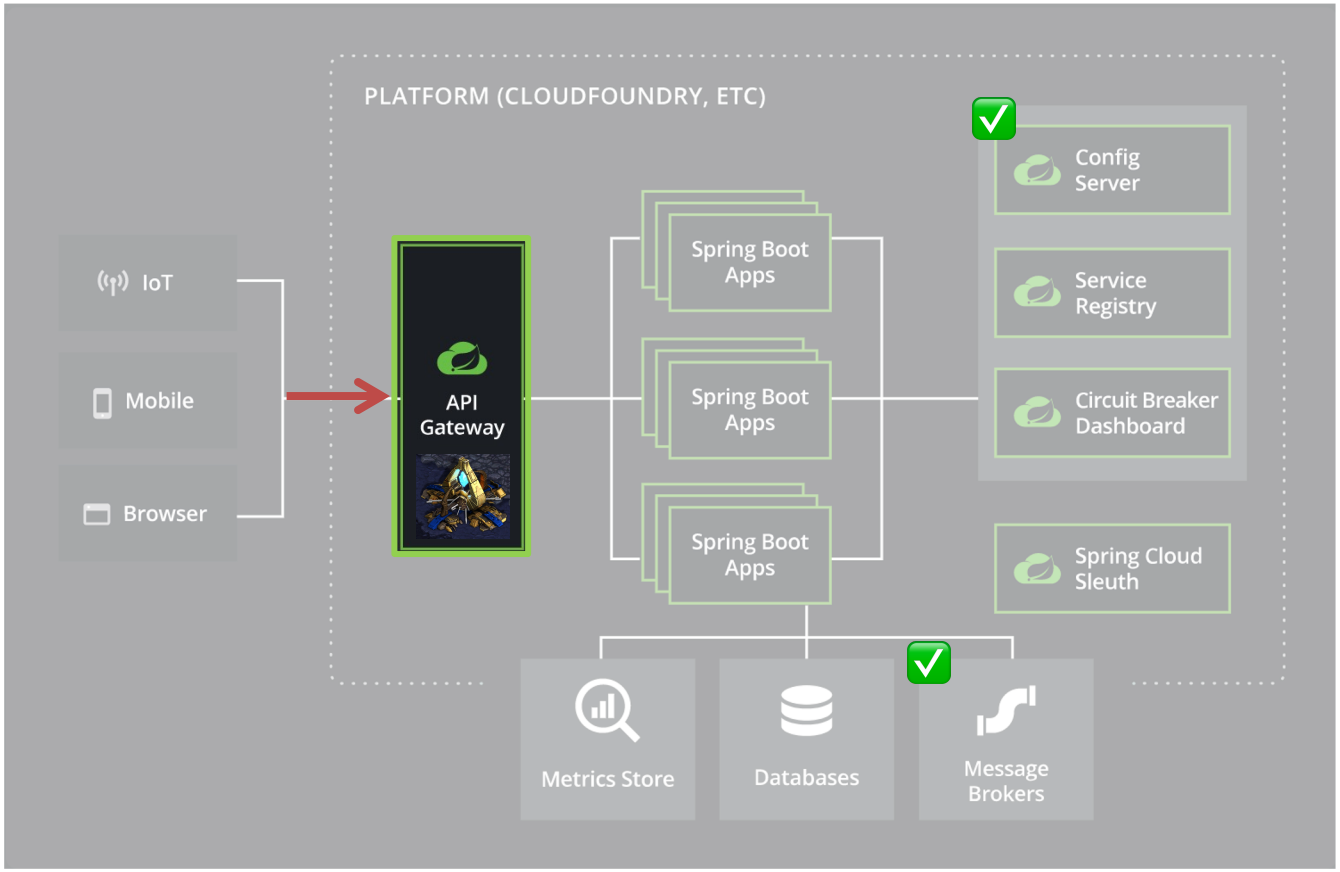
```
// RabbitMQ 서버 정보
spring:
  rabbitmq:
    host: localhost
    port: 5672
    username: guest
    password: guest
```

② config/application.yml

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

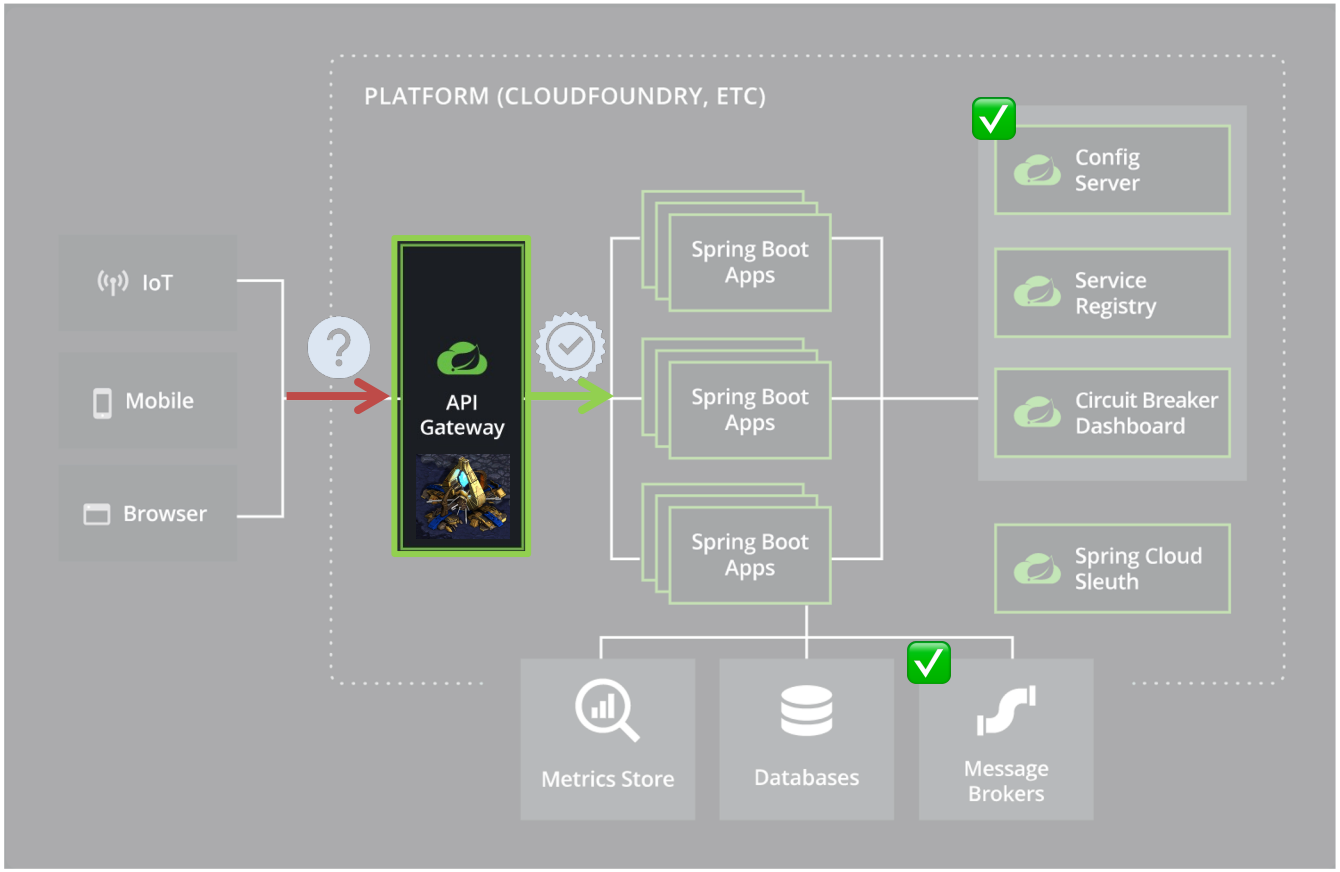
API Gateway

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



API Gateway

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



API Gateway

Spring Boot Application 생성 ➔ build.gradle 에 gateway 의존성 추가

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-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
}
```

API Gateway

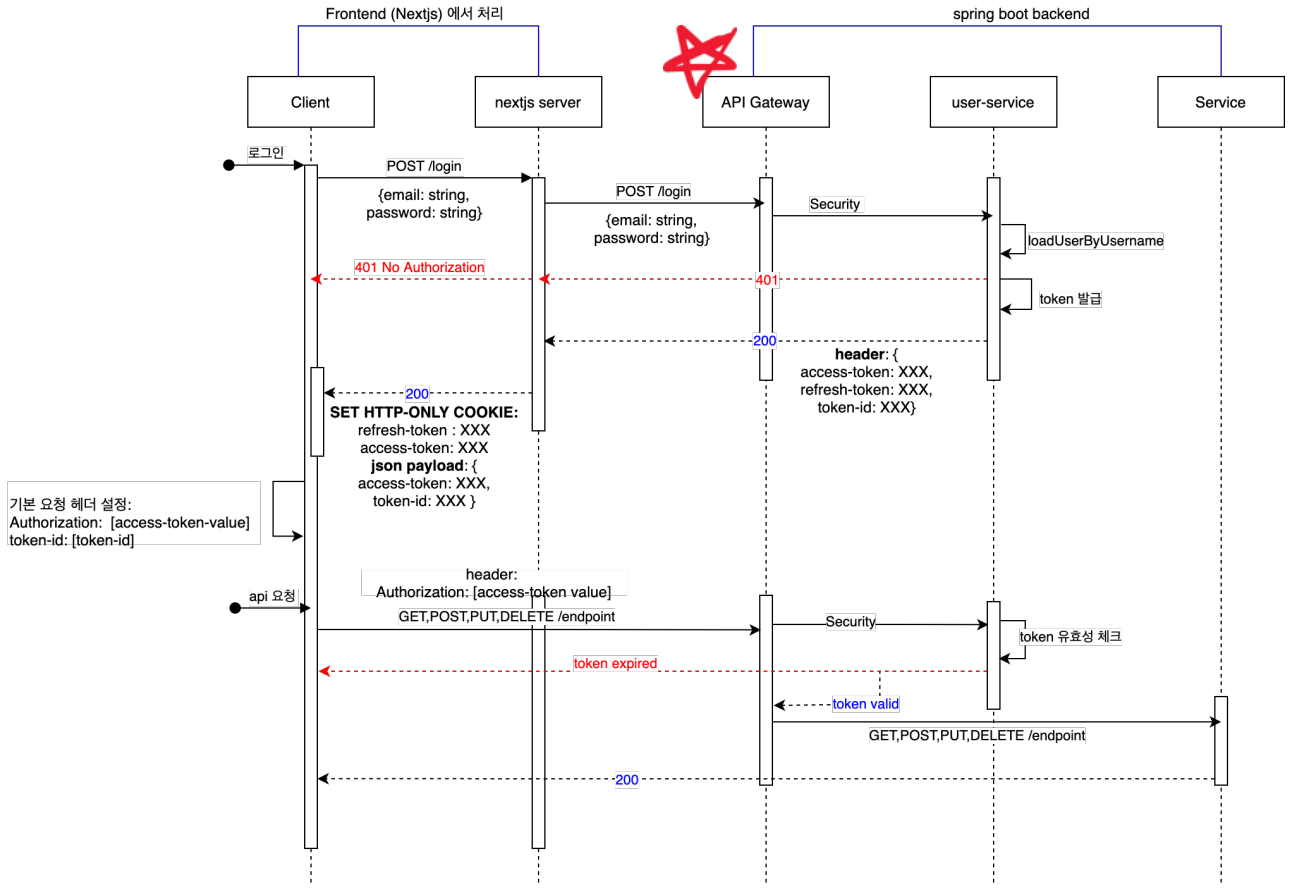
routes 설정 추가하여 uri 에 따라 호출되는 서비스가 달라지도록 구성

apigateway/application.yml

```yaml
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>.*), /$\{segment}
        - id: portal-service
          uri: lb://PORTAL-SERVICE
          predicates:
            - Path=/portal-service/**
          filters:
            - RewritePath=/portal-service/(?<segment>.*), /$\{segment}
```

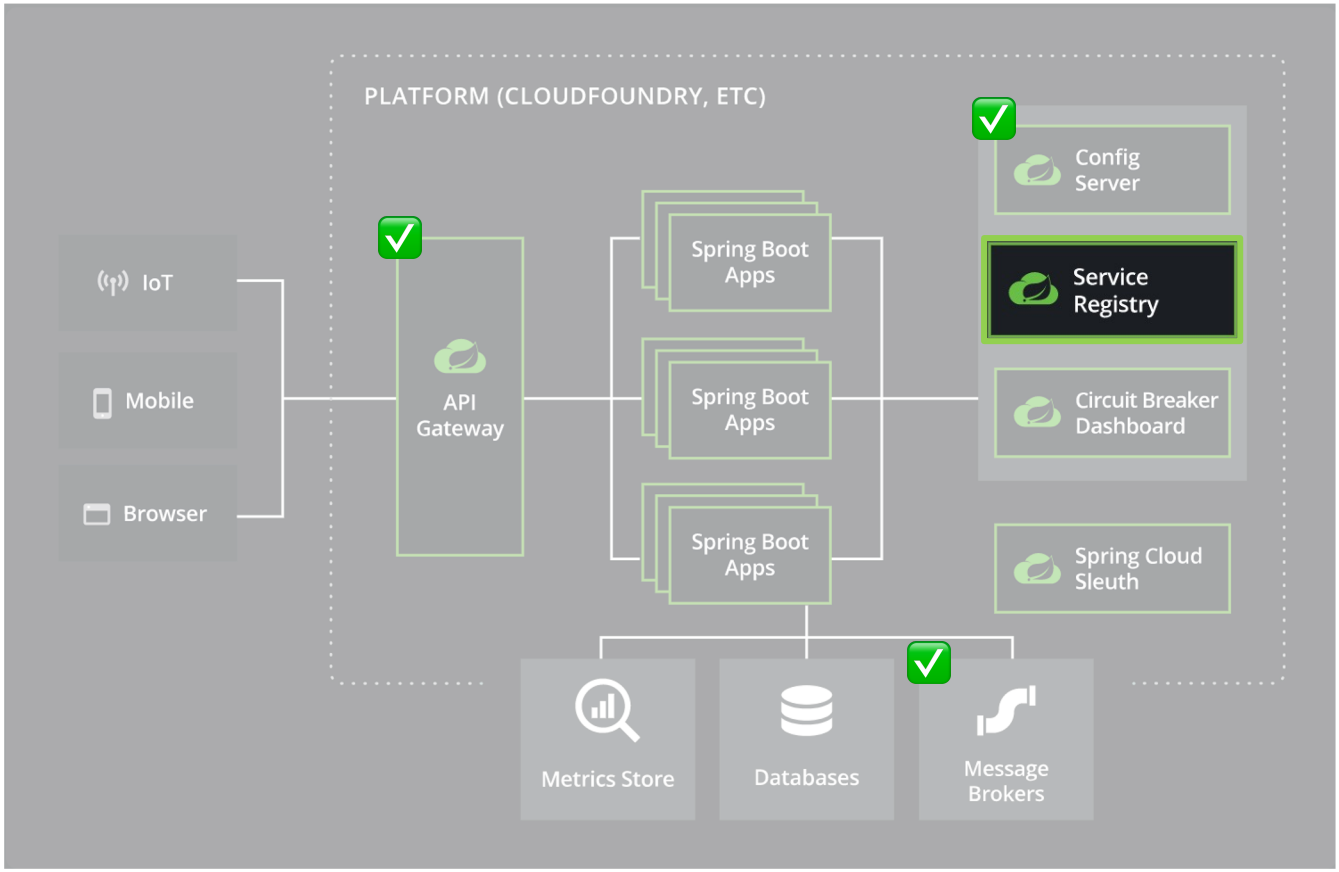
API Gateway – 인증/인가

인증/인가(Authentication/Authorization) – JWT



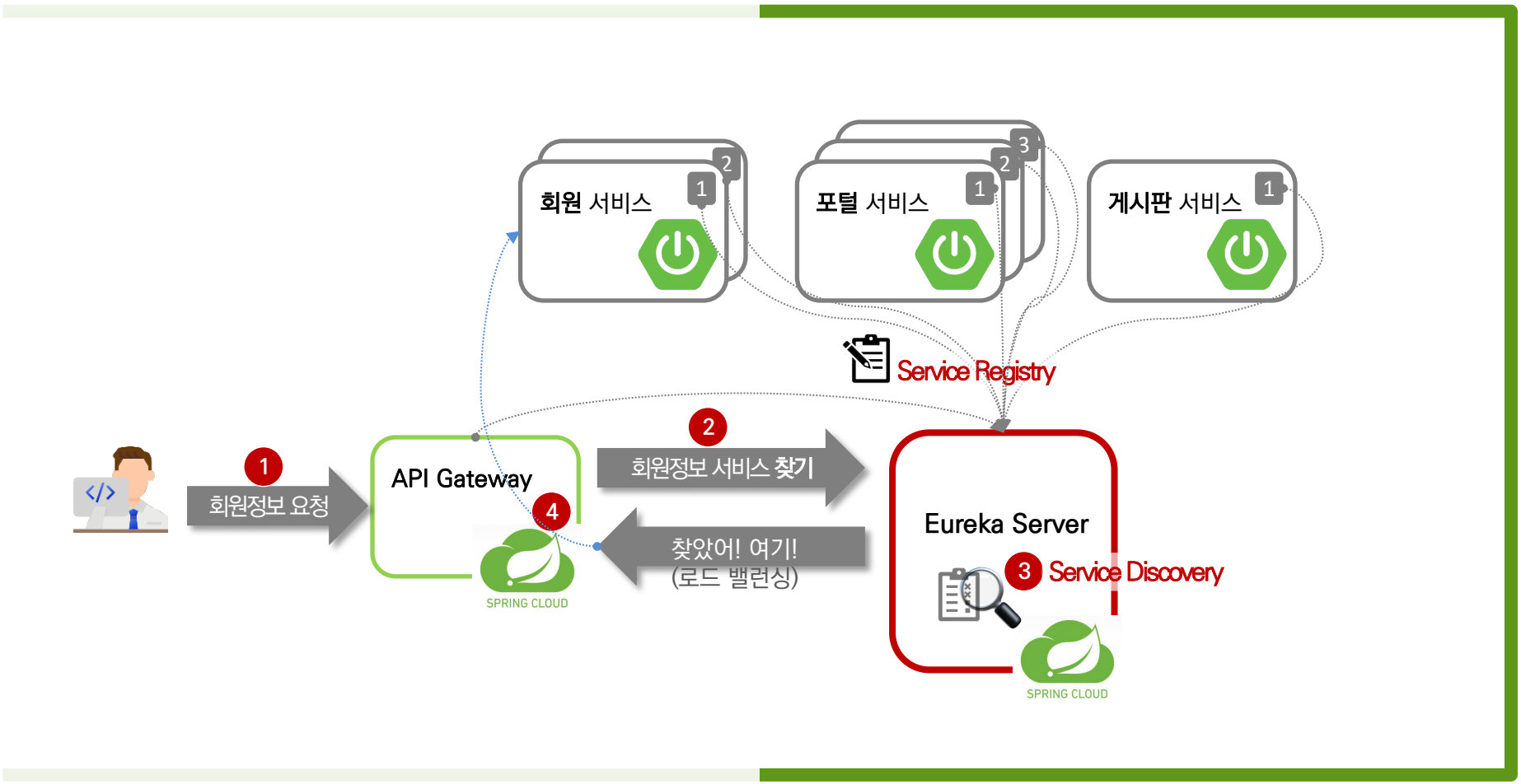
Service Registry/Discovery – Eureka Server

여러 마이크로 서비스들을 등록하여 관리하고
API 요청 시 해당 서비스를 찾아 호출



Service Registry/Discovery – Eureka Server

마이크로 서비스 인스턴스들이 Eureka Server 에 자신을 등록하면 API 요청 시 해당 서비스를 찾아 연결



Service Discovery – Eureka Server

Spring Boot Application 생성 ➔ build.gradle 에 Eureka Server 의존성 추가 ➔ @EnableEurekaServer 선언

build.gradle

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

DiscoveryApplication.java

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

application.yml

```
spring:
  application:
    name: discovery
  security:
    user:
      name: admin
      password: admin
```

Service Registry

build.gradle

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

서비스 Application.java

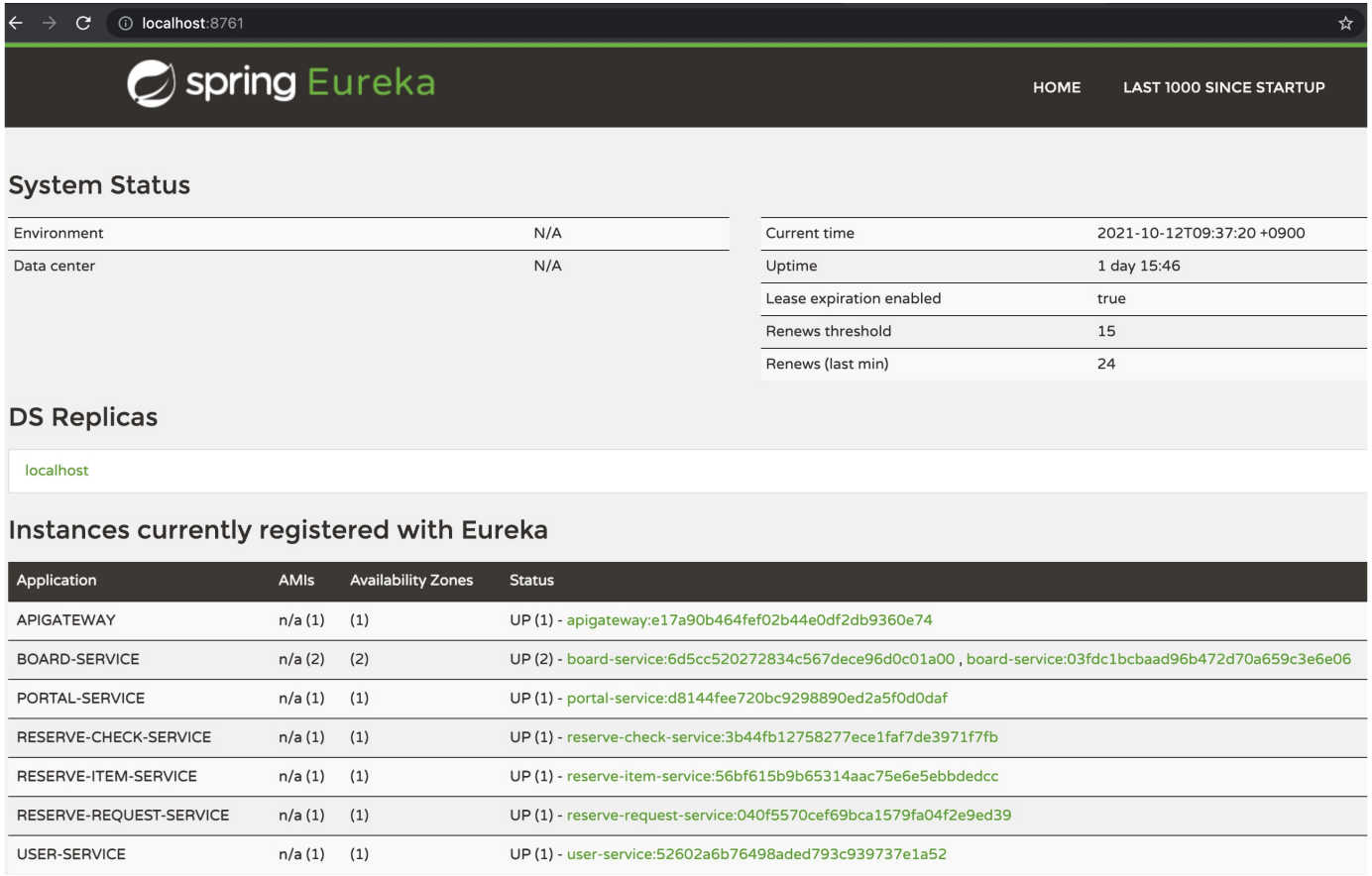
```
@EnableDiscoveryClient
@SpringBootApplication
public class PortalServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(PortalServiceApplication.class, args);
    }
}
```

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
```

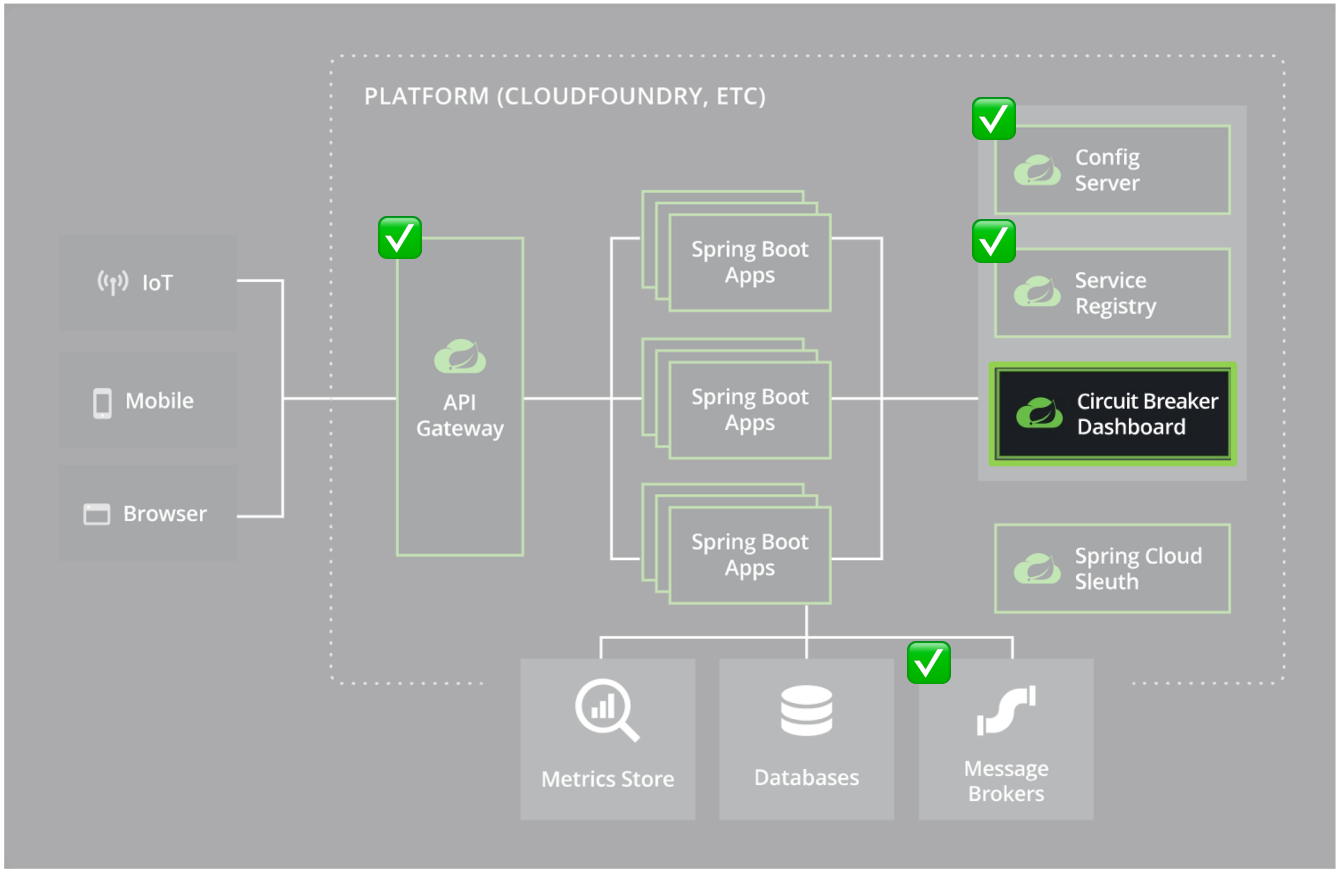
Service Registry/Discovery – Eureka Server

Eureka Server 에 접속해보면 등록된 서비스들을 확인 가능



Circuit Breaker – Resilience4j

서비스의 장애가 다른 서비스에 전파되지 못하도록 해야 한다.



Circuit Breaker – Resilience4j

서비스 간 통신 시 응답 지연 등의 장애 발생 시 요청을 차단 한다.

Circuit Breaker – Resilience4j

build.gradle 에 Circuitbreaker Resilience4j 의존성 추가
➔ Resilience4JConfig 설정

build.gradle

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

Circuit Breaker – Resilience4j

build.gradle 에 Circuitbreaker Resilience4j 의존성 추가
➔ Resilience4JConfig 설정

Resilience4JConfig.java

```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()
        );
    }
}
```

Circuit Breaker – Resilience4j

portal-service 에서 board-service 를 호출하는 경우 portal-service 의 코드

BoardServiceClient.java

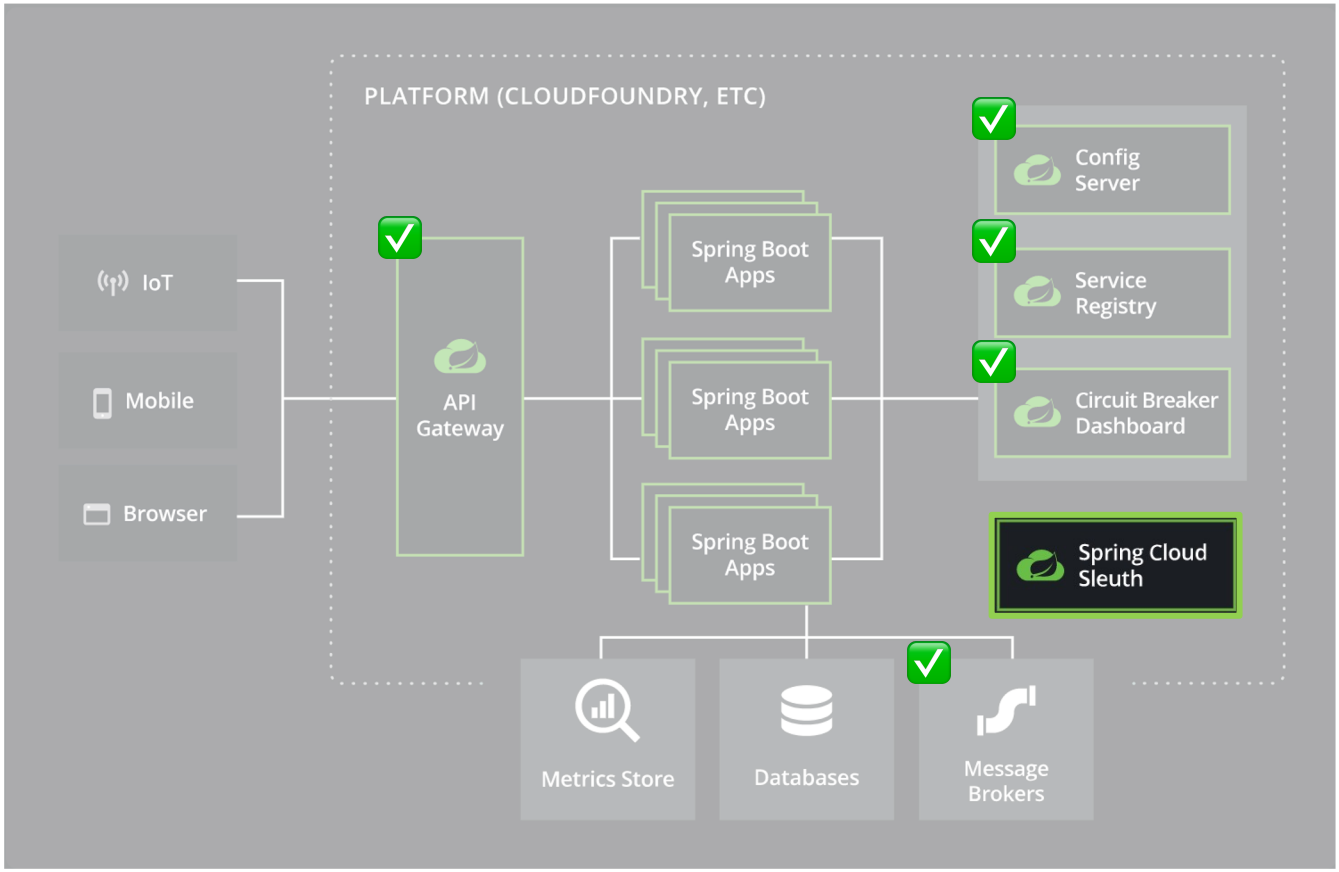
```java
@FeignClient(value = "board-service", configuration = CustomFeignConfiguration.class)
public interface BoardServiceClient {

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

MenuRoleService.java

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

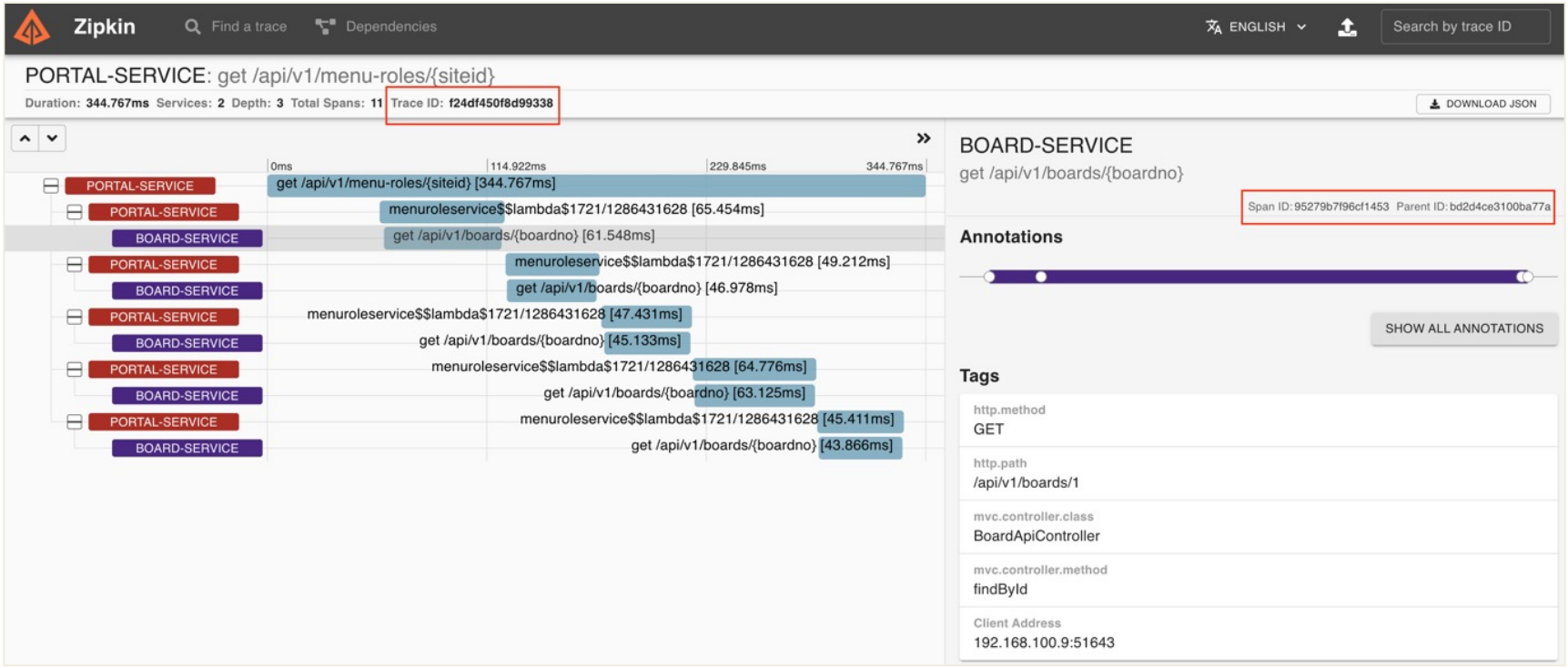
Distributed Tracing – Spring Cloud Sleuth + Zipkin

분산 환경에서 서비스 간 트래픽을 추적하고 분석하여 오류 지점 파악



Distributed Tracing – Spring Cloud Sleuth + Zipkin

Spring Cloud Sleuth 가 Zipkin 서버에 정보를 전송
Zipkin 에 접속하여 서비스 간 트래픽을 추적 하고 파악



Distributed Tracing – Spring Cloud Sleuth + Zipkin

Zipkin 서버 ➔ build.gradle 에 sleuth 의존성 추가

Zipkin dockder run

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

build.gradle

```
dependencies {
  implementation 'org.springframework.cloud:spring-cloud-starter-sleuth'
  implementation 'org.springframework.cloud:spring-cloud-sleuth-zipkin'
}
```

config/application.yml

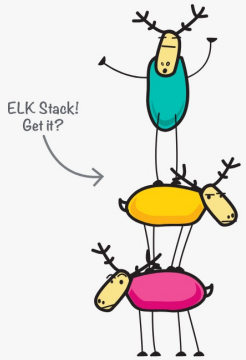
```
spring:
  ...
  zipkin:
    base-url: http://localhost:9411
```

Centralized Logging – ELK

흩어져 있는 로그 들을 한 곳에 모아 시각화 하여 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.



E Elasticsearch

L Logstash

K Kibana

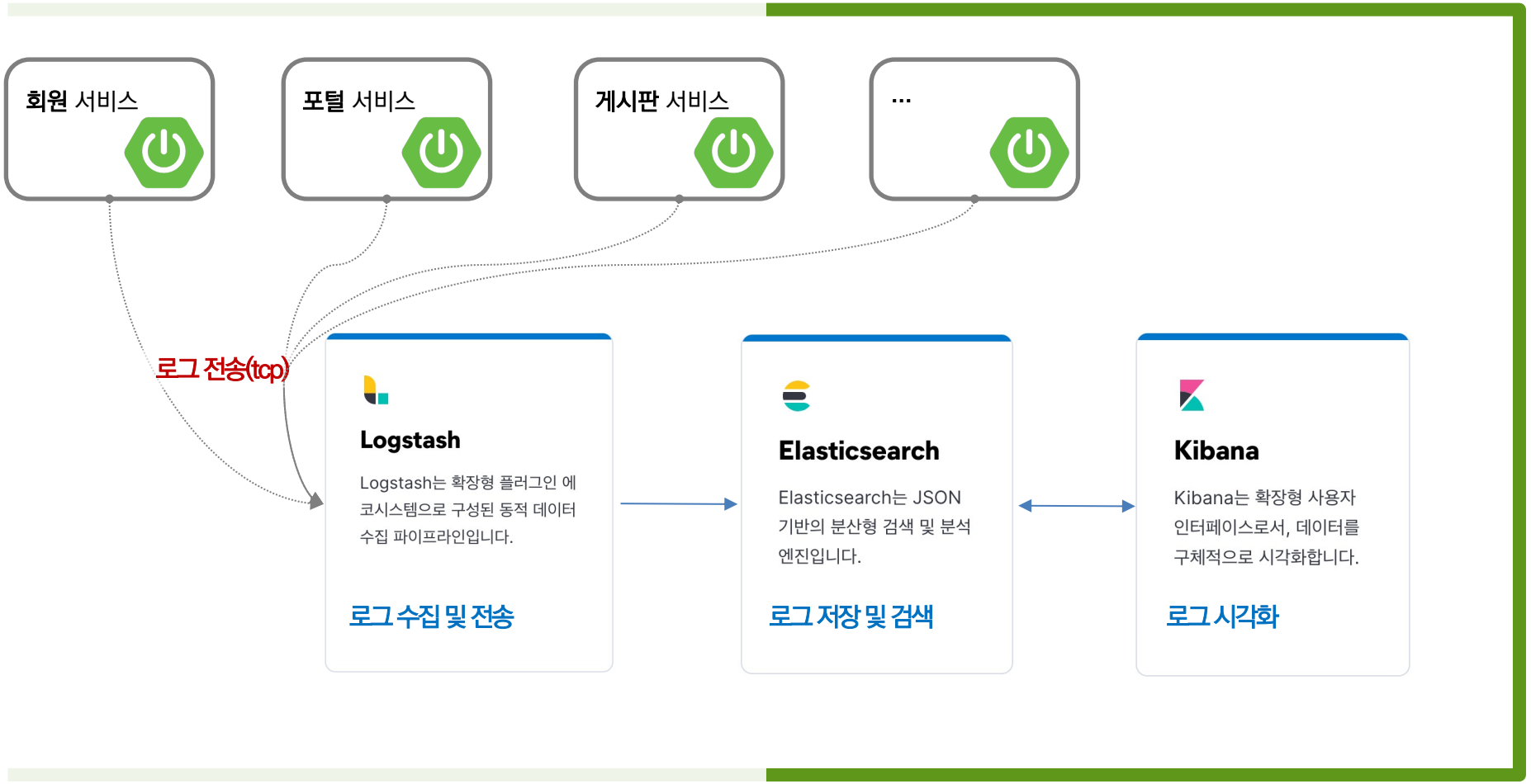
지금은 발전하여 Logstash, Kibana와 함께 제공됩니다

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

Centralized Logging – ELK

각 서비스의 로그를 Logstash 로 전송 ➔ Elasticsearch 저장, 검색 기능제공 ➔ Kibana 에서 로그 확인 및 검색



Centralized Logging – ELK

ELK 스택 ➔ build.gradle 에 logstash 에서 의존성 추가

ELK dockder-compose up

```
$ docker-compose/elk > docker-compose up -d
```

build.gradle

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

Centralized Logging – ELK

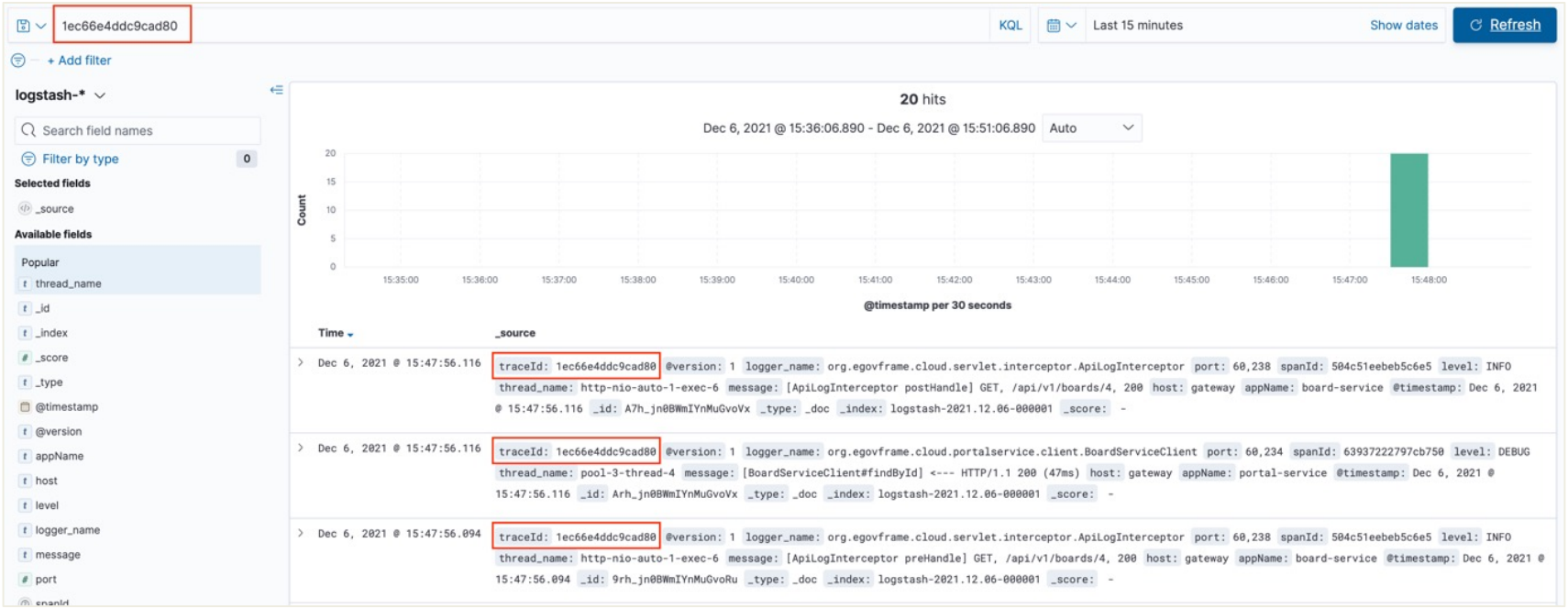
Application 에 Logback 설정하여 Logstash 로 로그 전송

logback-spring.xml

```
<property name="destination" value="${logstash_hostname:-localhost:5001}" />
<property name="app_name" value="${app_name:-portal-service}" />

<!-- ELK - Logstash 로 로그를 전송하기 위한 appender -->
<appender name="LOGSTASH" class="net.logstash.logback.appender.LogstashTcpSocketAppender">
    <destination>${destination}</destination>
    <encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
        <providers>
            <mdc/>
            <context/>
            <logLevel/>
            <loggerName/>
            <pattern>
                <pattern>
                    {"appName": "${app_name}"}
                </pattern>
            </pattern>
            <threadName/>
            <message/>
            <logstashMarkers/>
            <stackTrace/>
        </providers>
    </encoder>
</appender>
<root level="WARN">
    <appender-ref ref="LOGSTASH" />
    <appender-ref ref="STDOUT" />
</root>
```

Centralized Logging – ELK

모아진 로그들을 Kibana 에서 시각화 하고 검색하여 확인



Service Mesh

Service Mesh 시연

클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

MSA 템플릿
배포 환경

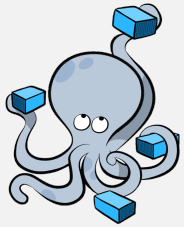
NIA 한국지능정보사회진흥원

MSA 템플릿 배포 환경

On-premise, IaaS, PaaS(K8S, CF) 배포 가능

1단계 온프레미스	2단계 클라우드	3단계 클라우드 네이티브
물리 서버	IaaS	PaaS

전자정부 클라우드 플랫폼



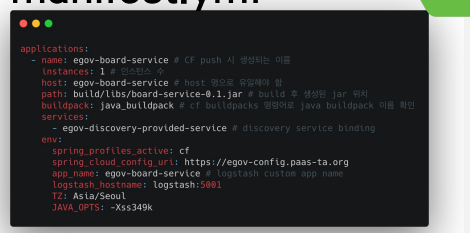
kubernetes

CLOUD FOUNDRY

Cloud Native – CF

CLOUD FOUNDRY

manifest.yml



CF PUSH

CLOUD FOUNDRY

Jenkins

ZIPKIN

docker

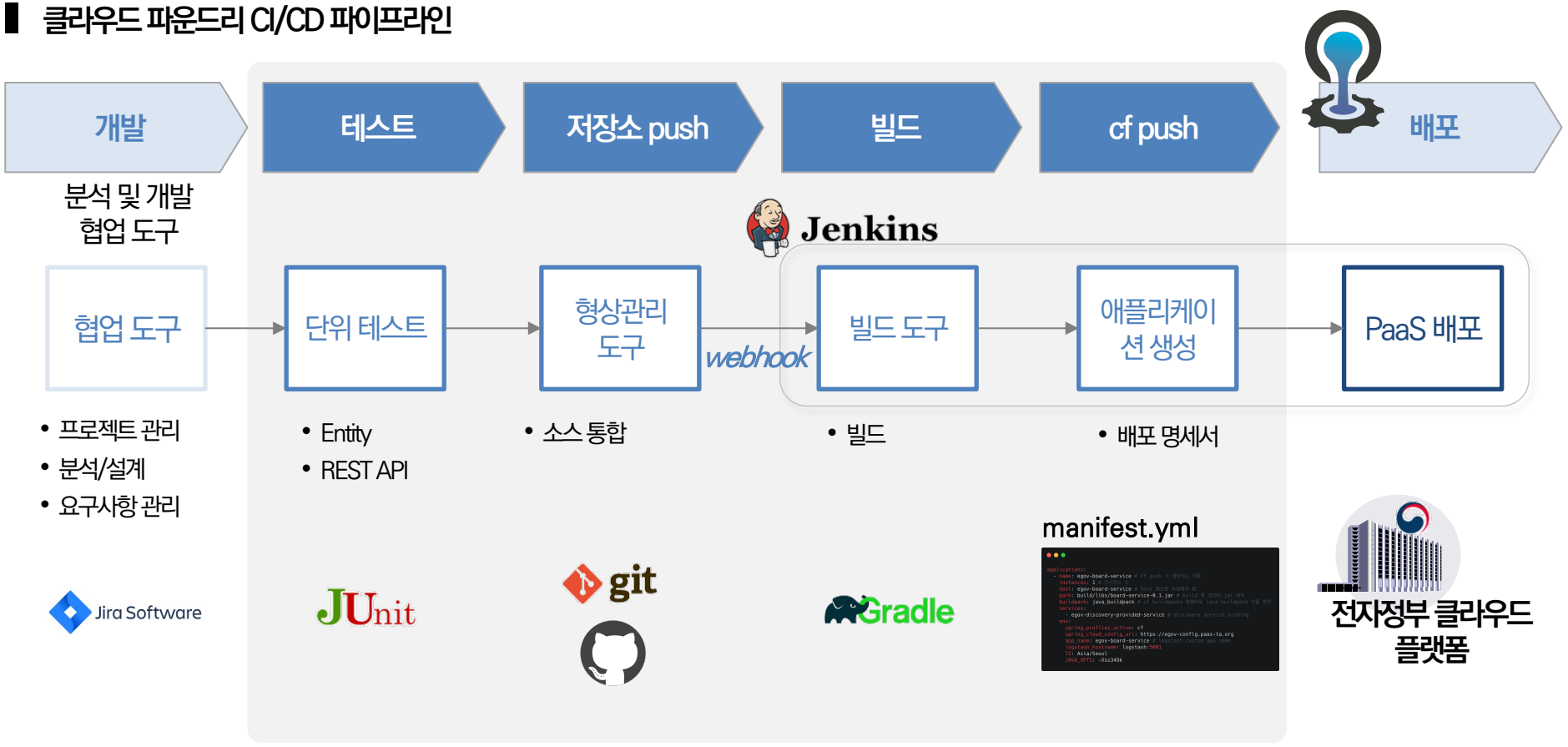
ubuntu

전자정부 클라우드 플랫폼

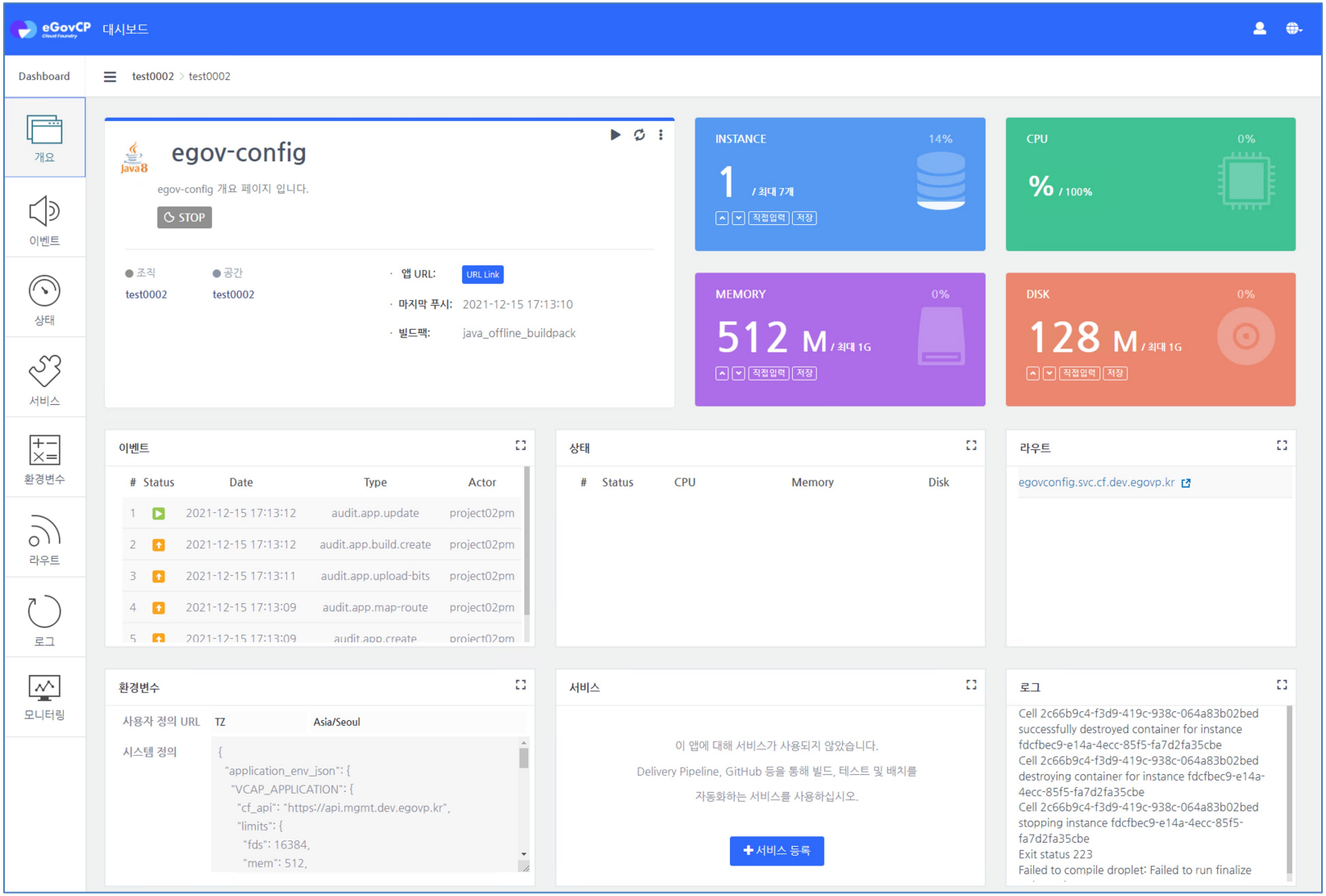
CI/CD 구성 – CF

개발된 소스가 저장소에 올라가면 자동으로 빌드하여 manifest.yml 기준으로 Cloud Foundry 에 서비스가 배포 된다.

클라우드 파운드리 CI/CD 파이프라인

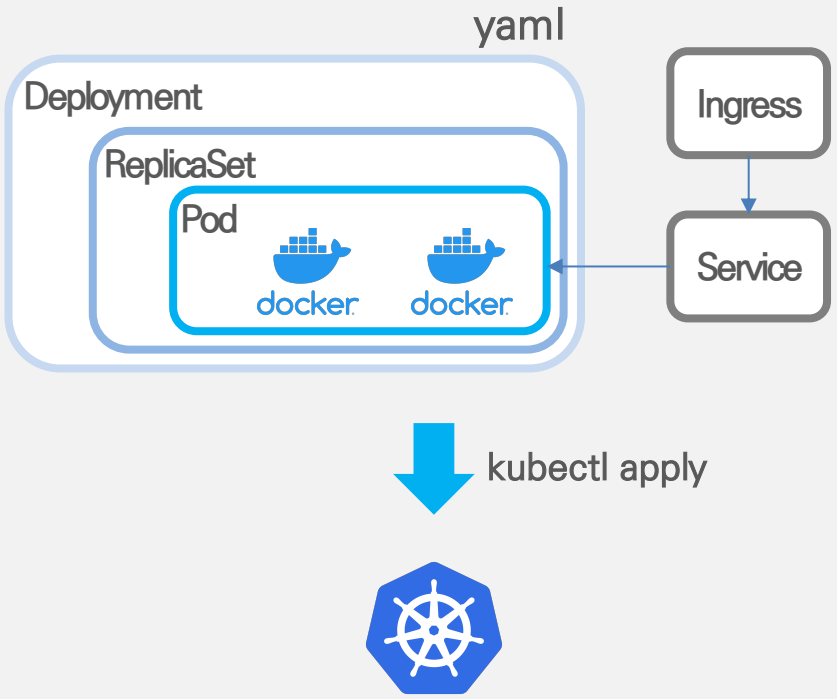


Cloud Native – CF



Cloud Native – K8S

kubernetes



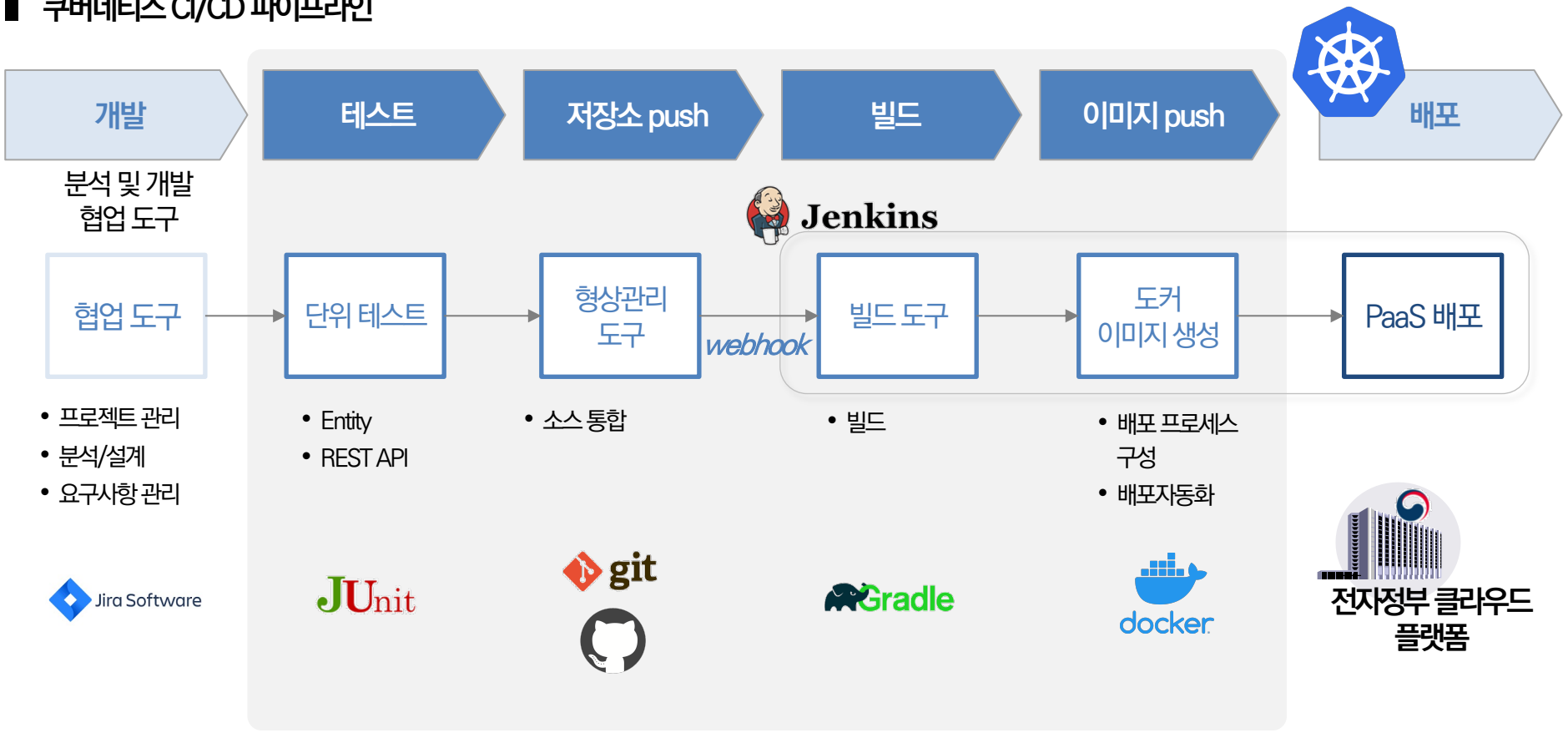
Jenkins
ZIPKIN
docker
ubuntu

전자정부 클라우드 플랫폼

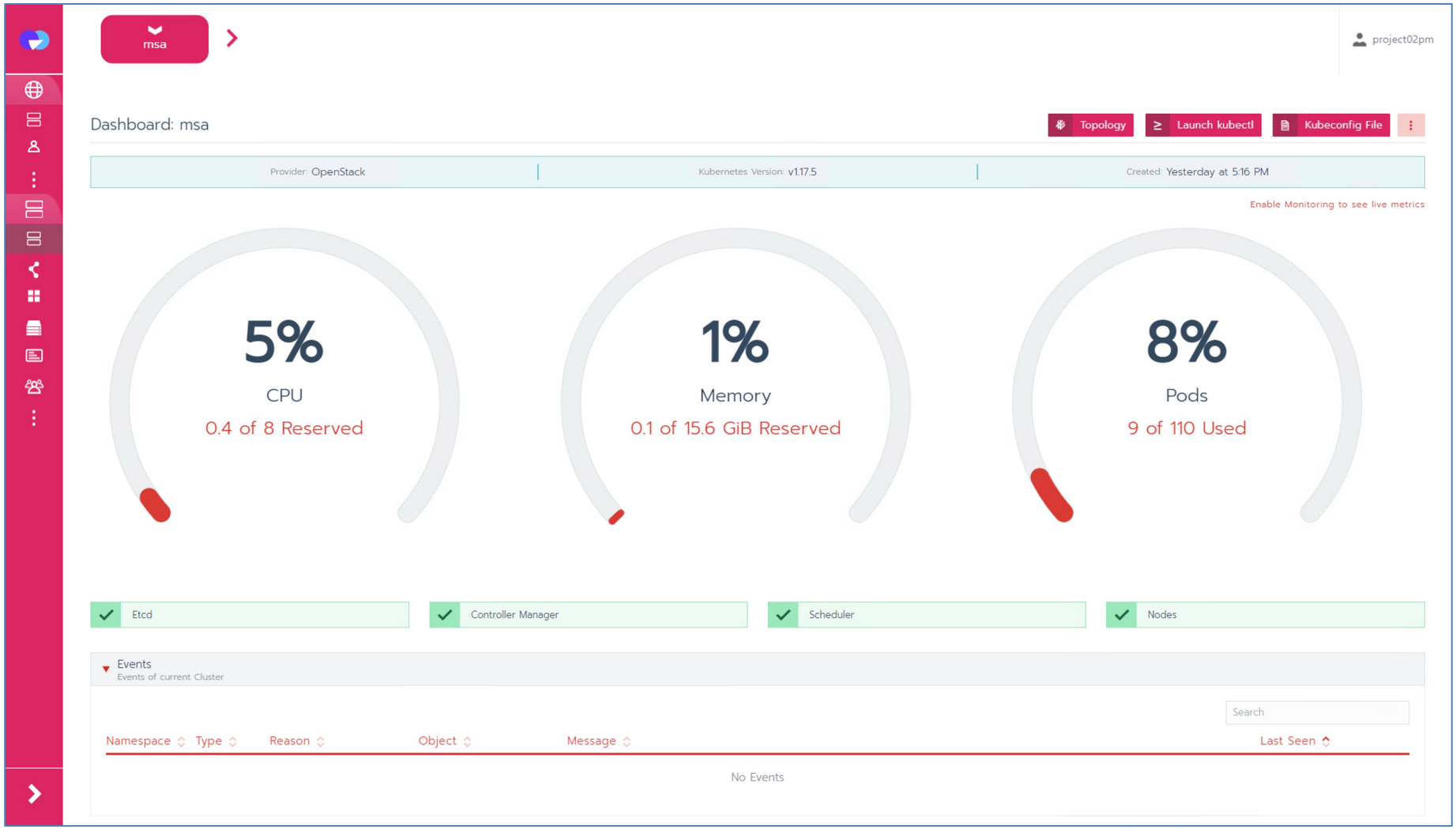
CI/CD 구성 – K8S

개발된 소스가 저장소에 올라가면 자동으로 빌드하여
도커 이미지를 생성하여 도커 레지스트리에 올린 후 서비스가 배포 된다.

쿠버네티스 CI/CD 파이프라인



Cloud Native – K8S



클라우드 네이티브 기반 행정·공공 서비스 확산 지원
MSA 기반의 표준프레임워크 템플릿 제작

감사합니다.